Multi-hop Byzantine Reliable Broadcast Made Practical

Silvia Bonomi, Giovanni Farina, Sébastien Tixeuil

Multi-hop Byzantine Reliable Broadcast Made Practical

Silvia Bonomi*, Giovanni Farina†*, Sébastien Tixeuil†

*Dipartimento di Ingegneria Informatica Automatica e Gestionale
"Antonio Ruberti",
Sapienza Università di Roma, Rome, Italy
bonomi@diag.uniroma1.it

†Sorbonne Université,
CNRS, Laboratoire d'Informatique de Paris 6, LIP6,
F-75005 Paris, France
Giovanni.Farina@lip6.fr, Sebastien.Tixeuil@lip6.fr

Abstract

In this paper, we revisit Byzantine-tolerant reliable broadcast algorithms in multi-hop networks. To tolerate up to f Byzantine nodes, previous solutions require an exponential number of messages to be sent over the network. We propose optimizations that preserve the safety and liveness properties of the original algorithms, while highly decreasing their observed message complexity when simulated on two families of random graphs with suitable connectivity.

1 Introduction

Designing dependable and secure distributed systems and networks, that are able to cope with various types of adversaries (ranging from simple errors to internal or external attackers), requires to integrate those risks from the very early design stages. The most general attack model in a distributed setting is the Byzantine model, where a subset of system participants may behave arbitrarily (including in a malicious manner), while the rest of the participants remain correct. Also, reliable communication primitives are a core building block of any distributed system.

In this paper, we consider the *reliable broadcast* problem in presence of Byzantine failures i.e., the problem of distributing informations from a *source* to every other process considering that a subset of nodes may act arbitrarily. The reliable broadcast primitive is expected to provide two guarantees: *(i) safety* i.e., every message m delivered by a correct node has been previously sent by the source and *(ii) liveness* i.e., every message m sent by the source is eventually delivered by every correct node.

Related Works. A solution to the reliable broadcast problem with Byzantine failures has been initially provided for complete networks [8], assuming that no more than one third of the nodes are Byzantine (i.e., $n > 3f$, where n denotes the size of the network ad f the maximum number of Byzantine nodes). Afterwards, necessary and sufficient conditions have been identified for general static networks [3], demonstrating that reliable broadcast can be solved if and only if the network is $2f + 1$-connected, where f the maximum number of Byzantine nodes.

Subsequently, research efforts followed three paths: *(i)* replacing global conditions with local conditions, *(ii)* employing cryptographic primitives, or *(iii)* considering weaker broadcast specifications.

The Certified Propagation Algorithm (CPA) [7, 16] is a protocol that solves reliable broadcast in static networks where the number of Byzantine nodes is *locally bounded*, *i.e.,* in any given neighborhood, at most f processes can be Byzantine. The original algorithm has been later extended in [15] along several directions: (i) considering different thresholds for each neighborhood, (ii) considering additional knowledge about the network topology, and (iii) considering the general adversary model.

Byzantine tolerant Reliable broadcast can also be solved by employing cryptography (e.g., digital signatures) [1,4] that enable all nodes to exchange messages guaranteeing authentication and integrity. The main advantage of cryptographic protocols is that they allow to solve problem with simpler solutions and weaker conditions (in terms of connectivity requirements). However, on the negative side, the safety of the protocols is bounded to the crypto-system.

Last, the broadcast problem has been considered weakening safety and/or liveness property e.g., allowing to a (small) part of correct processes to either deliver fake messages, or to never deliver a valid message [9–11].

Let us note that a common assumptions to Byzantine tolerant reliable broadcast protocols is to use authenticated point-to-point channels, which prevent a process from impersonating several ones (Sybil attack). The real difference between cryptographic and non-cryptographic protocols for reliable broadcast is how the cryptography is employed: non-cryptographic

protocols, in fact, may use digital signatures just within neighbours for authentication purposes, whereas the cryptographic protocols employs cryptographic primitives to enable the message verification even between non-directly connected nodes. Let us remark that an authenticated channel not necessarily requires the use of cryptography [17].

Although the Byzantine tolerant reliable broadcast problem has been extensively studied considering different settings, the solution provided in [3] is the only one for general settings and it has never been revisited from a performance perspective. Indeed, this solution hints at poor scalability since it requires an exponential number of copies of the same message to be spread and verified in order to be accepted by a correct node and let think that solving reliable broadcast in the weakest system model (i.e., the one in [3]) is practically infeasible.

Contributions. We review and improve previous solutions for reliable broadcast in static multi-hop networks, where at most f nodes can be Byzantine faulty, making no further assumption with respect to the original setting [3]. More in details, we propose and implement two optimizations that preserve both safety and liveness properties of the original algorithms. By extensive simulations for variously shaped random networks, we show that our optimizations enable to keep the message complexity close to quadratic (in the size of the network). Our work thus paves the way for the practical use of Byzantine tolerant reliable broadcast solution in realistic-size networks.

2 System Model and Problem Statement

System Model. We consider a distributed system composed by a set of n processes $\Pi = \{p_1, p_2, \dots p_n\}$, each one having a unique integer identifier. Processes are arranged in a *multi-hop* communication network. The network can be seen as an undirected graph where each node represents a process $p_i \in \Pi$, and each edge represents a communication channel between two elements $p_i, p_j \in \Pi$ such that p_i and p_j can communicate. The communication network is *static*. In the following, we interchangeably use terms *process* and *node* and we will refer to *edges* and *communication channels* interchangeably.

The processes communicate through message exchanges. Every message has a *source*, which is the id of the process that has created the message, and a *sender*, that is the id of the process that is relaying the message. The source and the sender may coincide. The sender is always a neighbor in the

communication network.

The passage of time is measured according to a fictional global clock spanning over natural numbers $\mathbb{N}$. The computation evolves in *rounds*. In each round, a process sends messages, it receives messages and then executes the computations required by the specific protocol to check if the message can be delivered or not and to prepare messages that should be sent in the next round.

We assume that the computation time is negligible with respect to communication and we consider it equal to 0. Let us notice that the timing assumptions we are considering here model a *synchronous system*.

We assume an omniscient adversary able to control up to f processes of the network allowing them to behave arbitrarily (including corrupting/ dropping messages or simply crashing). We call them *Byzantine* processes. Processes that are not Byzantine faulty are said to be *correct*. Correct processes do not a priori know the subset of Byzantine processes.

We assume *reliable* and *authenticated* communication channels, *i.e.* the exchanged messages are never altered or created by the channels and they are always delivered. Furthermore, the channels guarantee no forgery of the sender.

Processes have *no global knowledge about the system* (*i.e.* the size or the topology of the network) with the exception of the value of f.

Problem Statement. We consider the problem of *reliable broadcast* on a static distributed system from a *correct* [1] source s, assuming f Byzantine failures arbitrary spread in the network.

A protocol solves the Byzantine tolerant reliable broadcast problem if the following conditions are met:

- *(Safety)* if a correct process delivers a message m then it has been previously sent by the correct source;
- *(Liveness)* if a correct source broadcast a message m, then m will be eventually delivered by every correct process.

[1] The different assumption of a possibly faulty source leads to a more general problem, the *Byzantine Agreement* [3].

3 Background

In this section, we provide a review of the available solutions to the reliable broadcast problem highlighting the following metrics:

- *message complexity i.e.*, the total number of messages exchanged by the protocol in a single execution.
- *delivery complexity* the complexity of the procedure executed by a process to decide whether or not a message has to be accepted.

Dolev in [3] identified the necessary and sufficient condition enabling reliable broadcast on general static networks.

Remark 1 (Condition for Reliable Broadcast)**.** *The reliable broadcast can be achieved in a static network G composed of n processes considering at most f Byzantine processes if and only if the vertex connectivity of G is at least $2f + 1$.*

All of the available solutions that we are going to review rely on the Menger Theorems [2], remarked in the following:

Remark 2 (Global Menger Theorem)**.** *A graph is k-connected if and only if it contains k independent paths [2] between any two vertices.*

This, in turns, requires to remark the difference between Disjoint Paths and Vertex Cut:

Remark 3 (Vertex Cut VS Disjoint Paths)**.** *Let $G = (V, E)$ be a graph and $a, b \subseteq V$. Then the minimum number of vertices separating a from b in G is equal to the maximum number of disjoint $a - b$ paths in G.*

The following protocols are defined by a *propagation algorithm*, which rules how the message are spread over the network, and a *verification algorithm*, that decides if a message m_s can be accepted by a process guaranteeing the safety of reliable broadcast.

[2] Two paths are independent (or disjoint) if they do not have any internal vertex in common.

3.1 Dolev's Reliable Broadcast Protocol

This protocol assume that no global knowledge is given to the processes, with the exception of the value of f. The messages exchanged by the protocol have the format $msg := \langle m_s, path \rangle$, where m_s encapsulates the content and the source id s and $path$ represents a sequence of nodes.

Dolev Propagation Algorithm

- The source process s sends the message m_s to all of its neighbors, namely it multicasts $msg := \langle m_s, \emptyset \rangle$;
- a correct process p saves and relays a message $msg_i := \langle m_s, path_i \rangle$ sent by a neighbor q to all of other neighbors not included in $path_i$ appending to $path_i$ the id of the sender q, namely it multicasts $msg := \langle m_s, path_i \cup \{q\} \rangle$.

  The messages carrying not valid $path_i$, $path_i$ with loops or $path_i$ including p are discarded.

Dolev Verification Algorithm

- If a node receives copies of msg carrying the same m_s where it is possible to identify $f+1$ disjoint paths among the relative $path_i$, then m_s is delivered by the process.

The message complexity of the algorithm is exponential in the size of the network. This results in an exponential number of $path_i$ to elaborate in order to deliver a single message. Furthermore, to the best of our knowledge, the best method available to identify $f + 1$ disjoint $path_i$ is the reduction to a NP-Complete problem, *Set Packing*. We refer to this method as *DP* (disjoint paths).

The safety is guaranteed by the fact that Byzantine processes $b_1, b_2, \ldots b_f$ cannot propagate a fake message $\tilde{m}_s$ through no more than f disjoint paths. The liveness is guaranteed by the Menger theorem. Indeed assuming a vertex cut of size f, $f+1$ disjoint paths are still available between any pairs of nodes.

3.2 Maurer et al.

Maurer *et al.* [12] addressed the reliable communication problem on dynamic networks (*i.e.* the network where the topology changes over the time) where at most f processes are Byzantine and no global knowledge is given to the

processes, defining necessary and sufficient conditions to solve the problem, and providing a solution. Considering that a static network can be seen as a particular dynamic network where the topology never changes, the solution they proposed can also be employed on static networks.

They started by the fact that the Menger theorem reported in Remark 3 is not valid on dynamic networks. In particular, the Maximum number of Disjoint Paths *(MDP)* among those ones interconnecting two endpoints is less than or equal to the Minimum Vertex Cut *(MVC)* over the same paths. Thus, they defined the necessary and sufficient condition and a solution to the problem based on MVC.

The Maurer *et al.* protocol shares the basic idea behind Dolev algorithm for unknown topology: leverage the authenticated channels to collect the ids of the processes traversed by the messages.

Due to the fact that the order of traversing a set of nodes does not impact neither MVC or MDP, the message format has been changed from $msg := \langle m_s, path \rangle$ to $msg := \langle m_s, pathset \rangle$, namely carrying just the information about the ids of traversed nodes and discarding their order. Assuming that every node checks for duplicates and avoids to retransmit several copy of the same $msg_i := \langle m_s, pathset_i \rangle$, this message format reduces the message complexity and its effectiveness will be shown through simulation in a following section. As a matter of fact, this modification retains exponential message complexity.

Maurer et al. Propagation Algorithm

- The source process s sends the message m_s to all of its actual neighbors, namely it multicasts $msg := \langle m_s, \emptyset \rangle$.
  This action is iterated every time the network topology changes;

- a correct process p saves and relays a message $msg_i := \langle m_s, pathset_i \rangle$ sent by a neighbor q to all other of its actual neighbors not included in $pathset_i$ appending to $pathset_i$ the id of the sender q, namely it multicasts $msg := \langle m_s, pathset_i \cup \{q\} \rangle$.
  A correct process multicasts any saved message msg_i every time the network topology changes [3] . The messages carrying not valid $pathset_i$ or $pathset_i$ where p is included are discarded.

[3] this is due to the fact that a process does not known if a transmitted message is delivered or not by the channel.

Maurer et al. Verification Algorithm

- If a node receives copies of msg carrying the same m_s where it is not possible to identify a vertex cut of size less than or equal to f among all the relative *pathset*, then m_s is delivered by the process.

The message complexity of the algorithm is exponential in the size of the network (even discarding multiple retransmissions). This results in an exponential number of $pathset_i$ to elaborate in order to deliver a single message. Furthermore, to the best of our knowledge, the best method available to identify a vertex cut of size less than or equal to f is the reduction to a NP-Complete problem, *Hitting Set*. We refer to this method as *VC* (Vertex Cut).

The safety is guaranteed by the fact that Byzantine processes $b_1, b_2, \dots b_f$ cannot propagate a fake message $\tilde{m_s}$ through paths with a vertex cut greater than f.

4 Analysis and Contributions

To the best of our knowledge, there does not exists further solutions to the problem we are targeting that do not make extra or different assumptions (*e.g.* digital signatures, higher density networks, weaker versions of safety or liveness, etc.). We saw in Sections 3.1 and 3.2 that available solutions may not scale to larger networks, making them not practically employable. In this section, we further analyze some details of aforementioned solutions and propose simple optimizations that result in drastically reducing the message complexity.

4.1 Paths VS Pathsets, MDP VS MVC

The best method currently available to identify $f + 1$ disjoint paths generated in a Byzantine affected distributed system is the reduction to *Set Packing*, a NP-Complete problem *Set Packing* [6]. The best method currently available to check whether no vertex cut of size f exists over a set of path generated in a Byzantine affected distributed system is the reduction to *Hitting Set*, a NP-Complete problem [6].

A priori, there is no reason to prefer path over pathset as message format, unless another way to identify disjoint paths or vertex cut that preserves safety is identified. Indeed: *(i)* due to the reduction to a set problem, paths are converted to sets to be analyzed; *(ii)* two paths over the same set of nodes

are not disjoint and have a cut of size 1, and *(iii)* the pathset interconnecting two endpoints are lower size than the paths.

It follows from Menger's theorem in Remark 3 that, from a theoretical point of view, it is the same for a node to identify $f+1$ disjoint paths (DP) or to verify that no vertex cut (VC) of size lower or equal than f exists to deliver a message. Furthermore both problems are addressed by solving an NP-Complete problem.

4.2 Limiting message transmissions

From the previous discussion, we saw that a message travels unconditionally over the network collecting the ids of visited nodes before being delivered. Note that, even in the scenario where all the processes are correct, this leads to an exponential message complexity protocol. Also, every process eventually receives the visited sets of any pathset interconnecting the source with itself, resulting in an exponential input for the verification.

A process should attempt to verify the received messages continuously, because the number of messages to be verified only increases over time, until all possible pathsets are received by all processes. As a matter of fact, not all pathsets are required for the delivery.

For this reason, we propose a technique to let a process' neighbors know that a particular message m_s has been delivered, and argue that avoiding to forward a message carrying m_s to nodes that already delivered m_s does not hinder safety nor liveness of reliable broadcast.

Theorem 1. *It is safe for a correct process p that executes either Dolev or Maurer* et al. *algorithm to relay a message m_s with an empty path/pathset if m_s has already been delivered by p.*

Proof. The aim of the information about the nodes traversed by a message m_s is to enable a process p to decide whether m_s can be safely accepted. Once m_s is delivered, the information about the node traversed before reaching p is not useful, because m_s is already verified as safe by p. □

This modification has already been employed [14] for the purpose of topology reconstruction.

Theorem 2. *Let p be a process executing either Dolev or Maurer* et al. *algorithm to broadcast a message m_s. Even if p does not relay messages carrying m_s to its neighbors that already delivered m_s then liveness property is still satisfied.*

Proof. Let us assume that there exists three processes p, q, r such that only q has already delivered message m_s and that, among others, the following communication channels are available: (p, q), (q, r). From Theorem 1 we know that process q can relay m_s with an empty path/pathset. Thus, any further path/pathset containing p and q, after the delivery of q, does not affect the results of DP and VC about m_s computed on r. Thus, any further transmission from p to q can be avoided after the delivery of m_s by q without compromising liveness. □

4.3 Practical Reliable Broadcast Algorithm

Given the previous observations we revised the verification and propagation algorithms taking into account that: *(i)* if a process p receives a message $msg := \langle m_s, \emptyset \rangle$ from a neighbor q, it can conclude that q has delivered m_s, and *(ii)* if a neighbors q has delivered m_s, no further msg need to be forwarded to q.

The protocol we propose:

- employs *pathset* as message format;
- employs VC as verification algorithm;
- does not forward copies of a message to nodes that have already delivered it.

The pseudo code of our protocol is presented in Figure 1. Initially, every node is not aware about its neighborhood but it can easily retrieve it due to the authenticated channels. Every node keeps for every not delivered message m_s the list of neighbors that have already delivered such a message.

When a new msg is received from a process q, if the contained m_s is not yet delivered, it is enqueued for forwarding. If the contained *pathset* is empty, the process concludes that the neighbors q has delivered the message.

When there is a message msg to forward, it is relayed to all the neighbors that have not yet accepted the contained m_s.

When a node delivers a message m_s, it discards the relative enqueued msg for forwarding and it enqueues the message m_s with an empty *pathset*.

The forwarding policy picks randomly one message per time among the ones to relay. It is shown through simulations that this approach slightly reduce the message complexity with respect to a FIFO policy.

Notice that all the information about the visited sets of a message m_s and the neighbors that have delivered it can be dropped after the delivery.

```
Neigh = [p1, p2,...pj]
Neigh_del[ms] = []
Delivered = []
To_Forward[ms] = []
Pathset[ms] = []

When RECEIVED msg FROM q:
     If msg.ms not in Delivered:
          To_Forward[ms].append( < msg.ms, msg.pathset + {q} > )
          Pathset[ms].append( msg.pathset )
          If msg.pathset == {}
               Neigth_del[ms] = q
When |To_Forward[ms]| > 0 :
     msg = To_Forward[ms].random pop()
     ForAll neight not in Neight - Neigh_del[ms]:
          If neigth not in msg.pathset:
               SEND(neight, msg)

When |Pathset[ms]| > 0:
     If {s} in Pathset[ms] or not VC(Pathset[ms], f):
          Delivered.append( ms )
          To_Forward[ms].clear()
          To_Forward[ms].append( < msg.ms, {} > )
```

Figure 1: Practical Byzantine Tolerant Reliable Broadcast (pseudo-code)

4.4 Preventing Flooding

The verification complexity depends on the amount of pathsets that has been received by a process, and we highlighted that such a number is exponential in the size of the network even without considering the Byzantine processes. Indeed, a Byzantine process can potentially flood the network with fake messages (*i.e.*, $\tilde{msg} := \langle \tilde{m_s}, \tilde{pathset} \rangle$ where $\tilde{m_s}$ and/or $\tilde{pathset}$ is invented by the Byzantine) that are spread also by the correct processes (because they cannot distinguish between a message generated by the propagation algorithm from one made by a Byzantine). Thus a countermeasure must be adopted. Dolev's solution further assumes that every node knowns the id of the member of the system to prevent a Byzantine process to send $\tilde{msg}$ containing $\tilde{pathset}$ with fake ids. Nevertheless a Byzantine process can still diffuse a considerable amount of fake $\tilde{pathset}$ with valid ids.

A way to limit the flooding capability of Byzantine processes is to constraint the channel capacity of every process. Thus, we assume that every process can only multicast (*i.e.* send to all of its neighbors) one message per round.

5 Simulations

In this section, we present the simulation results we obtained in order to evaluate the effectiveness of the optimizations described by the previous section.

For the purpose of evaluation, we assume a synchronous setting that evolves in rounds identified by natural numbers (for every round r, $r \in \mathbb{N}$). In each round, a process sends messages, receives messages and then elaborates the next messages to be sent. The channel latency is assumed equal to 1 time unit with any load. We assume that the local computation time is negligible and equal to 0. We simulate a single broadcast that start at time 0. Every process attempts to deliver the message in every round until it succeeds to do so. We employed pathset as message format, and we used VC as delivery policy. Furthermore, we made use of implementation provided by Gainer-Dewar and Vera-Licona [5] for the algorithm defined by Murakami and Uno [13] to solve the reduction to the hitting set problem.

We considered two kinds of networks:

1. k-regular k-connected random graphs;
2. Erdős - Rényi random graphs, $G(n, p)$;

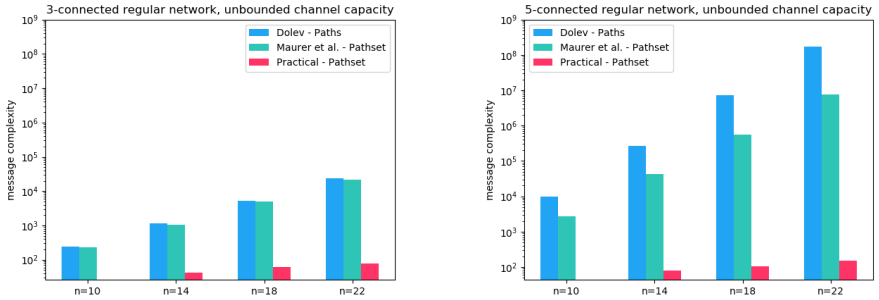

Figure 2: Original algorithms VS our algorithm, message complexity, unbounded channel capacity

The former one models managed systems where the communication links are minimized, the latter one is representative of self-organized environment (e.g P2P).

We also consider two settings for the channel capacity: *(i) unbounded*, namely a process can send an indefinite number of messages per round and they are all delivered within 1 time instant; *(ii) bounded*, in which a single message per round can be sent on any communication channel.

We refer with k to the network node connectivity. For all the simulations we consider the maximum number of f Byzantine tolerable, i.e. $f = \lfloor (k-1)/2 \rfloor$ and we assume all processes correct. This setting models the worst case scenario, indeed:

- the message complexity decrease assuming a lower value of f;
- in case the channel capacity is bounded, a Byzantine process can spread the same amount of messages as a correct one;
- assuming Byzantine processes that simply block the message retransmission reduces the message complexity, because the message are exchanged between a lower number of nodes.

For all results, we present 95% confidence intervals. The simulation code is available at `https://www-npa.lip6.fr/~farina/rbcode`

5.1 A Simple Setup

We start by comparing the message complexity while using: *(i)* paths as message format, *(ii)* pathset as message format, and *(iii)* our protocol. The results are presented in Figure 2. We consider k-regulars graph for small values

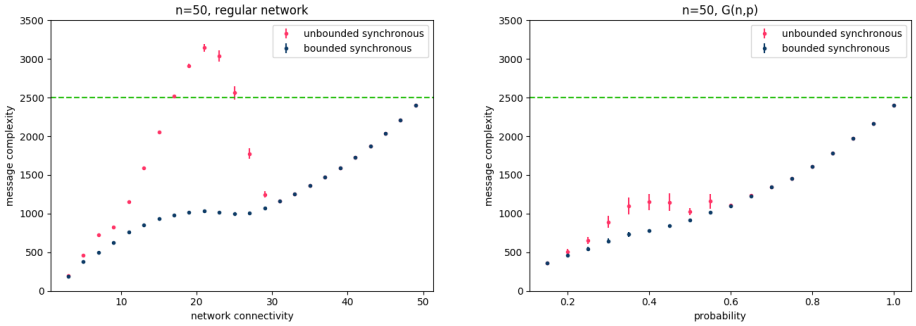

Figure 3: Message complexity, bounded *vs.* unbounded channel capacity

of k (respectively, $k = 3$ and $k = 5$) and n (respectively, $n \in [10, 14, 18, 22]$). For simulation purposes, all processes are supposed correct, and the parameter f of the protocol is assumed to be maximal, *i.e.* $f = \lfloor (k-1)/2 \rfloor$. We also assume unbounded capacity channels. We can see, as expected, that even in regular graphs the message complexity explodes even with small graph instances with small values of k. Nevertheless, the message complexity of our protocol is at least one order of magnitude smaller that previous approaches in this simple setting.

5.2 Bounded *vs.* Unbounded Capacity Channels

Figure 3 plots the message complexity of our protocol for a network of $n = 50$ nodes. Both unbounded and bounded channel capacity cases are considered. A regular network (on the left) and a $G(n, p)$ network (on the right) are considered. The constant function $y = n^2$, for $n = 50$ is also plotted (in dashed green). We can see that our protocol exhibits a message complexity below n^2 in a synchronous system with channels of bounded capacity. Furthermore, when channels have unbounded capacity, the message complexity is drastically reduced with respect to the state of the art protocols. Remind that the over message complexity we consider is the sum of all messages exchanged in the system thoughout execution, so any given process only verifies a small portion of those.

Figure 4 plots the message complexity of our protocol considering networks of various sizes, respectively 150, 200 and 250 nodes. The channel capacity is assumed bounded. It results that the message complexity of our protocol continues to follow a defined behavior, always upper bounded by n^2 (in green dashed line for both cases).

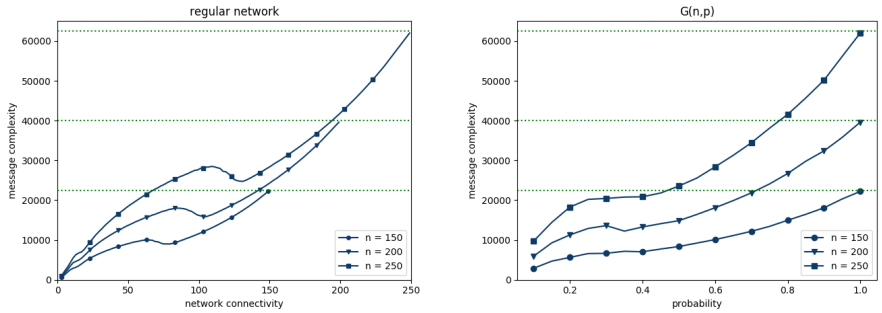

Figure 4: Message complexity, bounded channel capacity, network size $n = 150$, 200 and 250

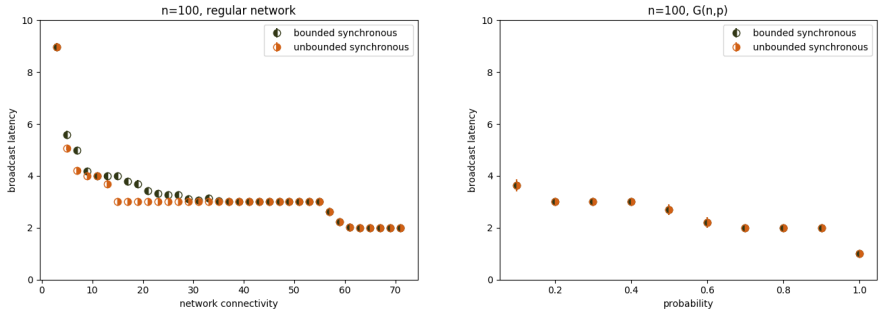

Figure 5: Broadcast latency

We mean by *broadcast latency* the length of the time interval $[r_s, r_e]$ where r_s is the round at which the broadcast starts and r_e is the first round at which every correct node delivered the message. Arguably, the unbounded channel capacity assumptions allows to minimize broadcast latency. Then, Figure 5 compares broadcast latency for unbounded and bounded channels. It turns out that a bounded channel capacity impacts broadcast latency in a negligible way, validating its use as an effective countermeasure against Byzantine processes.

5.3 Synchronous *vs.* Asynchronous Executions

One of the reasons that allows our protocol to perform efficiently is the synchrony assumption. Indeed, synchrony permits to avoid sending unnecessary pathsets. In order to evaluate the impact of asynchrony, we simulated our

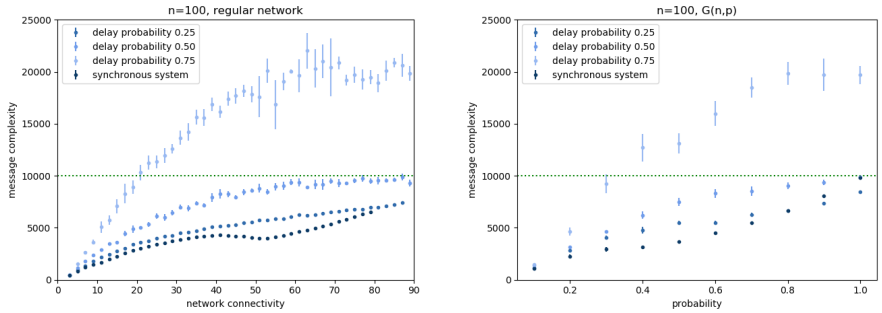

Figure 6: Message Complexity, Synchronous *vs.* random delays

protocol again introducing random delays. The computation still evolves in rounds but the channels no longer ensure that transmitted messages are always delivered within 1 time instant. In particular, whenever a message is sent, it is delivered by the channel during the same round with a fixed probability p. In case the message is *not* delivered during the same round, a new attempt for delivery is done in the next round. It follows that with a delay probability p of $\frac{1}{2}$, the probability of a timely transmission is $\frac{1}{2}$, the probability of a 1 time instant delay is $\frac{1}{4}$, and so on. The results obtained in this setting are presented in Figure 6, for a k-regular network and a $G(n,p)$ network of size 150. Notice that in our setting, the success of a transmission attempt is independent of the channel and of the time instant the message is sent. Our results show that considering random independent delays of the communication channels, the message complexity grows remaining constrained within a constant factor to the original quadratic bound.

5.4 FIFO *vs.* Random Selection when Transmitting

We advocated in the end of Section 4.3 that a random selection among messages to transmit is preferable with respect a FIFO policy. The comparison between the two choices is presented in Figure 7 for a k-regular network and a $G(n,p)$ of size 150. The gain is not huge, but is still visible.

6 Conclusions

We revisited available solutions for the reliable broadcast in general static network hit by up to f arbitrarily distributed Byzantine failures, and proposed two optimizations following performance related observations. Al-

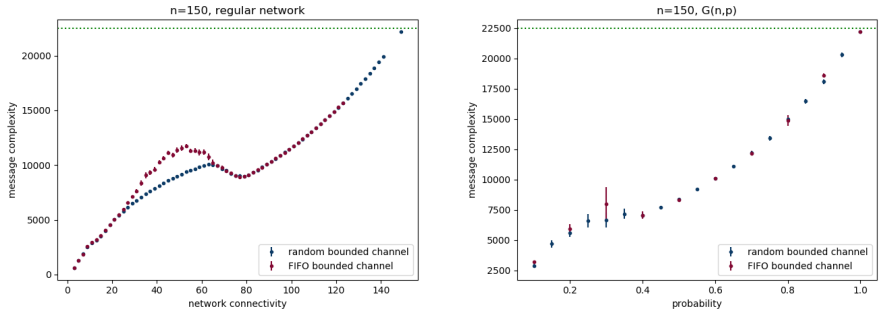

Figure 7: Message complexity, Synchronous FIFO *vs.* Synchronous Random

though the theoretical delivery complexity of our protocol remains unchanged with respect to previous solutions (that is, an exponential number of message exchanges is still required for some graph topologies and Byzantine placement), our experiments show that the message complexity obtained in simulation on two families of random networks with sufficient connectivity is considerably reduced (from exponential to polynomial in the size of the network), practically enabling reliable broadcast in larger systems and networks with authenticated channels. Our results open to the possibility of identifying a polynomial theoretical bound on message complexity solving the reliable broadcast problem for a large class of random topology networks. Another target to address consists in proving (or disproving) the equivalence between verifying a message in the system model we addressed and the related NP-Complete problem. Furthermore, the same problem should be analyzed also on dynamic networks. Even if the protocol we proposed can be directly employed, the achieved gain in message complexity is not guaranteed due to the weaker synchrony assumptions.

Acknowledgments

This work has been partially supported by the INOCS Sapienza Ateneo 2017 Project (protocol number RM11715C816CE4CB).

Giovanni Farina thanks the *Université Franco-Italienne/Universitá Italo-Francese* (UFI/UIF) for supporting his mobility through the Vinci grant 2018.

References

[1] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[2] Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2017.

[3] Danny Dolev. Unanimity in an unknown and unreliable environment. In *Foundations of Computer Science, 1981. SFCS'81. 22nd Annual Symposium on*, pages 159–168. IEEE, 1981.

[4] Vadim Drabkin, Roy Friedman, and Marc Segal. Efficient byzantine broadcast in wireless ad-hoc networks. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 160–169. IEEE, 2005.

[5] Andrew Gainer-Dewar and Paola Vera-Licona. The minimal hitting set generation problem: algorithms and computation. *SIAM Journal on Discrete Mathematics*, 31(1):63–100, 2017.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[7] Chiu-Yuen Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 275–282. ACM, 2004.

[8] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[9] Alexandre Maurer and Sébastien Tixeuil. Byzantine broadcast with fixed disjoint paths. *Journal of Parallel and Distributed Computing*, 74(11):3153–3160, 2014.

[10] Alexandre Maurer and Sébastien Tixeuil. Containing byzantine failures with control zones. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):362–370, 2015.

[11] Alexandre Maurer and Sebastien Tixeuil. Tolerating random byzantine failures in an unbounded network. *Parallel Processing Letters*, 26(01):1650003, 2016.

[12] Alexandre Maurer, Sébastien Tixeuil, and Xavier Defago. Communicating reliably in multihop dynamic networks despite byzantine failures. In *Reliable Distributed Systems (SRDS), 2015 IEEE 34th Symposium on*, pages 238–245. IEEE, 2015.

[13] Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, 170:83–94, 2014.

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

[15] Aris Pagourtzis, Giorgos Panagiotakos, and Dimitris Sakavalas. Reliable broadcast with respect to topology knowledge. *Distributed Computing*, 30(2):87–102, 2017.

[16] Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, 2005.

[17] Kai Zeng, Kannan Govindan, and Prasant Mohapatra. Non-cryptographic authentication and identification in wireless networks [security and privacy in emerging wireless networks]. *IEEE Wireless Communications*, 17(5), 2010.