Asynchronous Byzantine reliable broadcast with a message adversary

Timothé Albouy, Davide Frey, François Taïani, Michel Raynal

Asynchronous Byzantine Reliable Broadcast With a Message Adversary

Timothé Albouy[a], Davide Frey[a], Michel Raynal[a], François Taïani[a]

[a] *Univ Rennes, Inria, CNRS, IRISA, Rennes, 35000, France*

Abstract

This paper considers the problem of reliable broadcast in asynchronous authenticated systems, in which n processes communicate using signed messages and up to t processes may behave arbitrarily (Byzantine processes). In addition, for each message m broadcast by a correct (i.e., non-Byzantine) process, a message adversary may prevent up to d correct processes from receiving m. (This message adversary captures network failures such as transient disconnections, silent churn, or message losses.) Considering such a "double" adversarial context and assuming $n > 3t + 2d$, a reliable broadcast algorithm is presented. Interestingly, when there is no message adversary (i.e., $d = 0$), the algorithm terminates in two communication steps (so, in this case, this algorithm is optimal in terms of both Byzantine tolerance and time efficiency). It is then shown that the condition $n > 3t + 2d$ is necessary for implementing reliable broadcast in the presence of both Byzantine processes and a message adversary (whether the underlying system is enriched with signatures or not).

Keywords: Asynchronous system, Byzantine processes, Churn, Message adversary, Message losses, Message-passing, Message signatures, Reliable broadcast, Transient disconnection.

1. Introduction

Reliable broadcast. Introduced in the mid-eighties, *Reliable Broadcast* is a fundamental communication abstraction that lies at the center of fault-tolerant asynchronous distributed systems. Formally defined in [8, 9], it allows each process to broadcast messages in the presence of process failures, with well-defined delivery properties[1]. In turn, these properties make it possible to design provably correct distributed software for upper-layer applications based on such a broadcast abstraction.

Intuitively, reliable broadcast guarantees that the non-faulty processes deliver the same set of messages, which includes at least all the messages they broadcast. This set may also contain messages broadcast by faulty processes. The fundamental property of reliable broadcast lies in the fact that no two non-faulty processes deliver different sets of messages [10, 25].

[1] The term *delivery* refers here to the application layer where a process receives and processes the content of an application message (see Section 2.1).

When some processes may suffer from Byzantine failures [21], designing a reliable broadcast communication abstraction that tolerates such failures is far from trivial. Such an algorithm is called Byzantine-tolerant reliable broadcast (BRB) and we say that a process brb-broadcasts and brb-delivers messages. The most famous BRB algorithm is due to Bracha [8] (1987). For an application message, this algorithm gives rise to three sequential communication steps and up to $(n-1)(2n+1)$ implementation messages sent by correct processes. This algorithm requires $n > 3t$, which is optimal in terms of fault tolerance.

Recent works related to reliable broadcast. Due to its fundamental nature, BRB has been addressed by many authors. Here are a few recent results. Similarly to Bracha's algorithm, all these algorithms assume an underlying fully connected reliable network.

- The versatility dimension of Bracha's algorithm has been analyzed in [18, 26].
- Addressing efficiency issues, the BRB algorithm presented in [19] implements the reliable broadcast of an application message with only two communication steps and up to $n^2 - 1$ implementation messages sent by correct processes. The price to pay for this gain in efficiency is a weaker t-resilience, namely $t < n/5$. Hence, this algorithm and Bracha's algorithm differ in their trade-off between t-resilience and message/time efficiency.
- Scalable BRB is addressed in [17]. The goal of this work is to avoid paying the $O(n^2)$ message complexity price. To this end, the authors use a non-trivial message-gossiping approach which allows them to design a sophisticated BRB algorithm satisfying probability-dependent properties.
- BRB in dynamic systems is addressed in [16] (*dynamic* means that a process can enter and leave the system at any time). In their article, the authors present an efficient BRB algorithm for such a context. This algorithm assumes that, at any time, there are at least two times more correct processes than Byzantine ones in the system.
- An efficient algorithm for BRB with long inputs of b bits using lower costs than b single-bit instances is presented in [22]. This algorithm, which assumes $t < n/3$, achieves the best possible communication complexity of $\Theta(nb)$ input sizes. This article also presents an authenticated extension of this solution.

Hybrid Byzantine fault models. Byzantine process failures cover an extensive range of adversarial behaviors but remain pinned to specific processes (which are called *Byzantine*). An alternative fault model, proposed by Santoro and Widmayer for synchronous networks [28, 29], considers *mobile* (or dynamic) *link failures* between correct processes. In this model, failures are no longer bound to particular processes. Instead, a *message adversary* may corrupt or delete some share of the $n(n-1)$ possible transmissions taking place during one synchronous round, where n is the number of processes in the system.

Generalizing further, mobile link failures and Byzantine processes may be combined to produce a *hybrid fault model*, in which some failures are pinned to specific processes while others affect links dynamically. Biely, Schmid, and Weiss [6] have, in particular, proposed an extensive

hybrid fault model for synchronous systems that includes both a range of process failures (including Byzantine behaviors) along with mobile link failures between correct processes. As in the model of Santoro and Widmayer, link failures are mobile in that they might impact different processes in different (synchronous) rounds. The model is constrained by limiting the number of link failures that a process might experience during a synchronous round both as a sender (*send* link failures) and as a recipient (*receive* link failures).

This model was extended by Schmid and Fetzer [30][2] to *asynchronous round-based algorithms*. Schmid and Fetzer present, in particular, a Simulated Authenticated Broadcast algorithm (a weak form of Byzantine broadcast allowing duplicity by Byzantine senders) and a Randomized Consensus algorithm that work provided both send and receive link failures remain limited to specific patterns, ensuring in particular that no correct[3] process ever gets fully disconnected from other correct processes.

The work presented in this paper[4] departs from this model and introduces a hybrid Byzantine and link fault model that explicitly considers the disconnection of correct processes and applies to any message passing algorithm, whether it uses rounds or not[5]. (The broadcast algorithm presented in Section 3 in particular does not use explicit rounds.)

Our motivation for this novel hybrid fault model originated from our research on the reconciliation of local process states in distributed Byzantine-tolerant money transfer systems (a.k.a. cryptocurrencies), in which processes become temporarily disconnected. As in the model of Schmid and Fetzer [30], our model combines two types of adversary in an asynchronous network: processes may be *Byzantine*, but in addition, a *message adversary* may also remove implementation messages between correct processes[6]. Contrary to Schmid and Fetzer's approach, however, we set no limit on the number of receive link failures. As a consequence, our model allows correct processes to become disconnected for arbitrarily long periods of time. This design also allows us to eschew the notion of rounds entirely in the definition of our fault model. Building on this model, this work then addresses the problem of fault-tolerant reliable broadcast in asynchronous n-process message-passing systems enriched with message signatures, in which up to t processes are Byzantine, and a message adversary that may prevent up to d non-Byzantine processes from delivering an implementation message broadcast by a non-Byzantine process. Several researchers have indeed pointed out the fundamental role that broadcast abstractions play in Byzantine money transfer systems (see, for instance, [5, 12, 13, 14, 17, 16]). This crucial role naturally leads to considering how Byzantine broadcast can be expanded to more volatile and dynamic settings, thus motivating our proposal to combine traditional Byzantine faults with a message adversary.

The paper is made up of 5 sections.

- Section 2 defines the computing model and the Message Adversary-Tolerant Byzantine Reliable Broadcast communication abstraction (or MBRB for short).

[2]Although the work of Schmid and Fetzer [30] predates the publication of Biely, Schmid, and Weiss [6], it cites an earlier version of [6] available as a technical report.

[3]The model uses a finer notion of *obedient* process, which is ignored here for simplicity.

[4]A very preliminary version of this work appeared in [3].

[5]In Schmid and Fetzer's model, rounds are in particular essential to the definition of *receive* link failures.

[6]Schmid and Fetzer's model also encompasses arbitrary link failures which may corrupt messages.

Acronyms	Meaning
BRB	Byzantine-tolerant reliable broadcast
MA	Message adversary
MBRB	Message adversary- and Byzantine-tolerant reliable broadcast
Notations	**Meaning**
n	number of processes in the network
t	upper bound on the number of Byzantine processes
d	power of the message adversary
c	effective number of correct processes in a run ($n - t \leq c \leq n$)
ℓ	minimal nb of correct processes that mbrb-deliver a message
λ	time complexity of MBRB
μ	message complexity of MBRB

Table 1: Acronyms and notations

- Section 3 presents a signature-based algorithm implementing the MBRB abstraction, proves it is correct, and evaluates its cost. When there is no message adversary, this algorithm is optimal both in terms of Byzantine resilience and the number of communication steps[7].
- Section 4 shows that the condition $n > 3t + 2d$ is necessary and sufficient for implementing the MBRB communication abstraction (be the underlying system enriched with signatures or not).
- Finally, Section 5 concludes the article.

2. Computing Model and MBRB Abstraction

2.1. Computing Model

Process model. The system is composed of n asynchronous sequential processes denoted p_1, ..., p_n. Each process p_i has an identity, and all the identities are different and known by all processes. To simplify, we assume that i is the identity of p_i.

Regarding failures, up to t processes can be Byzantine, where a Byzantine process is a process whose behavior does not follow the code specified by its algorithm [21, 23]. Let us notice that Byzantine processes can collude to fool the non-Byzantine processes (also called correct processes). Let us also notice that, in this model, the premature stop (crash) of a process is a Byzantine failure.

Moreover, given an execution, c denotes the number of processes that effectively behave correctly in that execution. We always have $n - t \leq c \leq n$. While this number remains unknown to correct processes, it is used in the following to analyze and characterize (more precisely than using its lower bound $n - t$) the guarantees provided by the proposed algorithm.

[7]The signature-free BRB algorithm described in [8] is optimal with respect to Byzantine resilience ($t < n/3$), but requires three communication steps, while the signature-free BRB algorithm described in [19] is optimal with respect to the number of communication steps (2) but is not with respect to Byzantine resilience (it requires $t < n/5$).

Communication model. The processes communicate through a fully connected asynchronous point-to-point communication network. Although this network is assumed to be reliable—in the sense that it neither corrupts, duplicates, nor creates messages—it may nevertheless lose messages due to the actions of a message adversary (defined below).

Let MSG be a message type and v the associated value. A process can invoke the unreliable operation broadcast MSG(v), which is a shorthand for "**for all** $i \in \{1, \cdots, n\}$ **do** send MSG(v) **to** p_j **end for**". It is assumed that all the correct processes invoke broadcast to send messages. As we can see, the operation broadcast MSG(v) is not reliable. As an example, if the invoking process crashes during its invocation, an arbitrary subset of processes receive the implementation message MSG(v). Moreover, due to its very nature, a Byzantine process can send messages without using the macro-operation broadcast.

From a terminology point of view, at the system/network level, we say that messages are *broadcast* and *received*. Moreover, a message generated by the algorithm is said to be an *implementation* message (imp-message in short), while a message generated by the application layer is said to be an *application* message (app-message in short).

Message adversary. The notion of a *message adversary* (MA) was implicitly introduced in [28] (under the name *transient faults* and *ubiquitous faults*) and then used (sometimes implicitly) in many works (e.g., [2, 11, 27, 29, 31]). A short tutorial on message adversaries is presented in [24].

Let d be an integer constant such that $0 \leq d < c$. The communication network is under the control of an adversary that eliminates imp-messages sent by processes so that these imp-messages appear as being lost. More precisely, when a correct process invokes broadcast MSG(v), the message adversary is allowed to arbitrarily suppress up to d copies of the imp-message MSG(v) that were intended to correct processes. This means that, despite the fact the sender is correct, up to d correct processes may miss the imp-message MSG(v)[8].

As an example, consider a set D of correct processes, where $1 \leq |D| \leq d$, such that during some period of time, the adversary suppresses all the imp-messages sent to them. It follows that, during this period of time, this set of processes appears as a set of correct processes that are (unknowingly) input-disconnected from the other correct processes. Depending on the message adversary, the set D may vary with time. Let us notice that $d = 0$ corresponds to the weakest possible message adversary: it corresponds to a classical static system where some processes are Byzantine but no imp-message is lost (the network is fully reliable).

Let us remark that this type of message adversary is stronger, and therefore covers, the more specific case of *silent churn*, in which processes (nodes) may decide to disconnect from the network. While disconnected, such a process silently pauses its algorithm (a legal behavior in our asynchronous model) and is implicitly moved (by the adversary) to the D adversary-defined set. Upon returning, the node resumes its execution and is removed from D by the adversary.[9]

Informally, in a silent churn environment, a correct process may miss imp-messages sent by

[8] A close but different notion was introduced by Dolev in [15] (and explored in subsequent works, such as [7]) which considers static k-connected networks. If the adversary selects statically for each correct sender d correct processes that do not receive this sender's imp-messages, the proposed model includes Dolev's model with $k = n - d$.

[9] So the notion of a message adversary implicitly includes the notion of imp-message omission failures.

other processes while it is disconnected from the network. The adjective "silent" in *silent churn* expresses the fact that no notification is sent on the network by processes whenever they leave or join the system: there is no explicit "attendance list" of connected processes, and processes are given no information on the status (connected/disconnected) of their peers. In this regard, the silent churn model diverges from the classical approach when designing dynamic distributed systems, in which processes send imp-messages on the network notifying their connection or disconnection [16]. The silent churn model is a good representation of real-life large-scale peer-to-peer systems, where peers leaving the network typically do so in a completely silent manner (i.e., without warning other peers). (For more insights on this topic see for instance [20] that presents an in-depth study of distributed computation in dynamic networks.)

Let us also observe that silent churn allows us to model input-disconnections due to process mobility. When a process moves from one location to another, the sender's broadcasting range may not be large enough to ensure that the moving process remains input-connected. An even more prosaic example would be one where a user simply turns off her device or disables the device's Internet connection, preventing it from receiving or sending any further imp-messages. In this context, we consider that the message adversary removes all the incoming imp-messages from the corresponding process until the device reconnects.

Let us mention that the loss of imp-messages caused by a message adversary may be addressed using a reliable unicast protocol. These protocols were originally introduced to provide reliable channels on top of an unreliable network subject to imp-message losses. The principle is simple: the sender keeps sending idempotent imp-messages at regular intervals through an unreliable channel until it receives an acknowledgment from the receiver. This principle notoriously lies at the core of the Transmission Control Protocol (TCP), although with important practical adaptations, as TCP uses timeouts to close a malfunctioning or otherwise idle connection, typically after a few minutes.

But because there is no way to detect that a process has crashed or disconnected in an asynchronous environment, an ideal reliable unicast protocol (i.e., one that keeps on re-transmitting until success) needs to treat disconnected processes the same way as slow processes or as if there were packet losses in the network: the sender will thus potentially send infinitely many imp-messages to a disconnected receiver. To overcome this issue, some works leverage causal dependencies to avoid resending old imp-messages: if the sender receives an acknowledgment for a given imp-message, then it can stop resending the imp-messages that causally precede this imp-message and that have not been acknowledged yet (e.g., [14]). However, this approach still assumes that eventually, every communication channel lets some imp-messages pass, which is not always the case in our model, where the message adversary can permanently sever up to d channels.

Digital signatures. We assume the availability of an asymmetric cryptosystem to sign data (in practice, imp-messages) and verify its authenticity. We assume signatures are secure and, therefore, that the computing power of the adversary is bounded. Every process in the network has a public/private key pair. We suppose that the public keys are known to everyone and that the private keys are kept secret by their owner. Everyone also knows the mapping between any process' identity i and its public key. Additionally, we suppose that each process can produce at most one signature per imp-message.

The signatures are used to cope with the net effect of the Byzantine processes, and the fact that imp-messages broadcast (sent) by correct processes can be eliminated by the message adversary. A noteworthy advantage of signatures is that, despite the unauthenticated nature of the point-to-point communication channels, signatures allow correct processes to verify the authenticity of imp-messages that have not been directly received from their initial sender but rather relayed through intermediary processes. Signatures provide us with a *network-wide* non-repudiation mechanism: if a Byzantine process issues two conflicting imp-messages to two different subsets of correct processes, then the correct processes can detect the malicious behavior by disclosing to each other the Byzantine signed imp-messages.[10]

2.2. Message Adversary-Tolerant Byzantine Reliable Broadcast (MBRB)

This section introduces a new broadcast abstraction we have called *Message Adversary-Tolerant Byzantine Reliable Broadcast* (MBRB for short). The MBRB communication abstraction is composed of two matching operations, denoted mbrb_broadcast and mbrb_deliver. It considers that an identity (i, sn) (sender identity, sequence number) is associated with each app-message, and assumes that any two app-messages mbrb-broadcast by the same correct process have different sequence numbers. Sequence numbers are one of the most natural ways to design "multi-shot" reliable broadcast algorithms, that is, algorithms where the broadcast operation can be invoked multiple times with different app-messages.

When, at the application level, a process p_i invokes mbrb_broadcast(m, sn), where m is the app-message and sn the associated sequence number, we say p_i "mbrb-broadcasts (m, sn)". Similarly, when p_i invokes mbrb_deliver(m, sn, j), where p_j is the sender process, we say p_i "mbrb-delivers (m, sn, j)". We say that the app-messages are *mbrb-broadcast* and *mbrb-delivered* (while, as said previously, the imp-messages are *broadcast* and *received*).

Correctness specification. Because of the message adversary, we cannot always guarantee that an app-message mbrb-delivered by a correct process is eventually mbrb-delivered by all correct processes. Hence, in the MBRB specification, we introduce a variable ℓ which indicates the strength of the global delivery guarantee of the primitive: if one correct process mbrb-delivers an app-message then ℓ correct processes eventually mbrb-deliver this app-message.[11] The MBRB-broadcast abstraction is defined by the following properties:

- Safety:
  - MBRB-Validity (no spurious message). If a correct process p_i mbrb-delivers an app-message m from a correct process p_j with sequence number sn, then p_j mbrb-broadcast m with sequence number sn.
  - MBRB-No-duplication. A correct process p_i mbrb-delivers at most one app-message m from a process p_j with sequence number sn.

[10] The fact that the algorithm uses signed imp-messages does not mean that MBRB-broadcast requires signatures to be implemented, see [4].

[11] If there is no message adversary (i.e., $d = 0$), we should have $\ell = c \geq n - t$.

  - MBRB-No-duplicity. No two different correct processes mbrb-deliver different app-messages from a process p_i with the same sequence number sn.

- Liveness:
  - MBRB-Local-delivery. If a correct process p_i mbrb-broadcasts an app-message m with sequence number sn, then at least one correct process p_j eventually mbrb-delivers m from p_i with sequence number sn.
  - MBRB-Global-delivery. If a correct process p_i mbrb-delivers an app-message m from a process p_j with sequence number sn, then at least ℓ correct processes mbrb-deliver m from p_j with sequence number sn.

It is implicitly assumed that a correct process does not use the same sequence number twice. Let us observe that, since at the implementation level the message adversary can always suppress all the imp-messages sent to a fixed set D of d processes, the best-guaranteed value for ℓ is $c - d$. Furthermore, let us notice that the constraint $n > 2d$ prevents the message adversary from partitioning the system.

Performance metrics. In addition to the correctness specification, we define two metrics that capture the performance of an algorithm implementing the MBRB specification: λ and μ, which respectively denote the communication step complexity and the imp-message complexity of the algorithm. They are defined as follows:

- MBRB-Time-cost. If a correct process p_i mbrb-broadcasts an app-message m with sequence number sn, then ℓ correct processes mbrb-deliver m from p_i with sequence number sn in at most λ communication steps. As in similar analyses, a *communication step* is defined by assuming that Algorithm 1 (which is designed for an asynchronous network) executes in a synchronous model. In this model, all processes execute in lock-step synchronous rounds. Each synchronous round comprises a computation step followed by a communication step. In a computation step, invoked operations (e.g. mbrb_broadcast) are executed; pending messages (if any) are processed (**when** ... **do** statement); and sent messages are buffered (but not delivered). In the subsequent communication step, all buffered messages not suppressed by the message adversary get delivered to their destination. These messages are then processed in the computation step of the following synchronous round.

- MBRB-Message-cost. The mbrb-broadcast of an app-message by a correct process p_i entails the sending of at most μ imp-messages by correct processes.

Byzantine Reliable Broadcast (BRB). If $\ell = c$ (obtained when $d = 0$), the previous specification boils down to Bracha's seminal specification [8], which defines the Byzantine reliable broadcast (BRB) communication abstraction. Hence, the BRB abstraction is a sub-case of MBRB.

3. A Signature-based Algorithm Implementing the MBRB Abstraction

This section presents Algorithm 1, which implements the MBRB communication abstraction in an asynchronous setting under the constraint $n > 3t + 2d > 0$. When considering $d = 0$, this

algorithm provides both t-tolerance optimality (as in [8]) and step optimality (as in [19]): it only assumes $n > 3t$, and guarantees mbrb-delivery of an app-message in only two communication steps[12]. It follows that signatures can help save one communication step compared to classical signature-free BRB algorithms that assume $t < n/3$. Algorithm 1 fulfills the MBRB-Global-delivery property with $\ell = c - d$ under the following assumption:

- mbrb-Assumption: $n > 3t + 2d$.

3.1. Preliminaries

Implementation message types. The algorithm uses only one imp-message type, BUNDLE, that carries the signatures backing a given app-message m, along with m's content, sequence number, and emitter. BUNDLE imp-messages propagate through the network using controlled flooding.

Local data structures. Each (correct) process saves locally the valid signatures (i.e., the signed fixed-size digests of a certain data) that it has received from other processes using BUNDLE imp-messages. Each signature "endorses" a certain app-message (m, sn, j). When certain conditions are met (described below), a process further broadcasts in a BUNDLE imp-message all signatures it knows for a given triplet (m, sn, j). A correct process p_i saves at most one signature for a given triplet (m, sn, j) per signing process p_k.

Time measurement. For the proofs related to MBRB-Time-cost (Lemmas 7-10), we assume that the duration of local computations is negligible compared to that of imp-message transfer delays, and consider them to take zero time units. As the system is asynchronous, the time is measured under the traditional assumption that all the imp-messages have the same transfer delay.

3.2. Algorithm

At a high level, Algorithm 1 works by producing, forwarding, and accumulating *witnesses* of an initial mbrb-broadcast operation, until a large-enough quorum is observed by at least one correct process, at which point this quorum is propagated in one final unreliable broadcast operation.

Witnesses take the form of signatures for a given triplet (m, sn, i), where m is the app-message, sn its associated sequence number and i the identity of the sender p_i (which also produces a signature for (m, sn, i)). Signatures serve to ascertain the provenance and authenticity of these propagated BUNDLE imp-messages, thus providing a key ingredient to tolerate the limited reliability of the underlying network. They also authenticate the invoker of the mbrb_broadcast operation, and finally, in the last phase of the algorithm, they allow the propagation of a cryptographic proof that a quorum has been reached, thereby ensuring that enough correct processes eventually mbrb-deliver the app-message that was mbrb-broadcast.

In more detail, when a (correct) process p_i invokes mbrb_broadcast(m, sn), it builds and signs the triplet (m, sn, i) to guarantee its non-repudiation, and saves locally the resulting signature (line 1). Next, p_i broadcasts the BUNDLE imp-message containing the signature that it just produced (line 2).

[12]Signature-based BRB in only two communication steps is a known result [1], however, to the best of our knowledge, no existing BRB algorithm tolerates message adversaries as well as ours.

```
operation mbrb_broadcast(m, sn) is
(1)  save signature for (m, sn, i) by p_i;
(2)  broadcast BUNDLE(m, sn, i, {all saved signatures for (m, sn, i)}).

when BUNDLE(m, sn, j, sigs) is received do
(3)  if ((−, sn, j) not already mbrb-delivered
         ∧ sigs contains the valid signature for (m, sn, j) by p_j) then
(4)      save all unsaved valid signatures for (m, sn, j) of sigs;
(5)      if ((−, sn, j) not already signed by p_i) then
(6)          save signature for (m, sn, j) by p_i;
(7)          broadcast BUNDLE(m, sn, j, {all saved signatures for (m, sn, j)})
(8)      end if;
(9)      if (strictly more than (n+t)/2 signatures for (m, sn, j) are saved) then
(10)         broadcast BUNDLE(m, sn, j, {all saved signatures for (m, sn, j)});
(11)         mbrb_deliver(m, sn, j)
(12)     end if
(13) end if.
```

Algorithm 1: A signature-based implementation of the MBRB communication abstraction (code for p_i)

When a correct process p_i receives a BUNDLE$(m, sn, j, sigs)$ imp-message, it first checks if no app-message has already been mbrb-delivered for the given sequence number sn and sender p_j, and if p_j signed the app-message (line 3). If this condition is satisfied, p_i saves all the new valid signatures inside the $sigs$ set (line 4). Next, p_i creates and saves its own signature for (m, sn, j) and then broadcasts it in a BUNDLE imp-message, if it has not already done so previously (line 5-8). Finally, if p_i has saved a quorum of strictly more than $\frac{n+t}{2}$ signatures for the same triplet (m, sn, j), it broadcasts a BUNDLE imp-message containing all these signatures and mbrb-delivers the triplet (lines 9-12).[13]

Remark. The reader can notice that the system parameters n and t appear in the algorithm, whereas the system parameter d does not. Naturally, they all explicitly appear in the proof.

3.3. Algorithm proof

This section proves the correctness and performance properties of MBRB.

Theorem 1. *If the mbrb-Assumption is satisfied, Algorithm* 1 *implements MBR-broadcast with the following guarantees:*

- $\ell = c - d$ *correct processes,*

[13]The pseudo-code presented in Algorithm 1 favors readability and is therefore not fully optimized. For instance, in some cases, a process might unreliably broadcast exactly the same content at lines 7 and 10. This could be avoided by either using an appropriate flag or by tracking and preventing the repeated broadcast of identical BUNDLE imp-messages.

- $\lambda = \left\{ \begin{array}{ll} 2 & \textit{if } d < \frac{c - \lfloor \frac{n+t}{2} \rfloor}{\lfloor \frac{n+t}{2} \rfloor + 1} \\ 3 & \textit{if } d < c - \sqrt{c \times \frac{n+t}{2}} \\ > 3 & \textit{otherwise} \end{array} \right\}$ *communication steps,*
- $\mu = 2n^2$ *imp-messages.*

The proof follows from the next lemmas.

Lemma 1 (MBRB-Validity)**.** *If a correct process p_i mbrb-delivers m from a correct process p_j with sequence number sn, then p_j has previously mbrb-broadcast m with sequence number sn.*

Proof. If a correct process p_i mbrb-delivers (m, sn, j) (where p_j is correct) at line 11, then it has passed the condition at line 3, which means that it must have witnessed a valid signature for (m, sn, j) by p_j. Since signatures are secure, the only way to create this signature is for p_j to execute the instruction at line 1, during the mbrb_broadcast(m, sn) invocation. □

Lemma 2 (MBRB-No-duplication)**.** *A correct process p_i mbrb-delivers at most one app-message from a process p_j with a given sequence number sn.*

Proof. This property derives trivially from the condition at line 3. □

Lemma 3 (MBRB-No-duplicity)**.** *No two different correct processes mbrb-deliver different app-messages from a process p_i with the same sequence number sn.*

Proof. Let us consider two correct processes p_a and p_b which respectively mbrb-deliver (m, sn, i) and (m', sn, i). Due to the condition at line 9, p_a and p_b must have saved (and thus received) two sets Q_a and Q_b containing strictly more than $\frac{n+t}{2}$ signatures for (m, sn, i) and (m', sn, i), respectively. We thus have $|Q_a| > \frac{n+t}{2}$ and $|Q_b| > \frac{n+t}{2}$.

As we have $|A \cap B| = |A| + |B| - |A \cup B| \geq |A| + |B| - n > 2 \times \frac{n+t}{2} - n = t$, A and B have at least one correct process p_k in common, which must have signed both (m, sn, i) and (m', sn, i). But before signing (m, sn, i) at line 1 or 6, p_k checks that it did not sign a different app-message from the same sender and with the same sequence number, whether it be implicitly during a brb_broadcast(m, sn) invocation or at line 5. Thereby, m is necessarily equal to m'. □

Lemma 4 (MBRB-Local-delivery)**.** *If a correct process p_i mbrb-broadcasts an app-message m with sequence number sn, then at least one correct process p_j mbrb-delivers m from p_i with sequence number sn.*

Proof. If a correct process p_i mbrb-broadcasts (m, sn), then it broadcasts its own signature sig_i for (m, sn, i) in a BUNDLE$(m, sn, i, \{sig_i\})$ message at line 2. As p_i is correct, it does not sign another triplet (m', sn, i) where $m' \neq m$, therefore it is impossible for a correct process to mbrb-deliver (m', sn, i) at line 11, because it cannot pass the condition at line 3.

Let us denote by K the set of correct processes that receive a message BUNDLE$(m, sn, i, \{sig_i, ...\})$ at least once. Note that because p_i executes line 2, by definition of the message adversary, K contains at least $c - d$ processes. The assumption $n > 3t + 2d$ further yields that $c - d > 2t + d \geq 0$,

and therefore $K \neq \emptyset$. The first one of such BUNDLE messages that a process of K receives can be the one p_i initially broadcast at line 2, but it can also be a BUNDLE message broadcast by a correct process at lines 7 or 10, or it can even be a BUNDLE message sent by a Byzantine process. In any case, the first time the processes of K receive such a BUNDLE message, they pass the condition at line 3, and they also pass the condition at line 5, except for p_i if it belongs to K. Consequently, each process p_k of K necessarily broadcasts its own signature sig_k for (m, sn, i) in a BUNDLE$(m, sn, i, \{sig_k, sig_i, ...\})$ message.

By construction of the algorithm, the set K of correct processes that ever receive a BUNDLE$(m, sn, i, \{sig_i, ...\})$ message is therefore included into the set K' of correct processes p_k that ever broadcast a BUNDLE$(m, sn, i, \{sig_k, sig_i, ...\})$, $K \subseteq K'$. This inclusion in turn implies

$$|K| \leq |K'|. \tag{1}$$

By the definition of the message adversary, a message BUNDLE$(m, sn, i, \{sig_k, sig_i, ...\})$ broadcast by a correct process $p_k \in K'$ is eventually received by at least $c - d$ correct processes. Because K is the set of processes that ever receives sig_i in a BUNDLE message, these $c - d$ correct processes belong to K by construction. Hence, the minimum number of signatures for (m, sn, i) made by processes of K' that are also received by processes of K globally is $|K'|(c - d)$. Using $K \neq \emptyset$ and therefore $|K| \neq 0$, it follows that a given process of K individually receives on average the distinct signatures of at least $|K'|(c - d)/|K|$ processes of K'.

Using Equation (1) yields $|K'|(c - d)/|K| \geq c - d$. From mbrb-Assumption, we have $n > 3t + 2d \iff 2n > n + 3t + 2d \iff 2n - 2t - 2d > n + t \implies c - d \geq n - t - d > \frac{n+t}{2}$ (as $n - t \leq c$). As a result, by the pigeonhole principle, at least one process p_j of K (ergo one correct process) receives a set S (in possibly multiple BUNDLE messages) of strictly more than $\frac{n+t}{2}$ valid distinct signatures for (m, sn, i). When p_j receives the last signature of S, there are two cases:

- Case if p_j does not pass the condition at line 3.

  As processes of K are correct, then when they broadcast a BUNDLE$(m, sn, i, sigs)$ message, they necessarily include sig_i in $sigs$, which implies that sig_i is necessarily in S. Therefore, if p_j does not pass the condition at line 3, it is because p_j already mbrb-delivered some $(-, sn, i)$. But let us remind that, as p_i is correct, it is impossible for p_j to mbrb-deliver anything different from (m, sn, i). Therefore, p_j has already mbrb-delivered (m, sn, i).

- Case if p_j passes the condition at line 3.

  Process p_j then saves all signatures of S at line 4, and after it passes the condition at line 9 (as $|S| > \frac{n+t}{2}$) and finally mbrb-delivers (m, sn, i) at line 11. □

Lemma 5 (MBRB-Global-delivery)**.** *If a correct process p_i mbrb-delivers an app-message m from p_j with sequence number sn, then at least $\ell = c - d$ correct processes mbrb-deliver m from p_j with sequence number sn.*

Proof. If a correct process p_i mbrb-delivers (m, sn, j) at line 11, it must have saved a set $sigs$ of strictly more than $\frac{n+t}{2}$ valid distinct signatures because of the condition at line 9. Let us remark that $sigs$ necessarily contains the signature for (m, sn, i) by p_i because of the condition at line 3.

Additionally, p_i must also have broadcast BUNDLE$(m, sn, i, sigs)$ at line 10, that, by definition of the message adversary, is received by a set K of at least $c - d$ correct processes. For each process p_k of K:

- If p_k does not pass the condition at line 3, it is necessarily because it has already mbrb-delivered some $(-, sn, j)$ at line 11. But because of MBRB-No-duplicity, p_k has necessarily mbrb-delivered (m, sn, j).
- If p_k passes the condition at line 3, then it saves all signatures of $sigs$ at line 4 and then passes the condition at line 9 and finally mbrb-delivers (m, sn, j) at line 11.

Therefore, all processes of K (which, as a reminder, are at least $c - d = \ell$) necessarily mbrb-deliver (m, sn, j) at line 11. □

Remark. Neither Lemmas 1, 2, 3 or 5 use the assumption $n > 3t + 2d$ (mbrb-Assumption). As a result, if Algorithm 1 is used in a situation when the message adversary can partition the system ($n \leq 2d$), the safety properties of the algorithm (MBRB-Validity, MBRB-No-duplication, and MBRB-No-duplicity) and the MBRB-Global-delivery property continue to hold. In case of partition, however, the MBRB-Local-delivery property might get violated, as an application message mbrb-broadcast by a correct process might fail to gather a quorum of signatures at line 9 to trigger mbrb-delivery at line 11.

Lemma 6. $c - d > \left\lfloor \frac{n+t}{2} \right\rfloor$.

Proof. We have the following:

$$\begin{aligned} c - d \geq n - t - d &= \frac{2n - 2t - 2d}{2}, && \text{(by definition of } c\text{)} \\ &> \frac{n + 3t + 2d - 2t - 2d}{2}, && \text{(by mbrb-Assumption)} \\ &> \frac{n+t}{2} \geq \left\lfloor \frac{n+t}{2} \right\rfloor. && \square \end{aligned}$$

Lemma 7. *If a correct process p_i mbrb-broadcasts (m, sn), then at least $c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c-d-\lfloor \frac{n+t}{2} \rfloor} \right\rfloor$ correct processes mbrb-deliver (m, sn, i) at most two communication steps later.*

Proof. If a correct process p_i mbrb-broadcasts (m, sn), then it broadcasts its own signature sig_i for (m, sn, i) in a BUNDLE$(m, sn, i, \{sig_i\})$ imp-message at line 2. Let us denote by K the set of correct processes that receive this BUNDLE$(m, sn, i, \{sig_i\})$ imp-message from p_i during the same communication step, and let k be the number of processes in K, such that $c - d \leq k = |K| \leq c$ (by definition of the message adversary). By construction of the algorithm, every process p_x of K passes the condition at line 3, and therefore broadcasts a BUNDLE$(m, sn, i, \{sig_x, sig_i\})$ imp-message, whether it be at line 2 for p_i, or at line 7 for any other process of K.

Let A and B define two partitions of the set of all correct processes ($A \cup B$ is the set of all correct processes, and $A \cap B = \varnothing$). A denotes the set of correct processes that receive strictly

more than $\frac{n+t}{2}$ signatures for (m, sn, i) from processes of K two communication steps after p_i mbrb-broadcast (m, sn), while B denotes the set of remaining correct processes of K that receive at most $\frac{n+t}{2}$ signatures for (m, sn, i) from processes of K two communication steps after p_i mbrb-broadcast (m, sn). Let ℓ_2 be the size of A: $\ell_2 = |A|$. By construction, $|B| = c - \ell_2$. Let s_A and s_B respectively denote the number of signatures for (m, sn, i) from processes of K received by processes of A and B at most two communication steps after p_i mbrb-broadcast (m, sn). Figure 1 represents the distribution of such signatures among processes of K, sorted by decreasing number of signatures received. Each processes of A can receive at most k signatures (that is, all signatures) from processes of K, while each process of B can receive at most $\lfloor \frac{n+t}{2} \rfloor$ signatures from processes of K two communication steps after p_i mbrb-broadcasts (m, sn). For the sake of simplicity, we use q in the place of $\lfloor \frac{n+t}{2} \rfloor$ in some parts of this proof.

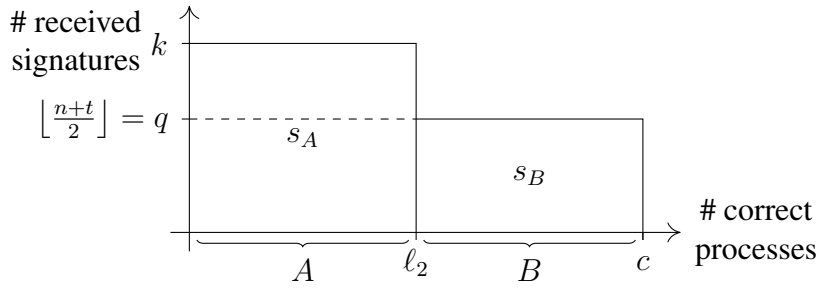

Figure 1: Distribution of signatures among processes of A and B two communication steps after p_i mbrb-broadcast (m, sn)

From these observations, we infer the following inequalities:

$$\ell_2 k \geq s_A,$$
$$(c - \ell_2)q \geq s_B.$$

By the definition of the message adversary, a BUNDLE$(m, sn, i, \{sig_x, sig_i\})$ imp-message broadcast by a correct process p_x is eventually received by at least $c - d$ correct processes. As a consequence, in total, the minimum number of signatures for (m, sn, i) collectively received by correct processes as a result of broadcasts by processes in K in the first two asynchronous communication steps is $k(c - d)$. We thus have:

$$s_A + s_B \geq k(c - d).$$

By combining the previous inequalities, we obtain:

$$\begin{aligned} \ell_2 k + (c - \ell_2)q &\geq k(c - d), \\ \ell_2 k + cq - \ell_2 q &\geq k(c - d), \\ \ell_2 k - \ell_2 q &\geq k(c - d) - cq, \\ \ell_2(k - q) &\geq k(c - d) - cq. \end{aligned} \tag{2}$$

By Lemma 6, we know that $k \geq c - d > \left\lfloor \frac{n+t}{2} \right\rfloor = q$, so we can rewrite (2) into:

$$\ell_2 \geq \frac{k(c-d) - cq}{k-q}. \tag{3}$$

Let us define a function f such that $f(k) = \frac{k(c-d)-cq}{k-q}$. As we seek the lowest guaranteed value for ℓ_2, we want to find the minimum of f on $k \in [c-d, c]$. To this end, let us first study the derivative of f. The image $f(k)$ is of the form $\frac{u}{v}$, so we have:

$$\begin{aligned} f'(k) &= \frac{u'v - uv'}{v^2} = \frac{(c-d)(k-q) - (k(c-d) - qc)}{(k-q)^2}, \\ &= \frac{(c-d)(k-q) - k(c-d) + qc}{(k-q)^2} = \frac{qc - q(c-d)}{(k-q)^2} = \frac{qd}{(k-q)^2}. \end{aligned}$$

As q and d are by definition positive, we know that $f'(k) = \frac{qd}{(k-q)^2}$ is positive, or null when $d = 0$. Therefore, f is monotonically increasing on $k \in [c-d, c]$, and the minimum value for ℓ_2 can be found when k is also minimum, that is when $k = c - d$. Thus, when we replace k by $c - d$ in (3), we obtain:

$$\begin{aligned} \ell_2 &\geq \frac{(c-d)(c-d) - cq}{c-d-q} = \frac{(c-d)(c-d-q) - qd}{c-d-q}, \\ &\geq c - d - \frac{qd}{c-d-q}. \end{aligned} \tag{4}$$

Let us denote by $\ell_{2,\mathsf{min}}$ the minimum number of correct processes that receive a quorum of strictly more than $\frac{n+t}{2}$ valid distinct signatures for (m, sn, i) two communication steps after p_i mbrb-broadcast (m, sn), such that $\ell_{2,\mathsf{min}} \leq \ell_2 = |A|$. As the right hand side of (4) is not always an integer, we have:

$$\begin{aligned} \ell_{2,\mathsf{min}} &= \left\lceil c - d - \frac{qd}{c-d-q} \right\rceil = c - d + \left\lceil - \frac{qd}{c-d-q} \right\rceil, \\ &= c - d - \left\lfloor \frac{qd}{c-d-q} \right\rfloor, && (\text{as } \forall\ x \in \mathbb{R}, \lceil -x \rceil = -\lfloor x \rfloor) \\ &= c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c-d-\lfloor \frac{n+t}{2} \rfloor} \right\rfloor. && (\text{by definition of } q) \end{aligned}$$

Hence, at least $\ell_{2,\mathsf{min}} = c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c-d-\lfloor \frac{n+t}{2} \rfloor} \right\rfloor$ processes of K receive strictly more than $\frac{n+t}{2}$ valid distinct signatures for (m, sn, i) two communication steps after p_i mbrb-broadcasts (m, sn). For every process p_a of A:

- If p_a does not pass the condition at line 3 after receiving the last signature of the quorum in a BUNDLE imp-message, it is necessarily because p_a already mbrb-delivered some $(-, sn, i)$, because processes of K are correct and all their BUNDLE imp-messages include the signature for (m, sn, i) by p_i. But let us remind that, as the sender p_i is correct, it is impossible for p_a to mbrb-deliver anything different from (m, sn, i). Therefore, p_a has already mbrb-delivered (m, sn, i) at line 11.

- If p_a passes the condition at line 3 after processing the last BUNDLE$(m, sn, i, \{sig_i, sig_x\})$ imp-message of the quorum from a process p_x, then p_a saves the signature sig_x at line 4, and after it passes the condition at line 9 (as it has saved strictly more than $\frac{n+t}{2}$ signatures) and finally mbrb-delivers (m, sn, i) at line 11.

Therefore, all processes of A, which are at least $\ell_{2,\mathsf{min}} = c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c-d-\lfloor \frac{n+t}{2} \rfloor} \right\rfloor$, mbrb-deliver (m, sn, i) at line 11 at most two communication steps after p_i mbrb-broadcast (m, sn). □

Lemma 8. *If a correct process p_i mbrb-broadcasts (m, sn) and $d < c - \sqrt{c \times \frac{n+t}{2}}$, then at least $c - d$ correct processes mbrb-deliver (m, sn, i) at most three communication steps later.*

Proof. Let us assume that a correct process p_i mbrb-broadcasts (m, sn) and that $d < c - \sqrt{c \times \frac{n+t}{2}}$. Process p_i must unreliably broadcast a first BUNDLE$(m, sn, i, \{sig_i\})$ imp-message (where sig_i is the signature of (m, sn, i) by p_i) at line 2. This initial imp-message is received by at least $(c-d-1)$ other correct processes, due to our assumption on the message adversary. This counts as a first communication step.

In the second communication step, each process p_j of these $(c - d - 1)$ correct processes unreliably broadcasts its own BUNDLE$(m, sn, i, \{sig_j, sig_i\})$ imp-message (where sig_j is the signature of (m, sn, i) by p_j) at line 7. At the end of the second communication step, in total, at least $(c - d)$ distinct signatures for (m, sn, i) have been created and unreliably broadcast by correct processes (counting that of p_i), resulting in at least $(c-d)^2$ receptions of said signatures by correct processes. As there are c correct processes, this means that, on average, each correct process has received at least $\frac{(c-d)^2}{c}$ signatures by the end of the second communication step, and that at least one correct process, p_k, receives (and saves at line 4) at least this number of signatures.

From the Lemma hypothesis $d < c - \sqrt{c \times \frac{n+t}{2}}$ and using simple algebraic transformations, we can derive $\frac{(c-d)^2}{c} > \frac{n+t}{2}$. Therefore, p_k reaches a quorum of signatures, that is, it passes the condition at line 9 and unreliably broadcast this quorum of signatures at line 10, two communication steps after the mbrb-broadcast of (m, sn) by p_i. By definition of the message adversary, this quorum of signatures is received by $c - d$ correct processes, which save it at line 4 and thus pass the condition at line 9 and finally mbrb-deliver (m, sn, i) at line 11, three communication steps after the mbrb-broadcast of (m, sn) by p_i. □

Lemma 9 (MBRB-Time-cost)**.** *If a correct process p_i mbrb-broadcasts an app-message m with sequence number sn, then $\ell = c - d$ correct processes mbrb-deliver m from p_i with sequence number sn at most*

$$\lambda = \left\{ \begin{array}{ll} 2 & \textit{if } d < \frac{c-\lfloor \frac{n+t}{2} \rfloor}{\lfloor \frac{n+t}{2} \rfloor + 1} \\ 3 & \textit{if } d < c - \sqrt{c \times \frac{n+t}{2}} \\ > 3 & \textit{otherwise} \end{array} \right\} \textit{communication steps later.}$$

Proof. Let us consider a correct process p_i that mbrb-broadcasts (m, sn). By exhaustion:

- Case where $d < \frac{c-\lfloor \frac{n+t}{2} \rfloor}{\lfloor \frac{n+t}{2} \rfloor + 1}$.

  By Lemma 7, at least $c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c-d-\lfloor \frac{n+t}{2} \rfloor} \right\rfloor$ correct processes mbrb-deliver (m, sn, i) two communication steps after p_i has mbrb-broadcast (m, sn). We have:

$$
\begin{aligned}
d &< \frac{c - \lfloor \frac{n+t}{2} \rfloor}{\lfloor \frac{n+t}{2} \rfloor + 1}, && \text{(case assumption)} \\
d\left\lfloor \frac{n+t}{2} \right\rfloor + d &< c - \left\lfloor \frac{n+t}{2} \right\rfloor, && \text{(as } \lfloor \tfrac{n+t}{2} \rfloor + 1 > 0\text{)} \\
d\left\lfloor \frac{n+t}{2} \right\rfloor &< c - d - \left\lfloor \frac{n+t}{2} \right\rfloor, \\
\frac{d\lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} &< 1, && \text{(as } c - d > \lfloor \tfrac{n+t}{2} \rfloor \text{ by Lemma 6)} \\
\left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor &\leq 0, \\
c - d - \left\lfloor \frac{d\lfloor \frac{n+t}{2} \rfloor}{c - d - \lfloor \frac{n+t}{2} \rfloor} \right\rfloor &\geq c - d = \ell.
\end{aligned}
$$

  Hence, ℓ correct processes mbrb-deliver (m, sn, i) at most two communication steps after p_i has mbrb-broadcast (m, sn).

- Case where $d < c - \sqrt{c \times \frac{n+t}{2}}$.

  Lemma 8 applies and at least $c - d = \ell$ correct processes mbrb-deliver (m, sn, i) at most three communication steps after p_i has mbrb-broadcast (m, sn). □

Lemma 10 (MBRB-Message-cost)**.** *The mbrb-broadcast of an app-message by a correct process p_i entails the sending of at most $\mu = 2n^2$ imp-messages by correct processes.*

Proof. The broadcast of an imp-message by a correct process at line 2 entails its forwarding by at most $n - 1$ other correct processes at line 7. As each broadcast by correct processes corresponds to the sending of n imp-messages, then at most n^2 imp-messages are sent in a first step.

In a second step, at least one correct process reaches a quorum of signatures and passes the condition at line 9, and then broadcasts this quorum of signatures at line 10. Upon receiving this quorum, every correct process also passes the condition at line 9 (if it has not done it already) and broadcasts the imp-message containing the quorum at line 10. Hence, at most n^2 imp-messages are also sent in this second step, which amounts to a maximum of $\mu = 2n^2$ imp-messages sent in total. □

An additional property. The reader can check from the previous proofs that the algorithm satisfies the following MBRB-delivery property. If there is a set K of k correct processes, $1 \leq k \leq d$, such that there is a finite time τ after which the message adversary never eliminates the imp-messages sent to them, then, after τ, each process of K mbrb-delivers all the app-messages mbrb-broadcast by correct processes.

4. A Tightness Bound

Definition. An algorithm implementing a broadcast communication abstraction is *event-driven* if, as far as the correct processes are concerned, only (i) the invocation of the broadcast operation that is provided to the application by the broadcast communication abstraction, or (ii) the reception of an imp-message—sent by a correct or a Byzantine process—can generate the sending of imp-messages (using the underlying unreliable network-level broadcast operation).

Theorem 2 (MBRB-Necessary-condition)**.** *When $n \leq 3t+2d$, there is no event-driven (signature-free or signature-based) algorithm implementing the* MBRB *communication abstraction on top of an n-process asynchronous system in which up to t processes may be Byzantine and where a message adversary may suppress up to d copies of each imp-message broadcast by a correct process.*[14]

Proof. Without loss of generality, the proof considers the case $n = 3t + 2d$. Let us partition the n processes into five sets Q_1, Q_2, Q_3, D_1, and D_2, such that $|D_1| = |D_2| = d$ and $|Q_1| = |Q_2| = |Q_3| = t$.[15] So, when considering the sets Q_1, Q_2, and Q_3, there are executions in which all the processes of either Q_1 or Q_2 or Q_3 can be Byzantine, while the processes of the two other sets are not.

The proof is by contradiction. So, assuming that there is an event-driven algorithm A that builds the MBRB-broadcast abstraction for $n = 3t + 2d$, let us consider an execution E of A in which the processes of Q_1, Q_2, D_1, and Q_2 are not Byzantine while all the processes of Q_3 are Byzantine.

Let us observe that the message adversary can isolate up to d processes by preventing them from receiving any imp-messages. Without loss of generality, let us assume that the adversary isolates a set of d correct processes not containing the sender of the app-message. As A is event-driven, these d isolated processes do not send imp-messages during the execution E of A. As a result, no correct process can expect imp-messages from more than $(n - t - d)$ different processes without risking being blocked forever. Thanks to the mbrb-Assumption $n = 3t + 2d$, this translates as "no correct process can expect imp-messages from more than $(2t + d)$ different processes without risking being blocked forever".

In the execution E, the (Byzantine) processes of Q_3 simulate the mbrb-broadcast of an app-message such that this app-message appears as being mbrb-broadcast by one of them and is mbrb-delivered as the app-message m to the processes of Q_1 (hence the processes of Q_3 appear, to the processes of Q_1, as if they were correct) and as the app-message $m' \neq m$ to the processes of Q_2 (hence, similarly to the previous case, the processes of Q_3 appear to the processes of Q_2 as if they were correct). Let us call m-messages (resp., m'-messages) the imp-messages generated by the event-driven algorithm A that entails the mbrb-delivery of m (resp., m'). Moreover, the execution E is such that:

[14] Let us recall that the underlying communication operation offered by the system is an unreliable broadcast defined in Section 2.1.

[15] For the case $n < 3t + 2d$, the partition is such that $\min(|Q_1|, |D_2|) \leq d$ and $\min(|Q_1|, |Q_2|, |Q_3|) \leq t$.

- concerning the m-messages: the message adversary suppresses all the m-messages sent to the processes of D_2, and asynchrony delays the reception of all the m-messages sent to Q_2 until some time τ defined below.[16] So, as $|Q_1 \cup D_1 \cup Q_3| = n - t - d = 2t + d$, Algorithm A will cause the processes of Q_1 and D_1 to mbrb-deliver m.[17]

- concerning the m'-messages: the message adversary suppresses all the m'-messages sent to the processes of D_1, and the asynchrony delays the reception of all the m'-messages sent to Q_1 until time τ. As previously, as $|Q_2 \cup D_2 \cup Q_3| = n - t - d = 2t + d$, Algorithm A will cause the processes of Q_2 and D_2 to mbrb-deliver m'.

- Finally, the time τ occurs after the mbrb-delivery of m by the processes of D_1 and Q_1, and after the mbrb-delivery of m' by the processes of D_2 and Q_2.

It follows that different non-Byzantine processes mbrb-deliver different app-messages for the same mbrb-broadcast (or a fraudulent simulation of it) issued by a Byzantine process (with possibly the help of other Byzantine processes). This contradicts the MBRB-No-Duplicity property, which concludes the proof of the theorem. □

Theorem 3 (Algorithm optimality)**.** *Considering an asynchronous n-process system in which up to t processes can be Byzantine and where a d-message adversary can suppress imp-messages, Algorithm* 1 *is optimal with respect to the pair of values $\langle t, d\rangle$.*

Proof. Theorem 2 has shown that the condition $n > 3t + 2d$ is necessary, while Algorithm 1 has shown that this condition is sufficient (Theorem 1). □

5. Conclusion

This article has presented a new communication abstraction (denoted MBRB) that extends Byzantine reliable broadcast (as defined by Bracha and Toueg [8, 9]) to systems where, at the underlying implementation level, an adversary may suppress some subset of implementation messages used by the processes to co-operate. From a practical point of view, this kind of message loss captures phenomena such as silent churn, input disconnection, etc. A signature-based algorithm implementing the corresponding Byzantine-tolerant reliable broadcast in the presence of a message adversary has been presented and proven correct. This algorithm assumes $n > 3t + 2d$ (where n is the number of processes, t is the maximum number of Byzantine processes, and d is an upper bound on the power of the message adversary), which has been shown to be a necessary and sufficient condition.

When there is no message adversary, this algorithm is optimal both in terms of Byzantine resilience and the number of communication steps. These properties are also satisfied in other

[16] Equivalently, we could also say that asynchrony delays the reception of all the m-messages sent to $D_2 \cup Q_2$ until time τ. The important point is here that, due to the assumed existence of Algorithm A, the processes of Q_1 and D_1 mbrb-deliver m with m-messages from at most $2t + d$ different processes.

[17] Let us notice that this is independent from the fact that the processes in Q_3 are Byzantine or not.

circumstances, including a message adversary whose power d is restricted to some well-defined threshold.

An intriguing question is whether the MBRB abstraction can be implemented in an unauthenticated setting (i.e., without signatures) while only assuming authenticated channels. A follow-up work [4] answers this question positively and shows how two existing unauthenticated Byzantine-tolerant reliable broadcast (BRB) algorithms, that of Bracha [8] and Imbs-Raynal [19], can be transformed into unauthenticated MBRB algorithms (i.e., made resistant to a message adversary without using signatures) by relying on a novel many-to-many abstraction called $k2\ell$-cast. However, the two resulting algorithms require stronger assumptions on n than $n > 3t + 2d$ and provide weaker delivery guarantees than $\ell = c - d$ (which is optimal). Whether an unauthenticated MBRB algorithm that guarantees delivery to at least $\ell = c - d$ correct processes can be implemented when $n = 3t + 2d + 1$ (such an algorithm would be optimal both in terms of resilience and delivery power), or whether the MBRB abstraction is subject to stronger bounds in an authenticated setting, remain to this date open questions.

Acknowledgments

This work was partially supported by the French ANR projects ByBloS (ANR-20-CE25-0002-01) and PriCLeSS (ANR-10-LABX-07-81) devoted to the modular design of building blocks for large-scale Byzantine-tolerant multi-users applications. The authors want to thank Colette Johnen, Elad Schiller, and Stefan Schmid for their kind invitation to participate in the SSS 2021 conference. The authors would also like to thank the anonymous reviewers whose careful reading and suggestions helped them improve their paper.

References

[1] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of Byzantine broadcast: a complete categorization. In *Proc. 40th ACM Symposium on Principles of Distributed Computing (PODC'21)*, pages 331–341. ACM, 2021.

[2] Yehuda Afek and Eli Gafni. Asynchrony from synchrony. In *Proc. 14th Int'l Conference on Distributed Computing and Networking (ICDCN'13)*, volume 7730 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2013.

[3] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. Byzantine-tolerant reliable broadcast in the presence of silent churn (invited talk). In *Proc. 23rd Int'l Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS'21)*, volume 13046 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2021.

[4] Timothé Albouy, Davide Frey, Michel Raynal, and François Taïani. A modular approach to construct signature-free BRB algorithms under a message adversary. In *Proc. 26th Int'l Conference on Principles of Distributed Systems (OPODIS'22)*, volume 253 of *LIPIcs*, pages 26:1–26:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[5] Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani. Money transfer made simple: a specification, a generic algorithm, and its proof. *Bull. EATCS*, 132, 2020.

[6] Martin Biely, Ulrich Schmid, and Bettina Weiss. Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.*, 412(40):5602–5630, 2011.

[7] Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. Practical Byzantine reliable broadcast on partially connected networks. In *Proc. 41st IEEE Int'l Conference on Distributed Computing Systems (ICDCS'21)*, pages 506–516. IEEE, 2021.

[8] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[9] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[10] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2nd edition, 2011.

[11] Bernadette Charron-Bost and André Schiper. The heard-of model: Computing in distributed systems with benign faults. *Distributed Comput.*, 22(1):49–71, 2009.

[12] Shir Cohen and Idit Keidar. Tame the wild with Byzantine linearizability: Reliable broadcast, snapshots, and asset transfer. In *Proc. 35th Int'l Symposium on Distributed Computing (DISC'21)*, volume 209 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[13] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *Proc. 50th IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN'20)*, pages 26–38. IEEE, 2020.

[14] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus. In *Proc. 17th European Conference on Computer Systems (EuroSys'22)*, pages 34–50. ACM, 2022.

[15] Danny Dolev. The Byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

[16] Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic Byzantine reliable broadcast. In *Proc. 24th Int'l Conference on Principles of Distributed Systems (OPODIS'20)*, volume 184 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[17] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. Scalable Byzantine reliable broadcast. In *Proc. 33rd Int'l Symposium on Distributed Computing (DISC'19)*, volume 146 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[18] Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. Multi-threshold asynchronous reliable broadcast and consensus. In *Proc. 24th Int'l Conference on Principles of Distributed Systems (OPODIS'20)*, volume 184 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[19] Damien Imbs and Michel Raynal. Trading off t-resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Process. Lett.*, 26(4):1650017:1–1650017:8, 2016.

[20] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 513–522. ACM, 2010.

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

[22] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved extension protocols for Byzantine broadcast and agreement. In *Proc. 34th Int'l Symposium on Distributed Computing (DISC'2020)*, volume 179 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[23] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[24] Michel Raynal. Message adversaries. In *Encyclopedia of Algorithms*, pages 1272–1276. Springer, 2016.

[25] Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018.

[26] Michel Raynal. On the versatility of Bracha's Byzantine reliable broadcast algorithm. *Parallel Process. Lett.*, 31(3):2150006:1–2150006:9, 2021.

[27] Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *Proc. 32th ACM Symposium on Principles of Distributed Computing (PODC'13)*, pages 166–175. ACM, 2013.

[28] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proc. 6th Symposium on Theoretical Aspects of Computer Science (STACS'89)*, volume 349 of *Lecture Notes in Computer Science*, pages 304–313. Springer, 1989.

[29] Nicola Santoro and Peter Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.*, 384(2-3):232–249, 2007.

[30] Ulrich Schmid and Christof Fetzer. Randomized asynchronous consensus with imperfect communications. In *22nd Symposium on Reliable Distributed Systems (SRDS 2003)*, pages 361–370. IEEE, 2003.

[31] Lewis Tseng, Qinzi Zhang, Saptaparni Kumar, and Yifan Zhang. Exact consensus under global asymmetric Byzantine links. In *Proc. 40th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*, pages 721–731. IEEE, 2020.