Efficient Byzantine Broadcast in Wireless Ad-Hoc Networks*

Vadim Drabkin Roy Friedman Marc Segal
Computer Science Department
Technion - Israel Institute of Technology
Haifa, 32000
Israel

Email:{dvadim,roy,marcs}@cs.technion.ac.il

Abstract

This paper presents an overlay based Byzantine tolerant broadcast protocol for wireless ad-hoc networks. The use of an overlay results in a significant reduction in the number of messages. The protocol overcomes Byzantine failures by combining digital signatures, gossiping of message signatures, and failure detectors. These ensure that messages dropped or modified by Byzantine nodes will be detected and retransmitted and that the overlay will eventually consist of enough correct processes to enable message dissemination. An appealing property of the protocol is that it only requires the existence of one correct node in each one-hop neighborhood. The paper also includes a detailed performance evaluation by simulation.

Keywords: Byzantine failures, broadcast, ad-hoc networks, unreliable failure detectors.

*This research was supported by the Israeli Science Foundation and by an academic grant from Intel Inc.

1 Introduction

Context of this Study: Wireless ad-hoc networks are formed when an ad-hoc collection of devices equipped with wireless communication capabilities happen to be in proximity to each other [46]. Clearly, each pair of such devices whose distance is less than their transmission range can communicate directly with each other. Moreover, if some devices occasionally volunteer to act as forwarders, it is possible to form a multiple hop ad-hoc network. An important distinguishing element of these networks from "standard" networks is that they do not rely on any pre-existing infrastructure or management authority. Also, due to their ad-hoc nature and device mobility, there is no sub-netting to assist routing and data dissemination decisions. Moreover, due to mobility, the physical structure of the network is constantly evolving.

Semi-reliable broadcast is a basic service for many collaborative applications as it provides nearly reliable dissemination of the same information to many recipients. It ensures that most messages will be received by most of their intended recipients. Yet, implementing semi-reliable broadcast in an efficient manner, and in particular over a wireless ad-hoc network, is far from trivial. It involves ensuring that a message is forwarded to all nodes as well as overcoming possible message losses.

Unlike infrastructure based networks in which routers are usually considered to be trusted entities, in ad-hoc networks routing is performed by the devices themselves. Thus, there is a high risk that some of the nodes of an ad-hoc network will act in a *Byzantine* manner, or in other words, would not respect the networking protocols. This can be due to maliciousness, or simply selfishness (trying to save battery power). Thus, the possibility of having Byzantine nodes in the system motivates the development of Byzantine tolerant broadcast protocols for ad-hoc networks.

The simplest way to obtain broadcast in a multiple hop network is by employing flooding [45]. That is, the sender sends the message to everyone in its transmission range. Each device that receives a message for the first time delivers it to the application and also forwards it to all other devices in its range. While this form of dissemination is very robust, it is also very wasteful and may cause a large number of collisions.

Hence, many multicast/broadcast protocols maintain an *overlay*, which can be thought of as a logical topology superimposed over the physical one, e.g., [25, 40, 47, 48]. The overlay typically covers all nodes, yet each node has a limited number of neighbors. Given an overlay, broadcast messages are flooded only along the arcs of the overlay, thereby reducing the number of messages sent as well as the number of collisions. The overlay composition and structure may be determined by either deterministic or probabilistic methods, and they can change dynamically over time.

On the other hand, having an efficient overlay reduces the robustness of the broadcast protocol against failures, and in particular against Byzantine behavior of overlay nodes. One way around this is to maintain $f + 1$ node independent overlays, where f is the assumed maximal number of Byzantine devices, and flood each message along each of these overlays, guaranteeing that each message will eventually arrive despite possible Byzantine nodes [15, 34, 36].

Of course, the price paid by this approach is that every message has to be sent $f + 1$ times even if in practice none of the devices suffered from a Byzantine fault. In this paper we propose an approach that reduces this overhead to a single overlay when there are no Byzantine failures.

Contribution of this Work: This paper presents an efficient Byzantine tolerant broadcast protocol for wireless ad-hoc networks. The protocol is based on the following principles: The protocol employs an overlay on which messages are disseminated. In parallel, signatures about these messages are being gossiped by all nodes in the system in an unstructured manner. This allows all nodes to learn about the existence of a message even if some of the overlay nodes fail to forward them, e.g., if they are Byzantine or due to

collisions. When a node learns about a message it is missing, it requests the missing message from another node that has it. The benefit of this approach comes from the fact that message signatures are typically much smaller than the messages themselves. Moreover, as gossips are sent periodically, multiple gossip messages are aggregated into one packet, thereby greatly reducing the number of messages generated by the protocol.

Additionally, the protocol employs several failure detectors in order to eliminate from the overlay nodes that act a noticeable Byzantine manner. Specifically, we rely on a *mute* failure detector, a *verbose* failure detector, and a *trust* failure detector. The mute failure detector detects when a process has failed to send a message with an expected header [17, 18]. The verbose failure detector detects when a node sends messages too often. Finally, the trust failure detector reports suspicions of a faulty behavior of nodes based on the other two failure detectors and the history of nodes.[1]

An interesting property of the failure detectors we use is that they only detect benign failures, such as a failure to send a message with an expected header, sending too many messages, or trying to forge a signed message. They do not detect, for example, sending messages with inconsistent data, or sending messages with different data to different processes. Thus, their properties can be detected locally and they can be implemented in an eventually synchronous environment, such as the timed-asynchronous model [16], regardless of the ratio between the number of Byzantine processes and the entire set of processes. Interestingly, combining this with signatures on messages is enough to overcome Byzantine failures.

An important aspect of our failure detector based approach is its modularity, as they encapsulate timing requirements behind a timeless functional specification. The use of failure detectors greatly simplifies the protocol's structure and enables us to present it with an asynchronous design. This is often considered more elegant and robust than synchronous alternatives, in which timing assumptions are explicit.

The result is a protocol that sends a small number of messages when all nodes behave correctly most of the time. The paper also includes a detailed performance evaluation carried by simulation.

Paper's road-map: The model and basic definitions and assumptions are described in Section 2. Section 3 describes the protocol and its proof of correctness, while Section 4 elaborate on the overlay construction. The results of the performance evaluation are given in Section 5. Section 6 compares our work with related work. We conclude with a discussion in Section 7.

2 System Model and Definitions

In this work we focus on wireless mobile systems. Specifically, we assume a collection of *nodes* placed in a given finite size area. A node in the system is a device owning an omni-directional antenna that enables wireless communication. A transmission of a node p can be received by all nodes within a disk centered on p whose radius depends on the transmission power, referred to in the following as the *transmission disk*; the radius of the transmission disk is called the *transmission range*. The combination of the nodes and the transitive closure of their transmission disks forms a wireless ad-hoc network.[2]

We denote the transmission range of device p by r_p. This means that a node q can only receive messages sent by p if the distance between p and q is smaller than r_p. A node q is a *direct neighbor* of another node p

[1] Notice that standard definitions of failure detectors require some properties to hold forever. However, this can be bounded along the lines of [23].

[2] In practice, the transmission range does not behave exactly as a disk due to various physical phenomena. However, for the description of the protocol it does not matter, and on the other hand, a disk assumption greatly simplifies the formal model. At any event, our simulation results are carried on a simulator that simulates a real transmission range behavior including distortions, background noise, etc.

if q is located within the transmission disk of p. In the following, $N(1,p)$ refers to the set of direct neighbors of a node p and $N(k,p)$ refers to the transitive closure with length k of $N(1,p)$. By considering $N(1,p)$ as a relation (defining the set $N(1,p)$), we say that a node p has a path to a node q if q appears in the transitive closure of the $N(1,p)$ relation.

As nodes can physically move, there is no guarantee that a neighbor q of p at time t will remain in the transmission disk of p at a later time $t' > t$. Additionally, messages can be lost. For example, if two nodes p and q transmit a message at the same time, then if there exists a node r that is a direct neighbor of both, then r will not receive either message, in which case we say that there was a *collision*. Yet, we assume that a message is delivered with positive probability.

Each device p holds a private key k_p, known only to itself, with which p can digitally sign every message it sends [44]. It is also assumed that each device can obtain the public key of every other device, and can thus authenticate the sender of any signed message.

Finally, we assume an abstract entity called an *overlay*, which is simply a collection of nodes. Nodes that belong to the overlay are called *overlay nodes*. Nodes that do not belong to the overlay are called *non-overlay nodes*. In the following $OVERLAY$ refers to the set of nodes that belong to the overlay and $OL(1,p) \equiv N(1,p) \bigcap OVERLAY$ (the neighbors of p that belong to the overlay). Later in this paper we give examples of a couple of known overlay maintenance protocols that we adapted to our environment.

2.1 Byzantine Failures

Up to f out of the total of n nodes in the system may be *Byzantine*, meaning that they can arbitrarily deviate from their protocol. In particular, Byzantine processes may fail to send messages, send too many messages, send messages with false information, or send messages with different data to different nodes. We also assume that correct and Byzantine processes are spread such that the transitive closure of the transmission disks of correct nodes form a connected graph (clearly, without this assumption, it is impossible to ensure dissemination of messages to all correct nodes).

Yet, a node cannot impersonate to another node, which is achieved using digital signatures [44].[3] Nodes that follow their protocol are called *correct*. If a node is correct, then it is presumed to be correct throughout the execution of the protocol. A node p that sends a message m is called the *originator* of m. We denote $sig(m)$ to be the cryptographic signature of a message m.

2.2 Failure Detectors and Nodes' Architecture

As already mentioned in the Introduction, we assume that each node is equipped with three types of failure detectors, MUTE, VERBOSE, and TRUST (see also illustration in Figure 1). In this work we assume that each message has a *header* part and a *data* part. The header part can be anticipated based on local information only while the data part cannot. For example, the type of message (application data, gossip, request for retransmission, etc.), the id of the originator, and a sequence number of the message are part of the header. On the other hand, the information that the application level intended to send, or the actual gossiped information, is part of the data.

Based on this, we define a *mute failure* as failure to send a message with an expected header. Similarly, a *verbose failure* is sending messages too often. Note that both types of failures can be detected accurately in a synchronous system based on local knowledge only. This is because in synchronous systems each message

[3] In the implementation of our protocol we use the DSA protocol [44]. Due to space limitations, we do not repeat a discussion about the infrastructure required for this, as this is can be found in many papers and text-books on the use of cryptography.

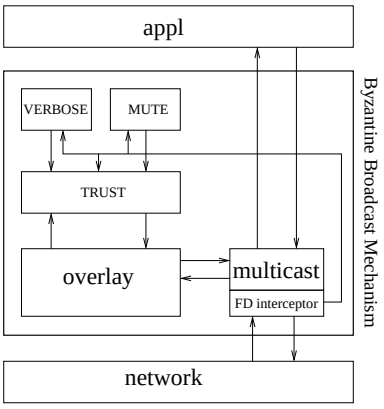

Figure 1: Node Architecture

has a known bounded deadline, so it is possible to tell that a message is missing. Similarly, it is possible to accurately measure the rate of messages received and verify that it is below an agreed upon threshold.

Obtaining synchronous communication in ad-hoc networks with standard hardware and operating systems is extremely difficult. On the other hand, observations of communication networks indicate that they tend to behave in a timely manner for large fractions of the time. This is captured by the notion of the class $\diamondsuit P_{mute}$ of failure detectors [5, 17, 18, 24]. Such failure detectors are assumed to eventually (i.e., during periods of timely network behavior) detect mute failures accurately. In this eventuality, all nodes that suffer a mute failure are suspected (known as *completeness*) and only such nodes are suspected (known as *accuracy*). This approach has the benefit that all synchrony assumptions are encapsulated behind the functional specification of the failure detector (i.e., its ability to eventually detect mute failures in an accurate manner). This also frees protocols that are based on such failure detectors from the implementation details related to timers and timeouts, thus making them both more general and more robust.

In a similar manner to $\diamondsuit P_{mute}$, we can defined $\diamondsuit P_{verbose}$ as a class of failure detectors that eventually reliably detect verbose failure. We assume that the failure detector MUTE is in the class $\diamondsuit P_{mute}$ while VERBOSE is in the class $\diamondsuit P_{verbose}$.

The failure detector TRUST collects the reports of MUTE and VERBOSE, as well as detections of messages with bad signatures and other locally observable deviations from the protocol. In return, TRUST maintains a trust level for each neighboring node. This information is being fed into the overlay, as illustrated in Figure 1. As we describe later in the paper, the information obtained from TRUST is used to ensure that there are enough correct nodes in the overlay so that the correct nodes of the overlay form a connected graph and that each correct node is within the transmission disk of an overlay node that does not exhibit detectable Byzantine behavior.

2.3 The Broadcast Problem

Intuitively, the broadcast problem states that a message sent by a correct node should usually be delivered to all correct nodes. We capture this by the *eventual dissemination* and the *validity* properties. The eventual dissemination property specifies the ability of a protocol to disseminate the message to all the nodes in the system. The validity property specifies that when a correct node accepts a message, then this message was indeed generated by the claimed originator.

Formally, we assume a primitive `broadcast`(p, m) that can be invoked by a node p in order to dis-

seminate a message m to all other nodes in the system, and a primitive `accept`(p,q,m) in which a message claimed to be originated by q is accepted at a node p.

Eventual dissemination: If a correct node p invokes `broadcast`($p, -$) infinitely often, then eventually every correct node q invokes `accept`(q,p,$-$) infinitely often.[4]

Validity: If a correct node q invokes `accept`(q,p,m) and p is correct, then indeed p invoked `broadcast`(p,m) beforehand. Moreover, for the same message m, a correct node q can only invoke `accept`(q,p,m) once.

3 The Dissemination Protocol

As indicated in the Introduction, our protocol includes three concurrent tasks. First, messages are disseminated over the overlay by the overlay nodes. Second, signatures about sent messages are gossiped among all nodes in the system. This allows all nodes to learn about the existence of messages they did not receive either due to collisions or due to a Byzantine behavior by an overlay node. When a node p discovers that it misses a message following a gossip it heard from q, then p requests the missing message from q as well as from its overlay neighbors. The third and final task is the maintenance of the overlay, whose goal is to ensure that the evolving overlay indeed disseminates messages to all correct nodes. Note that the dissemination and recovery tasks are independent of the overlay maintenance. Thus, this section deals with the first two tasks, while the overlay maintenance is described in Section 4. At any event, for performance reasons, overlay maintenance messages can be piggybacked on gossip messages.

As the protocol and overlay rely on failure detectors, we first describe the interface to these failure detectors in Figure 2 and in Section 3.1. The pseudo-code of the main protocol appears in Figures 3 and 4 and is described in detail in Section 3.2. These figures use two primitives. The primitive `broadcast` denotes a broadcast of a message with a given TTL value, i.e., it reaches by flooding all nodes in the corresponding hop distance from the sender. The primitive `lazycast` initiates periodic broadcasting of the given message only to the immediate neighbors of the sender.

3.1 Interfacing with the Failure Detectors

Recall that the goal of the MUTE failure detector is to detect when a process fails to send a message with a header it is supposed to. To notify this failure detector about such messages, its interface includes one method called `expect` (see Figures 1 and 2). This method accepts as parameters a message header to look for, a set of nodes that are supposed to send this message, and an indication if all of these nodes must send the message or only one of them is enough. Note that the header passed to this method can include wild cards as well as exact values for each of the header's fields. In this paper we do not focus on how such a failure detector is implemented. Intuitively, a simple implementation consists of setting a timeout for each message reported to the failure detector with the `expect` method. When the timer times out, the corresponding nodes that failed to send anticipated messages are suspected for a certain period of time (see discussion in [17, 18]).

The goal of the VERBOSE failure detector is to detect verbose nodes. Such nodes try to overload the system by sending too many messages that may cause other nodes to react with messages of their own, thereby degrading the performance of the system. Detecting such nodes is therefore useful in order to allow

[4]Clearly, with this property it is possible to implement a reliable delivery mechanism. In order to bound the buffers used by such a mechanism, it is common to use flow control mechanisms.

MUTE
`expect`(*message_header*,*set_of_nodes*,*one_or_all*)
This method notifies the MUTE failure detector about an expected message.
It accepts as parameters the expected *message_header*,
the *set_of_nodes* that are supposed to send the message,
and a *one_or_all* indication.
The latter parameters indicates if ALL nodes are assumed to send the message
or only ONE of them.

VERBOSE
`indict`(*node_id*)
This method indicts a node with *node_id* for being too verbose
It causes the VERBOSE failure detector to increment the suspicion level of *node_id*.
`implicate`(*message_header*,*set_of_nodes*,*one_or_all*,*node_id*)
This method notifies the VERBOSE failure detector that
if a message with *message_header* is received from ONE or ALL
nodes in *set_of_nodes*, as specified in *one_or_all*
then node *node_id* would be implicated by incrementing its suspicion level.

TRUST
`suspect`(*node_id*,*suspicion_reason*)
This method notifies the TRUST failure detectors that the
level of trust of node *node_id* should be reduced based on the provided *suspicion_reason*.

Figure 2: Failure Detectors' Interface

nodes to stop reacting to messages from these nodes. Similarly to MUTE, the VERBOSE failure detector also gets hints from the broadcast protocol about what would constitute a verbose fault. For this, the interface of VERBOSE exports the methods `indict` and `implicate`. The first method simply indicts a process that has sent too many messages of a certain type. The second method, `implicate`, is used when a verbose failure can only be noticed based on what other messages have been sent by other nodes. Thus, it tells the VERBOSE failure detector to suspect a given node only if some other nodes have sent messages with a given message; the existence of such messages "prove" that the node in question has generated an unnecessary message.

Practically, we assume that VERBOSE maintains a counter for each node that was listed in any invocation of one of its methods. The counter is incremented on each such event, and after a given threshold, the node is considered to be a suspect. VERBOSE also includes a method that allows to specify general requirements about the minimal spacing between consecutive arrivals of messages of the same type. Such a method is typically invoked at initialization time. As it it is not directly accessed by our protocol's code, we do not discuss it any further.

Finally, the TRUST failure detector maintains a trust level for any node known to it. Each time its `suspect` method is called, the trust level of the corresponding node is decreased by some number that depends on the suspicion reason. Once it goes below a threshold, the corresponding node is suspected. Moreover, to recover from mistakes, the trust level slowly grows, e.g., every few time units without `suspect` being invoked again (such an aging mechanism also exists in the MUTE and VERBOSE failure detectors).

```
Upon send(msg) by application do
    message := msg_id||node_id||msg||sig(msg_id||node_id||msg);
    gossip_message := msg_id||node_id||sig(msg_id||node_id);
    broadcast(message,DATA,ttl=1);
    lazycast(gossip_message,GOSSIP,ttl=1);

Upon receive(message,DATA,ttl) sent by p_j do
    if (have not received this message before) then
        if (authenticate-signature(message) = TRUE) then
            Accept(p_i,p_j,message) /* forward it to the application */;
            if (current_node ∈ OVERLAY) then
                broadcast(message,DATA,1);
            else /* the message is correct and I am not in the overlay */;
                if (ttl > 1) then
                    broadcast(message,DATA,ttl-1);
                endif;
            endif;
            if (already received a gossip_message about message before) then
                lazycast(gossip_message,GOSSIP,ttl=1);
            endif;
        else/* the message is not correct */;
            TRUST.suspect(p_j,"bad_signature_reason"); /* notify the trust failure detector */
        endif;
    endif;

Upon receive(gossip_message,GOSSIP,ttl) sent by p_j: do
    if (authenticate-signature(gossip_message) = TRUE) then
        if (there is no message that fits the gossip_message) then
            /* The node asks from the node that sent the gossip message or from overlay nodes to */
            /* send the real message */ ;
            broadcast(gossip_message,REQUEST_MSG,ttl=1,p_j);
            if (#messages with same signature (from different nodes) ≥ sig_proofs_threshold)
            and (the overlay neighbors have not sent the message) then
                broadcast(gossip_message,REQUEST_MSG,2,NULL);
                MUTE.expect(gossip_message.header,OL(2,current_node),ANY);
            endif;
        else /* the message that fits the gossip_message was received */ ;
                if (gossip_message have not been sent yet) then
                    lazycast(gossip_message,GOSSIP,ttl=1);
            endif;
        endif;
    else/* the message is not correct */;
            TRUST.suspect(p_j,"bad_signature_reason");
    endif;
```

Figure 3: Byzantine Dissemination Algorithm

Upon `receive`(*missing_message,REQUEST_MSG,ttl,*p_k) sent by p_j **do**
  if (`authenticate-signature`(*missing_message*) = TRUE) **then**
    if (*ttl* > 1) **then**
      `broadcast`(*gossip_message,REQUEST_MSG,ttl-1,*p_k);
    endif;
    if ($current\_node \in OVERLAY$) **then**
      if (a *message* that matches *missing_message* was received) **then**
        VERBOSE.`indict`(p_j);
        `broadcast`(*message,DATA,ttl=1,*p_j);
      else /* *the message that fits the gossip_message was not received* */;
        if (#REQUEST_MSG messages for the same *missing_message* from different nodes $\geq$ missing_msg_threshold) **then**
          message := `msg_id||node_id||REQUEST_MSG's||sig(msg_id||node_id||REQUEST_MSG's from other nodes)`;
          `broadcast`(*message,FIND_FAULTY_MSG,2*);
        endif;
      endif;
    /* *otherwise, current_node is not an overlay node* */
    elseif ((a *message* that matches *missing_message* was received) **and** (*current_node* = p_k)) **then**
      VERBOSE.`indict`(p_j);
      `broadcast`(*message,DATA,ttl=1,*p_j);
    endif;
  else/* *the message is not correct* */;
    TRUST.`suspect`(p_j,"bad_signature_reason");
  endif;

Upon `receive`(*faulty_message,FIND_FAULTY_MSG,ttl*) sent by p_j **do**
  if (`authenticate-signature`(*missing_message*) = TRUE) **then**
    if (*ttl* > 1) **then**
      `broadcast`(*faulty_message,FIND_FAULTY_MSG,ttl-1*);
    endif;
    if ($current\_node \in OVERLAY$) **then**
      if (a *message* that matches *faulty_message* was received) **then**
        /* *The overlay node that has a message will send it to all the 2 hop neighbors of* p_j, */;
        /* *in order to prevent overlay neighbors to suspect each other* */
        if ($p_j \in OL(1, current\_node)$) **then**
          `broadcast`(*message,DATA,1*);
          VERBOSE.`indict`(p_j);
        else
          `broadcast`(*message,DATA,2*);
          VERBOSE.`implicate`(*faulty_message.header,*OL(1,*current_node*),ALL,p_j);
          MUTE.`expect`(*faulty_message.header,*OL(1,*current_node*),ALL);
        endif;
      else /* *if we never heard of the message locally, continue to search it recursively* */
        `broadcast`(*message,FIND_FAULTY_MSG,2*);
      endif;
    endif;
  else/* *the message is not correct* */;
    TRUST.`suspect`(p_j,"bad_signature_reason");
  endif;

Figure 4: Byzantine Dissemination Algorithm – continued

3.2 The Main Protocol

3.2.1 The Dissemination Task in Detail

Dissemination consists of the following steps (described from the point of view of a node p): (1) The originator p of a message m sends $m||sig(m)$ to all nodes in $N(1,p)$. The header part of m includes a sequence number and the identifier of the originator. (2) The originator p of m then gossips $sig(m)$ to all nodes in $N(1,p)$. (3) When a node p receives a message m for the first time, then p first verifies that $sig(m)$ matches m. If it does, then p accepts m. Moreover, if p is also an overlay node, then p forwards m to all nodes in $N(1,p)$. However, if m does not fit $sig(m)$, then m is ignored and the process that sent it is suspected by the TRUST failure detector. (4) If a node p receives a message m it has already received beforehand, then m is ignored.

3.2.2 Gossiping and Message Recovery in Detail

Intuitively, the idea here is that nodes gossip about messages they received (or sent) to all their neighbors. This way, if a node hears a gossip about a message that it has never received, it can explicitly ask the message both from its overlay neighbor and from the node from which it received the gossip. If any of the contacted nodes has the message, it forwards it to the requesting node. Messages can be purged either after a timeout, or by using a stability detection mechanism. In this work, we have chosen to use a timeout based purging due to its simplicity.

Additionally, there are several mechanisms in place to overcome Byzantine failures (in addition to signatures that detect impersonations). In order to prevent a Byzantine overlay node from blocking the dissemination of a message, searching a missing message can be initiated by limited flooding with TTL 2, which ensures that the recovery request will reach beyond a single Byzantine overlay node. This, in addition to requesting the message from the process that gossiped about its existence. Also, when a node feels that it received a request for a missing message too often, or that such a request is unjustified, it notifies the VERBOSE failure detector about it.

More accurately, the gossiping and message recovery task is composed of the following subtasks:

1. When a node p receives a gossip $header(m)$ for a message m it has already received before, then p gossips $header(m)$ with the other $N(1,p)$ nodes. Otherwise, p ignores such gossips. In particular, p only gossips about messages it has already received and does not forward gossips about messages it has not receive yet. This is done in order to make the recovery process more efficient, and in order to help detect mute failures more accurately.[5]

2. When p receives a gossip $header(m)$ for a message m it has not received, p asks its overlay neighbors and the sender of the gossip to forward m to it using a REQUEST_MSG message.

3. If p receives $header(m)$ messages from more than *sig_proofs_threshold* other nodes, or when some timeout has passed since p got the first $header(m)$ message, yet p still has not received m, then p asks the overlay neighbors in 2 hop distance (using flooding with TTL=2) as well as one of the nodes q from which p received $header(m)$ to forward m to it. Essentially, this situation is likely to happen if the nearest overlay neighbor, or its neighbor, is mute. By approaching the overlay neighbors at distance 2, we can bypass such mute overlay nodes. Note that if p does not receive the requested

[5] It is possible to piggyback the first gossip of a message by the sender and by overlay nodes on the actual message. This saves one message and makes the recovery of messages a bit faster, since gossips about messages advance slightly faster this way. For clarity of presentation, we separate these two types of messages in the pseudo-code.

message, then eventually this will trigger the MUTE failure detector at p to suspect p's 2 hop overlay neighbors (which in return will eventually lead to electing another node to the overlay).

4. When an overlay node p receives a REQUEST_MSG for the same message m too many times from the same node q, it causes p's VERBOSE failure detector to suspect q.

5. When an overlay node p receives a REQUEST_MSG for the same message m from different nodes more than a threshold (*missing_msg_threshold*) of times, or a timeout has passed since it received the first such message, yet p has not received m, then p sends a FAULTY_MSG message to nodes in $OL(2, p)$ asking them to find the Byzantine node that is not forwarding m. (The message is sent to overlay nodes at distance 2 in order to bypass a potential neighboring Byzantine node.)

6. When an overlay node p receives a FAULTY_MSG message for a message m that it does not have yet, then p forwards m to the nodes in $OL(2, p)$. This is in order to make sure that if its overlay neighbor is mute, it will be detected and eventually replaced.

7. When an overlay node p receives a FAULTY_MSG message for m from a node $q \in N(1, p)$ and p has m, then p will broadcast m (to q and its other immediate neighbors). There are two reasons why p forwards the message in a broadcast and not using a point-to-point message: (1) if one of p's neighbors was missing the message, it is likely that many of them miss the message, and (2) as listed in Item 8 below, if any of the overlay neighbors r of p has forwarded m to p and does not hear p forwarding it again, then r will suspect that p is mute.

8. When an overlay node p receives a FAULTY_MSG message for m from a node $q \notin N(1, p)$ and p has m, then p first broadcast m. Following this, p instructs its MUTE failure detector to expect a retransmission of m by all its overlay neighbors. Failure by any of them to do so will eventually lead the MUTE failure detector of p to suspect such a node.

3.3 Correctness Proof

Let us remind the reader that in Section 2.1 we assumed that there are enough correct nodes so that non-Byzantine nodes form a connected graph. With this assumption, we prove the following validity and eventual dissemination properties.

Theorem 3.1 *The protocol satisfies the validity property.*

Proof: According to the protocol, the originator of a message m adds a signature $sig(m)$ and then disseminates the message $m||sig(m)$ to other nodes. Note that on receiving of $m||sig(m)$, every correct node checks if $sig(m)$ corresponds to m before the node accepts m. As a part of the model's basic assumptions, a Byzantine node cannot forge signatures. Therefore, no correct node will accept a message other than m as if it was m. Moreover, according to the protocol, correct nodes filter duplicates of messages they have already received. ■

Theorem 3.2 *The protocol satisfies the eventual dissemination property.*

Proof: We show that a message m that is sent infinitely often by a correct originator p is disseminated to all the correct nodes. Assume, by way of contradiction, that there is message m that is not received by some correct process. Let k be the smallest number such that there exists a correct node $q \in N(k, p)$ that does not receive m.

Recall that by assumption, the correct nodes form a connected graph. Therefore there exists a correct node $l \in N(k-1,p)$ such that the distance between q and l is smaller than r_l and l received m. According to the protocol, l will send a gossip about m to its neighbors and if requested by its neighbors, l will also send m. Thus, q will receive m either from its overlay node or from l. This is a contradiction to the assumption about the minimality of k. ■

In the following, we show that eventually, if the *MUTE* failure detector is indeed $\in \Diamond P_{mute}$, then messages are disseminated to all correct nodes by the overlay. The significance of this is that dissemination along overlay nodes is fast, since it need not wait for the periodic gossip mechanism.

Claim 3.3 *Assume that the* MUTE *failure detector* $\in \Diamond P_{mute}$*. Then eventually the non-mute overlay nodes form a connected graph* COL *such that every correct node is either in* COL*, or within the transmission range of a non-mute node in* COL*.*

Proof: Eventually, $\Diamond \mathcal{P}_{mute}$ of all correct nodes will suspect all the mute nodes. Thus, the goodness number in the overlay maintenance protocol for mute nodes will be lower than all other nodes. Consequently, the overlay built by the maintenance protocol will have the desired property. ■

Claim 3.4 *Every Byzantine overlay node is eventually suspected if it does not forward a message* m *that is sent by* s *infinitely often and if the* MUTE *failure detector* $\in \Diamond P_{mute}$*.*

Proof: Let p be the first correct overlay node that does not receive m. Let q be the first Byzantine overlay node between s and p that does not forward m. We assume that q is not forwarding m and we will show that q will be eventually suspected. The route between s and q is either $s->s_1->...->s_i->q$ when $i \geq 1$ or $s->q$. The proof for both cases is similar, so we will examine the more complicate case ($s->s_1->...->s_i->q$ when $i \geq 1$).
Let Q = {the non-overlay nodes $\in N(1,q)$} and S_i = {the non-overlay nodes $\in N(1,s_i)$}. Since q is the first overlay node that is not forwarding m, there exist at least one correct node in Q that will not get m. The correct nodes in S_i receive m from s_i and will eventually forward $sig(m)$ to nodes in Q. After receiving $sig(m)$, every correct node in Q that has not received m from q will activate its MUTE failure detector and eventually q will be suspected. ■

Claim 3.5 *Eventually, when there are no collisions, most messages propagate to all the nodes via the overlay nodes, if the* MUTE *failure detector* $\in \Diamond P_{mute}$*.*

Proof: Claim 3.4 shows that eventually every mute overlay node will be suspected. In Claim 3.3, we showed that eventually, the non-mute nodes of the overlay form a connected graph that covers all non-mute nodes. Therefore, eventually, all messages are propagated by overlay nodes to all correct nodes, which proves the claim. ■

4 Overlay Maintenance

Overlay maintenance is carried by a distributed protocol. There is no global knowledge and each node must decide whether it considers itself an overlay node or not. Thus, the collection of overlay nodes is simply the set of all nodes that consider themselves as such. At the same time, every correct overlay node periodically publishes this fact to its neighbors, so in particular, each overlay node eventually knows about all its correct overlay neighbors.

The goal of the protocol is to ensure that indeed the overlay can serve as a good backbone for dissemination of messages. This means that eventually between every pair of correct nodes p and q there will be a path consisting of overlay nodes that do not exhibit externally visible Byzantine behavior. At the same time, for efficiency reasons, the overlay should consist of as few nodes as possible.

For scalability and resiliency reasons, we are interested in a self-stabilizing distributed algorithm in which every node decides whether it participates in the overlay based only on the knowledge of its neighbors. Recall that the neighbors of p are the nodes that appear in the transmission disk of p. Thus, p can communicate directly with them and every message p sends is received by all of them. Additionally, we would like to influence the overlay construction process such that the overlay nodes will be the best nodes under a given metric. For example, since in mobile systems nodes are often battery operated, we may wish to use the energy level as the metric in order to have the nodes with highest energy levels members of the overlay. Alternatively, we might use bandwidth, transmission range, or local storage capacity, or some combination of several such metrics.

Following the work of [21], we define the *goodness number* as a generic function that associates each node with a value taken from some ordered domain. The goodness number represents the nodes appropriateness to serve in the overlay. This way, it is possible to compare any two nodes using their goodness number and to prefer to elect the one whose value is highest to the overlay. For example, it is easy to evaluate and compare the battery level of nodes. Intuitively, the idea is that a process that believes that it has the highest goodness number among its neighbors that are not covered by another overlay node elects itself to the overlay.

The protocol for deciding if a node should be in the overlay consists of computation steps that are taken periodically and repeatedly by each node. In each computation step, each node makes a local computation about whether it thinks it should be in the overlay or not, and then exchanges its local information with its neighbors. For simplicity, we concentrate below on the local computation steps only.

Each node has a local *status*, which can be either *active* or *passive*; active means that the node is in the overlay whereas passive means that it is not. The local state of each node includes a status (active or passive), its goodness number, and its knowledge of the local states of all its neighbors (based on the last local state they reported to it). Also, for each neighbor, the list of its active neighbors. We assume that these messages are signed as well.

Moreover, to ensure the appropriateness of the overlay, we need to ensure that the overlay includes alternatives to each detected mute or verbose node. Ideally, we would like to eliminate these nodes from the overlay, but as they are Byzantine, they may continue to consider themselves as overlay nodes. Thus, the best we can do is make sure that there is an alternative path in the overlay that does not pass through such nodes, and that correct nodes do not consider mute and verbose nodes as their overlay neighbors. So we refine the intuitive notion of a goodness number by ensuring that a node elects itself to the overlay if it has the highest goodness number among its trusted neighbors. Below we list a couple of overlay maintenance protocols that realize this intuition.

Specifically, we have implemented two overlay maintenance protocols, namely the *Connected Dominating Set* (CDS) and the *Maximal Independent Set with Bridges* (MIS+B) of [21], augmented with trust levels.[6] For lack of space, and since other than adding the trust level, the protocols are the same as in [21], we do not repeat them here.

[6]The CDS and MIS+B protocols in [21] are in fact self-stabilizing generalizations of the work of [48].

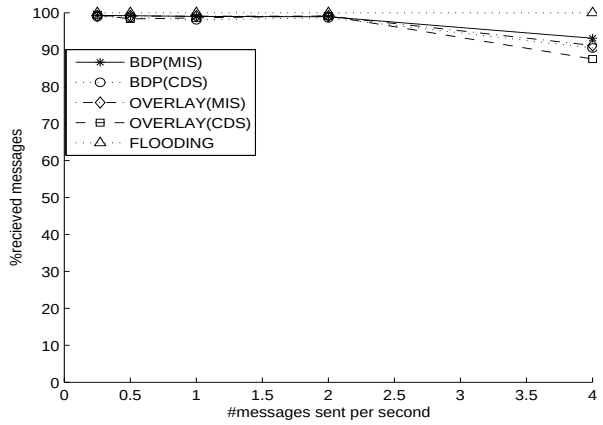

Figure 5: Message delivery ratio when all nodes are static

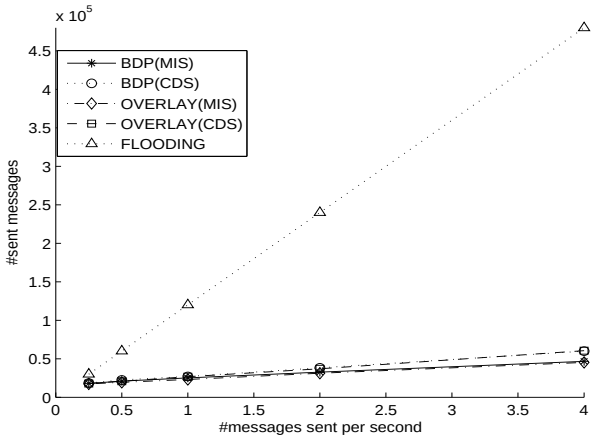

Figure 6: Network load in terms of total number of messages sent when all nodes are static

5 Results

We have measured the performance of our protocol using the SWANS/JIST simulator [1]. In the simulations, we have compared the performance of our protocol with the performance of flooding on one hand and of simple dissemination along an overlay (without recovery of lost messages). Here, flooding is an example of a very robust protocol against maliciousness, but also very wasteful. At the other extreme, dissemination along an overlay without message recovery is very efficient, but very unreliable as well. We have measured the percentage of messages delivered to all nodes, the latency to deliver a message to all and to most of the nodes, and the load imposed on the network. It is also important to note that our performance measurements included the overhead of the overlay maintenance as well as the gossip messages (although overlay maintenance are piggybacked on gossip messages).

In order to reduce the number of collisions, we have employed a staggering technique. That is, each time a node is supposed to send a message, it delays the sending by a random period of up to several milliseconds.

In the simulations, mobility was modelled by the Random-Waypoint model [28]. In this model, each node picks a random target location and moves there at a randomly chosen speed. The node then waits for a random amount of time and then chooses a new location etc. In our case, the speed of movement ranged from 0.5-1.5 m/s, which corresponds to walking speed. Also, the maximal waiting time was set to 20 seconds. Each simulation lasted 5 minutes (of simulation time) and each data point was generated as an average of 10 runs. The transmission range was set to roughly 80 meters[7] with a simulation area of 200x200 meters, the message size was set to 1KB (less than one UDP/IP packet), and the network bandwidth to 1Mbps. In each simulation, two nodes were generating messages at variable rates. We have run simulations with varying number of nodes, but discovered that with the exception of very sparse networks, the results are qualitatively the same. Thus, we only present the results when the number of nodes is fixed at 200. In the graphs, we denote the flooding protocol by FLOODING, our Byzantine dissemination protocol by BDP(MIS) and BDP(CDS) depending on the overlay mechanism used (see Section 4), and by OVERLAY(MIS) and OVERLAY(CDS) the simple overlay dissemination mechanism that has no message

[7] In fact in SWANS one can choose the transmission power which translates into a transmission range based on power degradation and background noise.

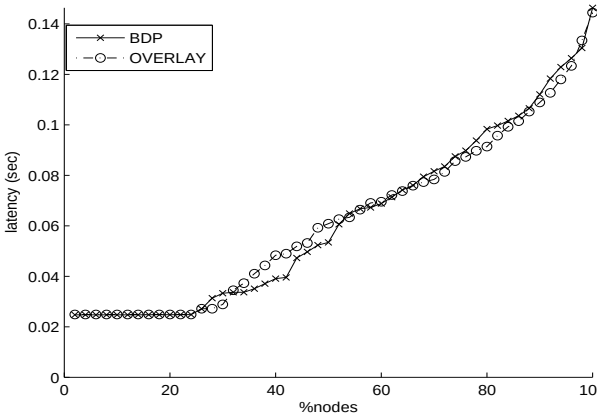

Figure 7: Latency to deliver a message to X% of the nodes when all nodes are static (with 200 nodes and sending one message per second)

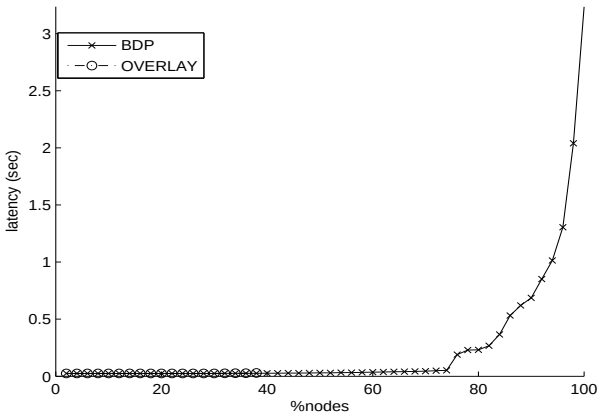

Figure 8: Latency to deliver a message to X% of the nodes when nodes are mobile (with 200 nodes and sending one message per second)

recovery. We limited the number of times each message is gossiped to two. Additional gossip attempts slightly improve the delivery ratios, but at the cost of additional messages. Finally, the main Byzantine behavior checked was of being mute, as this has the most adverse affect on the performance of the system.

The results of the simulations in static networks with no Byzantine nodes are presented in Figures 5, 6, and 7. As can be seen by the graphs, in this benign case, all protocols obtain very high delivery rate. Essentially, in all protocols the latency to deliver a message to all nodes remain well below $20ms$. However, the load on the network of the flooding protocol grows dramatically in the number of neighbors each node has (or in other words, the density of the network). Thus, from an energy stand point, flooding is much worse and less scalable than the others. Due to the staggering we used, even the flooding approach resulted in a relatively small number of collisions that were compensated for by its high redundancy, which explains its high delivery ratios. However, with higher sending rates, it is expected to perform much worse.

Since MIS+B and CDS performed almost the same, yet MIS+B is much more computationally efficient, during the rest of the this work, we only present the results for the MIS+B overlay. Figures 9, 10, and 8 present the simulation results for a mobile network. Here, flooding continues to behave well in terms of delivery ratio and latency (and bad in terms of network load). However, we start seeing a significant difference between our dissemination protocol (BDP) and a simple dissemination with no gossip and no recovery of messages (OVERLAY). While BDP maintains delivery rates close to flooding (and close to 100%), without gossip the delivery rate drops to 40%. Generally speaking, all protocols deliver messages fast. However, OVERLAY only delivers message to about 40% of the nodes. Also, in BDP the latency slightly grows for the last nodes proportionally to the frequency of a single gossip exchange.

Figures 11 and 12 explore the delivery ratio of the different protocols with varying number of Byzantine nodes. As can be seen, when no recovery mechanism is employed, the delivery rate drops dramatically. On the other hand, both our protocol and the flooding protocol maintain very high delivery rates. Interestingly, when nodes are mobile, the impact of Byzantine nodes is weakened. This can be explained by the fact that the overlay adapts itself to the evolving network topology. Thus, a Byzantine node does not necessarily remain in the overlay throughout the execution.

Figures 13 and 14 explore the network load imposed by the different protocols as a function of the

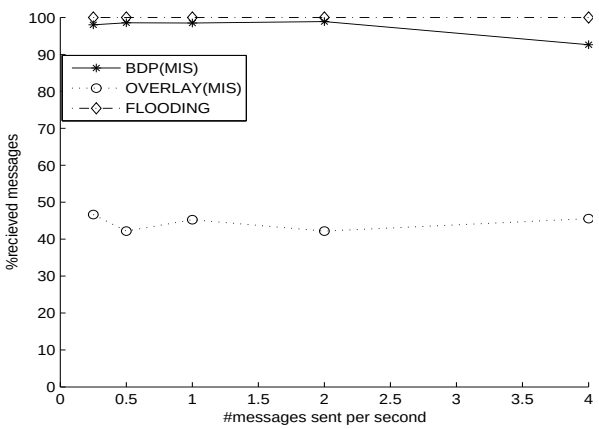

Figure 9: Message delivery ratio when nodes are mobile

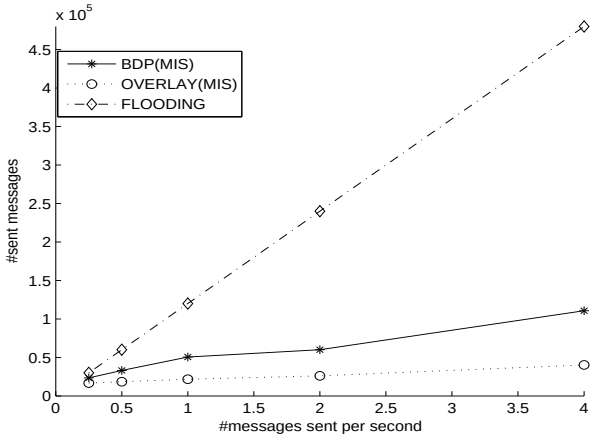

Figure 10: Network load in terms of total number of messages sent when nodes are mobile

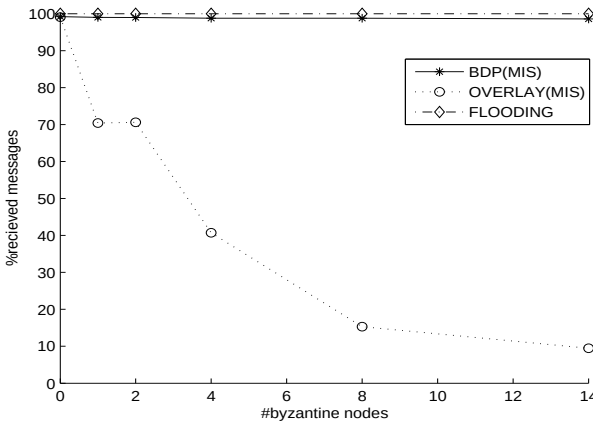

Figure 11: Message delivery ratio when all nodes are static with varying number of Byzantine nodes (out of a total of 200 nodes)

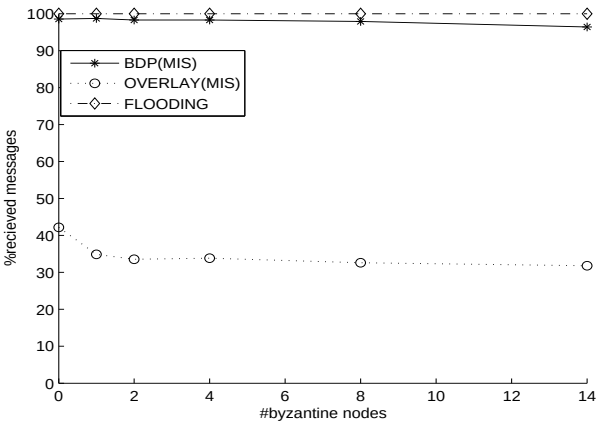

Figure 12: Message delivery ratio when nodes are mobile with varying number of Byzantine nodes (out of a total of 200 nodes)

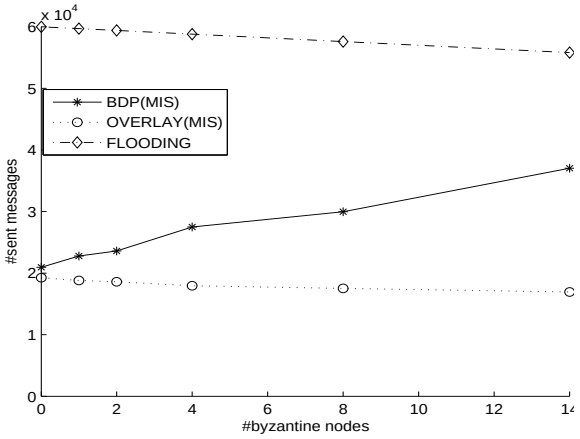

Figure 13: Network load when all nodes are static with varying number of Byzantine nodes (out of a total of 200 nodes)

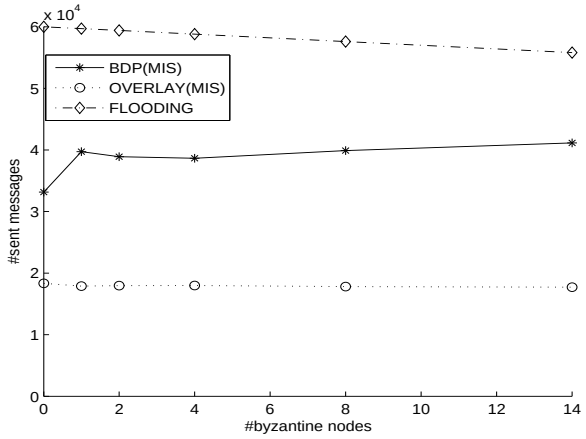

Figure 14: Network load when nodes are mobile with varying number of Byzantine nodes (out of a total of 200 nodes)

number of Byzantine nodes. In the static case, the network load imposed by BDP exhibit a linear increase with the number of Byzantine nodes. On the other hand, the network load imposed by flooding slightly improves. This can be explained by the fact that if Byzantine nodes avoid sending messages, then fewer messages are sent. As for the dynamic case, here we also observe the interesting phenomena that mobility improves the asymptotic behavior of the protocols. Again, this can be explained by the fact that the overlay structure evolves with the network topology, making it "harder" for Byzantine nodes to block message dissemination along the overlay.

Figures 15 and 16 explore the latency to deliver a message to X% of the nodes when some nodes are Byzantine (out of 200 nodes and a sending rate of 1 message per second). Clearly, the latency grows with the number of Byzantine nodes. Also, in the static Byzantine case, almost all nodes receive the message in less than a second and only when there are many Byzantine nodes, it may take several seconds to deliver a message to the last 20% of the nodes. In the mobile case we see the same qualitative behavior, but the latency starts growing beyond one second at 60% of the nodes. We would like to point out that by fine tuning the rate of gossips and the other timers in the system, it is possible to dramatically reduce the quantitative latency numbers. The numbers here do not include such tuning, yet we started exploring this option. However, the important thing to note is that with Byzantine nodes, without a best-effort recovery mechanism, it is almost impossible to ensure reliable delivery just by retransmission. This is because without additional recovery mechanism, the Byzantine nodes might collude to block all messages from reaching some parts of the network.

6 Related Work

A good survey of broadcast and multicast protocols for wireless ad hoc networks can be found in [46]. In particular, (multicast) routing in MANET can be classified into *proactive*, e.g., OLSR [13], *reactive*, e.g., AODV [40] and DSR [28], and mixtures of both, e.g., ZRP [26], as well as geographic routing [29, 30, 31, 41]. These protocols, however, ignore Byzantine failures.

Spanning tree based overlays have been often used as the main scheme for disseminating messages to

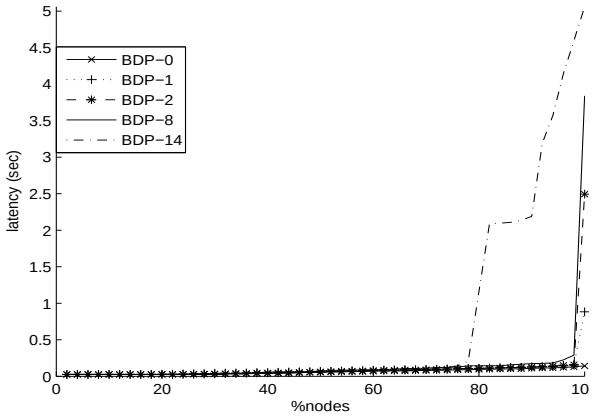

Figure 15: Latency to deliver a message to X% of the nodes when all nodes are static with varying number of Byzantine nodes

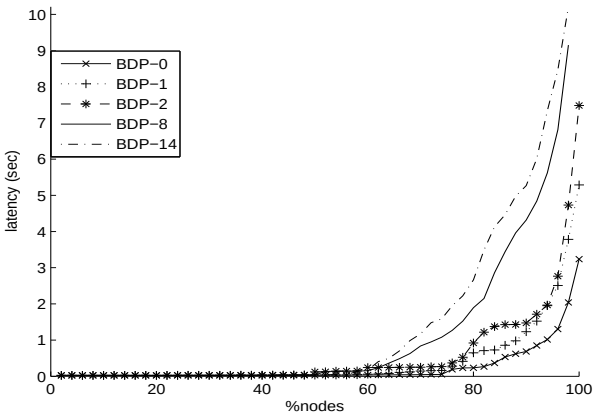

Figure 16: Latency to deliver a message to X% of the nodes when nodes are mobile with varying number of Byzantine nodes

large groups, e.g., in IP multicast [39, 45] and in the MBone [20, 33]. More sophisticated overlays such as hypercubes and Harary graphs have been explored, e.g., in [22, 32], as well as distributed hash tables like SCRIBE [42].

The idea that a process can detect that it is missing a message by exchanging messages with other processes first appeared, to the best of our knowledge, in the MNAK layer of the Ensemble system in 1996 [27]. Additionally, randomized gossip has been used as a method of ensuring reliable delivery of broadcast/multicast messages while maintaining high throughput in the PBcast/Bi-modal work [6] as well as in several followup papers, e.g., [19]. In these works it is assumed that a node can choose with whom it wishes to gossip, and does so in a random manner. In contrast, in our case gossiping is done with all neighbors that are physically decided by the movement of nodes and transmission ranges. Also, the works of [6, 19, 27] ignored Byzantine failures.

There has been a lot of work on securing point-to-point routing schemes against malicious nodes. Due to space limitations, we will only mention a few of them. One example is the protocol presented in [2]. In this work, the authors describe a mechanism for detecting malicious faults along a path and then discovering alternative paths. Another secure routing protocol (SRP) has been proposed in [37]. SRP requires a secure association between each pair of source and destination but assumes that Byzantine nodes do not collude. Yet another protocol, SMT [38], protects pairwise communication by breaking the message into several pieces based on a coding scheme that allows reconstructing the message even when some pieces are lost. Each piece is then sent along a different path. Additional examples of secure point-to-point routing include, e.g,. [43, 49, 50].

The work of Minsky and Schneider [36] explored disseminating information using gossip in wired networks, when some nodes can be faulty. This is by only trusting gossips that have gained the support of at least $f + 1$ nodes, where f is the number of potential Byzantine nodes. Several other works have also proposed a Byzantine multicast scheme that sends a message along $f + 1$ distinct paths [15, 34]. Similarly, using multiple paths chosen in a stochastic manner in order to reduce the possibility of interception have been studied in [7].

Reliable Byzantine tolerant broadcast and multicast in networks where all nodes can communicate di-

rectly with each other has been formally described in [8], and has been explored, e.g., in [15, 35]. Also, the works in [4, 9] have proposed a formal framework for defining and implementing reliable multicast protocols in a hybrid failure environment (Byzantine, crash, and omission) based on modern cryptography. In particular, they have investigated the computational complexity of such protocols.

A framework for fault-tolerance by adaptation was proposed in [11]. In this framework, a simple protocol is run during normal operation alongside some failure detection mechanism. Once a failure is detected, the execution switches to a masking protocol. This idea was demonstrated in [11] on the broadcast problem, which results in a somewhat similar solution to ours. However, in [11] it was not mentioned how the overlay (a tree in their case) is constructed and maintained. Also, the masking protocol was flooding, whereas we avoid flooding even when failures are detected. Instead, in our approach, local message recovery is first attempted. Moreover, in [11] it was not explained when and how to return to the simple protocol once a failure is compensated for. Finally, our work encapsulates failure detection behind failure detectors, which results in a modular implementation.

The notion of a failure detector, which capture the required functional properties of failure detection without specifying explicit timing assumptions, was initiated by Chandra and Toueg in the context of the Consensus problem [10]. Mute failure detectors were initially proposed in [17, 18] in order to solve Byzantine Consensus in otherwise asynchronous systems. They were later used also in [5, 24]. Moreover, the use of a trusted timely control channel, called *TTCB*, was explored as another mean of solving Byzantine Consensus efficiently in [14]. In fact, TTCB can be used to implement mute failure detectors. For example, when each node has both WiFi and cellular communication, one might be able to implement a TTCB using cellular communication while sending normal data using WiFi.

7 Discussion and Conclusions

In this work we have described a Byzantine tolerant broadcast protocol for mobile ad-hoc networks. The protocol disseminates messages along the arcs of a logical overlay. The protocol relies on signatures to prevent messages from being forged. It also employs gossiping of headers of known messages to prevent a Byzantine overlay node from stopping the dissemination of messages to the rest of the system. Additionally, for efficiency reasons, the overlay maintenance mechanism is augmented to ensure that enough correct nodes are elected to the overlay so that Byzantine nodes do not disconnect the overlay beyond the time required to detect such behavior. Finally, the detection of observable Byzantine behaviors, such as mute and verbose failures, are encapsulated within corresponding failure detectors modules. The use of failure detectors simplifies the presentation of the protocol and makes it more generic and robust. This is because the protocol need not deal explicitly with issues like timers and timeouts.

Our measurements confirm that for non-sparse networks, the protocol behaves very well. That is, our protocol obtains very high delivery ratios while sending much fewer messages than flooding. When there is no Byzantine activity, our protocol is almost as economical as a protocol that has no recovery mechanism (and in particular, much more efficient than flooding). When some Byzantine failures occur, our protocol still remains more efficient than flooding, while maintaining a comparable delivery rate. In contrast, when there are Byzantine failures or mobility, having no recovery mechanism results in a significant drop in delivery rates. Additionally, we discovered the interesting anecdote that Byzantine failures have a somewhat reduced impact when the nodes are mobile. Intuitively, when nodes are mobile, there is a lower chance that Byzantine nodes will constantly be at critical positions on the message dissemination paths for all messages.

In this work we detect and cope with verbose attacks. However, we do not address denial of service attacks caused by Byzantine nodes constantly sending messages in order to jam the network (at the MAC

level). This problem can only be solved at the hardware level, e.g., using frequency hopping techniques borrowed from electronic warfare [12]. While this is a very important issue, it is orthogonal to other forms of Byzantine failures. In particular, a solution to this denial of service problem will not provide remedy for the other forms of Byzantine behavior.

Finally, one of the main problems in mobile ad-hoc networks is power. As nodes are mobile, they are typically battery operated. It turns out that the network card consumes roughly the same levels of energy when it sends a message, receives a message, and listen for messages. The main source of energy saving is to put the card in sleeping mode. The IEEE 802.11 standard includes the Power Save Mode in order to deal with this problem in wireless LANs when all messages are point to point. There have also been a few attempts to extend this to multiple hops networks with point to point messages, such as [3]. An interesting problem is to develop a Byzantine broadcast protocol for multiple hop ad hoc networks that enables most nodes to sleep most of the time in order to reduce their energy consumption.

Acknowledgements: We would like to thank Eli Biham and Elad Barkan for advising us on the usage of cryptography for this work.

References

[1] http://jist.ece.cornell.edu/.

[2] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. An On-Demand Secure Routing Protocol Resilient to Byzantine Failures. In *Proc. ACM Workshop on Wireless Security (WiSe)*, Atlanta, GA, September 2002.

[3] B. Awerbuch, D. Holmer, and H. Rubens. The Pulse Protocol: Energy Efficient Infrastructure Access. In *Proc. of the 23rd Conference of the IEEE Communications Society (Infocom)*, March 2004.

[4] M. Backes and C. Cachin. Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, June 2003.

[5] R. Baldoni, J.M. Helary, and M. Raynal. From Crash-Fault Tolerance to Arbitrary-Fault Tolerance: Towards a Modular Approach. In *Proc. of the IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 273–282, June 2000.

[6] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, , and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, May 1999.

[7] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. Enhancing Security via Stochastic Routing. In *Proc. of the 11th IEEE International Conference on Computer Communications and Networks*, pages 58–62, May 2002.

[8] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4):824–840, October 1985.

[9] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In *Proc. of Advances in Cryptology: CRYPTO 2001*, pages 524–541, 2001.

[10] T. Chandra and S. Toueg. Unreliable Failure Detectors for Asynchronous Systems. *Journal of the ACM*, 43(4):685–722, July 1996.

[11] I. Chang, M.A. Hiltunen, and R.D. Schlichting. Affordable Fault Tolerance Through Adaptation. In *Proc. of Workshop on Fault-Tolerant Parallel and Distributed Systems (LNCS 1388)*, pages 585–603, April 1998.

[12] M.C.-H. Chek and Y.-K. Kwok. On Adaptive Frequency Hopping to Combat IEEE 802.11b with Practical Resource Constraints. In *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 391–396, May 2004.

[13] T. Clause, P. Jacquet, and A. Laouti. Optimized Link State Routing Protocol. In *Proc. IEEE International Multi Topic Conference (INMIC)*, December 2001.

[14] M. Correia, N.F. Neves, L.C. Lung, and P. Veríssimo. Low complexity byzantine-resilient consensus. Technical Report TR-03-25, DI-FCUL, University of Lisbon (Portugal), 2003. To appear in *Distributed Computing*.

[15] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic Broadcast: From Simple Diffusion to Byzantine Agreement. In *Proc. of the 15th International Conference on Fault-Tolerant Computing*, pages 200–206, Austin, Texas, 1985.

[16] F. Cristian and C. Fetzer. The Timed Asynchronous Distributed System Model. *IEEE Transanctions on Parallel and Distributed Systems*, 10(6):642–657, June 1999.

[17] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness Failure Detectors: Specification and Implementation. In *Proc. 3rd European Dependable Computing Conference (EDCC)*, pages 71–87, 1999.

[18] A. Doudou and A. Schiper. Muteness Detectors for Consensus with Byzantine Processes (Brief Announcement). In *Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC)*, page 315, 1998.

[19] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Transactions on Computing Systems*, 21(4):341–374, 2003.

[20] S. Floyd, van Jacobson, S. McCanne, C. Liu, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proc. ACM SIGCOMM'95*, August 1995.

[21] R. Friedman, M. Gradinariu, and G. Simon. Locating Cache Proxies in MANETs. In *Proc. 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 175–186, May 2004.

[22] R. Friedman, S. Manor, and K. Guo. Scalable Hypercube Based Stability Detection. *IEEE Transactions on Parallel and Distributed Systems*, 13(8), August 2002.

[23] R. Friedman, A. Mostefaoui, and M. Raynal. Asynchronous Bounded Lifetime Failure Detectors. *Information Processing Letters*. To appear.

[24] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and Efficient Oracle-Based Consensus Protocols for Asynchronous Byzantine Systems. In *Proc. of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 228–237, October 2004.

[25] J.J. Garcia-Luna-Aceves and E.L. Madruga. The Core-Assisted Mesh Protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1380–1394, August 1999.

[26] Z. Haas. A New Routing Protocol for the Reconfigurable Wireless Networks. In *Proc. IEEE Int. Conf. on Universal Personal Communications (ICUP)*, October 1997.

[27] M. Hayden. The Ensemble System. Technical Report TR98-1662, Department of Computer Science, Cornell University, January 1998.

[28] D.B. Johnson and D.A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[29] B. Karp. *Geographic Routing for Wireless Networks*. PhD thesis, Harvard University, 2000.

[30] Y. Ko and N. H. Vaidya. Geocasting in Mobile Ad Hoc Networks: Location-Based Multicast Algorithms. In *Proc. 2nd IEEE Workshop on Mobile Computer Systems and Applications*, page 101, 1999.

[31] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *Proc. 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 120–130, 2000.

[32] M.-J. Lin, K. Marzullo, and S. Masini. Gossip versus Deterministically Constrained Flooding on Small Networks. In *Proc1ˆ4th International Conference on Distributed Computing 2000*, pages 253–267, October 2000.

[33] M.R. Macedonia and D.P. Brutzman. MBone Provides Audio and Video Across the Internet. *IEEE Computer*, 27(4):30–36, April 1994.

[34] D. Malkhi, Y. Mansour, and M.K. Reiter. Diffusion Without False Rumors: on Propagating Updates in a Byzantine Environment. *Theoretical Computer Science*, 1–3(299):289–306, April 2003.

[35] Dahlia Malkhi and Michael Reiter. A High-Throughput Secure Reliable Multicast Protocol. *Journal of Computer Security*, 5:113–127, 1997.

[36] Y. Minsky and F.B. Schneider. Tolerating Malicious Gossip. *Distributed Computing*, 16(1):49–68, 2003.

[37] P. Papadimitratos and Z. Haas. Secure Routing for Mobile and Ad Hoc Networks. In *Proc. Communication Networks and Distributed Systems Modeling and Simulations Conference*, January 2002.

[38] P. Papadimitratos and Z. Haas. Secure Message Transmission in Mobile and Ad Hoc Networks. *Ad Hoc Networks*, 1, July 2003.

[39] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997. Special issue on Network Support for Multipoint Communication.

[40] C. Perkins. Ad Hoc On Demand Distance Vector (AODV) Routing. *Internet Draft, draft-ietf-manet-aodv-00.txt, citeseer.nj.nec.com/article/perkins99ad.html*, 1997.

[41] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proc. 9th Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 96–108, 2003.

[42] A. Rowstron, A-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large Scale Event Notification Infrastructure. In *Proceedings of 3rd International Workshop on Networked Group Communication*, November 2001.

[43] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E.M. Belding-Royer. A Secure Routing Protocol for Ad Hoc Networks. In *Proc. of the IEEE International Conference on Network Protocols (ICNP)*, November 2002.

[44] B. Schneier. *Applied Cryptography*. Wiley, 1996.

[45] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 1996. Third Edition.

[46] C.K. Toh. *Ad Hoc Mobile Wireless Networks*. Prantice Hall, 2002.

[47] C.W. Wu and Y.C. Tay. AMRIS: A Multicast Protocol for Ad-Hoc Wireless Networks. In *Proceedings of the IEEE MILCOMM '99*, November 1999.

[48] J. Wu, M. Gao, and I. Stojmenovic. On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks. In *Proc. of the 30th International Conference on Parallel Processing (ICPP)*, pages 346–353, 2001.

[49] S. Yi, P. Naldurg, and R. Kravets. Security Aware Ad Hoc Routing for Wireless Networks. In *Proc. ACM Syposium on Mobile Ad Hoc Networking and Computing*, October 2001.

[50] M.G. Zapata and N. Asokan. Secure Ad Hoc Routing Protocol. In *Proc. ACM Workshop on Wireless Security*, 2002.