Self-Stabilizing Byzantine-Resilient Communication in Dynamic Networks

Alexandre Maurer
Mohammed VI Polytechnic University, School of Computer Science, Ben Guerir, Morocco
alexandre.maurer@um6p.ma

Abstract

We consider the problem of communicating reliably in a dynamic network in the presence of up to k Byzantine failures. It was shown that this problem can be solved if and only if the dynamic graph satisfies a certain condition, that we call "RDC condition". In this paper, we present the first self-stabilizing algorithm for reliable communication in this setting – that is: in addition to permanent Byzantine failures, there can also be an arbitrary number of transient failures. We prove the correctness of this algorithm, provided that the RDC condition is "always eventually satisfied".

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Dynamic networks, Self-stabilization, Byzantine failures

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2020.27

1 Introduction

As networks grow larger and larger, it becomes more and more likely that some of their nodes will behave incorrectly at some point, for various reasons. Therefore, it is crucial to design *robust* networks, that is: networks that still satisfy some essential properties even when some nodes fail. There are many models of node failure, but the most general one is the Byzantine model [13]: we assume that the failing nodes can have any arbitrary behavior. Thus, we encompass any possible type of failure.

Here, we consider the problem of *reliable communication*: any two correct nodes of the network should be able to exchange messages reliably, despite the potentially malicious behavior of some Byzantine nodes.

One way to solve this problem is to use cryptography [6, 10]: the nodes use digital signatures to authenticate the sender across multiple hops. However, cryptography is not always reliable (see, for instance, the Heartbleed bug [1] discovered in the widely deployed OpenSSL software). The Defense in Depth paradigm [14] recommends the use of multiple security layers, including non-cryptographic layers. For instance, if the cryptographic layer is compromised (bug, virus, ...), a cryptography-free communication layer can be used to safely broadcast a patch, or to update cryptographic keys. Thus, even when cryptography is available, it is interesting to develop non-cryptographic solutions. In the following, we focus on these non-cryptographic solutions.

When it comes to non-cryptographic solutions, many solutions have been proposed for *static* networks (e.g. [4, 11, 22, 15, 17, 16, 7, 21, 8]). Fewer papers consider *dynamic* networks (e.g. [20, 3, 19]), where the topology changes over time. In particular, [19] showed the necessary and sufficient condition on the graph topology to tolerate up to k Byzantine nodes in a dynamic network. We call this condition the **RDC** (**R**eliable **D**ynamic **C**ommunication) condition. (This condition is described in Section 2.3, but its exact terms do not matter at this point.)

There already exists an algorithm for reliable communication in dynamic networks tolerating permanent Byzantine failures [19]. Hence, the question: is it possible to go further in terms of reliability guarantees?

The concept of *multitolerance* [2, 12] describes the property of a system to tolerate multiple fault-classes. One of the strongest possible level of multitolerance is the following: tolerating, not only a given number of permanent Byzantine failures, but also an *arbitrary* number of transient failures. This second point consists in assuming that the system can have any arbitrary initial state. More precisely: (1) each correct node can have any arbitrary initial state, and (2) communication channels between nodes can contain arbitrary messages, "sent but not yet received". In other words, such an algorithm is *self-stabilizing* [9]: it can recover from any incorrect initial state. Satisfying this property in the presence of Byzantine failures can be particularly challenging: Byzantine nodes, in addition to their usual malicious behavior, can also actively try to prevent stabilization. The problem of reliable communication despite transient and Byzantine failures was already considered in [18], but for a specific class of static networks and a non-standard criteria on Byzantine failures (i.e. the distance between Byzantine failures).

Our contribution is the following. we present the first *self-stabilizing* Byzantine-resilient algorithm for reliable communication in a dynamic network. We prove the correctness of our algorithm under the following assumption: the aforementioned RDC condition is "always eventually satisfied" – that is: for any time t, there always exists a time $t' \geq t$ where the RDC condition is satisfied.

The rest of the paper is organized as follows. In Section 2, we present the setting (definitions, assumptions...). In Section 3, we describe the problem. In Section 4, we motivate the main assumption w.r.t. the problem (i.e., that the RDC condition is "always eventually satisfied"). In Section 5, we describe our algorithm. In Section 6, we prove its correctness.

2 Preliminaries

The setting is mostly similar to [19]. We recall several definitions below.

2.1 Network model

We consider a continuous temporal domain $\mathbb{R}^+$. We model the system as a time varying graph, as defined by Casteigts, Flocchini, Quattrociocchi and Santoro [5], where vertices represent the processes and edges represent the communication links (or channels). A time varying graph is a dynamic graph represented by a tuple $\mathcal{G} = (V, E, \rho, \zeta)$ where:

- V is the set of *nodes*.
- $E \subseteq V \times V$ is the set of *edges*.
- $\rho : E \times \mathbb{R}^+ \to \{0, 1\}$ is the *presence* function: $\rho(e, t) = 1$ indicates that edge e is present at time t.
- $\zeta : E \times \mathbb{R}^+ \to \mathbb{R}^+$ is the *latency* function: $\zeta(e, t) = T$ indicates that a message sent at time t through edge e will be received, at worst, at time $t + T$.

(Note: although one could imagine a simpler definition, we prefer to stick with the standard definition of [5].)

2.2 Definitions

Informally, a *dynamic path* is a sequence of nodes a message can traverse, with respect to network dynamicity and latency.

▶ **Definition 1** (Dynamic path). *A sequence of distinct nodes* $(u_1, \ldots, u_n)$ *is a* "dynamic path *from* u_1 *to* u_n *starting after* t_0*" if there exists a sequence of times* $(t_1, \ldots, t_n)$ *such that* $t_0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$*, and,* $\forall i \in \{1, \ldots, n-1\}$*, we have:*

- $e_i = (u_i, u_{i+1}) \in E$*, i.e., there exists an edge connecting* u_i *to* u_{i+1}*.*
- $\forall t \in [t_i, t_i + \zeta(e_i, t_i)]$*,* $\rho(e_i, t) = 1$*, i.e.,* u_i *can send a message to* u_{i+1} *at time* t_i*.*
- $\zeta(e_i, t_i) \leq t_{i+1} - t_i$*, i.e., the aforementioned message is received by time* t_{i+1}*.*

With this definition, we can now define the following elements:

- Let $Dyn(p, q, t_0)$ be the set of node sets $\{u_1, \ldots, u_n\}$ such that $(p, u_1, \ldots, u_n, q)$ is a dynamic path starting after t_0 (if (p, q) is a dynamic path starting after t_0, $Dyn(p, q, t_0)$ contains an empty set).
- For any nonempty set of nonempty node sets $X = \{S_1, \ldots, S_n\}$, let $Cut(X)$ be the set of node sets C such that, $\forall i \in \{1, \ldots, n\}$, $C \cap S_i \neq \emptyset$ (C contains at least one node from each set S_i).[1]
- For any nonempty set of node sets X, we define $MinCut(X)$ as follows:
  - If X contains an empty set, $MinCut(X) = +\infty$.[2]
  - Otherwise: $MinCut(X) = \min_{C \in Cut(X)} |C|$ (the size of the smallest element of $Cut(X)$).

We say that a node *multicasts* a message m when it sends m to all nodes in its current local topology.[3]

2.3 Setting and assumptions

We make the same basic assumptions as previous works on the subject (e.g. [4, 7, 11, 15, 16, 17, 21, 22]):

1. Each node has a unique identifier.
2. When a node q receives a message through channel (p, q), it knows that p sent the message.
3. Each node u is aware of its *local topology* at any given time t (here, u knows the set of nodes v such that $\rho((u, v), t) = 1$)[4].
4. The time required for computation and sending messages is negligible w.r.t. the delays between changes in the dynamic graph.

Permanent Byzantine failures. An omniscient adversary can select up to k nodes as *Byzantine*. These nodes can have a totally arbitrary and unpredictable behavior defined by the adversary (including tampering or dropping messages, or simply crashing). Of course, correct nodes are unable to know *a priori* which nodes are Byzantine.

[1] Here, "one node" can be any node of S_i.

[2] Here, an empty set corresponds to the case where the two nodes p and q trying to communicate are directly connected at some point. Therefore, no amount of (other) nodes suppression will be sufficient to disconnect them.

[3] We use "multicast" instead of "broadcast" here to avoid common misunderstandings: "broadcast" can refer to some very specific problems in distributed computing (e.g. "Byzantine Broadcast"), not necessarily related to our problem.

[4] Note that this assumption is necessary to ensure that each existing dynamic path can be explored. For instance, it is possible that some edges appear during exactly the time required to send a message. In such a situation, the sending node must be immediately aware of the topology change.

Transient failures. In addition to Byzantine failures, for the correct nodes, any variable of their algorithm can have any arbitrary initial value. Besides, for any two neighbor nodes u and v, a channel connecting u and v can initially contain any set of messages "sent by u but not yet received by v". In the following, this set is called $Sen(u, v)$.[5]

RDC condition. In [19], it was shown that the necessary and sufficient condition for reliable communication in a dynamic network is the following: for each pair of nodes $\{p, q\}$, $MinCut(Dyn(p, q, 0)) > 2k$.

We call this condition the **RDC** (**R**eliable **D**ynamic **C**ommunication) condition.

Main assumption. In this paper, we assume that the RDC condition is "always eventually satisfied", that is: for any two nodes p and q and for any time t_0, $MinCut(Dyn(p, q, t_0)) > 2k$.[6] In Section 4, we motivate the "always" of "always eventually satisfied" w.r.t the problem.

3 The Problem

Let us assume that each correct node p has a *fixed* attribute $p.m_0$ (the message that p wants to broadcast, not part of the initial state) and a memory set $p.Acc$ (where received messages are stored). For each correct node q, when the set $q.Acc$ contains a tuple (p, m), we say that q *accepts* the message m as being from p.

The problem we try to solve is the following: finding an algorithm (for correct nodes) such that, given the aforementioned setting, we have the two following properties:

For any two correct nodes p and q, there exists a time t such that, after t...

1. **[Safety]** There exists no $m' \neq p.m_0$ such that $(p, m') \in q.Acc$.
2. **[Liveness]** $(p, p.m_0) \in q.Acc$.

The first property ensures that, after t, no wrong message (i.e., a message m' pretending to be the message $p.m_0$ from p) is accepted by q. The second property ensures that, after t, the correct message $p.m_0$ is accepted by q.

Such an algorithm would be *self-stabilizing* w.r.t. these two properties: no matter what the initial state is, these properties are always eventually satisfied.[7]

In the following, we motivate the main assumption w.r.t this problem (Section 4), then present our algorithm (Section 5) and prove that it solves the problem (Section 6).

4 Motivation of the main assumption

In this section, we motivate the main assumption (stated in Section 2.3). In particular, we motivate the "always" in "always eventually satisfied". In Theorem 2 below, we show that, if the RDC condition in just "eventually satisfied", no algorithm can solve the desired problem in the setting we consider. We do this by constructing two possible scenarios (allowed in this setting) that lead to incompatible outcomes.

---

[5] We do not assume any bound on the capacity of communication channels: the number of fake initial messages "sent but not yet received" can be arbitrarily high.

[6] If we take "always eventually satisfied" strictly, the proposition is: for any two nodes p and q and for any time t, there exists $t_0 \geq t$ such that $MinCut(Dyn(p, q, t_0)) > 2k$. But this proposition immediately implies the aforementioned (simpler) proposition.

[7] As usually done in self-stabilization papers, we assume that the algorithm itself is "hardwired", and cannot be corrupted (otherwise, it would be impossible to give any guarantee: the behavior of all nodes would be completely arbitrary). However, any variable used by the algorithm can have any arbitrary initial value.

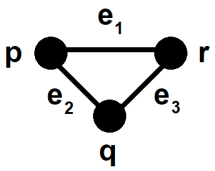

Figure 1 Graph used for the proof of Theorem 2.

▶ **Theorem 2.** *Let us assume the aforementioned setting (with transient failures, etc), but with one small change: the RDC condition is just eventually satisfied (and not "always eventually satisfied"). Then, there exists no algorithm solving the problem of Section 3.*

Proof. Suppose the opposite: there exists such an algorithm. Let m_1 and m_2 be two distinct messages.

Let us assume that $k = 0$ (no Byzantine failures). Then, the RDC condition simplifies as follows: there must exist one dynamic path between the sender and the receiver.

Consider the static network of Figure 1. In the following, we use this static network to describe two dynamic networks, and show a contradiction. Let x be any message that can be sent by q to r.

We first describe a scenario $S_1(x)$ (of which x is a parameter). In this scenario, $p.m_0 = m_1$. The evolution of the dynamic graph is the following:
Step 1: Edge e_1 appears, then disappears.
Step 2: Edge e_3 appears, then disappears. During this time, r receives x from q.[8]

With Step 1, there exists a dynamic path from p to r. Thus, as the algorithm solves the desired problem, there exists a time t_1 after which we have $(p, m_1) \in r.Acc$.[9] Let y be the state of r between Step 1 and Step 2.

We now describe a second scenario S_2. In this scenario, $p.m_0 = m_2$, and the initial state of r is y. The evolution of the dynamic graph is the following:
Step 1': Edge e_2 appears, then disappears.
Step 2': Edge e_3 appears, then disappears.

With these two steps, there exists a dynamic path from p to r. Thus, as the algorithm solves the desired problem, there exists a time t_2 after which we have $(p, m_2) \in r.Acc$. Let $t_3 = \max(t_1, t_2)$, and let x' be the message sent from q to r during Step 2'.

Now, let us consider (1) $S_1(x')$ after Step 1 and (2) S_2. From the point of view of r, what happens then is exactly the same: r starts in state y, then receives x' from q. Thus, according to both scenarios, after t_3, we have *both* $(p, m_1) \in r.Acc$ and $(p, m_2) \in r.Acc$.

As the algorithm solves the desired problem, it implies that $p.m_0 = m_1 = m_2$, which contradicts our initial assumption ($m_1 \neq m_2$). Thus, the result. ◀

5 Algorithm

In this section, we provide:
- An explanation of the main intuition behind the algorithm (5.1).
- A full description of our algorithm (5.2).
- A more detailed explanation of the algorithm, to facilitate its understanding (5.3).

[8] This can happen for any x, as the initial content of communication channels between nodes is arbitrary, according to our model (see Section 2.3).
[9] The fact that p sends a message to r through edge e_1 is implicit here, as the outcome of the algorithm is assumed to be guaranteed by the RDC condition (in the context of this proof by contradiction).

5.1 Intuition behind the algorithm

The initial idea of the algorithm is similar to [19]:

- Broadcast each message through each possible dynamic path, and register the identifier of the nodes forwarding the message.
- Before accepting the message: check if the various instances of the same message satisfy the RDC condition.

This is done through rules 2, 3 and 4 of the algorithm below. However, in a setting with transient failures, this is not sufficient: for instance, it is possible that a false message has already been accepted, or that enough messages have been sent to have a false message accepted in the future.

To solve this problem, we associate an integer α to each message. Now, each message looks like this: (s, m, S, α), where s is the sender (or pretending to be so), m is the content of the message, and S is the set of nodes crossed by the message.

The key idea here is the following: the algorithm is designed so that false transient messages are "stuck" with the value α they initially have (as shown in Lemma 3). Therefore, if correct nodes keep broadcasting their message with increasing values of α (Rule 1), and if we give priority to messages with the highest value of α (Rule 5), the correct messages are eventually accepted. Of course, Byzantine nodes can still broadcast messages with arbitrarily high values of α, but rules 2, 3 and 4 ensure that this will never be sufficient (according to the assumptions on the topology of the network).

5.2 Full description

For each correct node u, let $u.m_0$ be the message that u wants to broadcast; u also maintains the following variables:

- An integer $u.\alpha$ (with an arbitrary initial value).
- Three memory sets $u.\Omega$, $u.Acc_0$ and $u.Acc$ (the initial content of these sets is completely arbitrary).[10]

Let $A(u, s, m)$ be the set of integers α such that $(s, m, \alpha) \in u.Acc_0$. Let $Count(u, s, m) = |A(u, s, m)|$.

In **Rule 1** below, "keep doing X" means that the algorithm will always eventually do X (i.e., if t is the current time, there always exists a time $t' \geq t$ at which X is done).

Each correct node u obeys to the following rules:

- **Rule 1.** Keep doing the following: $u.\alpha := u.\alpha + 1$, and add $\{(u, u.m_0, \emptyset, u.\alpha)\}$ to $u.\Omega$.
- **Rule 2.** Whenever $u.\Omega$ or the set of neighbors of u changes[11]: multicast $u.\Omega$.
- **Rule 3.** When u receives a set Ω' from a neighbor v: $\forall (s, m, S, \alpha) \in \Omega'$, if $v \notin S$, add $(s, m, S \cup \{v\}, \alpha)$ to $u.\Omega$.
- **Rule 4.** When there exist s, m, α and n sets $S_1, \ldots, S_n$ such that the following 3 conditions are satisfied:
  1. $\forall i \in \{1, \ldots, n\}$, $(s, m, S_i \cup \{s\}, \alpha) \in u.\Omega$.
  2. $MinCut(\{S_1, \ldots, S_n\}) > k$.
  3. $(s, m, \alpha) \notin u.Acc_0$.

  Add (s, m, α) to $u.Acc_0$.

[10] See 5.3 for an explanation of the role of these memory sets.
[11] See assumption 3 in Section 2.3.

- **Rule 5.** When there exist s and m such that the following 3 conditions are satisfied:
  1. $Count(u, s, m) \geq 1$.
  2. There exists no $m' \neq m$ such that $Count(u, s, m) \leq Count(u, s, m')$.
  3. $(s, m) \notin u.Acc$.

  Do the following:
  1. $\forall m'$ such that $(s, m') \in u.Acc$, remove (s, m') from $u.Acc$.
  2. Add (s, m) to $u.Acc$.

5.3 Detailed explanation

In addition to the full description of the algorithm, we provide more detailed explanations here, to facilitate its understanding.

First, let us explain the role of the variables of each correct node u.

- The integer $u.\alpha$ is a counter that can only increase. This mechanism is used to defeat false transient messages, as we explain below.
- The memory set $u.\Omega$ stores all messages received by u, without discrimination, as described in Rule 3. The elements stored in this set are tuples (p, m, S, α), where...
  - p is supposedly the author of the message;
  - m is the content of the message supposedly sent by p (it can be any piece of information);
  - S is the set of nodes that supposedly forwarded the message;
  - α is an integer associated to the rest of the tuple.
- The role of the memory set $u.Acc_0$ is to store messages that are "pre-accepted", that is: messages that have been (or appear to have been) sent through several dynamic paths, in accordance with the RDC condition. These messages are added to $u.Acc_0$ in Rule 4.
- The role of the memory set $u.Acc$ is to store messages that are "truly accepted", that is: messages from $u.Acc_0$ satisfying a particular condition w.r.t their integers α. The goal is that, eventually, $u.Acc$ contains each message from each correct node, and no false message (that is: a tuple (s, m') such that $m' \neq s.m_0$).

Now, let us provide an informal explanation for each rule, for a given node u.

- **Rule 1:** This rule ensures that u always eventually does the following: increment its counter $u.\alpha$, and add its message $u.m_0$ (associated with the new value of $u.\alpha$) to its set $u.\Omega$.
- **Rule 2:** This rule sends the content of $u.\Omega$ to all neighbors of u whenever $u.\Omega$ is updated (which happens either in Rule 1 or 3).
- **Rule 3:** Whenever u receives a set Ω' from a neighbor node v, it adds each tuple contained in Ω' to its own set $u.\Omega$. However, before doing so, it adds v to the third element of the tuple (i.e., the set S supposed to register all the nodes that forwarded this tuple).
- **Rule 4:** This rule adds some elements to $u.Acc_0$ (the set of "pre-accepted" messages) when some conditions are satisfied in $u.\Omega$. More precisely: there must exist a node identifier s, a message m, a value α and n sets of nodes $(S_1, \ldots, S_n)$ such that...
  - For each of these node sets, $u.\Omega$ contains a tuple associating s, m, α and this node set (enriched with s). This condition ensures that at least one of these sets actually corresponds to the correct nodes that actually forwarded the message (as shown in the correctness proof).
  - The n sets of nodes cannot be (all) cut by removing k nodes ($MinCut(\{S_1, \ldots, S_n\}) > k$).
  - $u.Acc_0$ does not already contain (s, m, α).

The underlying idea of Rule 4 is to prevent k Byzantine nodes from cooperating to make u add false messages to $u.Acc_0$. Indeed, as shown in the correctness proof, k Byzantine nodes are not sufficient to bypass the filter of Rule 4.

- **Rule 5:** This rule adds some elements to $u.Acc$ (the set of "truly accepted" messages) when some conditions are satisfied in $u.Acc_0$. More precisely: when there exists a node identifier s and a message m such that the number of tuples $(s, m, \alpha) \in u.Acc_0$ is unmatched by any other message m', (s, m) is added to $u.Acc$, and replaces any previous message (s, m').
  The underlying idea of Rule 5 is to (eventually) eliminate and replace any false transient messages that $u.Acc_0$ may initially contain: as correct messages keep being generated with new values of α (according to Rule 1), they will eventually outnumber false messages.

6 Correctness proof

We prove the correctness of the algorithm in Theorem 9.

Before going further, we define Z, the set of values α such that a tuple (s, m, S, α) exists somewhere in the system at $t = 0$:

- Let Z_1 be the set of integers α such that, $\forall \alpha \in Z_1$, the following proposition is true: at $t = 0$, there exist s, m, S, u, v and Ω such that $\Omega \in Sen(u, v)$ and $(s, m, S, \alpha) \in \Omega$.
- Let Z_2 be the set of integers α such that, $\forall \alpha \in Z_2$, the following proposition is true: at $t = 0$, there exists s, m, S, and u such that $(s, m, S, \alpha) \in u.\Omega$.
- Let $Z = Z_1 \cup Z_2$.

Overview of the proof

- In Lemma 3, we show that no false message can be accepted when it is associated with an integer $\alpha \notin Z$. Thus, the number of integers α used by false messages is bounded by $|Z|$ (Lemma 4).
- In Lemma 5, we show that, for any α_0, a correct node eventually broadcasts messages with $\alpha \geq \alpha_0$.
- In Lemma 6, we show that correct messages successfully broadcast along each dynamic path. Thus, as shown in Lemma 7, each correct message with a large enough α is eventually "pre-accepted" (that is, added to the set Acc_0 of the receiver).
- From Lemmas 5 and 7, it follows that each correct message is eventually pre-accepted (Lemma 8).
- As the number of integers α used by false messages is bounded (Lemma 4), it follows that, eventually, the only messages accepted are the correct ones (Theorem 9).

▶ **Lemma 3.** *Let $\alpha \notin Z$. Let p and q be two correct nodes. Let $m' \neq p.m_0$. Then, we never have $(p, m', \alpha) \in q.Acc_0$.*

Proof. Suppose the opposite: $(p, m', \alpha) \in q.Acc_0$. Then, there exists $\{S_1, \ldots, S_n\}$ such that, $\forall i \in \{1, \ldots, n\}$, $(p, m', S_i \cup \{p\}, \alpha) \in q.\Omega$ and $MinCut(\{S_1, \ldots, S_n\}) > k$.

Suppose that each node set $S \in \{S_1, \ldots, S_n\}$ contains at least one Byzantine node. If C is the set of Byzantine nodes, then $C \in Cut(\{S_1, \ldots, S_n\})$ and $|C| \leq k$. This is impossible because $MinCut(\{S_1, \ldots, S_n\}) > k$. Therefore, there exists $S \in \{S_1, \ldots, S_n\}$ such that S does not contain any Byzantine node.

Now, let us use the correct dynamic path corresponding to S to show that $m' = p.m_0$. Let $n' = |S \cup \{p\}|$. Let us show the following property $\mathcal{P}_i$ by induction, $\forall i \in \{0, \ldots, n'\}$: there exists a correct node u_i and a set of correct nodes X_i such that $(p, m', X_i, \alpha) \in u_i.\Omega$ and $|X_i| = |S \cup \{p\}| - i$.

- As $S \in \{S_1, \ldots, S_n\}$, $(p, m', S \cup \{p\}, \alpha) \in q.\Omega$. Thus, $\mathcal{P}_0$ is true if we take $u_0 = q$ and $X_0 = S \cup \{p\}$.
- Let us now suppose that $\mathcal{P}_i$ is true, for $i < n'$. As $(p, m', X_i, \alpha) \in u_i.\Omega$, according to Rule 3 of our algorithm, it implies that u_i received Ω' from a node v, with $(p, m', X, \alpha) \in \Omega'$, $v \notin X$ and $X_i = X \cup \{v\}$. Thus, $|X| = |X_i| - 1 = |S \cup \{p\}| - (i + 1)$.
  As $v \in X_i$ and X_i is a set of correct nodes, v is correct and behaves according to our algorithm. Then, as v sent Ω', according to Rule 2 of our algorithm, we necessarily have $\Omega' \subseteq v.\Omega$. Thus, as $(p, m', X, \alpha) \in \Omega'$, we have $(p, m', X, \alpha) \in v.\Omega$. Hence, $\mathcal{P}_{i+1}$ is true if we take $u_{i+1} = v$ and $X_{i+1} = X$.

By the induction principle, $\mathcal{P}_{n'}$ is true. As $|X_{n'}| = 0$, $X_{n'} = \varnothing$ and $(p, m', \varnothing, \alpha) \in u_{n'}.\Omega$. As $u_{n'}$ is a correct node and follows our algorithm, and as $\alpha \notin Z$, the only possibility to have $(p, m', \varnothing, \alpha) \in u_{n'}.\Omega$, according to Rule 1, is that $u_{n'} = p$ and $m' = p.m_0$, which contradicts our initial hypothesis. Thus, the result. ◀

▶ **Lemma 4.** *For any two correct nodes p and q, $\forall m' \neq p.m_0$, we have $Count(q, p, m') \leq |Z|$.*

Proof. Suppose that, at some point, there exist q, p and m' such that $Count(q, p, m') > |Z|$. It implies that there exists $\alpha \notin Z$ such that $(p, m', \alpha) \in q.Acc_0$. According to Lemma 3, this is impossible. Thus, $Count(q, p, m') \leq |Z|$. ◀

▶ **Lemma 5.** *Let p be a correct node. For any integer α_0, there exists $\alpha \geq \alpha_0$ such that p multicasts a set Ω containing $(p, p.m_0, \emptyset, \alpha)$.*

Proof. According to the algorithm, either we initially have $p.\alpha \geq \alpha_0$, or we eventually have $p.\alpha \geq \alpha_0$. Thus, according to Rule 1, p eventually adds $\{(p, p.m_0, \emptyset, \alpha)\}$ to $p.\Omega$, with $\alpha \geq \alpha_0$.
Then, according to Rule 2, p multicasts a set Ω containing $(p, p.m_0, \emptyset, \alpha)$. ◀

▶ **Lemma 6.** *Let p and q be two correct nodes. Let α be an integer. Suppose that p multicasts a set Ω containing $(p, p.m_0, \emptyset, \alpha)$ at time t. Suppose that there exists a dynamic path $(u_1, \ldots, u_n)$, starting after t, such that $p = u_1$, $q = u_n$, and $\forall i \in \{1, \ldots, n\}$, u_i is correct. Then, eventually, we have $(p, p.m_0, \{u_1, \ldots, u_{n-1}\}, \alpha) \in q.\Omega$.*

Proof. Let us prove the following property $\mathcal{P}_i$ by induction, $\forall i \in \{1, \ldots, n\}$: u_i eventually multicasts a set Ω containing $(p, p.m_0, \{u_1, \ldots, u_{i-1}\}, \alpha)$.

- $\mathcal{P}_1$ is true, as p multicasts a set Ω containing $(p, p.m_0, \emptyset, \alpha)$.
- Suppose that $\mathcal{P}_i$ is true, for some $i \in \{1, \ldots, n-1\}$. Then, u_{i+1} eventually receives a set Ω from u_i containing $(p, p.m_0, \{u_1, \ldots, u_{i-1}\}, \alpha)$, and adds $(p, p.m_0, \{u_1, \ldots, u_i\}, \alpha)$ to $u_i.\Omega$. Thus, according to Rule 2, u_i multicasts a set Ω' containing $(p, p.m_0, \{u_1, \ldots, u_i\}, \alpha)$, and $\mathcal{P}_{i+1}$ is true.

As $\mathcal{P}_n$ is true, $u_n = q$ eventually multicasts a set Ω containing $(p, p.m_0, \{u_1, \ldots, u_{n-1}\}, \alpha)$. According to the algorithm, it implies that $(p, p.m_0, \{u_1, \ldots, u_{n-1}\}, \alpha) \in q.\Omega$. ◀

▶ **Lemma 7.** *Let p and q be two correct nodes. Let α be an integer. Suppose that p multicasts a set Ω containing $(p, p.m_0, \emptyset, \alpha)$ at time t. Then, eventually, we have $(p, p.m_0, \alpha) \in q.Acc_0$.*

Proof. Let $\{S_1, \ldots, S_n\}$ be the set of node sets $S \in Dyn(p, q, t)$ that contain no Byzantine node. Similarly, let $\{X_1, \ldots, X_{n'}\}$ be the set of node sets $X \in Dyn(p, q, t)$ that contain at least one Byzantine node.

Let us suppose that $MinCut(\{S_1, \ldots, S_n\}) \leq k$. Then, there exists a node set $C \in Cut(\{S_1, \ldots, S_n\})$ such that $|C| \leq k$. Let $C' = C \cup B$ (where B is the set of Byzantine nodes). Thus, $C' \in Cut(\{S_1, \ldots, S_n\} \cup \{X_1, \ldots, X_{n'}\}) = Cut(Dyn(p, q, t))$, and $|C'| \leq 2k$. Thus, $MinCut(Dyn(p, q, t)) \leq 2k$, which contradicts our hypothesis. Therefore, $MinCut(\{S_1, \ldots, S_n\}) > k$.

$\forall S = \{v_1, \ldots, v_n\} \in Dyn(p, q, t)$, $(p, v_1, \ldots, v_n, q)$ is a dynamic path. Therefore, according to Lemma 6, $\forall S \in \{S_1, \ldots, S_n\}$, we eventually have $(p, p.m_0, \{p\} \cup \{v_1, \ldots, v_n\}, \alpha) \in q.\Omega$. Thus, according to the algorithm, we eventually have $(p, p.m_0, \alpha) \in q.Acc_0$. ◀

▶ **Lemma 8.** *Let p and q be two correct nodes. $\forall \alpha_0$, there exists $\alpha \geq \alpha_0$ such that we eventually have $(p, p.m_0, \alpha) \in q.Acc_0$.*

Proof. The result follows from Lemma 5 and Lemma 7. ◀

▶ **Theorem 9.** *For any two correct nodes p and q, there exists a time t such that, after t, (1) there exists no $m' \neq p.m_0$ such that $(p, m') \in q.Acc$, and (2) $(p, p.m_0) \in q.Acc$.*

Proof. According to Lemma 4, $\forall m' \neq p.m_0$, $Count(q, p, m') \leq |Z|$. According to Lemma 8 we eventually have $Count(q, p, p.m_0) \geq |Z| + 1$. Thus, the result, according to Rule 5 of our algorithm. ◀

7 Conclusion

In this paper, we provided the first self-stabilizing and Byzantine-resilient algorithm for reliable communication in dynamic networks, and proved its correctness. To go further, one could consider more probabilistic settings: random positions for Byzantine nodes (each node may have a given probability to be Byzantine), random evolution of the dynamic graph... One could also consider permanent failures at the level of communication channels (e.g., regularly and randomly dropping messages). Finally, an interesting open question would be to consider the time and space complexity of solving this problem, and see if some optimizations could be made w.r.t. these metrics.

References

1 The Heartbleed Bug (http://heartbleed.com).

2 Anish Arora and Sandeep S. Kulkarni. Component based design of multitolerant systems. *IEEE Trans. Software Eng.*, 24(1):63–78, 1998. doi:10.1109/32.663998.

3 B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens. ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks. *ACM Transactions on Information and System Security*, 11:18:1–18:35, 2007.

4 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. doi:10.1145/1073814.1073841.

5 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

6 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *OSDI*, pages 173–186, 1999. doi:10.1145/296806.296824.

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

8 Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40, January 1993.

9 Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000. URL: `http://www.cs.bgu.ac.il/%7Edolev/book/book.html`.

10 Vadim Drabkin, Roy Friedman, and Marc Segal. Efficient Byzantine broadcast in wireless ad-hoc networks. In *DSN*, pages 160–169. IEEE Computer Society, 2005. `doi:10.1109/DSN.2005.42`.

11 Chiu-Yuen Koo. Broadcast in radio networks tolerating Byzantine adversarial behavior. In Soma Chaudhuri and Shay Kutten, editors, *PODC*, pages 275–282. ACM, 2004. `doi:10.1145/1011767.1011807`.

12 Sandeep S. Kulkarni and Anish Arora. Compositional design of multitolerant repetitive byzantine agreement. In *Foundations of Software Technology and Theoretical Computer Science, 17th Conference, Kharagpur, India, December 18-20, 1997, Proceedings*, pages 169–183, 1997. `doi:10.1007/BFb0058030`.

13 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. `doi:10.1145/357172.357176`.

14 R. Lippmann, K. Ingols, C. Scott, and K. Piwowarski. Validating and restoring defense in depth using attack graphs. *IEEE Military Communications Conference*, 2006.

15 Alexandre Maurer and Sébastien Tixeuil. Limiting Byzantine influence in multihop asynchronous networks. In *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems (ICDCS 2012)*, pages 183–192, June 2012.

16 Alexandre Maurer and Sébastien Tixeuil. On Byzantine broadcast in loosely connected networks. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC 2012)*, volume 7611 of *Lecture Notes in Computer Science*, pages 183–192. Springer, 2012.

17 Alexandre Maurer and Sébastien Tixeuil. A scalable Byzantine grid. In *Proceedings of the 14th International Conference on Distributed Computing and Networking (ICDCN 2013)*, volume 7730 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2013.

18 Alexandre Maurer and Sébastien Tixeuil. Self-stabilizing byzantine broadcast. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*, pages 152–160. IEEE Computer Society, 2014. `doi:10.1109/SRDS.2014.10`.

19 Alexandre Maurer, Sébastien Tixeuil, and Xavier Défago. Communicating reliably in multihop dynamic networks despite byzantine failures. In *34th IEEE Symposium on Reliable Distributed Systems, SRDS 2015, Montreal, QC, Canada, September 28 - October 1, 2015*, pages 238–245, 2015. `doi:10.1109/SRDS.2015.10`.

20 Henrique Moniz, Nuno Ferreira Neves, and Miguel Correia. Turquois: Byzantine consensus in wireless ad hoc networks. *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010)*, 2010.

21 Mikhail Nesterenko and Sébastien Tixeuil. Discovering network topology in the presence of Byzantine faults. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 20(12):1777–1789, December 2009. `doi:10.1109/TPDS.2009.25`.

22 Andrzej Pelc and David Peleg. Broadcasting with locally bounded Byzantine faults. *Inf. Process. Lett.*, 93(3):109–115, 2005. `doi:10.1016/j.ipl.2004.10.007`.