On the Versatility of Bracha’s Byzantine Reliable Broadcast Algorithm
Michel Raynal

To cite this version:
Michel Raynal. On the Versatility of Bracha’s Byzantine Reliable Broadcast Algorithm. Parallel Processing Letters, 2021, 31 (03), pp.1-7. 10.1142/S0129626421500067. hal-03347874

HAL Id: hal-03347874
https://inria.hal.science/hal-03347874
Submitted on 17 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On the Versatility of Bracha’s Byzantine Reliable Broadcast Algorithm

Michel Raynal

Univ Rennes, Irisa, INRIA, CNRS, 35042 Rennes Cedex, France, and Department of Computing, Polytechnic University, Hong Kong

Abstract

G. Bracha presented in 1987 a simple and efficient reliable broadcast algorithm for $n$-process asynchronous message-passing systems, which tolerates up to $t < n/3$ Byzantine processes. Following an idea recently introduced by Hirt, Kastrato and Liu-Zhang (OPODIS 2020), instead of considering the upper bound on the number of Byzantine processes ($t$), the present short article considers two types of Byzantine behavior: the ones that can prevent the safety property from being satisfied, and the ones that can prevent the liveness property from being satisfied (a Byzantine process can exhibit only one or both types of failures). This Byzantine differentiated failure model is captured by two associated upper bounds denoted $t_s$ (for safety) and $t_l$ for liveness). The article shows that only the threshold values used in the predicates of Bracha’s algorithm must be modified to obtain an algorithm that works with this differentiated Byzantine failure model.

Keywords: Algorithm, Asynchronous system, Byzantine process, Distributed computing, Fault-tolerance, Liveness property, Message-passing, Reliable broadcast, Safety property, Upper bound.

1 Computing model

Asynchronous processes  This paper consider the classical system model made up of a finite set of $n > 1$ asynchronous sequential processes, namely the set $\{p_1, \ldots, p_n\}$. “Asynchronous” means that each process proceeds at its own speed, which can vary arbitrarily with time, and always remains unknown to the other processes.

Communication network  The processes communicate by exchanging messages through an asynchronous reliable point-to-point network. “Point-to-point” means that there is a bi-directional communication channel between each pair of processes (hence, no process can impersonate another process). “Reliable” means that the network neither lose, duplicate, modify, nor create messages. “Asynchronous” means that, while a message that has been sent is eventually received by its destination process, there is no bound on message transfer delays.

A process $p_i$ sends a message to a process $p_j$ by invoking the primitive “\texttt{send TYPE}(m) \to p_j””, where $\texttt{TYPE}$ is the type of the message and $m$ its content. To simplify the presentation, it is assumed that a process can send messages to itself. A process receives a message by executing the primitive “\texttt{receive}()”. The (non-atomic) macro-operation “\texttt{broadcast TYPE}(m)” is a shortcut for “\texttt{for $j \in \{1, \ldots, n\}$ do send TYPE}(m) \to p_j \texttt{end for}”.
**Failure model** A **Byzantine** process is a process that behaves arbitrarily: it can crash, fail to send or receive messages, send arbitrary messages, start in an arbitrary state, perform arbitrary state transitions, etc. [8, 12]. A process that exhibits a Byzantine behavior is also called *faulty*. A non-Byzantine process is called *correct* or *non-faulty*.

Byzantine processes can collude to “pollute” the computation. They can also control the network in the sense that they can re-order message deliveries at correct processes, and determine the message delay of each message (but neither modify its content nor discard it).

As in [6] we consider a *differentiated* process failure model that distinguishes two types of Byzantine behavior\(^1\): the ones that can prevent the safety property from being satisfied (e.g., sending fake values), and the ones that can prevent the liveness property from being satisfied (e.g., not sending or not receiving a message). The integer \(t_s\) (resp., \(t_\ell\)) denotes the maximal number of processes beyond which the safety (resp., liveness) property cannot be ensured. More precisely, if a Byzantine process exhibits only one type of bad behavior, it counts for “1” in the corresponding bound \(t_s\) or \(t_\ell\). If it exhibits both types of bad behavior it is counted for “1” in both \(t_s\) and \(t_\ell\). In the case where each Byzantine exhibits both types of bad behavior we have \(t_s = t_\ell = t\).

### 2 Byzantine Reliable Broadcast

**Definition** This communication abstraction (in short BRB) provides each process with two operations, denoted \(\text{BRB\_broadcast}()\) and \(\text{BRB\_deliver}()\). As in [5, 13], we use the following terminology: when a process invokes \(\text{BRB\_broadcast}()\), we say that it “brb-broadcasts a message”, and when it executes \(\text{BRB\_deliver}()\), we say that it “brb-delivers a message”. BRB is defined by the following properties.

- **BRB-Validity.** If a correct process brb-delivers a message \(m\) from a correct process \(p_i\), then \(p_i\) brb-broadcast \(m\).
- **BRB-Integrity.** Given a process \(p_i\), a correct process brb-delivers at most one message from \(p_i\).
- **BRB-Termination.** If (i) a correct process \(p_i\) brb-broadcasts a message \(m\), or (ii) a correct process brb-delivers a message \(m\) from a process \(p_j\), then all the correct processes brb-deliver the message \(m\).

**On safety properties** The safety properties are on the nature of the messages that are brb-delivered (hence, they are trivially satisfied if no message is ever delivered). BRB-validity relates outputs (messages brb-delivered) to inputs (the brb-broadcast messages). BRB-integrity states that there is no duplication.

**On liveness properties** The liveness properties are on the fact that messages must be brb-delivered. BRB-Termination (i) states that a message brb-broadcast by a correct process is brb-delivered by all correct processes. BRB-Termination (ii) gives its name to reliable broadcast. Whether \(p_j\) is correct or not, every message brb-delivered from \(p_j\) by a correct process is brb-delivered by all correct processes.

**On agreement** It follows from the BRB-Integrity property and the BRB-Termination (ii) property that all the correct processes BRB-deliver the same set of messages, and this set contains at least all the messages BRB-broadcast by the correct processes.

---

\(^1\)Other approaches differentiating failure types have been proposed since a long time. An example, the types *dormant* and *arbitrary* are introduced in [10], which correspond to omission/timing failures and **Byzantine** behaviors, respectively.
On computability  It is shown in [6] (with slightly different notations) that BRB can be solved in the differentiated Byzantine failure model, if and only if \( n > 2t_\ell + t_s \). Hence, if \( t_\ell = t_s = t \) this condition boils down to \( n > 3t \), which is the bound in the non-differentiated Byzantine failure model [1, 2, 13].

From a message to a sequence of messages  BRB is a one-shot communication abstraction (a correct process invokes \texttt{brb\_broadcast()} at most once). Hence, the BRB instance invoked by \( p_i \) can be identified by the process index \( i \).

A simple way to obtain a multi-shot BRB abstraction (M-BRB) consists in associating a unique tag (e.g., a sequence number) with each invocation of \texttt{brb\_broadcast()}, so that a process \( p_i \) now invokes \texttt{brb\_broadcast}(\( sn_i, m \)), where \( sn_i \) is its current sequence number value. This M-MRB instance is then identified by the pair \( \langle i, sn_i \rangle \). For the M-BRB abstraction, the BRB-Validity and BRB-Integrity properties become:

* M-BRB-Validity. If a correct process \( p_i \)-delivers a message \( m \) from a correct process \( p_i \) with sequence number \( sn \), then \( p_i \)-brb-broadcasts \( m \) with sequence number \( sn \).

* M-BRB-Integrity. Given a sequence number \( sn \) and a process \( p_i \), a correct process \( p_i \)-delivers at most one message \( m \) associated with \( sn \) from \( p_i \).

the same as their BRB counterparts, with the addition that sequence numbers are also preserved.

On BRB algorithms  Bracha’s BRB algorithm [1] implements the BRB abstraction, with optimal \( t \)-resilience (\( t < n/3 \)). Let an application message be a message sent by the BRB abstraction, and a protocol message be a message used to implement BRB. The BRB of an application message by a correct process requires 3 communication steps and \( (n-1)(2n+1) \) protocol messages.

It is natural that, as it is a fundamental communication abstraction, BRB has been addressed by many authors. Here are a few recent results.

* The BRB algorithm presented in [7], implements the \texttt{brb\_broadcast} of an application message with only two communication steps, two message types, and \( n^2 - 1 \) protocol messages. 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 the trade-off \( t \)-resilience versus message/time efficiency.

* Scalable BRB is addressed in [4]. The issue is here not to pay 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 fixed probability-dependent properties.

* A flexible Byzantine fault-tolerant approach is introduced in [9]. This new and very general approach relies on appropriate commit and BRB algorithms.

* BRB in dynamic systems is addressed [3]. 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, the number of Byzantine processes present in the system is less than one third of total number of processes present in the system.

* An efficient algorithm for BRB with long inputs of \( \ell \) bits using lower costs than \( \ell \) single-bit instances is presented in [11]. This algorithm, which assumes \( t < n/3 \), achieves the best possible communication complexity of \( \Theta(n\ell) \) input sizes. This article also presents an authenticated extension of the previous algorithm.
3 A Generic View of Bracha’s Algorithm

Skeleton of Bracha’s algorithm The algorithm proposed by Bracha in [1] to implement BRB is based on a “double echo” mechanism of the message brb-broadcast by the sender process. The corresponding algorithm skeleton is described in Fig. 1, where the values $\alpha$, $\beta$, and $\gamma$ (which appear inside squares to make them more visible) are specific integer values.

When a process $p_j$ invokes `brb_broadcast(m)` it sends the protocol message `INIT(j, m)` to all (line 1). When a process $p_i$ receives a message `INIT(j, m)`, $p_i$ echoes it by sending to all the protocol message `ECHO(j, m)` (line 2). When $p_i$ has received “enough” `ECHO(j, m)` messages (namely $\alpha$), $p_i$ echoes it for the second time (message `READY(j, m)` sent at line 4). Then, when $p_i$ has received “enough” `READY(j, m)` messages (namely $\beta$), it participates to second echo if not yet done (lines 6-8). The aim of this second echo is to prevent a possible blocking. Finally, when $p_i$ has received “enough” `READY(j, m)` messages (namely $\gamma$), it `brb_deliver` the pair $(j, m)$.

It is easy to see that the BRB of an application message requires three types of protocol messages and three consecutive communication steps (one for each message type) which generates $(n-1)(2n+1)$ protocol messages.

Three instances of Bracha’s algorithm skeleton Table 1 presents three instantiations of Bracha’s algorithm skeleton, each defined by specific values of the integers $\alpha$, $\beta$, and $\gamma$.

The second column corresponds to Bracha’s algorithm which considers the bound $t$ on Byzantine processes, while the last two columns consider the pair of parameters $(t_s, t_\ell)$ as defined in the “Failure model” Section. The necessary and sufficient requirement $n > 2t_s + t_\ell$, which applies to the two rightmost columns, is proved in [6]. As we can see, the values of $\beta$ are the same in all the instantiations. Differently, the values of $\alpha$ and $\gamma$ are the same in the Bracha’s column and the rightmost column, and smaller than the ones of the HKL’s column as soon as $n \geq 3t + x$ (for the Bracha’s column) and $n \geq 2t_\ell + t_s + x$ (for the HKL’s column), for $x \geq 1$. Moreover, in some circumstances, the algorithm described in [6] requires one more communication step than the two other instantiations. Let us also notice that when $t_s = t_\ell = t$, the instantiation of the rightmost column boils down to Bracha’s algorithm.
4 Proof of the Third Instantiation

The reader can find a proof of Bracha’s algorithm in [1, 13], and the proof of Hirt, Kastrato, and Liu-Zhang’s algorithm in [6]. So, only the proof of the last instantiation is presented here.

Preliminary remark In the following \( f \) denotes the actual number of Byzantine processes in a run of the algorithm instantiated with \( \alpha = \lceil \frac{n + t}{2} \rceil + 1, \beta = t_s + 1, \) and \( \gamma = t_s + t + 1 \) (rightmost column of the table). As any subset of the \( f \) Byzantine processes can participate only in \( t_s \) or only in \( t, \) and this is not known in advance (as in [6]), it is assumed that \( f \leq t_s \) and \( f \leq t. \) Moreover, \( n > 2t + t_s. \)

Lemma 1. Let \( f \leq t_s \) and \( n > 2t + t_s. \) If a correct process \( p_i \) brb-delivers the message \( m \) from a correct process \( p_j, \) then \( p_j \) brb-broadcasts \( m. \)

Proof As no correct process forges fake messages, the only cause for a correct process \( p_i \) to brb-deliver a message \( m \) from a correct process \( p_j \) is because \( p_j \) brb-broadcasts \( m \) or because the Byzantine processes forged messages that create the illusion that \( m \) was brb-broadcast by \( p_j. \) We show that the second scenario is not possible.

To this end, let us assume by contradiction that \( p_j \) never issued \( \text{brbroadcast}(m) \) and the pair \(( j, m)\) has been forged by Byzantine processes. As there are at most \( t_s \) Byzantine processes, they can generate at most \( t_s \) fake messages \( \text{ECHO}(j, m), \) and \( t_s \) fake messages \( \text{READY}(j, m). \) So a correct process can receive the fake message \( \text{ECHO}(j, m) \) from at most \( t_s \) Byzantine processes and similarly for the fake message \( \text{READY}(j, m). \)

As \( n > 2t + t_s, \) and \( \alpha = \lceil \frac{n + t}{2} \rceil + 1 \) we have \( \alpha > t_s. \) Hence, a correct process cannot send the message \( \text{READY}(j, m) \) at line 4 (because \( \alpha > t_s), \) and at line 7 (because \( \beta > t_s). \) Finally, as \( \gamma > t_s, \) the brb-delivery predicate of line 9 cannot be satisfied at a correct process \( p_i. \) It follows that, if a correct process \( p_i \) brb-delivers the message \( m \) from a correct process \( p_j, \) then \( p_j \) brb-broadcasts \( m. \) \( \square \)

Lemma 2. A correct process brb-delivers at most one message \( m \) from a process \( p_j. \)

Proof The proof follows directly from the brb-delivery predicate at line 9. \( \square \)

Lemma 3. Let \( f \leq t \) and \( n > 2t + t_s. \) If a correct process brb-broadcasts a message \( m, \) all the correct processes brb-delivers \( m \) from \( p_i. \)

Proof Let \( p_j \) be a correct process that invokes \( \text{brbroadcast}(m). \) It follows that each correct process \( p_i \) receives the message \( \text{INIT}(j, m) \) from \( p_j \) and sends the message \( \text{ECHO}(j, m) \) to all the processes. Hence, each correct process receives the message \( \text{ECHO}(j, m) \) from \( n - f \) correct processes. As \( n - f \geq n - t \geq \lceil \frac{n + t}{2} \rceil + 1 = \alpha, \) each correct process sends \( \text{READY}(j, m) \) to all the processes (line 4). Finally,
as \( n - f \geq n - t_\ell \geq t_\ell + t_s + 1 \), each correct process receives enough messages \( \text{READY}(j, m) \) that allows it to brb-deliver \( m \).

**Lemma 3**  
Let \( f \leq t_\ell \). If a correct process brb-delivers a message \( m \) from a process \( p_j \), all the correct processes brb-delivers \( m \) from \( p_j \).

**Proof**  
Let \( p_i \) be a process that brb-delivers \( m \) from \( p_j \). It follows from line 9 that \( p_i \) received the message \( \text{READY}(j, m) \) from at least \( \gamma = t_\ell + t_s + 1 \) different processes. We consider two cases.

- **Case 1:** \( f \leq t_\ell \leq t_s \). As there are \( f \leq t_\ell \) Byzantine processes, we can conclude from the value of \( \gamma = t_\ell + t_s + 1 \) that at least \( t_s + 1 \) correct processes sent the message \( \text{READY}(j, m) \). It then follows that each correct process receives the message \( \text{READY}(j, m) \) from \( t_s + 1 \) correct processes, and (due to the predicate of line 6) sent this message to all the correct processes. It follows that each correct process receives the message \( \text{READY}(j, m) \) from \( n - f \geq t_\ell + t_s + 1 \) different processes, which allows it to brb-deliver \( m \) from \( p_j \).

- **Case 2:** \( f \leq t_s < t_\ell \). Exchanging \( t_s \) and \( t_\ell \), and observing that we have now \( t_s + 1 < t_\ell + 1 \), the reasoning is the same as in previous case\(^2\).

**Lemma 4**  
Theorem 1. Let \( n > 2t_\ell + t_s \). Given a run, let \( f \) be the number of Byzantine processes. If \( f \leq t_s, t_\ell \), Algorithm 1 instantiated with \( \alpha = \lceil \frac{n + t_s}{2} \rceil + 1 \), \( \beta = t_s + 1 \), and \( \gamma = t_s + t_\ell + 1 \), implements the BRB broadcast abstraction.

**Proof**  
The proof derives from the Lemmas 1 and 2 for the safety properties, and from the Lemmas 3 and 4 for the liveness properties.

**Theorem 1**

5 Conclusion

Considering Bracha’s Byzantine reliable broadcast (BRB) abstraction and the differentiated approach (as introduced in [6, 10]), this article has shown that Bracha’s algorithm is based on a powerful skeleton (made up of predicates associated with message exchange patterns), that has a versatility dimension which allows us to have different instantiations according to the thresholds associated with each message forwarding (echo mechanism). As shown by the last two columns of Table 1 it follows that the proposed instantiation of Bracha’s BRB skeleton is more efficient that the one described in [6] (which may require four sequential communication phases instead of three).

As, due to its very definition, the behavior of a Byzantine process cannot be predicted, a practical approach consists in taking \( t_s = t_\ell = t \). Nevertheless, the differentiated approach, that distinguishes the behaviors that prevent safety from the behaviors that prevent liveness is worth studying because it allows us to better understand the essence of algorithms implementing the BRB abstraction, which is one of the fundamental abstractions of fault-tolerant message-passing distributed computing.

**Acknowledgments**

The author want to thank the reviewers for their constructive comments, which help improve both the presentation and the content of the article.

\(^2\)This comes from the fact that, in this lemma, \( t_s \) and \( t_\ell \) plays a symmetrical role, the important point being that \( f \leq t_s \) and \( f \leq t_\ell \).
This work was partially supported by the French ANR Project (devoted to layered and modular structures in distributed computing) and ByBLoS (devoted to the design of building blocks for large-scale trustless multi-users applications).

References