Limiting Byzantine Influence in Multihop Asynchronous Networks

Alexandre Maurer, Sébastien Tixeuil

Limiting Byzantine Influence in Multihop Asynchronous Networks

Alexandre Maurer and Sébastien Tixeuil
UPMC Sorbonne Universités, Paris, France
Email: Alexandre.Maurer@lip6.fr, Sebastien.Tixeuil@lip6.fr

January 27, 2012

Abstract

We consider the problem of reliably broadcasting information in a multihop asynchronous network that is subject to Byzantine failures. That is, some nodes of the network can exhibit arbitrary (and potentially malicious) behavior. Existing solutions provide deterministic guarantees for broadcasting between *all* correct nodes, but require that the communication network is highly-connected (typically, $2k + 1$ connectivity is required, where k is the total number of Byzantine nodes in the network).

In this paper, we investigate the possibility of Byzantine tolerant reliable broadcast between *most* correct nodes in low-connectivity networks (typically, networks with constant connectivity). In more details, we propose a new broadcast protocol that is specifically designed for low-connectivity networks. We provide sufficient conditions for correct nodes using our protocol to reliably communicate despite Byzantine participants. We present experimental results that show that our approach is especially effective in low-connectivity networks when Byzantine nodes are randomly distributed.

1 Introduction

In this paper, we revisit the problem of reliably broadcasting information in an arbitrary shaped network that is subject to Byzantine failures. As distributed systems and networks grow larger, faults and attacks are more likely to appear and resiliency to those faults needs to be adressed in the very early design stages of the protocols that target those systems. One of the strongest fault models is *Byzantine* [11]: the faulty node behaves arbitrarily. This model encompasses a rich set of fault scenarios. Moreover, Byzantine fault tolerance has security implications, as the behavior of an intruder can be modeled as Byzantine. However, in most studies to date, Byzantine faults are considered in completely connected networks.

One approach to deal with Byzantine faults is by enabling the nodes to use cryptographic operations such as digital signatures or certificates. This limits the power of a Byzantine node as a correct node can verify the validity of received information and authenticate the sender across multiple hops. However, this option may not be available. For example, the nodes may not have enough resources to manipulate digital signatures. Moreover, cryptographic operations implicitly assume the presence of a trusted infrastructure: secure channels to a key server or a public key infrastructure. Establishing and maintaining such infrastructure in the presence of Byzantine faults may be problematic.

Another way to limit the power of a Byzantine process is to assume synchrony: all processes proceed in lock-step. Indeed, if a process is required to send a message with each pulse, a Byzantine process cannot refuse to send a message without being detected. However, the synchrony assumption may be too restrictive for practical systems.

Related works Many recent Byzantine-robust protocols make use of *cryptography* (see [3, 5] and references herein) to contain the influence of Byzantine nodes. As previously stated, this requires a trusted infrastructure that we do not assume.

Cryptography-free Byzantine failures have first been studied in completely connected networks [11, 1, 12, 13, 16]: a node can directly communicate with any other node. If the underlying network is not completely connected (as are most networks), such a setting raises the problem to reliably transmit information between nodes that are not direct neighbors.

In practice, broadcasting a message in the network requires to rely upon other nodes. Dolev [4] considers Byzantine agreement on arbitrary graphs. He states that, for agreement in the presence of up to k Byzantine nodes, it is necessary and sufficient that the network is $(2k + 1)$-connected, and that the number of nodes in the system is at least $3k + 1$. Also, this solution assumes that the underlying graph is known to every node, and that nodes are scheduled according to the synchronous execution model. Nesterenko and Tixeuil [19] relax both requirements (graph is unknown and scheduling is asynchronous) yet retain $2k + 1$ connectivity for resilience and $k + 1$ connectivity for detection (that is, detecting that there is at least one Byzantine node in the network). In a low-connectivity network such as a torus (where nodes have degree at most four), both approaches can cope only with a single Byzantine node, independently of the torus size.

Byzantine resilient broadcast was recently investigated in the context of *radio networks*: each node is a robot or a sensor with a physical position. A node can only communicate with nodes that are located within a certain radius, called neighbors. Broadcast protocols have been proposed [10, 2] for nodes organized on a grid (the wireless medium typically induces much more than four neighbors per node, otherwise the broadcast does not work). Both approaches are based on a local voting system, and perform correctly if every node has less than a $1/4\pi$ fraction of Byzantine neighbors. This criterion was later generalized [20] to other topologies, assuming that each node knows the exact topology. Again, in low-connectivity networks, the local constraint on the proportion of Byzantine nodes in any neighborhood may be difficult to assess.

A notable class of algorithms tolerates Byzantine faults with either space [15, 18, 21] or time [14, 9, 8, 7, 6] locality. Yet, the emphasis of space local algorithms is on containing the fault as close to its source as possible. This is only applicable to the problems where the information from remote nodes is unimportant such as vertex coloring, link coloring or dining philosophers. Also, time local algorithms presented so far can hold at most one Byzantine node and are not able to mask the effect of Byzantine actions. Thus, the local containment approach is not applicable to reliable broadcast.

Our contribution All aforementioned results rely on a strong *connectivity* of the communication graph, and on Byzantine proportions assumptions in the network. In other words, tolerating more Byzantine nodes requires an increase of the degree of each node, which can be a heavy constraint on large networks such as a peer-to-peer overlays.

In this paper, we introduce the idea to trade the perfectly reliable communication between

correct nodes for the ability to support low-connectivity communication graphs with many Byzantine nodes. Informally, we accept that a small minority of correct nodes are denied reliable communication, provided that a large majority of correct nodes can reliably communicate. Such a loss in safety guarantees looks acceptable to us since, by hypothesis, many Byzantine nodes (with arbitrary behavior) are already present in the system. Also, our results demonstrate that this tradeoff permits to tolerate a high number of Byzantine nodes, even in constant connectivity networks such as grids or torus. Yet, experimental results show that our scheme preserves the communication capabilities of a huge majority of correct processes.

Our approach is not based on voting proportions, but on *control zones* and *authorizations*. Intuitively, control zones act as filters in the network: they limit the diffusion of Byzantine messages.

The sequel of the paper is organized as follows. In Section 2, we define a new broadcast protocol based on *control zones*. In Section 3, we prove sufficient conditions to achieve reliable communication in a particular subgraph of the network, using our protocol. In Section 4, we provide an experimental evaluation of our protocol on *torus* and *grid* shaped networks, with randomly distributed Byzantine failures. Using the sufficient conditions of Section 3, we evaluate the probability for two randomly choosen nodes to communicate reliably.

2   Description of the protocol

In this section, we give an informal description of our protocol, set the formal background, and describe how the protocol is locally executed.

2.1   Informal Description

The network is described by a set of processes, called *nodes*. Some pairs of nodes are linked by a *canal*, and can send messages to each other: we call them *neighbors*. The network is *asynchronous*: the nodes can send and receive messages at any time. Our only hypothesis is that any message that is sent is eventually received.

A node may want to broadcast a specific information m_0 to the the network. For instance, in a sensors network, m_0 can be a temperature; in a mobile robots network, m_0 can be the position of the current robot; etc. In a network where all nodes are correct, the following would happen:

- A given node p sends a message containing the couple (p, m_0) to its neighbors,

- the neighbors of p send (p, m_0) to their neighbors,

- and so forth, until every node receives (p, m_0). Then the entire network knows that p broadcasted the information m_0.

In our setting however, some nodes can be Byzantine and send arbitrary messages. Those messages are potentially malicious. For instance, a Byzantine node can send (p, m_1), with $m_1 \neq m_0$, to make the network believe that p broadcasted the information m_1. Therefore, one single Byzantine node can lie about the information of every node, and then deceive the whole network.

To limit the action of Byzantine nodes, we define a set of *control zones*. A control zone is defind by:
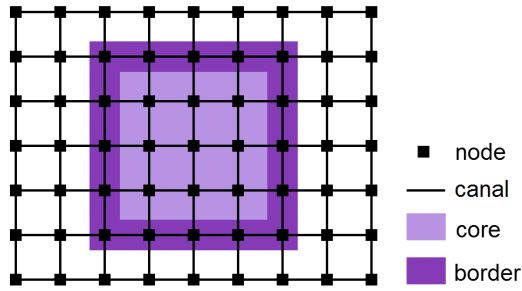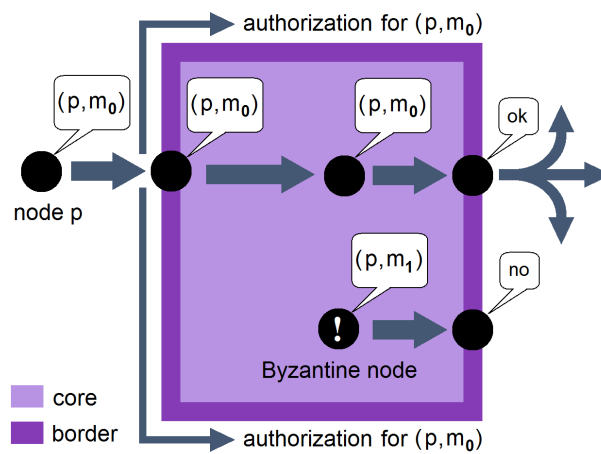
Figure 1: Example of control zone

Figure 2: Principle of a control zone

- Its *core*, an arbitrary set of nodes.

- Its *border*, a node-cut isolating the core from the rest of the network.

An example of control zone is given in Fig. 1. The important point is that messages must pass through the *border* to access the *core*.

Here is the main idea of the protocol:

- When a message enters the *core* of a control zone, an *authorization* is broadcasted on its *border*.

- When the same message wants to exit the *core*, this *authorization* is required.

This mechanism does not disturb the broadcasting of correct messages.

Now, suppose that a Byzantine node is in the core of the control zone, and sends a lying message (p, m_1), whereas p is *not* in the core of the control zone. Then, this message never gets the authorization to exit the core, as it never entered it. This is illustrated in Fig. 2.

Intuitively, this mechanism of control zones enables to limit the broadcasting of Byzantine messages. The underlying idea is to define a lot of control zones on the network, intersecting each other, in order to minimize the broadcasting of Byzantine messages. Then eventually, under certain conditions, we could achieve reliable broadcast in a specific part of the network.

2.2 Definitions and Hypothesis

2.2.1 Network Model

Let (G, E) be a non-oriented graph representing the topology of the network.

- G denotes the *nodes* of the network.

- E denotes the *neighborhood* relationships: two given nodes are or are not neighbors. A node can only send messages to its neighbors.

Let $Corr$ be the set of *correct* nodes. These nodes follow the protocol described further. We assume that the other nodes are totally unpredictable (*Byzantine*).

We assume asynchronous message passing: any message sent is eventually received, but it can be at any time. We assume that, in an infinite execution, any process is activated inifinitely often. We make no hypothesis on the order of activation of the processes. We assume local topology knowledge: each node knows the identifiers of its neighbors. Therefore, its direct neighbors cannot lie on their identity when sending a message. Also, a slightly stronger topology knowledge is required (see definition of $myCtr$ in 2.2.4).

2.2.2 Control zones

A set of nodes S is *connected* if, for any nodes p and q in S, there exists a chain of neighborhood relationships linking p and q. A *node-cut* is a set of nodes C such that $G \backslash C$ is disconnected.

Definition 1 (Control zone) *A* control zone *is a pair* $(Core, Border)$ *of disjoint, connected node sets, such that* $Border$ *is a node-cut isolating* $Core$ *from the rest of the network.*

We denote the core and the border of a control zone z by $core(z)$ and $border(z)$.

Before running the protocol, we choose an arbitrary set Ctr of control zones. These are the control zones used in the protocol.

2.2.3 Messages formalism

In the protocol, two types of messages can be exchanged:

- *Standard messages*, of the form (s, m): a message claiming that the node s (*source*) initially sent the information m.

- *Authorization messages*, of the form (s, m, z): a message authorizing the *standard message* (s, m) to exit the control zone z.

2.2.4 Attributes of the correct nodes

Each correct node p posesses two static attributes:

- m_0: the information that p wants to broadcast.

- $myCtr$: set of control zones $z \in Ctr$ such that $p \in border(z)$. We assume that, for each zone $z \in myCtr$, p knows which nodes belong to $core(z)$ and $border(z)$.

It also possesses three dynamic attributes:

- $Wait$: set of messages received, but waiting for an authorization (initially empty). When $(s, m, q) \in Wait$, it means that p received the *standard message* (s, m) from a neighbor q.

- $Auth$: set of authorizations received (initially empty). When $(s, m, z) \in Auth$, it means that p has received the authorization for the *standard message* (s, m) on the control zone z.

- Acc: set of accepted messages (initially empty). When $(s, m) \in Acc$, it means that p has received (s, m) and all the corresponding authorizations, and has sent it to its neighbors.

The attribute X of a node p is denoted by $p.X$.

2.3 Local Execution of the Protocol

The protocol is locally executed by each correct node.

Let p be the identifier of the current node executing the protocol. $Wait$, $Auth$, Acc, $myCtr$ and m_0 refer to the corresponding attributes of p.

The protocol contains the following actions:

- *Send* a particular message to the neighbors of p.

- *Add* an element x to a set X ($X := X \cup \{x\}$).

The protocol is divided in four sections, executed in specific moments: INIT, ENTER, DIFF and EXIT.

2.3.1 INIT - Initial broadcast

Executed initially.

- Send (p, m_0) to all neighbors.

- Add (p, m_0) to Acc.

- $\forall z \in myCtr$, send (p, m_0, z) to all neighbors.

2.3.2 ENTER - Message entering control zones

Executed when a standard message (s, m) is received from a neighbor q.

- If $(s, m) \in Acc$, ignore it.

- Else, add (s, m, q) to $Wait$.

2.3.3 DIFF - Diffusion of authorizations

Executed when an authorization message (s, m, z) is received from a neighbor q.

- If $(s, m, z) \in Auth$, or $q \notin border(z)$, ignore it.

- Else:

  - Add (s, m, z) to $Auth$.
  - Send (s, m, z) to all neighbors.

2.3.4 EXIT - Message exiting control zones

Executed when an element (s, m, q) of $Wait$ verifies the following condition: $\forall z \in myCtr$, such that $q \in core(z)$ and $s \notin core(z)$, we have $(s, m, z) \in Auth$.

- Add (s, m) to Acc.

- Send (s, m) to all neighbors.

- $\forall z \in myCtr$, send (s, m, z) to all neighbors.

3 Properties of the protocol

In this section, we adopt the point of view of an omniscient observer, knowing the topology of the network and the position of all Byzantine nodes. The following theorems enable to determine sets of nodes satisfying certain properties in *any* execution: *safety, communication, reliability* (see Definitions 4, 5, 6).

- In Theorem 1, we show how to determine the *safe* nodes (who never accept a false message).

- In Theorem 2, we show how to construct a *communicating* node set (where all correct messages are received).

- In Theorem 3, we show that a *safe* and *communicating* node set achieves reliable communication.

Notice that it does not require that any correct node knows the position of the Byzantine nodes: this is just a global vision of the network. These theorems are used in Section 4, to evaluate the performances of the protocol for a given placement of Byzantine nodes.

We also analyze the message complexity of the protocol.

3.1 Notations and Definitions

We say that a correct node p *accepts* a message (s, m), when (s, m) is added to the set $p.Acc$.

Definition 2 (Correct and false messages) *A message (s, m) is* correct *if $m = s.m_0$. Else, it is* false*.*

Definition 3 (Correct path) *Let S be a node set. Let p and q be two nodes of S. A* correct path *on S between p and q is a serie $(i_1, \ldots, i_n)$ of correct nodes of S such that:*

- $i_1 = p$.

- $i_n = q$.

- i_k *and* i_{k+1} *are neighbors.*

Definition 4 (Safe node set) *A node is* safe *if it never accepts a false message. A node set is safe if all its nodes are safe.*

Definition 5 (Communicating node set) *A set S of correct nodes is* communicating *if, for any nodes p and q of S, q eventually accepts* $(p, p.m_0)$.

Definition 6 (Reliable node set) *A set S of correct nodes is* reliable *if, for any nodes p and q of S, q never accepts a false message, and eventually accepts* $(p, p.m_0)$.

3.2   Determination of a safe node set

The following theorem enables, under certain conditions, to determine a safe node set in the network. The condition is the existence of a particular set Z of control zones, such that:

- The union of the cores and the union of the borders are disjoint.

- The union of the cores contains all Byzantine nodes.

Then, all the nodes outside of Z are safe. Notice that no correct node needs to determine Z: we just have to know that this set exists.

Theorem 1 (Determination of safe nodes) *If there exists a set Z of control zones $z \in Ctr$ such that:*
   With $Cores = \cup_{z \in Z} core(z)$
With $Borders = \cup_{z \in Z} border(z)$

- *(1) The node sets $Cores$ and $Borders$ are disjoint.*

- *(2) All Byzantine nodes are in $Cores$.*

   Then any node $v \notin Cores$ is safe.

Proof: The proof is by contradiction. Suppose the opposite: let (s, m) be any false message, that is $m \neq s.m_0$. And let v be the first correct node such that:

- (a) $v \notin Cores$.

- (b) v accepts (s, m), that is: v is *not* safe.

Obviously, v did not accept (s, m) in INIT, as $m \neq s.m_0$. So it was in EXIT. Thus, there exists $(s, m, q) \in v.Wait$ verifying the condition of EXIT. And the only way for (s, m, q) to have joined $v.Wait$, is that v received (s, m) from q in ENTER. Then, two possibilities:

- Either q is a correct node, and accepted (s, m) in EXIT. As v is the first node to verify (a) and (b), it implies that $q \in Cores$.

- Either q is a Byzantine node. Then, according to (2), $q \in Cores$.

So, in any case, $q \in Cores$. Therefore, let $z \in Z$ be a control zone such that $q \in core(z)$. As $v \notin Cores$, $v \notin core(z)$. But v is neighbor of $q \in core(z)$. So, by definition of a control zone (see Definition 1), $v \in border(z)$: otherwise, the border would not be a node-cut isolating the core.

Then, by definition of $myCtr$ (see 2.2.4), $z \in v.myCtr$. As (s, m, q) verifies the condition of EXIT, $z \in v.myCtr$ implies that $(s, m, z) \in v.Auth$.

The only way for (s, m, z) to have joined $v.Auth$, is that v received (s, m, z) in DIFF, from a neighbor in $border(z)$. Let u be the first node of $border(z)$ to send (s, m, z). As $Cores$ and $Borders$ are disjoint, according to (2), u is correct. And u did not send (s, m, z) in DIFF: otherwise, it would not be the first to do so. So it was in EXIT, implying that u accepted (s, m). So u verified (a) and (b) before v. This contradiction achieves the proof. □

3.3 Construction of a communicating node set

The following theorem enables to construct a communicating set node by node, when it is possible. Let S be a given communicating set, and v a given correct node: then the theorems tells us if $S \cup \{v\}$ is communicating, and so forth. The conditions are the following:

- The node v has a neighbor in S.

- We have enough *correct paths* (see Definition 3) to *always* receive the authorizations required for the communication between v and S.

To initiate the construction of S: simply notice that any correct node p forms a communicating node set $\{p\}$.

Theorem 2 (Construction of a communicating node set) *Let S be a communicating node set. Let v be a correct node verifying the following conditions:*

- *(1) v has a neighbor $u \in S$*

- *(2) Let Z be the set of control zones $z \in Ctr$, such that $u \in core(z)$ and $v \in border(z)$. Then $\forall z \in Z$, there exists a correct path on $border(z)$ between v and a node $w \in S$.*

Then $S \cup \{v\}$ is also communicating.

Proof: We use Lemma 1 and Lemma 2, detailed below. According to these lemmas:

- $\forall x \in S$, v eventually accepts $(x, x.m_0)$ (Lemma 1)

- $\forall x \in S$, x eventually accepts $(v, v.m_0)$ (Lemma 2)

Then, according to Definition 5, $S \cup \{v\}$ is communicating.

$\square$

Lemma 1 *Let there be the same hypothesis as Theorem 2. Then $\forall x \in S$, v eventually accepts $(x, x.m_0)$.*

Proof: Let x be a node of S.

As S is communicating, u eventually accepts $(x, x.m_0)$ in EXIT, and sends it to its neighbors. So according to (1), v receives $(x, x.m_0)$ from u. To accept it, according to the condition of EXIT, v only needs the authorizations $(x, x.m_0, z)$ with $z \in Z$ (possibly less, if $x \in core(z)$). Let us show that v eventually receives these authorizations.

Let $z \in Z$ be. Then, according to (2), there exists a correct path on its border between v and a node $w \in S$. As S is communicating, w eventually accepts $(x, x.m_0)$. So, according to EXIT, w sends $(x, x.m_0, z)$ to its neighbors. So does each node of the correct path in DIFF, until $(x, x.m_0, z)$ reaches v. Thus, the result. $\square$

Lemma 2 *Let there be the same hypothesis as Theorem 2. Then $\forall x \in S$, x eventually accepts $(v, v.m_0)$.*

Proof: Let x be a node of S.

First, let us establish two preliminary results:

- (a) Initially, v sends $(v, v.m_0)$. So u receives $(v, v.m_0)$ from v. According to ENTER, $(v, v.m_0, v)$ is added to $u.Wait$. As there is no zone z such that $v \in core(z)$ and $v \notin core(z)$, no authorization is required in EXIT. Thus, u eventually accepts $(v, v.m_0)$.

- (b) Let $z \in Z$ be. Let y be any node of S in $border(z)$.

    - As $z \in v.myCtr$, v sent $(v, v.m_0, z)$ in INIT.
    - According to (2), there exists a correct path on $border(z)$ between v and w.

    So w eventually receives the authorization $(v, v.m_0, z)$. S is a set of correct nodes, $w \in S$ and $border(z)$ is connected. Thus, there also exists a correct path on $border(z)$ between w and y. Then y eventually receives $(v, v.m_0, z)$.

Now, let $\mathcal{C}_1$ be a configuration in which we have reached the states described in (a) and (b). Then, in such a configuration, $(v, v.m_0)$ becomes *indistinguishable* from $(u, u.m_0)$. Indeed, the only part of the protocol that could distinguish these messages is the condition of EXIT, for the nodes of $border(z)$ with $z \in Z$. But, as we have reached the state described in (b), all these nodes have received the authorizations $(v, v.m_0, z)$, $z \in Z$. So EXIT behaves the same way in both cases. Thus, as x eventually accepts $(u, u.m_0)$, x eventually accepts $(v, v.m_0)$.

This notion of *indistinguishability* is deliberately intuitive: the detailed proof is by exhaustion, and presents no particular interest. However, let us give the sketch of this proof.

As x eventually accepts $(u, u.m_0)$, let $(\mathcal{A}_1(u), \ldots, \mathcal{A}_n(u))$ be the list of actions related to $(u, u.m_0)$, by order of execution. These actions can be of the following types:

- p sends $(u, u.m_0)$ to q

- p sends $(u, u.m_0, z)$ to q, $z \in Ctr$

- p receives $(u, u.m_0)$ from q

- p receives $(u, u.m_0, z)$ from q, $z \in Ctr$

- p accepts $(u, u.m_0)$ from q

Let there be any configuration occuring after $\mathcal{C}_1$. Let $k < n$ be such that:

- The actions $(\mathcal{A}_1(v), \ldots, \mathcal{A}_k(v))$ have been executed.

- $\mathcal{A}_{k+1}(v)$ has not been executed yet.

Then, as $\mathcal{A}_{k+1}(u)$ eventually occurs, $\mathcal{A}_{k+1}(v)$ eventually occurs by the same mechanism. Notice that $\mathcal{A}_{k+1}(v)$ can also occur by another mechanism: this one is by default. Thus, by recursion, the result. □

3.4 Determination of a reliable node set

This theorem is the combination of the two previous theorems: we simply show that a *safe* and *communicating* node set is *reliable*. In pratice, to determine a reliable node set, we just have to:

- Determine a safe node set S_1, if we manage to.

- Construct a communicating node set S_2.

- Make the intersection of S_1 and S_2.

Theorem 3 (Determination of a reliable node set) *Let S_1 be a safe node set. Let S_2 be a communicating node set. Then $S = S_1 \cap S_2$ is a reliable node set.*

Proof: Let p and q be two nodes of S.

- As q also belongs to S_1, q never accepts a false message.

- As p and q also belong to S_2, q eventually accepts $(p, p.m_0)$.

Then, by definition 6, S is reliable. □

3.5 Message complexity

Let us evaluate the message complexity of this protocol when the whole network is a reliable node set (according to Definition 6).

We define the following parameters:

- n, the number of nodes of the network.

- d, the *degree* of the network, that is: the maximal number of neighbors for a node.

- N_{Ctr}, the number of control zones in Ctr.

- N_{Border}, the maximal number of nodes in the border of a control zone.

Let u be any node. Let us evaluate the number of messages related to u:

- All nodes once accept and send $(u, u.m_0)$ to their neighbors, which makes at most dn messages.

- All nodes on the border of a control zone z once send $(u, u.m_0, z)$ to their neighbors, which makes at most $dN_{Border}N_{Ctr}$ messages.

Thus, at most $dn(n + N_{Border}N_{Ctr})$ messages are sent in the network. Therefore, if we assume that N_{Border} is $o(1)$ and that N_{Ctr} is $o(n)$, the message complexity is $o(n^2)$, the same as a standard broadcast protocol (see 2.1).

4 Evaluation of the Protocol

In this section, we make an experimental evaluation of our protocol. We describe our methodology and our case of study, then comment on the results.

4.1 Methodology

We want to evaluate the performances of our protocol for a given network topology and a given choice of control zones. As we are most interested in obtaining a high *proportion* of correct nodes which communicate reliably, assuming a probabilistic distribution of the Byzantine nodes is a sensible option. It is also motivated by real constraints of actual networks that, we believe, could be supporting our protocol: for instance, most peer-to-peer overlay networks give newcomers a random identifier and thus a random position in the virtual topology for the purpose of load balancing, and many virus propagation mechanisms use random epidemic schemes to spread across networks. The main metric for evaluating the efficiency of a reliable communication scheme is the probability for two correct nodes, selected uniformly at random, to communicate properly.

Our evaluation scheme has the following input and output:

- *Input*: n_B, the number of Byzantine nodes on the network, randomly distributed.

- *Output*: $P(n_B)$, the probability that two randomly choosen nodes communicate reliably. That is: each node eventually receives the message of the other, and never accepts any lying message.

To evaluate $P(n_B)$, we use simulations and a Monte Carlo method [17]. For a given value of n_B, we run a large number of simulations. The fraction of succesful simulations converges to $P(n_B)$. It is actually impossible to simulate a distributed algorithm in the presence of Byzantine failures. Indeed, it would imply to predict the worst possible behavior of Byzantine nodes, which is a far too difficult problem. Also, giving a specific behavior to faulty nodes would weaken the problem tremendously. Therefore, instead of simulating the protocol, we use the theorems of Section 3.

Here are the main steps of a single simulation:

- Among the nodes of the network, choose n_B nodes that are Byzantine, uniformly at random. That is, all possible distributions of Byzantine nodes have the same probability to occur.

- Use Theorem 3 to construct a *reliable node set* (see Definition 6). In the worst case, this set is empty. An illustration of this step is given in 4.2.4.

- Choose two nodes uniformly at random. If both nodes are in the reliable node set, the simulation is a success. Else, it is a failure.

Notice that we only construct *one* reliable node set, which may not necessarily be the best one. Therefore, the estimated value of $P(n_B)$ is a lower bound of the real value of $P(n_B)$. This is not a problem, as we only want to give guarantees.

4.2 Topology and Control Zones

In this subsection, we choose an particular network topology and a particular set of control zones to perform our evaluation. Then we present an example of the construction of a reliable node set.
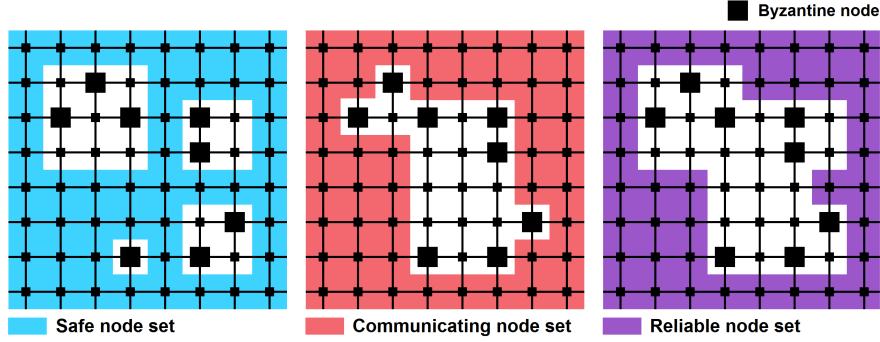
Figure 3: Toy example of safe, communicating and reliable node sets

4.2.1 Network Topology

We consider a *torus* network and a *grid* network. A torus network can be seen as a grid network with a continuity between the left-right and up-down extremities. Our topology choice is motivated by the fact that those are the most simple yet two-dimensional topologies with fixed degree.

Definition 7 (Torus and grid) *A $N \times N$ torus (resp. grid) network is a network such that:*

- *Each node has a unique identifier (i, j) with $1 \leq i \leq N$ and $1 \leq j \leq N$.*

- *Two nodes (i_1, j_1) and (i_2, j_2) are neighbors if and only if one of these two conditions is satisfied:*

  - *$i_1 = i_2$ and $|j_1 - j_2| = 1$ or N (resp. 1).*
  - *$j_1 = j_2$ and $|i_1 - i_2| = 1$ or N (resp. 1).*

4.2.2 Control Zones

Our protocol is given a set of *square control zones*. See 2.2.2 for the definition of a control zone.

Definition 8 (Square control zone) *A square control zone $Sqr(i_0, j_0, w)$ is a pair $(Core, Border)$ such that:*

- *Core if the set of nodes (i, j) such that:*

  - *$i_0 < i < i_0 + w + 1$*
  - *$j_0 < j < j_0 + w + 1$*

- *Border is the set of nodes (i, j) such that:*

  - *$i = i_0$ or $i_0 + w + 1$, and $j_0 \leq j \leq j_0 + w + 1$*

13

$$- \ j = j_0 \ or \ j_0 + w + 1, \ and \ i_0 \leq i \leq j_0 + w + 1$$

The parameter w denotes the width *of the control zone.*

For instance, Figure 1 represents a square control zone of width 3; (i_0, j_0) corresponds to the upper-left node of the border of the control zone.

The set of control zones Ctr used by the protocol (see 2.2.2) has parameter W, the *order* of the protocol.

Definition 9 (Order of the protocol) *A protocol of* order *W on a torus network is defined by the following set Ctr of control zones:*

- $Ctr(W) = \bigcup_{1 \leq w \leq W, 1 \leq i_0 \leq N, 1 \leq j_0 \leq N} Sqr(i_0, j_0, w)$

For instance, $Ctr(3)$ is the set of all square control zones of width 1, 2 or 3. Notice that a node knowing its identifier (i, j) and the order W of the protocol can easily determine its set $myCtr$ (see 2.2.4), without any offline implementation.

4.2.3 Message complexity

Let us evaluate the exact number of messages sent, when the whole network is a reliable node set.

Let n be the number of nodes of the network, and W the order of the protocol (according to Definition 9). A square control zone of width w has $4(w+1)$ nodes on its border. According to Definition 9, there are n square control zones of a given width w. Let u be any node. Let us evaluate the number of messages related to u. First, all nodes once accept and send $(u, u.m_0)$ to their 4 neighbors, which makes $4n$ messages. Second, all nodes on the border of a control zone z once send $(u, u.m_0, z)$ to their 4 neighbors, which makes $4n \sum_{w=1}^{W} 4(1+w) = 8nW(W+3)$ messages. Thus, $4n^2$ standard messages are sent, the same as a standard broadcast protocol (see 2.1). In addition, $8W(W+3)n^2$ authorization messages are sent. However, in practice, they can contain a hash code of the authorized message: they are potentially way lighter.

4.2.4 Example of construction of a reliable node set

The main step of a simulation is the construction of a *reliable node set*, if it exists. Figure 3 represent a toy example (extracted from a supposedly larger network), for a protocol of order 3. Let us comment on this figure.

First, we determine a *safe node set* (see Definition 4), using Theorem 1. There actually exists a set Z of square control zones satisfying the conditions of Theorem 1. The blank squares correspond to the cores of the control zones of Z. These cores contain all Byzantine nodes, and do not intersect the union of the corresponding borders. Here, having control zones of width 3 is an advantage: with width 2, the upper-left group of Byzantine nodes could not be neutralized.

Then, we construct a *communicating node set S* (see Definition 5), using Theorem 2. We notice that the correct nodes surrounded by too many Byzantine nodes cannot be added to S, as they do not satisfy the conditions of Theorem 2. Here, having control zones of width 3 is a drawback: the presence of these zones make the conditions of Theorem 2 harder to satisfy, which limits the size of the communicating node set.
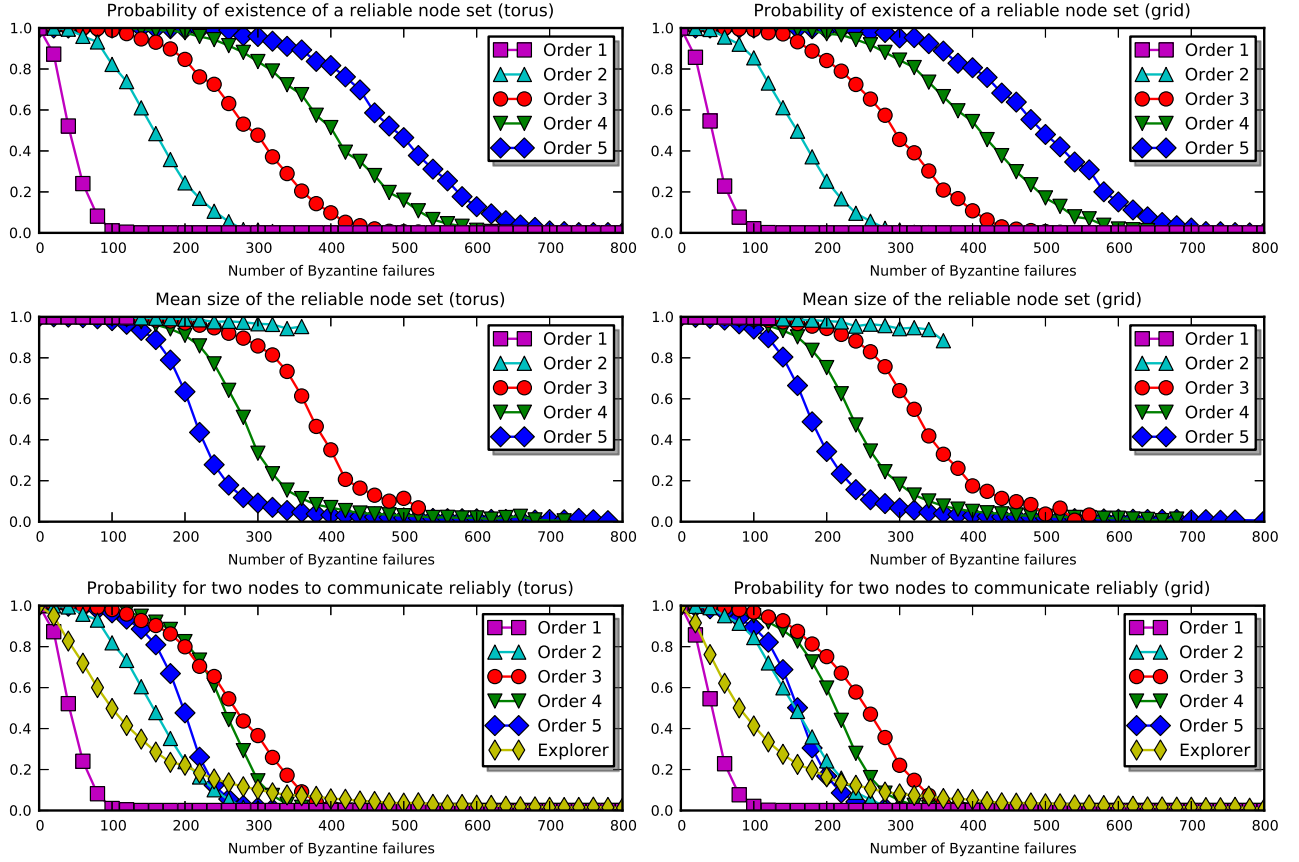
Figure 4: Influence of the parameters of simulation

4.3 Experimental results

In Figure 4, we represented the influence of the number of Byzantine nodes and of the order of the protocol. The left column gives the results for a 100×100 torus network. The right column gives the results for the corresponding grid network, obtained by an edge-cut on the torus. This edge-cut separates some control zones in 2 or 4 new zones. We consider that these new control zones have independent identifiers.

The first row of Figure 4 represents the probability that a reliable node set *exists*. As any correct node forms a communicating node set, this corresponds to the probability of existence of a safe node set. That is, the probability that a set Z of control zones satisfies the conditions of Theorem 1. Figure 4 shows that this probability *increases* with the order. To give an illustration: a group of 3×3 Byzantine nodes cannot be neutralized at order 2. But with the new zones added at order 3, it becomes possible. And so forth.

The second row of Figure 4 represents the mean size of the reliable node set, when it

exists. That is, the fraction of correct nodes covered by the reliable node set. Figure 4 shows that this fraction *decreases* with the order. Indeed, increasing the order makes the conditions of Theorem 2 harder to satisfy, as the set Z of Theorem 2 contains more zones. Therefore, the construction of the communicating node set is made more difficult.

The third row of Figure 4 represents the probability $P(n_B)$ for two correct nodes, choosen uniformly at random, to communicate reliably. In the previous plots, we showed that the order of the protol had a *positive* influence on the existence of a reliable node set, but a *negative* influence on its size. Therefore, an compromise between these two tendancies appears for order 3, for which the probability is optimal.

It is difficult to compare our proposal with the high-connectivity based approaches [4, 19] since they always assume the worst possible placement of Byzantine nodes. For example, Explorer [19] has the sender use all paths and the receiver validates the first message coming from $2k + 1$ node-disjoint paths by a majority. So an asynchronous schedule in a torus may always choose that node-disjoint paths going through Byzantine nodes arrive first, defeating the protocol anytime the number of Byzantine is greater than 1. In order to maximize the efficiency of Explorer in our context (*i.e.* assuming random placement of the nodes), we force the sender to use exactly 4 node-disjoint fixed paths. Then, for a particular sender and receiver, the Explorer protocol works if and only if Byzantine nodes are located on at most one of those four paths. We present the corresponding guarantees of the modified Explorer protocol in Figure 4. It turns out that our protocol outperforms the modified Explorer by a significant margin. For example, if the goal probability is $P(n_B) \geq 0.99$, then on the grid (resp. torus) topology, the modified version of Explorer can tolerate at most 5 (resp. 7) Byzantine nodes. Our approach can tolerate up to 50 (resp. 80) Byzantine nodes.

5 Conclusion

In this paper, we showed that, if we accept that a small minority of correct nodes does not communicate reliably in the presence of Byzantine failures, we make it possible to tolerate a large number of those failures, even in a low-connectivity network. We proposed an experimental methodology to obtain probabilistic guarantees, and illustrated this on torus and grid shaped networks, with an uniform distribution of Byzantine failures. Yet, the same principle could be applied to any network and any distribution of Byzantine failures. However, it requires to define a proper set of control zones to limit the Byzantine influence. Defining optimal sets of control zones for a given communication graph and Byzantine node distribution is a challenging open question.

References

[1] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* McGraw-Hill Publishing Company, New York, May 1998. 6.

[2] Vartika Bhandari and Nitin H. Vaidya. On reliable broadcast in a radio network. In Marcos Kawazoe Aguilera and James Aspnes, editors, *PODC*, pages 138–147. ACM, 2005.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[4] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[5] Vadim Drabkin, Roy Friedman, and Marc Segal. Efficient byzantine broadcast in wireless ad-hoc networks. In *DSN*, pages 160–169. IEEE Computer Society, 2005.

[6] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. The impact of topology on byzantine containment in stabilization. In *Proceedings of DISC 2010*, Lecture Notes in Computer Science, Boston, Massachusetts, USA, September 2010. Springer Berlin / Heidelberg.

[7] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. On byzantine containment properties of the min+1 protocol. In *Proceedings of SSS 2010*, Lecture Notes in Computer Science, New York, NY, USA, September 2010. Springer Berlin / Heidelberg.

[8] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. Bounding the impact of unbounded attacks in stabilization. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2011.

[9] Swan Dubois, Toshimitsu Masuzawa, and Sébastien Tixeuil. Maximum metric spanning tree made byzantine tolerant. In David Peleg, editor, *Proceedings of DISC 2011*, Lecture Notes in Computer Science (LNCS), Rome, Italy, September 2011. Springer Berlin / Heidelberg.

[10] Chiu-Yuen Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In Soma Chaudhuri and Shay Kutten, editors, *PODC*, pages 275–282. ACM, 2004.

[11] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[12] D. Malkhi, Y. Mansour, and M.K. Reiter. Diffusion without false rumors: on propagating updates in a Byzantine environment. *Theoretical Computer Science*, 299(1–3):289–306, April 2003.

[13] D. Malkhi, M. Reiter, O. Rodeh, and Y. Sella. Efficient update diffusion in byzantine environments. In *The 20th IEEE Symposium on Reliable Distributed Systems (SRDS '01)*, pages 90–98, Washington - Brussels - Tokyo, October 2001. IEEE.

[14] Toshimitsu Masuzawa and Sébastien Tixeuil. Bounding the impact of unbounded attacks in stabilization. In Ajoy Kumar Datta and Maria Gradinariu, editors, *SSS*, volume 4280 of *Lecture Notes in Computer Science*, pages 440–453. Springer, 2006.

[15] Toshimitsu Masuzawa and Sébastien Tixeuil. Stabilizing link-coloration of arbitrary networks with unbounded byzantine faults. *International Journal of Principles and Applications of Information Science and Technology (PAIST)*, 1(1):1–13, December 2007.

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

[17] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[18] Mikhail Nesterenko and Anish Arora. Tolerance to unbounded byzantine faults. In *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pages 22–29. IEEE Computer Society, 2002.

[19] Mikhail Nesterenko and Sébastien Tixeuil. Discovering network topology in the presence of byzantine nodes. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(12):1777–1789, December 2009.

[20] Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Inf. Process. Lett.*, 93(3):109–115, 2005.

[21] Yusuke Sakurai, Fukuhito Ooshita, and Toshimitsu Masuzawa. A self-stabilizing link-coloring protocol resilient to byzantine faults in tree networks. In *Principles of Distributed Systems, 8th International Conference, OPODIS 2004*, volume 3544 of *Lecture Notes in Computer Science*, pages 283–298. Springer, 2005.