Practical Byzantine Reliable Broadcast on Partially-Connected Networks

Bonomi, Silvia; Decouchant, Jérémie; Farina, Giovanni; Rahli, Vincent; Tixeuil, Sébastien

DOI
10.1109/ICDCS51616.2021.00055

Publication date
2021

Document Version
Accepted author manuscript

Published in
Proceedings - 2021 IEEE 41st International Conference on Distributed Computing Systems, ICDCS 2021

Practical Byzantine Reliable Broadcast on Partially Connected Networks

Silvia Bonomi*, Jérémie Decouchant†, Giovanni Farina*, Vincent Rahli‡, Sébastien Tixeuil§
*Sapienza Università di Roma, †Delft University of Technology,
‡University of Birmingham, §Sorbonne University, CNRS, LIP6
bonomi@diag.uniroma1.it, j.decouchant@tudelft.nl, gfarina@diag.uniroma1.it,
vincent.rahli@gmail.com, sebastien.tixeuil@lip6.fr

***Abstract*—In this paper, we consider the Byzantine reliable broadcast problem on authenticated and partially connected networks. The state-of-the-art method to solve this problem consists in combining two algorithms from the literature. Handling asynchrony and faulty senders is typically done thanks to Gabriel Bracha's authenticated double-echo broadcast protocol, which assumes an asynchronous fully connected network. Danny Dolev's algorithm can then be used to provide reliable communications between processes in the global fault model, where up to f processes among N can be faulty in a communication network that is at least 2f+1-connected. Following recent works that showed that Dolev's protocol can be made more practical thanks to several optimizations, we show that the state-of-the-art methods to solve our problem can be optimized thanks to layer-specific and cross-layer optimizations. Our simulations with the Omnet++ network simulator show that these optimizations can be efficiently combined to decrease the total amount of information transmitted or the protocol's latency (e.g., respectively, -25% and -50% with a 16B payload, N=31 and f=4) compared to the state-of-the-art combination of Bracha's and Dolev's protocols.**

***Index Terms*—Byzantine reliable broadcast, partially connected networks, synchronous or asynchronous communications**

I. Introduction

At their very core, distributed systems consist of autonomous computing entities (or processes) that *communicate* to globally solve non-trivial tasks. Over the years, distributed systems became ubiquitous and increased the possibility that some of the processes fail in some unpredictable manner. The most general fault model is the Byzantine model, that allows a process to simply exhibit arbitrary behavior. Byzantine processes are opposed to correct processes, that trustfully follow their prescribed algorithm. In this context, communicating reliably may become difficult, especially when processes have to rely on other (possibly Byzantine) processes to convey the information they want to send (that is, the network is partially connected).

Two useful global communication abstractions have been defined in this context. First, the *reliable communication* (RC) abstraction requires that: *(i)* when a correct process broadcasts a message, this message is then delivered by all correct processes, and *(ii)* when a message originating from a correct process is delivered, it was indeed sent by this correct process. Reliable communication is also referred to as *reliable broadcast with honest dealer*, outlining that the source of a message is always correct. Second, the *reliable broadcast* abstraction considers the additional case where the sender of a message may be arbitrarily faulty. In this case, all correct processes are supposed to deliver the same message. Reliable broadcast, often abbreviated as BRB (Byzantine Reliable Broadcast), guarantees stronger properties than reliable communication, yet both abstractions can be implemented in a fully asynchronous network. Recently, protocols that rely on Byzantine reliable broadcast have been used to implement Blockchain consensus [1] and payments [2].

The most well known asynchronous BRB protocol is probably Gabriel Bracha's algorithm, which is sometimes named double echo authenticated broadcast [3]. This protocol assumes a fully connected network of N processes (among which at most $f<N/3$ are Byzantine) and authenticated network links.

Questioning whether it is possible to remove the full connectivity assumption in an asynchronous setting leads to Danny Dolev's RC algorithm [4], as long as the network is "sufficiently well connected" (a necessary and sufficient condition for RC in arbitrary networks is that the connectivity of the network is at least $2f + 1$, where f is the number of Byzantine processes). However, Dolev's algorithm does not solve BRB, as it does not prevent a Byzantine faulty process from causing correct processes to *not all* deliver the same message. Nevertheless, Bracha's and Dolev's protocols can be combined to solve BRB in sufficiently well-connected synchronous or asynchronous network topologies [5].

In this paper, we start from this state-of-the-art solution to the BRB problem in partially connected networks, and make the following contributions (code available online[1]):
(i) We describe how state-of-the-art optimizations of Dolev's RC algorithm [6] can be extended to the BRB combination of Bracha's and Dolev's algorithms.
(ii) We present a total of 12 novel modifications that can be applied to the BRB combination of Bracha's and Dolev's algorithms, some of them being cross-layer.
(iii) We evaluate the impact of each modification on the protocol latency and throughput using Omnet++ simulations in a large variety of settings.
(iv) We detail how these modifications should be combined depending on the network asynchrony and connectivity, and on the payload size to optimize latency and/or throughput.

[1] https://github.com/jdecouchant/BRB-on-partially-connected-networks

The remainder of this paper is organized as follows. Sec. II discusses the related work. Sec. III describes the system model and the BRB problem. Sec. IV provides some background on BRB algorithms on asynchronous partially connected networks. Sec. V explains how we modify the interface of the Bracha-Dolev protocol to include state-of-the-art improvements of Dolev's protocol, and to add functionalities. Sec. VI and Sec. VII respectively present our latency and throughput modifications of the Bracha-Dolev protocol, which we call MBD.1 to MBD.12. Sec. VIII details our performance evaluation. Finally, Sec. IX concludes the paper.

II. Related Work

The first protocol to consider the Byzantine Reliable Broadcast (BRB) problem assumed authenticated links[2].

Bracha and Toueg formalized the Reliable Broadcast problem [8, 9] and then Bracha described the first BRB protocol for asynchronous and fully connected reliable networks (i.e., networks where each process is able to communicate with any other in the system and where messages cannot be lost) and the proposed solution is able to tolerate f Byzantine nodes (where $f < N/3$ and N is the number of processes in the system) [3]. This protocol is characterized by three different all-to-all communication phases (namely, *send*, *echo* and *ready*) and processes progress in the algorithm as soon as they have heard from a quorum of nodes in a given phase.

This protocol assumes a fully connected communication network and thus its applicability in general networks (like the ones considered in this paper) is not directly possible but requires some adaptation.

Concerning generic networks where full connectivity cannot be assumed, Dolev [4] showed that correct processes can reliably communicate in presence of f Byzantine nodes if, and only if, the network graph is $(2f+1)$-connected.

In particular, Dolev's algorithm allows a process p_i to deliver a message when it receives it through at least $f+1$ disjoint paths on which processes behaved correctly (which is made possible when it flows through at least $2f+1$ disjoint paths). In order to do that, it requires to solve a maximum disjoint paths problem. Also in this case, the solution assumes authenticated and reliable point-to-point communication links.

Despite its theoretical correctness, Dolev's solution is not practical in large networks due to its worse-case complexity both in terms of messages and computational complexity.

Bonomi et al. [6] proposed several optimizations that improve the performance of Dolev's algorithm on unknown topologies. Indeed, a few modifications allow to save messages, making the algorithm more appealing from a practical point of view even if its worst-case complexity still remains high. Maurer et al. [10] considered the reliable communication problem in settings where the topology can vary with time. In Maurer et al.'s protocol, a process needs to solve the minimum vertex cut problem instead of the not equivalent (in dynamic networks) maximum disjoint paths problem.

Instead of assuming the global fault model, Koo presented a broadcast algorithm under the t-locally bounded fault model [11], which was later coined CPA (Certified Propagation Algorithm) [12]. Tseng et al. provided a necessary and sufficient condition for CPA to work correctly on a given topology [13]. Several authors have defined weaker reliable communication primitives for improved scalability [14, 15].

All the aforementioned works [4, 6, 10–13] solve a weaker problem than BRB, indeed they guarantee that all correct processes eventually deliver messages diffused by a correct source but no agreement in case of a faulty one.

More recently, BRB on partially connected networks has been achieved by combining Bracha's and Dolev's algorithms[3]. For example, Wang and Wattenhoffer used this method to design a randomized Byzantine agreement protocol [5]. We show in this work that the two protocols can be combined in a more efficient way. We evaluate the impact of our modifications to the combination of Bracha's algorithm with Bonomi et al. [6]'s improved version of Dolev's algorithm. Our new modifications apply to Bracha's and Dolev's protocols, and the cross-layer combination of the two protocols.

Other approaches have assumed authenticated processes (i.e., processes can use digital signatures) instead of authenticated communication channels [16]. Relying on cryptography provides integrity and authenticity properties that simplify the algorithms. In particular, weaker connectivity is required to solve the BRB problem. However, cryptography has a computational cost and requires a trusted public key infrastructure (PKI). From a theoretical point of view, cryptography-based approaches are limited to the case of computationally bounded adversaries (e.g., that cannot generate hash collisions). Also, having a trusted agent (TA) that is never Byzantine trivializes the BRB problem (the sender can simply send its message to the TA, and every correct node can then reliably collect it from there). By contrast, we aim at solutions that can cope with computationally unbounded adversaries.

Recently, Contagion [17] replaced quorums by smaller stochastic samples, and described an abstraction whose properties can be violated with a probability that depends on the size of the samples. Differently, RT-ByzCast [18] and PISTIS [19] aimed at providing real-time guarantees in probabilistically synchronous and reliable networks. The idea of using pseudo-randomized message dissemination has been used to tolerate malicious or selfish behaviors in several works [20, 21], which however only provide probabilistic guarantees, while we target exact deterministic guarantees.

III. System Model and Problem Statement

Processes. We assume a set $\Pi = \{p_1, p_2, \ldots, p_N\}$ of N processes, uniquely identified by an ID, which are interconnected. We assume that up to $f < \lfloor N/3 \rfloor$ of the N processes are Byzantine, i.e., that they can behave arbitrarily or maliciously. Processes know about the number N of processes, the IDs of the processes in the system, and the fault threshold f.

[2]Let us recall that authenticated links guarantee that the identity of the sender cannot be forged and can be implemented without cryptography [7].

[3]One can also combine Bracha and CPA, but the different local Byzantine conditions yields a stronger requirement to be satisfied.

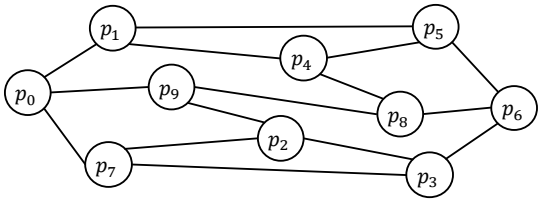

Fig. 1. A reliable communication graph with $N = 10$ and $f = 1$.

Communications. The processes are interconnected by a communication network, which can be represented by an undirected graph $G = (V, E)$ where each node represents a process $p_i \in \Pi$ (i.e., $V = \Pi$), and each edge represents a communication channel. Two processes can directly communicate with each other if and only if they are connected by an edge. Otherwise, processes have to rely on others to relay their messages and communicate. Intermediary nodes might be Byzantine, i.e., drop, modify or inject messages. We assume that communication channels are authenticated, i.e., that messages received by a node p_i on the link interconnecting it with a node p_j always come from p_j. Communication channels can be synchronous or asynchronous, but they are reliable, i.e., a message cannot be lost. We assume that the communication network is at least $2f+1$ connected. Fig. 1 illustrates a 3-vertex-connected communication network. We assume that the network topology is unknown to processes.

We assume that processes broadcast payload data, which can be of variable size, and that they might have to broadcast the same payload data at different times during the system's life. To do so, processes use a header that uniquely identifies the payload data and contains control information.

Byzantine Reliable Broadcast. We consider the following Byzantine Reliable Broadcast (BRB) abstraction. Its interface is as follows: nodes initiate broadcasts using `Broadcast` events, while a `Deliver` event indicates that a message has been delivered by a node. Moreover, this abstraction guarantees the following properties:

[BRB-Validity] If a correct process p broadcasts m, then some correct process eventually delivers m.

[BRB-No duplication] No correct process delivers message m more than once.

[BRB-Integrity] If a correct process delivers a message m with correct sender p_i, then m was previously broadcast by p_i.

[BRB-Agreement] If some correct process delivers m, then every correct process eventually delivers m.

Reliable Communication. In addition to the BRB abstraction, we also consider the weaker Reliable Communication (RC) abstraction, which has the same interface as BRB, and guarantees the same properties, except BRB-Agreement, which is only guaranteed when the sender is correct.

IV. Background: BRB on Asynchronous and Partially Connected Networks

In this section we first recall Bracha's and Dolev's algorithms and we then present how they can be used to provide BRB in asynchronous partially connected networks.

A. Bracha's Algorithm for BRB in Asynchronous and Fully Connected Networks

Bracha's algorithm (sometimes called *authenticated double-echo broadcast*) describes the first BRB protocol for asynchronous and authenticated communication networks [3]. Let us recall that this protocol assumes a fully connected network made of reliable and authenticated point-to-point links and it tolerates up to f Byzantine processes (where $f < N/3$ and N is the number of processes in the network).

The protocol works in three phases. When a process p_i wants to BRB-broadcast some payload data, it sends a `SEND` message along with the payload data to all nodes in the system (phase 1). Upon receiving a `SEND` message, a process sends an `ECHO` message to all the nodes in the system (along with the payload data) and moves to phase 2 where it remains waiting for a quorum of `ECHO` messages. Upon receiving $\left\lceil \frac{N+f+1}{2} \right\rceil$ `ECHO` messages for a given payload, a process sends a `READY` message to other nodes and moves to phase 3. Let us note that a `READY` message can also be sent upon receiving $f+1$ `READY` messages as it will ensure that al least one correct process is moving to phase 3 and thus it is safe to broadcast a `READY` message and to move to phase 3 as well. Finally, upon receiving $2f+1$ `READY` messages, a process can BRB-deliver the payload data by concluding the procedure.

For the sake of completeness, we report in Algo. 1 the pseudo-code of the algorithm.[4]

B. Dolev's Algorithm for Reliable Communication in $(2f+1)$-Connected Networks

Dolev's protocol provides reliable communication if processes are interconnected by reliable and authenticated communication channels, and if the communication network is at least $(2f+1)$-connected [4].

Dolev's protocol leverages the authenticated channels to collect the labels of processes traversed by a content. Processes use those labels to compute the maximum number of node-disjoint paths among all the paths received for a particular content. A process delivers a content as soon as it has verified its authenticity, i.e., as soon as it has received the content through at least $f+1$ node-disjoint paths. The keystone of Dolev's proof is that by Menger's theorem, if the communication network is $(2f+1)$-connected then there are at least $2f+1$ node-disjoint paths between any two processes in the network [24], and since at most f processes are faulty, according to the pigeonhole principle, a process will receive a message sent by another process through at least $f+1$ disjoint paths.

Depending on whether the processes know the network topology or not, Dolev presented two variants of its protocol, which respectively use predefined routes between processes, or flooding. We focus on the unknown topology version of Dolev's algorithm, whose pseudo-code is presented in Algo. 2.

Dolev's algorithm made practical. Let us note that the worst case complexity of Dolev's algorithm is high (both in terms of number of messages and complexity to verify

[4]We extract, and refer the reader to the pseudo-code from [22] and [23].

Algorithm 1 *BRB in asynchronous and fully connected networks (Bracha's protocol) at process p_i*

1: **Parameters:**
2: Π: the set of all processes.
3: N: total number of processes.
4: $f < N/3$: maximum number of Byzantine processes.
5: **Uses:** Auth. async. perfect point-to-point links, instance *al*.
6:
7: **upon event** $\langle Bracha, \texttt{Init} \rangle$ **do**
8: $sentEcho = sentReady = delivered =$ **False**
9: $echos = readys = \emptyset$
10:
11: **upon event** $\langle Bracha, \texttt{Broadcast} \mid m \rangle$ **do**
12: **forall** $q \in \Pi$ **do** { **trigger** $\langle al, \texttt{Send} \mid q, [\texttt{SEND}, m] \rangle$ }
13:
14: **upon event** $\langle al, \texttt{Deliver} \mid p, [\texttt{SEND}, m] \rangle$ **and not** $sentEcho$ **do**
15: $sentEcho =$ **True**
16: **forall** $q \in \Pi$ **do** { **trigger** $\langle al, \texttt{Send} \mid q, [\texttt{ECHO}, m] \rangle$ }
17:
18: **upon event** $\langle al, \texttt{Deliver} \mid p, [\texttt{ECHO}, m] \rangle$ **do**
19: $echos.\texttt{insert}(p)$
20:
21: **upon event** $echos.\texttt{size}() \geq \left\lceil \frac{N+f+1}{2} \right\rceil$ **and not** $sentReady$ **do**
22: $sentReady =$ **True**
23: **forall** $q \in \Pi$ **do** { **trigger** $\langle al, Send \mid q, [\texttt{READY}, m] \rangle$ }
24:
25: **upon event** $\langle al, \texttt{Deliver} \mid p, [\texttt{READY}, m] \rangle$ **do**
26: $readys.\texttt{insert}(p)$
27:
28: **upon event** $readys.\texttt{size}() \geq f + 1$ **and not** $sentReady$ **do**
29: $sentReady =$ **True**
30: **forall** $q \in \Pi$ **do** { **trigger** $\langle al, \texttt{Send} \mid q, [\texttt{READY}, m] \rangle$ }
31:
32: **upon event** $readys.\texttt{size}() \geq 2f + 1$ **and not** $delivered$ **do**
33: $delivered =$ **True**
34: **trigger** $\langle Bracha, \texttt{Deliver} \mid s, m \rangle$

if a specific content can be safely delivered). Bonomi et al. presented several modifications that reduce the number of messages transmitted along with their size [6] in practical executions[5]. We recall these modifications in the following.

[MD.1] If a process p receives a content directly from the source s, then p directly delivers it.

[MD.2] If a process p has delivered a content, then it can discard all the related paths and relay the content only with an empty path to all of its neighbors.

[MD.3] A process p relays path related to a content only to the neighbors that have not delivered it.

[MD.4] If a process p receives a content with an empty path from a neighbor q, then p can abstain from relaying and analyzing any further path related to the content that contains the label of q.

[MD.5] A process p stops relaying further paths related to a content after it has been delivered and the empty path has been forwarded.

C. BRB in $(2f + 1)$-connected networks

The state-of-the-art method to implement the BRB abstraction in a partially connected network consists in combining

[5]Let us stress that the proposed modifications do not improve the worst case theoretical complexity of the protocol, but the authors showed, through experimental evaluations, that in several practical settings, the performance gain is significant.

Algorithm 2 *Reliable communication in $(2f+1)$-connected networks (Dolev's protocol) at process p_i*

1: **Parameters:**
2: f : max. number of Byzantine processes in the system.
3: **Uses:** Auth. async. perfect point-to-point links, instance *al*.
4:
5: **upon event** $\langle Dolev, \texttt{Init} \rangle$ **do**
6: $delivered =$ **False**
7: $paths = \emptyset$
8:
9: **upon event** $\langle Dolev, \texttt{Broadcast} \mid m \rangle$ **do**
10: **forall** $p_j \in \texttt{neighbors}(p_i)$ **do**
11: **trigger** $\langle al, \texttt{Send} \mid p_j, [m, [\,]] \rangle$
12: $delivered =$ **True**
13: **trigger** $\langle Dolev, \texttt{Deliver} \mid m \rangle$
14:
15: **upon event** $\langle al, \texttt{Deliver} \mid p_j, [m, path] \rangle$ **do**
16: $paths.\texttt{insert}(path + [p_j])$
17: **forall** $p_k \in \texttt{neighbors}(p_i) \setminus (path \cup \{p_j\})$ **do**
18: **trigger** $\langle al, \texttt{Send} \mid p_k, [m, path + [p_j]] \rangle$
19:
20: **upon event** (p_i is connected to the source through $f + 1$ node-disjoint paths contained in $paths$) **and** $delivered = \texttt{False}$ **do**
21: **trigger** $\langle Dolev, \texttt{Deliver} \mid m \rangle$
22: $delivered =$ **True**

Bracha's algorithm with a second algorithm that is in charge of providing the abstraction of a reliable point-to-point link (i.e., with a protocol ensuring Reliable Communications on generic networks). For that purpose, one can either use Dolev's algorithm, CPA, or even topology specific protocols [25], depending on the assumptions one is willing to make regarding the communication network. In the global fault model and without additional network assumptions, one has to rely on Dolev's algorithm. The combination of Bracha's and Dolev's algorithms has recently been used to design a randomized Byzantine agreement protocol on partially connected networks [5].

In practice, Bracha's and Dolev's protocols can be combined by replacing each send-to-all operation at process p_i (i.e, **forall** $q \in \Pi$ **do** {**trigger**$\langle al, \texttt{Send} \mid q, [mType, m] \rangle$}) in Algo. 1 (lns. 12, 16, 23, 30) by a $\langle Dolev, \texttt{Broadcast} \mid [mType, m] \rangle$ operation, where $mType \in \{\texttt{SEND}, \texttt{ECHO}, \texttt{READY}\}$. In addition, one has to replace $\langle al, \texttt{Deliver} \mid q, [msgType, m] \rangle$ in Algo. 1 (lns. 14, 18, 25) by $\langle Dolev, \texttt{Deliver} \mid [msgType, m] \rangle$, where q is the source of the $f+1$ disjoint paths identified to trigger a Dolev-deliver in Algo. 2, ln. 20. The resulting protocol stack is illustrated in Fig. 2, which also shows the interface of each module. This combination can be made more practical by applying modifications MD.1–5. However, given that the number of messages generated by Bracha's and Dolev's protocols are respectively $\mathcal{O}(N^2)$ and $\mathcal{O}(N!)$, and that Dolev's algorithm requires solving a problem with exponential complexity (i.e., finding disjoint paths in a unknown network), the combination of these two protocols does not scale well with the number of processes.

V. Cross-layer Bracha-Dolev Implementation and Functional Modifications

In the following, we describe how Bracha's and Dolev's protocols can be combined and leverage modifications MD.1-5.

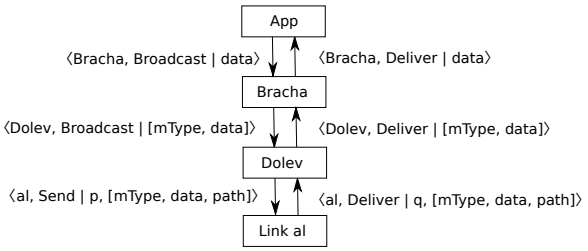

Fig. 2. Composition of Bracha's and Dolev's algorithms to implement BRB on partially connected networks.

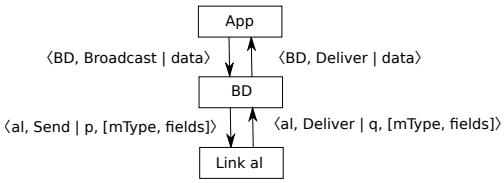

Fig. 3. Interfaces in a cross-layer combination of Bracha's and Dolev's protocols to implement BRB on partially connected networks.

Protocol stack and interfaces. Fig. 3 illustrates the interface of our combination of Bracha's and Dolev's protocols, modified to support modifications MD.1–5 and repeatable broadcasts. We detail below the fields used in this interface, and Sec. VI-C shows how some of those fields can be made optional.

Repeatable broadcast. Across the system's life, we assume that a process might broadcast several times the same payload data. For example, this would be required for sensing applications (e.g., temperature monitoring). We then assume that a broadcast message contains the payload data m, the ID of its source process s, and a broadcast ID bid (a sequence number) that the source process monotonically increases after each broadcast. If the source is correct, the source and broadcast IDs of a broadcast message uniquely identify a payload data. A Byzantine process might reuse a broadcast ID for several payload data, in which case all processes will agree on delivering at most one payload and ignore the others.

We therefore modify the interface of the BRB protocol so that Broadcast and Deliver operations include the s and bid fields: $\langle BD, \texttt{Broadcast} \,|\, [(s, bid), m]\rangle$ and $\langle BD, \texttt{Deliver} \,|\, [(s, bid), m]\rangle$. This modification limits the Byzantine processes' ability to replay messages. Each process maintains a transmission graph for each message it receives that is uniquely identified by the broadcasting process ID, the broadcast ID, the message type, the sending process ID and the payload (which is necessary to handle Byzantine senders).

Applying modifications MD.1–5. For MD.1–5 to be used in the combination of Bracha's and Dolev's algorithms, the format of the ECHO and READY messages has to be slightly modified to contain the ID of their original sender, so that ECHO or READY messages that are broadcast by different processes can be distinguished. Using the original specification of Dolev's protocol, one could rely on the full paths received along with messages to identify the sender of an ECHO or READY message. However, optimization MD.2 of Dolev's protocol replaces a path by an empty path upon delivery of a message, which then prevents the identification of the original sender of the message from its path. For this reason, ECHO and READY messages must contain an additional field that identifies their sender. For example, an ECHO message generated by process p_i would then have the format $[\texttt{ECHO}, p_i, (s, bid), m, path]$.

In the following, to avoid repetitions, we will use BD to refer to the Bracha-Dolev protocol combination; BDopt to refer to the version of BD optimized with the modifications MD.1–5; and name our novel modifications MBD.1–12.

VI. Optimizing for Latency

A. Limiting payload transmissions

Bracha's and Dolev's protocols make processes include the payload data (i.e., the content they wish to broadcast) in each message sent to their neighbors. However, when the payload data is large, and because of the large number of messages generated, it is worthwhile to reduce the number of times the payload data is exchanged. Some Byzantine-resilient protocols replace the payload by hashes or signatures for this purpose. These methods could be used in the Bracha-Dolev combination too. However, we aim at avoiding the use of cryptographic methods to contain the computational overhead of the protocol but also to tolerate computationally unbounded adversaries.

Modification [MBD.1] We modify the protocol as follows. Upon receiving a message that contains a previously unknown payload data, a process p_i associates it to a unique local ID. Note that simply replacing a payload data by its source ID and sequence number would not allow this modification in asynchronous settings, because Byzantine processes might send several payloads with the same sequence number and because messages might be reordered during their transmission, which would threaten the BRB-Agreement property. When sending a message to its neighbors related to this payload data for the first time, the process then includes the local ID it chooses. In later messages, p_i only sends the local ID of that payload data. A neighbor of p_i, say p_j, might generate its own local ID for a given payload before receiving p_i's local ID. This is not a problem, and p_i can either use its local ID or the one received from p_j to interact with p_j.

Because of the asynchrony of communications, a process might actually receive the message that contains the payload data after subsequent messages that only use the local ID. In that case, we modified the protocol so that a process stores the messages that mention an unknown payload in a queue, before processing them all when the payload data is finally received.

In the worst case, our method requires a process to receive a given payload data once from each of its neighbors. In practice, the size of local IDs should be decided in an ad-hoc manner so that a process does not run out of available IDs.

B. Bracha phase transitions

Processes receive and forward Send, Echo and Ready messages that have been created according to Bracha's protocol, following a dissemination that Dolev's protocol determines. In

the following, we describe several modifications that processes can implement when they create messages, following the reception of a message that has been delivered using Dolev's algorithm. These modifications reduce the total number of messages exchanged, and the overall bandwidth consumption.

Echo and Ready amplifications. In Bracha's protocol, it is known (and necessary) that the reception of $f+1$ Readys in Bracha's protocol allows a process to generate its own ready message, if it has not done so already [22, 23]. This situation can happen in Bracha's protocol because the links are asynchronous. However, we also observe that the delivery of $f+1$ Echos allows a process to generate its own Echo message, and send it if it was not sent before because the original Send message has not yet been delivered. Our modification MBD.2 of Send messages into single-hop messages makes this amplifications of Echos necessary.

Single-hop Send messages [MBD.2]. In the default combination of Bracha's and Dolev's protocols, the Send messages that a correct process creates reach all the processes. We modify the protocol so that upon receiving a Send message, a neighbor of the source stops disseminating it, and creates instead an Echo message that it relays to all its neighbors. When a process p_i Dolev-delivers a $[\texttt{SEND}, (s, bid), m, path]$ message that it received from a neighbor p_j, the combination of Bracha's and Dolev's algorithms makes it forward a modified Send message to its neighbors not included in $path$, i.e., $[\texttt{SEND}, (s, bid), m, path{+}[p_j]]$. A process can avoid generating and forwarding the modified Send message without loss of information. A Send message is therefore a single-hop message and does not need to carry a $path$.

Proof. Bonomi et al. proved that upon Dolev-delivery of a message $[\texttt{SEND}, (s, bid), m, path]$ a process p_i can send a $[\texttt{SEND}, (s, bid), m, [\,]]$ message instead of the $[\texttt{SEND}, (s, bid), m, path{+}[p_j]]$ message to its neighbors not included in $path$ (MD.2 in Sec. IV-B). This modification decreases the size of the forwarded Send message. Following Bracha's algorithm, after having validated a Send message, process p_i should also send an $[\texttt{ECHO}, p_i, (s, bid), m, [\,]]$ message to all its neighbors. However, upon receiving an $[\texttt{ECHO}, p_i, (s, bid), m, path]$ message, processes can extract the $[\texttt{SEND}, (s, bid), m, path]$ message they should have received in the unmodified protocol and process it.

Echo to Echo transitions [MBD.3]. When a process p_i receives an $[\texttt{ECHO}, p_k, (s, bid), m, path]$ from a neighbor p_j that makes p_i Dolev-deliver the message, according to Dolev's protocol and MD.2, p_i forwards $[\texttt{ECHO}, p_j, (s, bid), m, [\,]]$ to its neighbors not included in $path$. As a result of delivering the Echo message, process p_i might also send an $[\texttt{ECHO}, p_i, (s, bid), m, [\,]]$ message to all its neighbors (after having delivered the Send message of the source, or received $f+1$ Echos - using the *Echo amplification*).

These two messages can be merged into a single one, in particular because they both have to be transmitted using empty paths (the first one because it was delivered, due to MD.2, and the second one because it has just been created). For that purpose, we introduce a new message type Echo_Echo formatted as $[\texttt{ECHO\_ECHO}, p_i, p_j, (s, bid), m, path]$, where p_i is the ID of the process whose Echo message was received, while p_j is the ID of the process whose Echo was delivered by p_i and triggered the emission of the Echo message by p_i. We explain below how these messages are handled.

Finally, a process p_i can decide to which neighbors it should send the Echo_Echo message, and to which it should only send its own Echo message (cf. MD.3, IV-B): (i) $[\texttt{ECHO\_ECHO}, p_i, p_j, (s, bid), m, [\,]]$ is sent to the neighbors that have not yet Dolev-delivered $[\texttt{ECHO}, p_j, (s, bid), m]$ and are not included in the received path; and (ii) $[\texttt{ECHO}, p_i, (s, bid), m, [\,]]$ is sent to the remaining neighbors.

Proof. Upon receiving an $[\texttt{ECHO\_ECHO}, p_i, p_j, (s, bid), m, [\,]]$ message, a process can extract $[\texttt{ECHO}, p_j, (s, bid), m, [\,]]$ and $[\texttt{ECHO}, p_i, (s, bid), m, [\,]]$ messages. It is correct for p_i to forward the Echo message with an empty path (MD.2, IV-B). In addition, p_i does not have to forward the $[\texttt{ECHO}, p_j, (s, bid), m, [\,]]$ message to its neighbors that have delivered the $[\texttt{ECHO}, p_j, (s, bid), m]$ message (MD.3, IV-B), which justifies the use of an Echo message instead of an Echo_Echo message.

Echo to Ready transitions [MBD.4]. When a process p_i Dolev-delivers an $[\texttt{ECHO}, p_j, (s, bid), m, path]$ message that it received from a neighbor p_k, the standard Bracha-Dolev protocol combination makes p_i forward a modified Echo message to its neighbors not included in $path$, i.e., $[\texttt{ECHO}, p_j, (s, bid), m, path + [p_k]]$, while MD.2 makes p_i forward an empty path to those neighbors. As a result of delivering the Echo message, process p_i might also send a $[\texttt{READY}, p_i, (s, bid), m, [\,]]$ message to all its neighbors (after having Dolev-delivered $2f+1$ Echo messages).

We introduce a second novel message type Ready_Echo associated to messages of the format $[\texttt{READY\_ECHO}, p_i, p_j, (s, bid), m, path]$, where p_i is the ID of the process whose Ready was received, while p_j is the ID of the process whose Echo was delivered by p_i and triggered the emission of the Ready message by p_i. The $path$ field of a Ready_Echo message describes an empty path upon creation, but this is not the case if the message is relayed. As for Echo_Echo messages, process p_i decides whether to send a Ready_Echo, Ready, Echo, or no message at all for each of its neighbors depending on whether or not it has transmitted an empty path for the Echo or the Ready.

Proof. Upon receiving a $[\texttt{READY\_ECHO}, p_i, p_j, (s, bid), m, [\,]]$ message, a process can extract $[\texttt{ECHO}, p_j, (s, bid), m, [\,]]$ and $[\texttt{READY}, p_i, (s, bid), m, [\,]]$ messages. It is correct for p_i to forward the Echo message with an empty path (MD.2, IV-B). In addition, p_i does not have to forward the $[\texttt{ECHO}, p_j, (s, bid), m, [\,]]$ message to its neighbors that have delivered the $[\texttt{ECHO}, p_j, (s, bid), m]$ message (MD.3, IV-B), which justifies the use of a Ready message instead of a Ready_Echo message for those neighbors.

Reception of Echo_Echo and Ready_Echo messages. We detail here how processes handle Ready_Echo messages (Echo_Echos are handled similarly). Upon receiving a $[\texttt{READY\_ECHO}, p_i, p_j, (s, bid), m, path]$ message, we have

Algorithm 3 *READY_ECHO message reception at process* p_i

upon event $\langle$ al, Deliver | p_l, [READY_ECHO, p_r, p_e, (s, bid), m, path] $\rangle$ **do**
  insert $path{+}[p_l]$ in $[\texttt{ECHO}, p_e, (s, bid)]$'s graph
  insert $path{+}[p_l, p_e]$ in $[\texttt{READY}, p_r, (s, bid)]$'s graph

  // Dolev-delivered(msg) checks whether msg has already been
  // Dolev-delivered prior to this event (MD.5)
  $sendEcho = !\texttt{Dolev-delivered}([\texttt{ECHO}, p_e, (s, bid)])$
  $sendReady = !\texttt{Dolev-delivered}([\texttt{READY}, p_r, (s, bid)])$

  // actOnEchos and actOnReadys create and send Echo/Ready
  // messages if necessary, according to Bracha's algorithm
  // Dolev-delivering(msg) checks whether msg is being
  // Dolev-delivered at this event (MD.2)
  $(\texttt{Dolev-delivering}([\texttt{ECHO}, p_e, (s, bid)]))?$ $\{\texttt{actOnEchos}(s, bid, p_e);\ epath = [\,]\} : epath = path{+}[p_l]$
  $(\texttt{Dolev-delivering}([\texttt{READY}, p_r, (s, bid)]))?$ $\{\texttt{actOnReadys}(s, bid, p_r);\ rpath = [\,]\} : rpath = path{+}[p_l]$

  // Avoiding the neighbors that sent empty paths (MD.3)
  $NDE = \texttt{neighborsThatDelivered}([\texttt{ECHO}, p_e, (s, bid)])$
  $NDR = \texttt{neighborsThatDelivered}([\texttt{READY}, p_r, (s, bid)])$

  if $sendEcho$ **and** $!sendReady$ **then**
    forall $x \in \texttt{neighbors}(p_i) \setminus (NDE \cup path \cup \{p_l\})$
      trigger $\langle al, \texttt{Send} \mid x, [\texttt{ECHO}, p_e, (s, bid), m, epath]\rangle$
  else if $!sendEcho$ **and** $sendReady$ **then**
    forall $x \in \texttt{neighbors}(p_i) \setminus (NDR \cup path \cup \{p_l\})$
      trigger $\langle al, \texttt{Send} \mid x, [\texttt{READY}, p_r, (s, bid), m, rpath]\rangle$
  else
    forall $x \in NDR \setminus (NDE \cup path \cup \{p_l\})$
      trigger $\langle al, \texttt{Send} \mid x, [\texttt{ECHO}, p_r, (s, bid), m, rpath]\rangle$
    forall $x \in NDE \setminus (NDR \cup path \cup \{p_l\})$
      trigger $\langle al, \texttt{Send} \mid x, [\texttt{READY}, p_r, (s, bid), m, rpath]\rangle$
    forall $x \in neighbors(p_i) \setminus (NDE \cup NDR \cup path \cup \{p_l\})$
      if $epath == rpath$ **then**
        trigger $\langle al, \texttt{Send} \mid x, [\texttt{READY\_ECHO}, p_e, p_r, (s, bid), m, epath]\rangle$
      else
        trigger $\langle al, \texttt{Send} \mid x, [\texttt{ECHO}, p_e, (s, bid), m, epath]\rangle$
        trigger $\langle al, \texttt{Send} \mid x, [\texttt{READY}, p_r, (s, bid), m, rpath]\rangle$
      end if
  end if

seen that processes can extract $[\texttt{ECHO}, p_j, (s, bid), m, path]$ and $[\texttt{READY}, p_i, (s, bid), m, path]$ messages. However, it might not always be necessary for processes to forward the Ready_Echo message to their neighbors, as a Dolev dissemination would dictate. Indeed, some of these neighbors might have already delivered the Echo or the Ready message (MD.3, IV-B), or neighbors that have delivered can be included in the received path (MD.4, IV-B). Algo. 3 provides the pseudocode that processes use to handle the Ready_Echo messages they receive.

Empirical lessons. We observed that the system's latency is lower when processes first forward a message they have received before sending the message they might have created. In addition, one might also remark that we have not mentioned the possibility for a received Ready message to trigger the emission of a Ready message, which might be exploited to create a Ready_Ready message. Similarly, one could say that receiving an Echo_Echo message might trigger the emission of an Echo message, which would justify the need to create a message type that would carry three Echos. In practice, we have not observed that these phenomena appeared sufficiently often to justify this implementation.

C. Optimized messages [MBD.5]

So far, we have assumed that messages in the Bracha-Dolev protocol combination contain, besides the payload data, a payload ID (s, bid), a message type $mType \in \{\texttt{SEND}, \texttt{ECHO}, \texttt{READY}\}$, the ID of the process that generated the message, and a list of process IDs to indicate the path the message followed in the topology. We now detail several observations that allow these messages to be more concisely transmitted without loss of information.

Send messages. As we have seen in the previous section, Send messages are single-hop messages. Therefore, they do not have to carry the source ID in the payload's ID (s, bid), because communication links are authenticated. Send messages also always carry the payload data along with the local ID that was chosen by the source. Send messages therefore have the following format: $[\texttt{SEND}, bid, localPayloadID, payloadSize, payload]$.

Optional fields. We now consider the other types of message we have mentioned: Echo, Ready, Echo_Echo, and Echo_Ready. A process receives a message in a buffer, and decodes the information it encloses according to a header that contains the message type, and several bits that indicate the fields to be read in the message.

First, messages might contain or not the payload data, which is indicated by a bit $payloadBit$. If $payloadBit$ is set, a process expects to read the ID of the payload data (s, bid), the local ID selected by the sending process $localPayloadID$, and the payload data. If $payloadBit$ is not set, a process only expects to read $localPayloadID$.

Second, the ID of the source that generates a message is originally included in the path of a message, and therefore does not always have to be transmitted in a message. However, when processes replace a propagation path by an empty path (using MD.2), this is not the case anymore, and processes therefore restore the sender field in their message. To handle both situations, we add a bit $senderBit$ that indicates whether the sender ID is included in the message.

Finally, if the creator of the message is not the neighbor from which it is received (indicated by the authenticated link), a message contains a path, which is decoded using a $pathLen$ field that indicates the number of process IDs to be read, followed by a list $path$ that contains the process IDs.

For example, an Echo message can have the following format $[\texttt{ECHO}, payloadBit{=}1, senderId{=}1, (s, bid), echoSenderID, localPayloadID, payloadSize, payload, pathLen, path]$ at most once between any pair of processes for a given sender. In a more positive situation, an Echo message can be transmitted as $[\texttt{ECHO}, payloadBit{=}0, senderId{=}0, localPayloadID]$, which happens every time a process generates its own Echo.

VII. Optimizing for Throughput

A. Handling asynchrony

We present several optimizations that can be implemented when the Bracha part of the protocol generates novel messages

while the Dolev propagation of a received message has to continue. These optimizations reduce the overall amount of information exchanged over the network. **Ignore Echos received after Dolev-delivering the corresponding Ready [MBD.6].** If a process p_i Dolev-delivers a Ready message that was sent by a process p_j then p_i can stop disseminating and discard any Echo message emitted by p_j.

Proof. If p_i Dolev-delivered the Ready message of a process p_j, it means that p_i verified that p_j did send the Ready message. The Echo message that p_j sent contains less information and reflects an old state of p_j.

Ignore Echos received after delivering the content [MBD.7]. If a process p_i Bracha-delivers a content (because it has Dolev-delivered $2f+1$ Readys), then it can stop disseminating and discard all Echo messages it might receive that are related to this content.

Proof. If p_i Bracha-delivered a message, then p_i has Dolev-delivered Ready messages from $2f+1$ distinct processes and at least $2f+1$ processes are reliably exchanging the related Ready messages. Thus, all processes will eventually Bracha-deliver the message and the Echo message is not needed.

Receiving Readys before transmitting Echos [MBD.8]. If a node p_i has Dolev-delivered the Ready message of its neighbor p_j, it can avoid sending any future Echo message it receives to p_j. Note that this happens as soon as p_j's Ready is received, since it is transmitted with an empty path and therefore immediately delivered.

Proof. If p_j is faulty, whether or not it transmits a message to p_j will not impact the protocol's guarantees. If p_j is correct, it has Dolev-delivered $\lceil \frac{N+f+1}{2} \rceil$ Echos, or $f+1$ Readys, before sending its Ready, and each of these Echos or Readys have been forwarded with empty paths. These messages will eventually be Dolev-delivered by all processes.

Avoiding neighbors that delivered [MBD.9]. If a node p_i has received $2f+1$ Readys (generated by $2f+1$ different processes) with empty paths that are related to the same content from a neighbor p_j, then p_i can avoid sending any message related to that content to p_j in the future.

Proof. If p_j is correct it sent $2f+1$ Readys with an empty path to all its neighbors (MD.2). These neighbors will eventually receive these $2f+1$ Readys and be able to deliver independently from p_i's message transmissions through p_j. Again, if p_j is faulty, it will not make a difference for the protocol's properties whether or not it receives messages from p_i.

Ignore messages whose path is a superpath of a message previously received [MBD.10]. If a node p_i receives a message m_1 (e.g., an Echo from process p_0) with path $path_1$, for which a previous message m_0 with $path_0$ was received (and which only differs from m_1 by its path) and such that $path_0$ is a subpath of $path_1$, then p_i can ignore m_1.

Proof. The path $path_1$ does not help p_i identify $2f+1$ disjoint paths towards the source, because $path_0$ contains a subset of the processes that are included in $path_1$. Similarly, it is not useful to forward the received message with a modified path equal to $path_1+[p_i]$, because a subpath such as $path_0+[p_i]$ or $[\,]$ has already been transmitted.

Upon receiving a message, a process checks whether a subpath has been previously received, and if so discard the message, otherwise the received path is saved. Processes represent paths using bit arrays, and store them in a list. The observed performance impact of this method does not justify the use of more efficient data structures.

B. Non-tight cases

We introduce several modifications that decrease the number of processes that participate in various phases of the Bracha-Dolev protocol combination when the number of processes in the network is larger than $3f+1$.

Reduced number of messages in Bracha [MBD.11]. To implement this method, we assume that processes know the ID of all processes in the system. The processes with the $\lceil \frac{N+f+1}{2} \rceil + f$ smallest IDs generate Echos, while the processes with the $2f+1+f$ smallest IDs generate Readys. The other processes simply relay the messages they receive. All processes deliver when they have collected $2f+1$ Readys.

Proof. To realize this, one has to reason starting from the end of Bracha's protocol. A process needs $2f+1$ Readys to deliver, therefore only $3f+1$ processes are required to send a Ready, since there are at most f faulty processes in the system, and there are no message losses. To be able to send a Ready, a process needs to receive $\lceil \frac{N+f+1}{2} \rceil$ Echos. Therefore, $\lceil \frac{N+f+1}{2} \rceil + f$ processes are required to send an Echo. Finally, the source needs to transmit its Send message to $\lceil \frac{N+f+1}{2} \rceil + f$ processes. Note that when $N = 3f+1$, the number of processes that are chosen to participate in each phase is always equal to $3f+1$, as indicated by Bracha's protocol.

Send messages in Bracha-Dolev. [MBD.12] If the source has more than $2f+1$ neighbors, it can transmit its Send message to $2f+1$ of its neighbors, instead of to all of them.

Proof. If the source sends its Send message to $2f+1$ of its neighbors, it means that at least $f+1$ correct neighbors will eventually receive the Send message. These correct neighbors will then initiate a Dolev-broadcast of their Echo message (cf. Send to Echo transition) to all other nodes in the system. Since every pair of processes are connected through at least $2f+1$ disjoint paths, every process in the system will eventually receive at least $f+1$ Echo messages. Using the Echo amplification, each of these processes will then send their own Echo, which means that every correct process will have delivered the original Send message of the source.

Discussion. Those modifications decrease the number of exchanged messages drastically. However, they also often increase the protocol's latency. Indeed, the processes selected to generate Echo/Ready messages might be located far from each other in the network. Without these modifications all processes send their Echo/Ready messages, which allows processes to collect the required numbers of Echos/Readys faster.

VIII. Performance evaluation

A. Settings

We use the Omnet++ network simulator v.5.6.2 [26], which runs C++ code. We use an Intel Xeon E5-4650 CPU (2.10GHz)

TABLE I
MESSAGE FIELDS.

Msg field	Description	Size
$mtype$	Message type	4 bits
s	ID of the source process	32 bits
bid	Message ID	32 bits
$data\_size$	Payload data size in Bytes	32 bits
$data$	Payload data	16B/16KB
$erId_1$	ECHO/READY sender ID	32 bits
$erId_2$	Embedded ECHO/READY sender ID	32 bits
$path_size$	Path length (number of processes)	16 bits
$path$	List of process IDs	Variable

machine. We evaluate the impact of modifications MBD.1–12 on BDopt, the state-of-the-art combination of Bracha's algorithm with Dolev's algorithm modified with Bonomi et al.'s improvements [6]. We vary the number N of processes, the number f of Byzantine processes, and the network connectivity k, so that $N \geq 3f+1$ and $k \geq 2f+1$. For each experiment with a (N, k, f) tuple, we generate a regular random graph [27] using the NetworkX Python library [28], and report the average of 5 runs. The latency and throughput of each network link are set to 0.5ms and 1Mbps, respectively. We measure the protocols' latency as the time when all correct processes have delivered the broadcast message. Note that further messages might still be exchanged after this point. We consider payloads of 16B and 16KB, to represent small and large messages. We simulate the protocol execution for a single broadcast message (i.e., only one process broadcasts a message), and make all processes participate to maximize the bandwidth consumption. Table I details the size of the message fields. In plots, vertical bars indicate the minimum and maximum observed values.

B. Impact of individual modifications

We evaluated the impact of each modification on BDopt's latency and network consumption with a large set of parameters. Table II summarizes the protocol layer on which a modification is applied, and its impact on latency and throughput for small and large payload sizes. We observed that one can decide whether a modification should be used to improve either latency or the network consumption based on the payload size and the network connectivity k. We illustrate some modifications when $N = 31$ and a 16KB payload. Fig. 4 shows that MBD.4 almost always improves latency (from -50% to 1.2%), and is more effective when k increases or f decreases. Figs. 5 and. 6 show MBD.11's impact on latency and on the network consumption, respectively. MBD.11 limits the number of processes that generate Echo/Ready messages while the others only relay messages. MBD.11 decreases the network consumption (-66% to -16%) but also often increases latency (-31% to +240%). Larger payloads lead to larger performance improvements.

C. CPU and memory consumption

We measured the total processing time per process during an experiment where $N=31$, $f=4$ and a 16KB payload. This time decreases when the network connectivity increases for all combinations of modifications from 6s to 0.5s. Processing

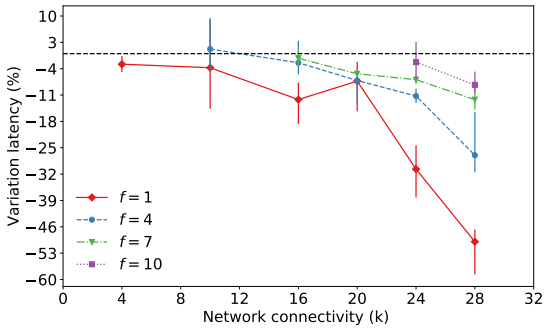

Fig. 4. MBD.4's latency impact - $N = 31$, 16KB payload.

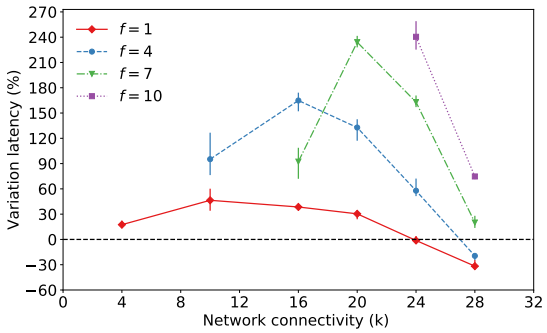

Fig. 5. MBD.11's latency impact - $N = 31$, 16KB payload.

times cannot be directly associated with the protocol's latency. First, because processes continue to treat messages after they have delivered a content. And second, because Omnet++ does not advance the simulation time when messages are processed. Compared to real deployments this effect would create differences only when several messages are simultaneously received. We partially compensate it so that, at creation time, messages are delayed by a time that corresponds to the processing time of the message that triggered their emission. Latencies measured in a real-deployment would probably be slightly higher than those we observed. However, the effect of our modifications on latency and bandwidth would be the same since the processing times were not significantly impacted by our modifications. The modifications have a very small impact on memory usage, since the most costly ones require storing a boolean per process.

D. Latency vs. Network Consumption

Based on the observed impact of our modifications, we compare the latency and network consumption improvements of three combinations of modifications to BDopt that respectively contain: (i) *lat.*: only the modifications that decrease latency; (ii) *bdw.*: only the modifications that decrease bandwidth consumption; and (iii) *lat. & bdw.*: only the modifications that decrease both latency and bandwidth consumption. Figs. 7 and 8 respectively show the latency decrease and the network consumption of these configurations with $(N, f)=(31, 4)$ and a 16B payload. Configurations *lat.* and *lat. & bdw.* always decrease latency (from -25% to 0%), while *bdw.* almost always increase it (up to 130%), because of MBD.11. However,

TABLE II
IMPACT OF THE MODIFICATIONS WITH RANDOM GRAPHS AND SYNCHRONOUS COMMUNICATIONS.

		Small payload (16B)				Large payload (16KB)			
MBD	**Layer**	**Lat. var. %**	**Useful when**	**# bits var.**	**Useful when**	**Lat. var. %**	**Useful when**	**# bits var.**	**Useful when**
1	B	$[-22, 4.3]$	always	-63	always	$[-93, -78]$	always	-97	always
2	BD	$[-21, 62]$	large k	$[-1.8, 9]$	small f$\vee$large k	$[-15.4, 107]$	large k	$[-4.3, 0.4]$	always
3	BD	$[-21, 99]$	large k	$[-1.8, 12]$	large k	$[-17, 104]$	large k	$[-4.3, 0.4]$	large k
4	BD	$[-25, 5]$	large k	$[-1.4, 19.7]$	f=1 $\vee$ large k	$[-50, 1.2]$	always	$[-1.3, 0.8]$	f=1 $\vee$ large k
5	BD	$[-5.3, 9.4]$	small f and k	$[15, 19]$	never	$[-4.1, 4.5]$	small f and k	$[-0.9, 0]$	always
6	B	$[-5.3, 1.6]$	-	$[-9.6, 0]$	always	$[-4, 4.5]$	-	$[-0.9, 0.13]$	always
7	B	$[-3, 1.4]$	-	$[-13.2, -1.4]$	always	$[-2, 5.4]$	-	$[-5.8, 0.07]$	always
8	B	$[-8.7, 3.8]$	-	$[-12.8, -3.1]$	always	$[-3.6, 2]$	-	$[-4.9, 0.07]$	always
9	BD	$[-5.7, 3.0]$	-	$[-43, 0]$	always	$[-3.9, 1.9]$	-	$[-38, 0]$	always
10	D	$[-5.1, 1.3]$	small f and k	$[-1.9, 0]$	always	$[-4.2, 6.2]$	small f and k	$[-1.5, 0]$	always
11	B	$[-25, 149]$	small f$\wedge$large k	$[-66, -16]$	always	$[-31, 240]$	small f$\wedge$large k	$[-66, -16]$	always
12	B	$[-15, 27]$	large k	$[-2.7, 2.6]$	small f and k	$[-17, 57]$	large k	$[0.26, 4.7]$	never

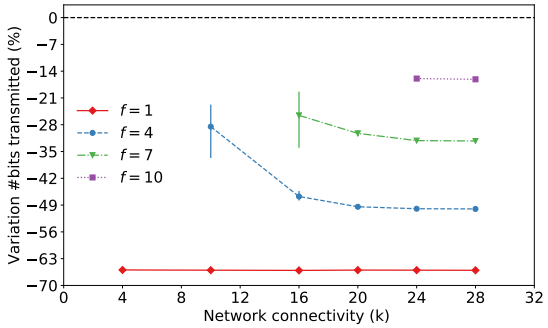

Fig. 6. MBD.11's bandwidth consumption impact - $N = 31$, 16KB payload.

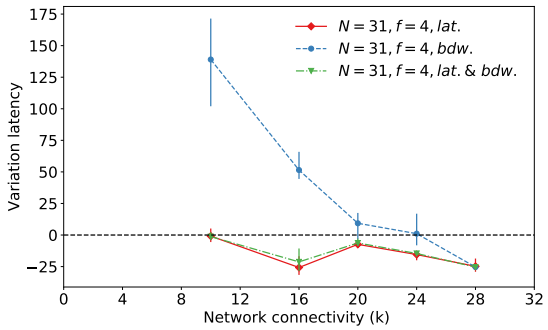

Fig. 7. Latency impact of the configurations - $(N, f) = (31, 4)$, 16B payload.

bdw. decreases the network consumption the most (from 25 to around 12KB in average, i.e., around -50%), while *lat.* and *lat. & bdw.* have a similar impact (from 25 to around 20KB in average, i.e., -20%). These results show that there is no single best configuration to optimize both latency and network consumption. In the same settings with 16KB payload, the latency is decreased between -93% and -83%, while the bandwidth consumption is decreased by -99.4% to -97%.

E. Evolution with the number of processes

In this experiment, we considered graphs that include 25, 31, 40, and 73 processes. We chose the average possible f value for each N value. Simulating systems with larger N values is challenging because of the exponential number of

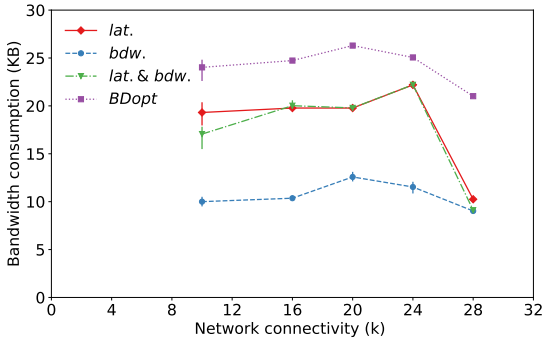

Fig. 8. Bandwidth consumption per configuration - $(N, f) = (31, 4)$, 16B payload.

messages, memory and computing time that processes require to execute the Bracha-Dolev protocol combination. Figs. 9 and 10 respectively show the bandwidth and latency improvement of the *lat.* and *bdw.* configurations over BDopt with a 16B payload. We omit configuration *lat. & bdw.* in those figures for clarity. As one can see, configurations *bdw.* and *lat.* improve the bandwidth consumption respectively by -80% to -70%, and by -63% to -38%. Configuration *lat.* also improves the latency (-50% to +2%). However, *bdw.* increases the latency by up to 400% when k is small, which goes out of the plot, but otherwise also manages to improve latency down to -40%. Latency and network consumption improvements are even higher with the larger 16KB payload (resp. at least -97.6% and -97.3%).

F. Asynchronous networks

In addition to the above experiments, we also performed experiments with links whose transmission delays were computed per message using a normal distribution centered around 5ms with a standard deviation of 20ms truncated between 0 and 80ms. With these settings, messages were frequently reordered during their transmission. Using the three sets of optimizations obtained for synchronous networks, we observed a latency decrease of 10%, and a network consumption decrease of 50% with 16KB messages and $N = 31$. This suggests that overall, our modifications also perform well in asynchronous networks,

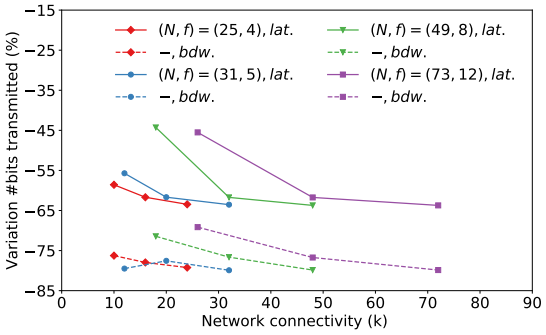

Fig. 9. Bandwidth consumption improvement over BDopt - 16B payload.

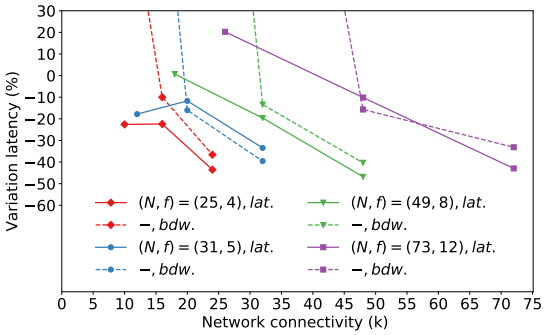

Fig. 10. Latency improvement over BDopt - 16B payload.

but also that the configurations might have to be fine tuned to improve performance.

IX. Conclusion

In this paper, we discussed the Byzantine reliable broadcast problem on unknown partially connected topologies. This problem can be solved by combining two seminal protocols, Dolev's and Bracha's algorithms. We first explained how recent improvements of Dolev's algorithms can be used in this protocol combination. We then described a total of 12 new modifications one can apply to this protocol combination to optimize latency and/or bandwidth consumption. We benchmarked the impact of each modification using the Omnet++ network simulator on random graphs. When simultaneously applied, our modifications allow the BRB problem to be solved with a lower latency and/or bandwidth consumption, e.g., respectively down to -25% and -50% with a 16B payload when N=31 and f=4. Future works include to also consider the local fault model used for example in the CPA line of work [11–13].

References

[1] J. Yu, D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo. "Repucoin: Your reputation is your power". In: *IEEE Transactions on Computers* 68.8 (2019), pp. 1225–1237.

[2] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y.-A. Pignolet, D.-A. Seredinschi, A. Tonkikh, and A. Xygkis. "Online payments by merely broadcasting messages". In: *DSN*. 2020.

[3] G. Bracha. "Asynchronous Byzantine Agreement Protocols". In: *Inf. Comput.* 75.2 (1987), pp. 130–143.

[4] D. Dolev. "Unanimity in an unknown and unreliable environment". In: *FOCS*. IEEE. 1981.

[5] Y. Wang and R. Wattenhofer. "Asynchronous Byzantine Agreement in Incomplete Networks". In: *AFT*. ACM, 2020.

[6] S. Bonomi, G. Farina, and S. Tixeuil. "Multi-hop Byzantine reliable broadcast with honest dealer made practical". In: *Journal of the Brazilian Computer Society* 25.1 (2019), 9:1–9:23.

[7] K. Zeng, K. Govindan, and P. Mohapatra. "Non-cryptographic authentication and identification in wireless networks". In: *IEEE Wireless Communications* 17.5 (2010), pp. 56–62.

[8] G. Bracha. "An asynchronous [(n-1)/3]-resilient consensus protocol". In: *PODC*. 1984.

[9] G. Bracha and S. Toueg. "Asynchronous Consensus and Broadcast Protocols". In: *J. ACM* 32.4 (1985), pp. 824–840.

[10] A. Maurer, S. Tixeuil, and X. Defago. "Communicating reliably in multihop dynamic networks despite byzantine failures". In: *SRDS*. 2015.

[11] C.-Y. Koo. "Broadcast in radio networks tolerating byzantine adversarial behavior". In: *PODC*. 2004, pp. 275–282.

[12] A. Pelc and D. Peleg. "Broadcasting with locally bounded byzantine faults". In: *Information Processing Letters* 93.3 (2005), pp. 109–115.

[13] C. Litsas, A. Pagourtzis, and D. Sakavalas. "A graph parameter that matches the resilience of the certified propagation algorithm". In: *AdHoc-Now*. 2013.

[14] A. Maurer and S. Tixeuil. "Byzantine broadcast with fixed disjoint paths". In: *Journal of Parallel and Distributed Computing* 74.11 (2014), pp. 3153–3160.

[15] A. Maurer and S. Tixeuil. "Containing byzantine failures with control zones". In: *IEEE TPDS* 26.2 (2014), pp. 362–370.

[16] M. Castro and B. Liskov. "Practical Byzantine Fault Tolerance". In: *OSDI*. 1999.

[17] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D.-A. Seredinschi. "Scalable Byzantine Reliable Broadcast". In: *DISC*. 2019.

[18] D. Kozhaya, J. Decouchant, and P. Esteves-Verissimo. "RT-ByzCast: Byzantine-Resilient Real-Time Reliable Broadcast". In: *IEEE ToC* 68.3 (2018), pp. 440–454.

[19] D. Kozhaya, J. Decouchant, V. Rahli, and P. Esteves-Verissimo. "PISTIS: An Event-Triggered Real-time Byzantine Resilient Protocol Suite". In: *IEEE Transactions on Parallel and Distributed Systems* (2021).

[20] S. B. Mokhtar, J. Decouchant, and V. Quéma. "Acting: Accurate freerider tracking in gossip". In: *SRDS*. 2014.

[21] J. Decouchant, S. B. Mokhtar, A. Petit, and V. Quéma. "Pag: Private and accountable gossip". In: *ICDCS*. 2016.

[22] C. Cachin, R. Guerraoui, and L. E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

[23] M. Raynal. *Fault-tolerant message-passing distributed systems: an algorithmic approach*. Springer, 2018.

[24] K. Menger. "Zur allgemeinen kurventheorie". In: *Fundamenta Mathematicae* 10.1 (1927), pp. 96–115.

[25] J. Behrens, S. Jha, K. Birman, and E. Tremel. "RDMC: A reliable RDMA multicast for large objects". In: *DSN*. 2018.

[26] *Omnet++ Discrete Event Simulator*. URL: https://omnetpp.org/.

[27] A. Steger and N. C. Wormald. "Generating random regular graphs quickly". In: *Combinatorics, Probability and Computing* 8.04 (1999), pp. 377–396.

[28] A. Hagberg, P. Swart, and D. S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. LANL, 2008.