Reliable Broadcast Despite Mobile Byzantine Faults

Silvia Bonomi
Sapienza University of Rome, Italy

Giovanni Farina
Sapienza University of Rome, Italy

Sébastien Tixeuil
Sorbonne Université, CNRS, LIP6, Institut Universitaire de France, Paris, France

Abstract

We investigate the solvability of the Byzantine Reliable Broadcast and Byzantine Broadcast Channel problems in distributed systems affected by Mobile Byzantine Faults. We show that both problems are not solvable even in one of the most constrained system models for mobile Byzantine faults defined so far. By endowing processes with an additional local failure oracle, we provide a solution to the Byzantine Broadcast Channel problem.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases Byzantine fault-tolerance, Reliable Broadcast, Mobile Byzantine Faults

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.18

Related Version *Full Version*: https://hal.science/hal-04277831 [7]

Funding *Giovanni Farina*: This work was partially supported by the *CBFTFDS project – Concrete Byzantine Fault Tolerance and Forecasting in Distributed Systems* (Bando di Ateneo per la Ricerca 2022, Sapienza University of Rome) and partially funded by the EURASIA project by the Italian MAECI.
Sébastien Tixeuil: This work was partially funded by the ANR projects SAPPORO (ref. 2019-CE25-0005-1) and ESTATE (ref. ANR-16-CE25-0009-03).

1 Introduction

Byzantine Reliable Broadcast (BRB) is a fundamental primitive in fault-tolerant distributed systems ensuring that all correct processes eventually deliver the same message from a defined sender regardless of its correctness. Defined by Bracha [12] as a building block for a Byzantine-tolerant consensus protocol, BRB has been widely adopted and investigated since then, thanks to its ability to prevent arbitrarily (i.e., Byzantine) faulty processes from *equivocating* by sending different messages to different processes. It has been introduced as a *one-shot* primitive that allows a pre-defined process in the system to spread a single message and generalized as a Byzantine Broadcast Channel (BBC) primitive [14] to allow every process to spread an arbitrary number of messages. BRB has been used to construct several fault-tolerant distributed solutions, solving more complex problems such as register abstractions, consensus problems, and distributed ledgers. Thus, it has been analyzed in the literature from various perspectives, such as minimizing bandwidth consumption [2], or latency [20, 1].

A fundamental perspective to consider is the investigation of the feasibility of BRB and BBC when assuming no permanent failures. In this paper, we are interested in analyzing BRB and BBC solvability considering a *dynamic process failure model*, i.e., a model in which every process may potentially fail and recover, causing a potentially continuous change in a process's failure state throughout the system's lifetime. Some examples of systems

27th International Conference on Principles of Distributed Systems (OPODIS 2023).
Editors: Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi; Article No. 18; pp. 18:1–18:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

considering dynamic process failures are crash-recovery systems [28, 5], self-stabilizing systems [15, 16], and Mobile Byzantine tolerant systems [17, 6]. In this work, we consider the *Mobile Byzantine Failure* (MBF) model, in which all processes may alternate between periods of correct behavior and periods of arbitrary behavior (i.e., Byzantine). Indeed, the failure state of processes is governed by an external attacker capable of compromising and controlling a set of processes in the system, and such a set is dynamic. The MBF model captures some of the features of the most frequent attacks targeting distributed systems and related countermeasures, where the process's faults are primarily due to external malicious causes rather than internal misbehavior, and tools such as software rejuvenation techniques [21], intrusion detection systems [23], and trusted execution environments [29] are available.

Despite several fundamental distributed problems have been analyzed in the literature considering the MBF model (i.e., Byzantine agreement [17, 6], approximate Byzantine agreement [32, 9], and registers emulation [8]), to the best of our knowledge the BRB problem has never been studied so far in such settings.

Thus, our objective in this paper is the investigation of BRB and BBC in the presence of MBFs. In particular, our contributions are:

1. we formalize the *Mobile Byzantine Reliable Broadcast* (MBRB) and *Mobile Byzantine Broadcast Channel* (MBBC) as a natural extension of the BRB and BBC specifications to deal with MBFs. Indeed, the standard specifications for BRB and BBC primitives consider a *static failure model*, where every process is either permanently correct or faulty;
2. we prove several impossibility results, mainly showing that MBRB and MBBC cannot be implemented without additional knowledge provided by a powerful oracle reporting about processes' failure state;
3. we introduce such a powerful oracle and provide a protocol for solving MBBC in a synchronous round-based system;
4. we analyze a weaker MBBC specification that can be realized without the oracle.

Let us note that being a natural extension of BRB and BBC primitives, the MBRB and MBBC primitives prevent faulty processes from equivocating, namely from sending different information to different processes, and can be used as building block for other fault-tolerant primitives. For example, MBRB/MBBC primitives can extend mobile Byzantine fault-tolerant register abstractions to support Byzantine clients [8]. Our work not only offers an analysis of a specific problem but also provides several insights for other distributed system problems where the failure state of a process is dynamic and partially or entirely unknown. We consider relatively strong assumptions in our system model, the same as those considered in related work, in order determine fundamental solvability conditions. Relaxation of most of these assumptions has already been partially investigated [11].

The rest of the paper is structured as follows. After reviewing related work on implementations of the BRB primitive and contributions considering mobile Byzantine failures in Section 2, we formalize the system model in Section 3. We introduce the new specifications for the Mobile Byzantine Reliable Broadcast and the Mobile Byzantine Broadcast Channel problems in Section 4. Section 5 presents some impossibilities for the specifications we defined. To overcome some of the identified impossibilities and solve the Mobile Byzantine Broadcast Channel problem, we consider a powerful oracle, we propose a protocol in Section 6, and we analyze a weaker Mobile Byzantine Broadcast Channel specification that is realizable without any oracle in Section 7. Due to space constraints, some of the proofs are delegated to the companion technical report [7].

2 Related Work

The Byzantine Reliable Broadcast (BRB) abstraction has been introduced by Bracha [12] as a building block for a Byzantine-tolerant consensus protocol in a distributed system where at most f processes are permanently arbitrary (Byzantine) faulty. Thanks to its ability to guarantee agreement among correct processes over the set of delivered messages, a BRB primitive has been used as a building block from several fault-tolerant solutions, and has been intensely investigated under several system and failure models, with the final aim of extending its power and optimizing different performance metrics.
Imbs and Raynal [20] proposed a protocol that improves latency (in terms of the number of rounds of message exchanges) compared to Bracha. Guerraoui et al. [19] relaxed the BRB specification, allowing each property to be violated with a fixed and arbitrarily small probability. Backes and Cachin [3] and Raynal [26] discussed extensions of the BRB problem; the former assuming both Byzantine faulty processes and fail-stop failures, the latter distinguishing between two different kinds of Byzantine behaviors, i.e. those attempting to prevent the liveness and those attempting to prevent the safety of the BRB. Recently, Guerraoui et al. [18] and Li et al. [22] extended BRB to distributed systems with dynamic membership: in any given view (i.e. set of participating processes, governed by the processes themselves), the set of Byzantine processes remains the same; however, two consecutive views allow for different sets of Byzantine processes. By contrast, our work considers a static system membership (i.e., a fixed set of processes participating in the protocol) but a dynamic failure model, where Byzantine processes may change (that is, recover, and get Byzantine again) during the *same* view. To the best of our knowledge, all existing BRB protocols that assumed arbitrary process failures, except the aforementioned works by Guerraoui et al. [18] and Li et al. [22], considered a *static failure model* i.e., they assumed that the set of Byzantine processes does not change.

Mobile Byzantine Failure (MBF) models have been introduced to capture various types of faults, such as external attacks, virus infections, or even arbitrary behaviors caused by software bugs, using a single model encompassing detection and rejuvenation capabilities. In all these models, failures are abstracted by an omniscient adversary that can control up to f mobile Byzantine agents. Every agent is located in a process and makes it Byzantine faulty until the omniscient adversary decides to move it to another process. The main differences between existing MBF models are in the power of the omniscient adversary (i.e., when it can move the agents) and in the awareness that every process has about its failure state. Most MBF models considered *round-based computations* and can be classified according to Byzantine mobility constraints: under *constrained mobility* [13] the adversary can move agents only when protocol messages are sent (similarly to how viruses would propagate), while under *unconstrained mobility* [4, 6, 17, 24, 31, 27] agents do not move with messages but rather during specific phases of the round. More in detail, Reischuk [27] considered malicious agents stationary for a given period; Ostrovsky and Yung [24] introduced the notion of mobile viruses and defined the adversary as an entity that can inject and distribute faults; finally, Garay [17], Banu et al. [4], Sasaki et al. [31], and Bonnet et al. [6] considered that processes execute synchronous rounds and mobile agents can move from one process to another in a specific phase of the round, which subsequently affects each process's ability to adhere to the algorithm. As a result, the set of Byzantine faulty processes at any given moment is limited in size; however, its composition may change from one round to the next, and the impact of past compromises may linger if not properly addressed by the protocol. The aforementioned works [17, 4, 31, 6] also differ due to the assumption about the knowledge that processes have about their previous infection. In the Garay model [17], a process can detect its infection after the agent leaves it. Conversely, Sasaki et al. [31] investigated a model where processes cannot detect when agents leave. Finally, Bonnet et al. [6] considered an intermediate setting where not faulty processes control the messages they send (in particular, they send the same message to all destinations, and they do not send spurious information). Bonomi et al. [10, 11] decoupled algorithm rounds from Mobile Byzantine agent movement (*round-free model*). The problems analyzed under MBF models are Byzantine agreement [17, 4, 31, 6], approximate Byzantine agreement [32, 30, 9], and Byzantine-tolerant registers [10, 8, 11]. To the best of our knowledge, no efforts have been made to investigate the BRB problem in the presence of MBFs. All existing works that assume MBFs rely on some kind of best-effort communication subsystem (i.e., no guarantees exist when a process is controlled by a Mobile Byzantine agent), potential equivocations and omissions introduced by faulty processes are directly addressed by the main investigated primitive (e.g., consensus, register). The existence of a BRB primitive can simplify the definition of other mobile Byzantine fault-tolerant primitives, similar to the case of the static failure model [12].

3 System Model

We consider a distributed system composed of a set of n processes $\Pi = \{p_1, p_2 \dots p_n\}$, each associated with a unique identifier.

Processes communicate through message passing. We assume that a process can communicate with any other process through a *reliable*, *authenticated*, *point-to-point link* abstraction [14]. This means that messages sent over such channels cannot be altered, dropped, or duplicated, and the identity of the sender cannot be forged. A reliable authenticated point-to-point link abstraction exposes two operations: *(i)* P2P.send(p_{rcv}, m) which sends the message m to the receiver process p_{rcv}, and *(ii)* P2P.deliver(p_{snd}, m) which notifies the reception of the message m from a sender process p_{snd}.

We measure the time according to a fictional global clock $\mathbb{T}$ (not accessible to processes) spanning over the set of natural numbers $\mathbb{N}$. We refer to the starting time of the system as t_0, the i-th time instant since the beginning of the execution as t_i, and a period of time between time t_b and t_e as $T_{b,e} := [t_b, t_e) : t_b, t_e \in \mathbb{T}; t_b < t_e$.

Each process executes a distributed protocol $\mathcal{P}$ consisting of a set of local algorithms. Each algorithm in $\mathcal{P}$ is represented by a finite state automaton whose transitions correspond to computation and communication steps. A computation step denotes a computation executed locally by a given process, while a communication step denotes the sending or receiving of a message. Computation steps and communication steps are generally called *events*. Each process maintains a set of variables. This set and the current value of those variables denote the *state* of a process.

▶ **Definition 1** (Local Execution History). *A local execution history is an alternating sequence* $s_0, e_0, s_1, e_1, \dots$ *of states and events of a process* p_i*, such that state* s_{j+1} *results from state* s_j *by executing event* e_j*.*

We assume that the local algorithms composing $\mathcal{P}$ are stored in a tamper-proof read-only memory.

Processes may fail and we assume that they are affected by *Mobile Byzantine Failures* (MBF). That is, we assume the existence of an omniscient adversary that controls up to $f > 0$ mobile Byzantine agents and that can “move” such agents from one set of processes to

another. When the adversary places a Byzantine agent on a process p_i, the agent takes control of p_i, letting it behave arbitrarily. For example, p_i may omit to send/receive messages, alter the content of messages, alter its process state regardless of its local algorithm, and execute arbitrary code. However, we assume that the mobile Byzantine agents cannot compromise the code stored in the tamper-proof memory. Thus, when the Byzantine agent leaves p_i, p_i resumes executing its local algorithm correctly (albeit from a possibly corrupted state). We assume that the adversary can move each mobile agent independently of the others. Still, any agent must remain on a process for a period of time lasting at least $\Delta_s \in \mathbb{Q}^+$ (rational positive numbers), i.e., once arrived, an agent compromises a node for at least Δ_s consecutive time units, and when $\Delta_s < 1$ we have that an agent can move multiple times in the same time unit. As an example, if $\Delta_s = 2$ we have that every mobile Byzantine agent must remain on the same process for at least 2 consecutive time units, while $\Delta_s = \frac{1}{2}$ means that the agent may move $\left\lceil \frac{1}{\Delta_s} \right\rceil = 2$ times in a time unit and compromise $\left\lceil \frac{1}{\Delta_s} \right\rceil = 2$ different processes in the same time unit.
Let us note that, in the MBF model, no single process is guaranteed to remain correct forever and we may have processes that alternate between correct and incorrect behavior infinitely often. This fundamental difference from the classical static Byzantine failure model commands to redefine the notion of correct and faulty processes (i.e., *the process failure states*).

▶ **Definition 2** (Faulty process). *A process p_i is said to be* faulty *at time t_k if it is controlled by a mobile Byzantine agent at time t_k. By extension, if at each time between t_b and t_e, process p_i is* faulty*, then p_i is faulty during the period $T_{b,e}$.*

When a process p_i is faulty, it may execute a protocol $\mathcal{P}' \neq \mathcal{P}$, and its local state may be altered arbitrarily.

We denote by $B(t)$ the set of faulty processes at time t and by $B(T_{b,e})$ the set of faulty processes during the whole period $T_{b,e}$ (i.e., $B(T_{b,e}) = \bigcap_i B(t_i)$ for $b \leq i < e$).

▶ **Definition 3** (Correct process). *A process p_i is* correct *when it is not faulty, that is, p_i is correct at time t_k if it is not controlled by a Byzantine agent at time t_k. Similarly, a process p_i is correct in the period $T_{b,e}$ if it remains correct between times t_b and t_e.*

Let us remark that when a process p_i is correct, it executes $\mathcal{P}$ but potentially it may start its execution from a compromised state (due to a previous corruption performed by a mobile Byzantine agent).
We denote by $C(t_k)$ the set of correct processes at time t_k and by $C(T_{b,e})$ the set of correct processes throughout the period $T_{b,e}$ (that is, $C(T_{b,e}) = \bigcap_i C(t_i)$ for $b \leq i < e$).

Note that, due to the mobility of Byzantine agents, every process may potentially alternate between correct and faulty states infinitely often. To this aim, we also introduce the notion of *infinitely often correct processes*:

▶ **Definition 4** (Δ_c-Infinitely often correct process). *Let $\Delta_c \in \mathbb{N}^+$. A process p_i is* Δ_c-infinitely often correct *if, for every time t_j, there exists a following period $T_{b,e}$ lasting at least Δ_c where p_i is correct. Formally: $\forall t_j \in \mathbb{T}$, $\exists t_b, t_e$ such that $t_b > t_j$, $t_e - t_b \geq \Delta_c$, $p_i \in C(T_{b,e})$.*

Informally, the notion of Δ_c-infinitely often correct process captures the possibility that a process is not permanently faulty, but correct for at least Δ_c units of time after mobile Byzantine agents have left it.

In the following, we will consider several alternative settings for our system model:

- **system timing assumptions**: we consider either a *synchronous* (SYNC) or an *asynchronous* (ASYNC) system. When considering a synchronous system, we assume that there is an upper bound on the time required to perform local computation on the processes and an upper bound on the time required by a message to be delivered via a P2P link, both of them known by all processes. In addition, we assume that the computation evolves in sequential synchronous rounds $r_1, r_2, \ldots, r_j, \ldots$. Every round r_j is divided into three phases: *(i) send* where processes transmit messages to their intended receivers, *(ii) receive* where processes collect messages sent during the send phase of the current round, and *(iii) compute* where processes process received messages, and prepare those that need to be sent in the following round. Contrarily, in an asynchronous setting, we are not assuming any upper bound, and the computation progresses as soon as an event is generated by a process.
- **mobile Byzantine agent synchronization**: we consider three different types of mobility with different degrees of synchronization between mobile Byzantine agents. In particular, we will consider movement that are either *synchronized* ($\texttt{S-MOB}^+$), *synchronous* (S-MOB), or *asynchronous* (A-MOB) that abstract MBF models existing in the literature. In the A-MOB model, mobile Byzantine agents move independently and once the movement occurs, the agent remains at the destination node for at least Δ_s, with Δ_s unknown to the processes (see ITU model in [10]). In the S-MOB model, mobile Byzantine agents move independently, and, also in this case, once the movement happens the agent remains on the destination node for at least Δ_s. Unlike the previous case, Δ_s is known to the processes (see the ITB model in [10]). The $\texttt{S-MOB}^+$ model is a particular case of the S-MOB model specific for synchronous systems where the computation evolves in synchronous rounds. Indeed, in this case Δ_s is expressed in terms of round, and mobile Byzantine agents can move only between two consecutive rounds, i.e. after the computation phase of a round r_i and before the send phase of round r_{i+1}[1](see Garay's MBF model [17]). Let us stress that in the $\texttt{S-MOB}^+$ setting every process is either faulty or correct for an entire round. Therefore, for ease of presentation, we say that a process is *faulty or correct in the round* r_k in the $\texttt{S-MOB}^+$ systems and extend the notation of $C(t)$ and $B(t)$ accordingly, that is, with $C(r_k)$ and $B(r_k)$, respectively, referring to the sets of correct and faulty processes in the round r_k. Furthermore, we measure the time with the number of rounds.
- **failure awareness**: we assume that every process p_i is either *aware* or *unaware* about a mobile Byzantine agent moving away from p_i. We abstract this knowledge by introducing two different local oracles that reveal information to process p_i. Specifically, we consider: *basic failure awareness* ($\mathcal{O}_{BFA}$) and *full failure awareness* ($\mathcal{O}_{FFA}$). In the $\mathcal{O}_{BFA}$ case, a process p_i knows when (i.e., in which time unit) a mobile agent moves away from p_i; in the $\mathcal{O}_{FFA}$ case, a processes p_i additionally know when the agent arrived to p_i (i.e., p_i know the entire period $T_{b,e}$ in which it was faulty).

More formally:

▶ **Definition 5** (*Basic Failure Awareness Oracle* $\mathcal{O}_{BFA}$). *If a mobile Byzantine agent leaves from a process p_i at time t_j, then the failure awareness oracle $\mathcal{O}_{BFA}$ generates a* CURED() *event on p_i at time t_{j+1}.*

Observe that $\mathcal{O}_{BFA}$ informs p_i as soon as p_i becomes free from mobile Byzantine agents, and thus allows p_i to take corrective actions (*e.g.* to avoid spreading compromised information).

[1] The agents' movements are thus synchronized with the synchronous rounds.

However, $\mathcal{O}_{BFA}$ does not provide any information about the length of the period p_i was faulty.

▶ **Definition 6** (*Full Failure Awareness Oracle $\mathcal{O}_{FFA}$*)**.** *If a mobile Byzantine agent takes control of a process p_i at time t_j and leaves p_i at time t_k, then the full failure awareness oracle $\mathcal{O}_{FFA}$ generates a* CURED() *event on p_i at time t_{k+1}, and returns the time label t_j when invoking operation* FAULTY_AT()*.*

For the sake of notation, we refer to setting where no oracle is available as $\mathcal{O}_{NFA}$. Let us remark that both $\mathcal{O}_{BFA}$ and $\mathcal{O}_{FFA}$ are *local* oracles, i.e., they provide information to the actual process where the events occurred; thus, a process p_i is not aware of the failure state of any other process p_j.

Note that the assumptions considered in our system model are equivalent to or less constrained than those in other works dealing with mobile Byzantine agents [17, 4, 31, 6]. The only exceptions are the $\mathcal{O}_{FFA}$ oracle and the notion of Δ_c-*infinitely often correct* process, which have not been considered before.

In the remainder of the paper, we will characterize the specific setting considered in terms of system timing assumptions, agent synchronization, and failure awareness by specifying a triple $\langle \alpha, \beta, \gamma \rangle$ where $\alpha \in \{\texttt{SYNC}, \texttt{ASYNC}\}$, $\beta \in \{\texttt{A-MOB}, \texttt{S-MOB}, \texttt{S-MOB}^+\}$ and $\gamma \in \{\mathcal{O}_{BFA}, \mathcal{O}_{FFA}, \mathcal{O}_{NFA}\}$. With slight abuse of notation, we will use " $*$ " in a triple when the specific dimension is not relevant to prove our claims.

4 Mobile BRB and BBC Specification

Informally *Byzantine Reliable Broadcast* (BRB) [12, 14] is a communication primitive that enables all processes of a distributed system to agree on the delivery of a single message disseminated by a pre-defined process called the *source*, while the *Byzantine Broadcast Channel* (BBC) [14] primitive extends BRB allowing all processes to disseminate an arbitrary number of messages so that all correct processes eventually deliver the same set of messages [2].

Let us note that in the original BRB and BBC specifications the source is either always correct or always faulty in a given execution. Conversely, in our settings, it is possible that the source of a message changes its failure state multiple times (even during a single broadcast instance) making the original specification no more suitable. Thus, we extend the BRB and BBC, by formalizing the *Mobile Byzantine Reliable Broadcast* (MBRB) and the *Mobile Byzantine Broadcast Channel* (MBBC) problems to capture challenges imposed by mobile Byzantine faults. We aim to specify two communication primitives accessible by every process and exposing the MBRB/MBBC.Broadcast(m) and MBRB/MBBC.Deliver(s,m) operations, where m is a message and s is a process identifier. We say that a process p_i "MBRB/MBBC-broadcasts a message m" when it executes MBRB/MBBC.Broadcast(m), and p_i "MBRB/MBBC-delivers a message m from p_s" when p_i generates the MBRB/MBBC.Deliver(s, m) event. Similarly to other communication primitives, the MBRB/MBBC-broadcast operation is triggered to disseminate a message, while MBRB/MBBC-deliver notifies message deliveries. We associate two additional parameters to both primitives, $\Delta_b \in \mathbb{N}^+$ and $\Delta_c \in \mathbb{N}^+$, characterizing the length of two periods (detailed in the specifications' properties). We use the character "*" in our specifications when the actual value of the reference parameter is irrelevant.

[2] The formal specification of BRB and BBC primitives are provided in the Appendix A.

Informally, a MBRB(Δ_b, Δ_c) communication primitive guarantees that, given a source process p_s and a message m generated by p_s while it is correct (for at least Δ_b time units), m is reliably delivered by any Δ_c-infinitely often correct process p_j in a period where p_j is correct. Similarly to BRB, this primitive is specified by considering an instance for every message generated by the identified source. More formally, a MBRB(Δ_b, Δ_c) communication primitive must guarantee the following properties:

- (Δ_b, Δ_c)-*Validity*: If there exists a period $T_{i,j}$ lasting at least Δ_b where a process p_s is correct in $T_{i,j}$ and executes MBRB.Broadcast(m), then at least one Δ_c-infinitely often correct process p_d eventually executes MBRB.Deliver(s,m) while correct.
- *No duplication*: Every process p_d executes MBRB.Deliver(s,$*$) at most once when correct, namely p_d MBRB-delivers at most one message from p_s among all times $t_k \in \mathbb{T}$ such that $p_d \in C(T_{k,k+1})$.
- Δ_b-*Integrity*: If a process p_d is correct at time t_k and executes MBRB.Deliver(s,m), then either p_s was correct in $T_{i,j} = [t_i, t_{i+\Delta_b})$, with $t_i \leq t_k$, and executed MBRB.Broadcast(m) at time t_i, or p_s was faulty at some $t_i \leq t_k$.
- *Consistency*: If some process is correct at time t_k and executes MBRB.Deliver(s, m), and another process is correct at time t_l and executes MBRB.Deliver(s, m'), then $m = m'$.
- Δ_c-*Totality*: If some process is correct at time t_k and executes MBRB.Deliver($s, *$), then every Δ_c-infinitely often correct process eventually executes MBRB.Deliver($s, *$).

The MBBC communication primitive is the natural extension of the BBC and its specification extends the one of the MBRB. In particular, the MBBC primitive guarantees that multiple messages generated by a source process (while it is correct for at least Δ_b consecutive time units) will be eventually delivered by any process p_j that is Δ_c-infinitely often correct in a period in which p_j is correct. More formally, a MBBC(Δ_b,Δ_c) communication primitive must guarantee the following properties:

- (Δ_b, Δ_c)-*Validity*: If there exists a period $T_{i,j}$ lasting at least Δ_b where a process p_s is correct in $T_{i,j}$ and executes MBRB.Broadcast(m), then at least one Δ_c-infinitely often correct process p_d eventually executes MBRB.Deliver(s,m) while correct.
- *No duplication*: Every process p_d executes MBBC.Deliver(s,m), with message m and source s, at most once when correct, namely, it MBBC-delivers a message m from p_s at most once among all times t_k such that $p_d \in C(T_{k,k+1})$.
- Δ_b-*Integrity*: If a process p_d is correct at time t_k and executes MBRB.Deliver(s,m), then either p_s was correct in $T_{i,j} = [t_i, t_{i+\Delta_b})$, with $t_i \leq t_k$, and executed MBRB.Broadcast(m) at time t_i, or p_s was faulty at some $t_i \leq t_k$.
- Δ_c-*Agreement*: If some process is correct at time t_k and executes MBRB.Deliver(s, m), then every Δ_c-infinitely often correct process eventually executes MBRB.Deliver(s, m).

Note that the specifications rule the MBRB/MBBC.Deliver(s, m) operations in times when processes are correct. Operations executed when a process is faulty cannot be controlled and thus are not relevant to the specification. Furthermore, note that when a process is controlled by a mobile Byzantine agent, it may execute arbitrary code and alter its local memory. Such a process has no information about what occurred when compromised (except the fact of being previously compromised in case an oracle is available). This makes the implementation of the presented communication primitives particularly challenging and will lead to proving several impossibility results that are specific to mobile Byzantine faults in the following sections.

5 Impossibility Results

This section presents several impossibility results for the MBRB and MBBC problems. In particular, Theorems 7 and 9 prove the impossibility of solving both MBRB and MBBC if the system is asynchronous, or if the agents' movements are asynchronous. Then, assuming a synchronous system and synchronized agents, Theorems 10 and 12 state the impossibility of solving MBRB with the strongest failure oracle we considered, $\mathcal{O}_{FFA}$, and the impossibility of solving MBBC with the weaker failure oracle, $\mathcal{O}_{BFA}$. These latter impossibilities arise from the fact that a correct process cannot infer other processes' failure state from their behavior. Thus, they cannot distinguish messages that must be delivered from those that can be safely dropped. Table 1 provides an overview of the impossibilities proved in this Section based on the specific considered settings.

▶ **Theorem 7.** *There exists no protocol $\mathcal{P}$ implementing the Mobile Byzantine Reliable Broadcast (resp. Mobile Byzantine Broadcast Channel) in $\langle$ASYNC, S-MOB, $\mathcal{O}_{FFA}\rangle$.*

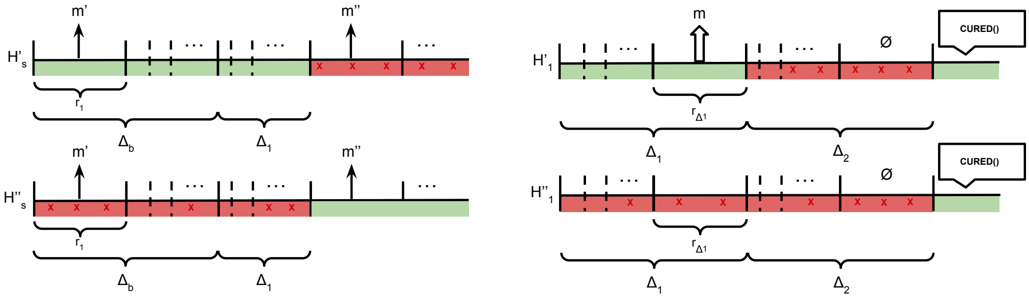
Let us note that Theorem 7 holds assuming the most constrained agent's mobility model available in an asynchronous system (i.e., `S-MOB`) and the most powerful failure oracle ($\mathcal{O}_{FFA}$) considered. It follows that the MBRB and MBBC problems cannot be solved in `ASYNC` assuming a less constrained environment, as stated in the following Corollary.

▶ **Corollary 8.** *There exists no protocol $\mathcal{P}$ implementing the Mobile Byzantine Reliable Broadcast (resp. Mobile Byzantine Broadcast Channel) in $\langle$ASYNC, $M, O\rangle$, with $M \in \{$A-MOB, S-MOB$\}$ and $O \in \{\mathcal{O}_{FFA}, \mathcal{O}_{BFA}\}$.*

▶ **Theorem 9.** *There exists no protocol $\mathcal{P}$ implementing the Mobile Byzantine Reliable Broadcast (resp. Mobile Byzantine Broadcast Channel) in $\langle$SYNC, A-MOB, $\mathcal{O}_{FFA}\rangle$.*

▶ **Theorem 10.** *If $\Delta_b \in \mathbb{N}^+$ and $\Delta_b \geq 2$ rounds, then there exists no protocol $\mathcal{P}$ implementing a Mobile Byzantine Reliable Broadcast primitive in $\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$.*

Proof. For the sake of contradiction, let us assume that such a protocol $\mathcal{P}$ exists. Let us consider the local execution history $\mathcal{H}'_s$ of a process p_s that is *correct* for $\Delta_b \geq 2$ rounds and executes MBRB.Broadcast(m') in round r_1. Subsequently, p_s remains correct for the successive Δ_1 rounds, it gets permanently *faulty* from round $r_{\Delta_b+\Delta_1+1}$ (namely $\forall r_j \in [r_{\Delta_b+\Delta_1+1}, \infty)$, $p_s \in B(r_j)$), and it executes MBRB.Broadcast(m'') in round $r_{\Delta_b+\Delta_1+1}$. We remark that the failure state of any process may change unexpectedly due to the movement of a Byzantine agent. Let us consider another local execution history $\mathcal{H}''_s$ of process p_s where the failure state of p_s evolves in the opposite way from $\mathcal{H}'_s$, that is process p_s is *faulty* in rounds $r_j \in [r_1, r_{\Delta_b+\Delta_1}]$ and executes MBRB.Broadcast(m') in round r_1; subsequently, p_s is permanently *correct* from round $r_{\Delta_b+\Delta_1+1}$ (namely $\forall r_j \in [r_{\Delta_b+\Delta_1+1}, \infty), p_s \in C(r_j)$) and executes MBRB.Broadcast(m'') in round $r_{\Delta_b+\Delta_1+1}$. Notice that in both histories p_s executes the MBRB.Broadcast operation only once while correct. We provide a graphical representation of the two histories in Figure 1a. Let us consider a process $p_1 \neq p_s$ that is correct for the entire lifetime of the system (i.e. $\forall r_j$, $p_1 \in C(r_j)$), thus p_1 is also an Δ_c-infinitely often correct process for any value of $\Delta_c \in \mathbb{N}$. The two execution histories $\mathcal{H}'_s$ and $\mathcal{H}''_s$ are indistinguishable to p_1 because the same operations and events occurred on p_s. Process p_1 is not aware of the failure state of p_s (i.e. it has no access to the failure oracle on p_s). Even defining an algorithm $\mathcal{A}$ that allows process p_s to share the information obtained from $\mathcal{O}_{FFA}$ with process p_1 through the point-to-point primitive, process p_1 cannot distinguish an execution of $\mathcal{A}$ where p_s is correct and reveals a previous faulty state, from another where p_s is faulty, and maliciously reports the same information.



(a) Graphical representations for Theorem 10. **(b)** Graphical representations for Theorem 12.

Figure 1 Graphical representations for Theorems' proof.

According to the *Validity* property of the MBRB specification, process p_1 executing $\mathcal{P}$ must MBRB-deliver a message from p_s considering both histories because process p_s MBRB-broadcasts a message when correct. If $\mathcal{P}$ makes process p_1 eventually MBRB-deliver message m', then the *Validity* property is violated in $\mathcal{H}''_s$, because process p_1 never MBRB-delivers m'' (according to the *No-duplication* property) that is broadcast when p_s is correct. If $\mathcal{P}$ makes process p_1 eventually MBRB-deliver message m'', then the *Validity* property is violated in $\mathcal{H}'_s$ for the same reason. This is a contradiction and the claim follows regardless of the value of Δ_b and Δ_c. ◀

Theorem 10 states the impossibility in solving MBRB assuming the most constrained assumptions we considered. Corollary 11 extends the result to less constrained settings.

▶ **Corollary 11.** *If $\Delta_b \in \mathbb{N}^+$ and $\Delta_b \geq 2$ rounds, then there exists no protocol $\mathcal{P}$ implementing a Mobile Byzantine Reliable Broadcast primitive in $\langle$SYNC, S-MOB$^+, \mathcal{O}_{BFA}\rangle$ or in $\langle$SYNC, S-MOB, $*\rangle$.*

▶ **Theorem 12.** *If $\Delta_b \in \mathbb{N}^+$ and $\Delta_b \geq 2$ rounds, then there exists no protocol $\mathcal{P}$ implementing a Mobile Byzantine Reliable Channel primitive in $\langle$SYNC, S-MOB$^+, \mathcal{O}_{BFA}\rangle$.*

Proof. For the sake of contradiction, let us assume that such a protocol $\mathcal{P}$ exists. Let us assume a permanently correct process p_s (i.e. $\forall r_j,\ p_s \in C(r_j)$) that executes MBBC.Broadcast(m) in rounds r_1. Let us consider the local execution history $\mathcal{H}'_1$ of a process p_1 that is *correct* in rounds $r_j \in [r_1, r_{\Delta_1}]$, $\Delta_1 \in \mathbb{N}$, and executes MBBC.Deliver(m) in round r_{Δ_1}; subsequently, p_1 gets *faulty* for Δ_2 consecutive rounds, $\Delta_2 \in \mathbb{N}$, it wipes its local state (i.e. initialises all the process variables) in round $r_{\Delta_1+\Delta_2}$, and it gets *permanently correct* from round $r_{\Delta_1+\Delta_2+1}$ (namely $\forall r_i \in [r_{\Delta_1+\Delta_2+1}, \infty),\ p_1 \in C(r_i)$).
Let us consider another local execution history $\mathcal{H}''_1$ of process p_1 that is *faulty* in rounds $r_j \in [r_1, r_{\Delta_1+\Delta_2}]$ and it wipes its local state in round $r_{\Delta_1+\Delta_2}$; subsequently, p_1 gets *permanently correct* from round $r_{\Delta_1+\Delta_2+1}$ (namely $\forall r_j \in [r_{\Delta_1+\Delta_2+1}, \infty),\ p_1 \in C(r_j)$). We provide a graphical representation in Figure 1b. In round $r_{\Delta_1+\Delta_2+1}$, process p_1 has the same local state in both histories and the $\mathcal{O}_{BFA}$ oracle generates the same cured() event on process p_1. Process p_1 does not know what happened during the previous rounds. It is even defining an algorithm $\mathcal{A}$ that allows any process p_i to share and retrieve the state and events occurred on the process through the point-to-point primitive: process p_i can execute such a protocol either as correct or as faulty, and the two executions would be indistinguishable by any other process.

Table 1 Summary of the solvability results.

(a) MBRB.

	ASYNC	SYNC	
S-MOB$^+$		$\mathcal{O}_{BFA}$	$\mathcal{O}_{FFA}$
		✗ (Cor. 11)	✗ (Th. 10)
S-MOB	✗ (Cor. 8)	$\mathcal{O}_{BFA}$	$\mathcal{O}_{FFA}$
		✗ (Cor. 11)	✗ (Cor. 11)
A-MOB	✗ (Cor. 8)	✗ (Th. 9)	

(b) MBBC.

	ASYNC	SYNC	
S-MOB$^+$		$\mathcal{O}_{BFA}$	$\mathcal{O}_{FFA}$
		✗(* **Sec 7**) (Th. 12)	✓ **(Th. 16)**
S-MOB	✗ (Cor. 8)	$\mathcal{O}_{BFA}$	$\mathcal{O}_{FFA}$
		✗	?
A-MOB	✗ (Cor. 8)	✗ (Th. 9)	

According to the *Validity* property of the MBBC specification, process p_1 executing $\mathcal{P}$ must MBBC-deliver message m from p_s in both histories. In round $r_{\Delta_1+\Delta_2+1}$ process p_1 has the same local state on both histories, thus it can act in one only way, specifically it can command or not process p_i to deliver message m from p_s. In the positive case, the protocol violates the *No duplication* property in history $\mathcal{H}_1'$, in the negative case the *Validity* property is violated by the protocol in $\mathcal{H}_1''$. This leads to a contradiction and the claim follows regardless to the value of Δ_1,Δ_2, and Δ_c. ◀

Discussion. Contrarily to what we could expect, the MBRB and MBBC problems are impossible to solve in settings (*e.g.*, ⟨`SYNC`, `S-MOB`$^+$, $\mathcal{O}_{NFA/BFA}$⟩) where the register abstraction and consensus problems are solvable [17, 4, 31, 6, 10, 8, 11]. The intuition behind this is that other problems addressed under the MBF model have a semantics that do not require to execute a particular operation (the delivery of a message in our case) at most once and depending on a precedent failure state of the process. Indeed, both the register abstractions and consensus set constraints on a local value stored by the processes (respectively, the shared value and the decided value) but no primitive operation is associated with their update in their specification. Contrarily, MBRB and MBBC introduce constraints on the deliveries of messages that depend on the actual and previous failure states of the processes, generating thus symmetry conditions that are impossible to break without violating one of the properties characterizing the specification. In particular, the main challenge is to ensure that a single broadcast instance does not generate multiple deliveries to the same process while it is correct. Another counter-intuitive result is that considering a setting stronger than the one considered in related works (*e.g.*, ⟨`SYNC`, `S-MOB`$^+$, $\mathcal{O}_{FFA}$⟩), the MBRB problem is impossible to solve while the MBBC one is possible (see Section 6). In the static Byzantine failure model (where every process is always either correct or faulty in a given execution), the channel specification extends the broadcast one allowing multiple broadcast from the same source. As a matter of fact, in the mobile Byzantine failure model such an extension is less constrained with respect to the broadcast: in MBRB, every process can execute only one broadcast operation for the entire lifetime of the system, whereas MBBC allows multiple broadcasts from the same source; if a process is faulty and executes a broadcast, then it is not allowed to execute a subsequent broadcast when correct in the future in the MBRB specification (*No duplication* property), while it is in MBBC. Finally, note that other primitives, such as consensus or register abstractions, are not useful in solving the MBRB/MBBC problems. Consider again the execution depicted in Figure 1a, correct process may agree or may store a set of delivered messages (according to the MBRB/MBBC specifications) but a single process (p_s in the example), in the settings we characterized, cannot infer if it has already delivered or not a message if it was previously compromised.

6 A Protocol for MBBC in $\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$

Theorem 12 and Corollary 11 motivate the definition of a stronger local oracle than those considered in related work dealing with mobile Byzantine faults, $\mathcal{O}_{FFA}$: both MBRB and MBBC are impossible to solve in the ($\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{NFA/BFA}\rangle$) settings. Theorem 10 states the impossibility in solving MBRB even in ($\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$). This Section investigates the remaining open problem-setting: the solvability of MBBC in ($\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$). Specifically, we start by defining $\mathcal{P}_{MBBC-RB}$, a protocol implementing the MBBC(Δ_b, Δ_c) communication primitive. Then, we prove its correctness and fault-tolerance optimality.

6.1 $\mathcal{P}_{MBBC-RB}$: Protocol Description

$\mathcal{P}_{MBBC-RB}$ is an extension of Bracha's algorithm [12] aimed to solve the MBBC problem. It inherits Bracha's diffusion mechanism: a payload message m is exchanged inside three protocol messages, SEND, ECHO, and READY. The former is initially sent by the source process to all peers, and the latter are subsequently diffused by all correct processes to all peers if certain conditions are met, namely certain quorums are reached.
The pseudo-code of $\mathcal{P}_{MBBC-RB}$ is shown in Algorithm 1. This solution overcomes the impossibility stated in Theorem 12 by leveraging on $\mathcal{O}_{FFA}$ and by fixing the round index (i.e., the moment in time) where MBBC-deliveries must occur. Every protocol's message contains the information about a specific MBBC-broadcast instance, specifically the source process label s, the message (payload) m, and the round counter r_b when the broadcast instance started. An MBBC-broadcast instance proceeds in four consecutive rounds in $\mathcal{P}_{MBBC-RB}$. In the first round r_b, the protocol's message SEND is computed by p_s and enqueued to P2P-send to all processes in the subsequent round. Every process that P2P-receives a SEND message in round r_{b+1} from p_s computes the ECHO protocol's message for $\langle s, r_b, m\rangle$ and enqueues it to P2P-send to all peers. In round r_{b+2}, the processes that receive sufficiently many ECHO messages (more than $(n+f)/2$) for an MBBC-broadcast instance from distinct peers generate the related READY protocol's message to P2P-send to all processes. Finally, in round r_{b+3}, the processes that receive a sufficient number of READY messages (more than $2f$) for an MBBC-broadcast instance from distinct peers MBBC-deliver the associated message m from p_s. An additional protocol's message with respect to Bracha [12], i.e. ABORT, is exchanged in $\mathcal{P}_{MBBC-RB}$ to guarantee the *Agreement* property in case of a faulty source. In $\mathcal{P}_{MBBC-RB}$, if a correct process p_s executes MBBC.BROADCAST(m) in round r_b, then every process that is correct in round r_{b+3} triggers MBBC.DELIVER(s,m) in the *compute* phase of that round; every process that is faulty in round r_{b+3} MBBC-delivers the message m from p_s at the first round $r_k > r_{b+3}$ it is correct.

We plug the fault-tolerant round counter defined by Bonnet at al. [6] inside the $\mathcal{P}_{MBBC-RB}$ protocol, enabling all correct processes to share the same value for the round index (that is assumed as an integer value). Its purpose is to fix the single round where the delivery of a certain message can take place. The round counter features are summarised in the following remark.

▶ Remark 13 (Round counter correctness [6]). In $\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{BFA/FFA}\rangle$, if $n > 3f$ then every correct process p_i in round r_j stores the same value for the round index (namely the variable rc in Algorithm 1) during compute phase.

We stress the fact that protocol's messages in $\mathcal{P}_{MBBC-RB}$ (SEND, ECHO, READY, and ABORT) must be propagated in specific rounds with respect to the beginning of the MBBC-broadcast, in order to progress till the delivery of the associated message m.

A detailed description of $\mathcal{P}_{MBBC-RB}$ appears in the companion technical report [7], and we illustrate some execution examples in Appendix B, and within the proof of Lemma 14.

Algorithm 1 $\mathcal{P}_{MBBC-RB}$.

```
procedure Init
    To_send ← ∅, Sends ← ∅, cured ← False, rc ← 1
    Echos ← {}, Readys ← {}, Aborts ← {}        ▷ map, ⟨s, r, m⟩ : set of process ids
    RC ← {}                                      ▷ map, process id : round value
procedure Broadcast(m)
    To_send ← To_send ∪ {⟨SEND, s, rc, m⟩}
upon O_FFA.cured do
    cured ← True
Send Phase
if cured then
    To_send ← ∅
for pk ∈ To_send do
    for q ∈ Π do
        P2P.send(q, pk)
Receive Phase
Sends ← ∅, Echos ← {}, Readys ← {}, Aborts ← {}, RC ← {}
upon P2P.deliver(q, ⟨Type, s, r_b, m⟩) do
    if s = q and Type = SEND then
        Sends ← Sends ∪ {⟨s, r_b, m⟩}
    if Type = ECHO then
        Echos[⟨s, r_b, m⟩] ← Echos[⟨s, r_b, m⟩] ∪ {q}
    if Type = READY then
        Readys[⟨s, r_b, m⟩] ← Readys[⟨s, r_b, m⟩] ∪ {q}
    if Type = ABORT then
        Aborts[⟨s, r_b, m⟩] ← Aborts[⟨s, r_b, m⟩] ∪ {q}
upon P2P.deliver(q, ⟨ROUND, j⟩) do
    RC[q] ← j
Compute Phase
To_send ← ∅, rc ← getMajority(RC.values)
for ⟨s, r_b, m⟩ ∈ Sends do
    if rc = r_{b+1} then
        To_send ← To_send ∪ {⟨ECHO, s, r_b, m⟩}
for ⟨s, r_b, m⟩ ∈ Echos do
    if |Echos[⟨s, r_b, m⟩]| > (n + f)/2 then
        To_send ← To_send ∪ {⟨READY, s, r_b, m⟩}
    else if |Echos[⟨s, r_b, m⟩]| > f then
        To_send ← To_send ∪ {⟨ABORT, s, r_b, m⟩}
for ⟨s, r_b, m⟩ ∈ Aborts do
    if |Aborts[⟨s, r_b, m⟩]| > f then
        |Readys[⟨s, r_b, m⟩] ← ∅
for ⟨s, r_b, m⟩ ∈ Readys do
    if |Readys[⟨s, r_b, m⟩]| > 2f then
        if ((rc = r_{b+3}) or (cured and rc > r_{b+3} and O_FFA.faulty_at ≤ r_{b+3}))
         and (∄⟨s, r_k, m⟩ ∈ Readys : (|Readys⟨s, r_k, m⟩| > 2f) ∧ (r_k < r_b)) then
            Deliver(s,m)
        To_send ← To_send ∪ {⟨READY, s, r_b, m⟩}
cured ← False, rc ← rc+1, To_send ← To_send ∪ {⟨ROUND, rc⟩}
```

6.2 Correctness Proofs

We remark that in S-MOB$^+$ mobile agents can move only between the *compute* and *send* phase of two consecutive rounds. This implies that Δ_s is assumed greater than or equal to one round. Such mobility model has the following effects to the agents' capabilities: at the beginning of a round r_j, mobile agents can potentially control the messages that are diffused by $2f$ processes, the ones where the mobile agents are placed in r_j and the others where they were in the previous round r_{j-1} (they can set in round r_{j-1} the messages that will be exchange by freed processes in round r_j). This capability can partially be mitigated by the local failure detector $\mathcal{O}_{FFA}$: a process can discard all messages queued to be send right after the failure detector notifies the CURED() event. It follows that, at the beginning of a round, at most f processes may not participate in the protocol and at most f may have a Byzantine behavior.

The following Lemmas and Theorem state the correctness of $\mathcal{P}_{MBBC-RB}$ in solving the MBBC problem and its fault-tolerance optimality with respect to the number of tolerated mobile agents.

▶ **Lemma 14.** *If $\Delta_b \geq 2$ rounds and $\Delta_c \geq 1$ round, then $\mathcal{P}_{MBBC-RB}$ solves the Mobile Byzantine Broadcast Channel problem (MBBC) in $\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$ if $n > 5f$.*

Proof. For simplicity, we give the proof assuming the minimum values for Δ_b and Δ_c. The arguments extend to higher values.

($\Delta_b = 2$ rounds, $\Delta_c = 1$ round)-Validity. We prove that if we assume $\Delta_b = 2$ rounds, $\Delta_c = 1$ round, and a process p_s is correct in round r_b when it executes MBBC.BROADCAST(m), then every process that is Δ_c-infinitely often correct eventually triggers MBBC.DELIVER(s,m), that implies the (Δ_b, Δ_c)-*Validity* property. The MBBC-delivery of a message m from a process p_s may occur either because p_s was correct in round r_b and executed MBBC.BROADCAST(m) or since p_s was faulty at some round $r_d < r_b$ and P2P-sent a SEND message with payload m. Let us assume that process p_s has not P2P-sent yet the SEND message with payload m neither as correct or faulty before round r_b, that it is correct in rounds r_b and r_{b+1} ($\Delta_b = 2$) and executes the procedure Broadcast with parameter m in round r_b. The $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ message is then prepared (line 6) to be relayed to all other processes (lines 11-13). In round r_{b+1}, the $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ message is P2P-sent by p_s to all processes and it is received by all but f (the ones controlled by mobile agents); it follows that $n - f$ processes executes lines 15-17 during the *receive* phase in round r_{b+1} and lines 27-29 in the *compute* phase, preparing the $\langle \mathsf{ECHO}, \mathrm{s}, r_b, \mathrm{m}\rangle$ message to P2P-send in round r_{b+2}. In round r_{b+2}, at least $n - 2f$ processes relay the message $\langle \mathsf{ECHO}, \mathrm{s}, r_b, \mathrm{m}\rangle$ (f process may be faulty in round r_{b+2} and f process may have been faulty in round r_{b+1}) and it is received by $n-f$ processes (again, the ones not controlled by mobile agents). These processes execute lines 15, 18 and 19 in the *receive* phase and lines 31 and 32 in the *compute* phase. In particular, the condition inside the *if* statement at line 31 is verified due to the assumption $n > 5f$, given that $n - 2f > (n + f)/2$, and line 32 is executed preparing $\langle \mathsf{READY}, \mathrm{s}, r_b, \mathrm{m}\rangle$ message to P2P-send in round r_{b+3}. Finally, in round r_{b+3}, the same reasoning given for round r_{b+2} applies and $n - f$ processes execute lines 39-41, given $n - 2f > 2f$ and Remark 13, and thus they trigger Deliver with parameters s and m. At every round $r_j > r_{b+3}$ the $\langle \mathsf{READY}, \mathrm{s}, r_b, \mathrm{m}\rangle$ message is P2P-sent by all the correct processes not faulty in round r_{j-1} (that are at least $n - f$). The *if* statement at line 40 guarantees that every process that was faulty in round r_{b+3} delivers message m from p_s at the first round $r_k > r_{b+3}$ it is correct. Finally, in case *(i)* process p_s was faulty and P2P-sent the SEND message with payload m in round $r_k < r_b$, *(ii)* every Δ_c-infinitely correct process MBBC-delivered m from p_s, and *(iii)* process p_s is correct in round $r_b > r_k$ and executes MBBC.BROADCAST(m), then the claim still follows: the message m has been already MBBC-delivered (further details can be found in the *Agreement* property's proof).

No duplication. The second sub-condition of the *if* statement at line 40 guarantees that the entire *if* statement is verified only for the minimum r_j among all the tuples $\langle s, *, m\rangle$ (i.e. the MBBC-delivery is independent from the r_b parameter). The first sub-condition inside the *if* statement at line 40 is verified only once among all the rounds a mobile agent does not control the process. More in detail, if the *cured* variable is FALSE, the condition is verified only in round r_{b+3} for the tuple $\langle s, r_b, m\rangle$. Otherwise, the *if* statement in line 40 is verified in round $r_k > r_{b+3}$ when a mobile agent, arrived on the process in round $r_j \leq r_{b+3}$, leaves the process, that occurs only once on a process during the entire lifetime of the system given Remark 13. The condition $rc > r_{b+3}$ in line 40 is not required but simplifies this proof.

($\Delta_b = 2$)-Integrity. For the sake of contradiction, let us assume that a process p_i is correct in round r_k and executes MBBC.DELIVER(s,m), that process p_s is correct in rounds r_b and r_{b+1} (that is, $\Delta_b = 2$), and that it does not execute MBBC.BROADCAST(m) in round r_b. Process p_i MBBC-delivers m from p_s either in round $r_k = r_{b+3}$ if p_i is correct, or at the first round $r_k > r_{b+3}$ when p_i is correct. In the former case, more than $2f$ processes sent message $\langle \mathsf{READY}, \mathrm{s}, r_b, \mathrm{m}\rangle$ in round r_{b+3}, therefore more than $(n+f)/2$ processes sent message $\langle \mathsf{ECHO}, \mathrm{s}, r_b, \mathrm{m}\rangle$ in round r_{b+2}, that implies that at least $(n+f)/2-f$ processes were correct in round r_{b+1} and received $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ in round r_{b+1} from p_s (lines 28-29). No procedure in $\mathcal{P}_{MBBC-RB}$ allows a correct process p_s to P2P-send $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ messages except BROADCAST(m). It follows that the latter scenario occurred and process p_i was faulty in round r_{b+3}. As a matter of fact, correct process p_i P2P-received more than $2f$ $\langle \mathsf{READY}, \mathrm{s}, r_b, \mathrm{m}\rangle$ messages from distinct processes in round r_k. For the same reasoning as in the former case, this implies that a correct process p_s sent $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ messages but no procedure except BROADCAST(m) allows it. This leads to a contradiction and the claim follows.

($\Delta_c = 1$)-Agreement. We proved, in the *Validity* proof, that this property is satisfied in the case of a correct source. Faulty processes cannot collude to make one of the *if* statements at lines 31, 33, 36 and 39 verified for a message m never sent over the P2P links of a process p_s. More in detail, the attacker cannot attempt to make any correct process MBBC-deliver a message m from p_s without compromising p_s. We prove that if p_s is faulty and P2P-sends $\langle \mathsf{SEND}, \mathrm{s}, r_b, \mathrm{m}\rangle$ messages in round r_b, then either all Δ_c-infinitely correct processes delivers m from p_s or no Δ_c-infinitely correct processes delivers m from p_s. For the sake of contradiction, let us assume that all Δ_c-infinitely often correct processes but some, $p_1, p_2, \ldots, p_i$, MBBC-delivered a message m from p_s. It follows that there is no round r_j where more than $2f$ correct processes concurrently P2P-send $\langle \mathsf{READY}, \mathrm{s}, r_b, \mathrm{m}\rangle$. This implies that the correct processes that delivered m are at most $2f$. According to the protocol, such processes receive a quorum of ECHO messages and at most f ABORT messages about m, to generate the required READY messages. More in detail, they received ECHO messages from at least $2f+1$ correct processes. At that point, the faulty processes decided which correct processes reached the quorum of ECHO messages. Nevertheless, each correct process that did not reach the quorum generated an ABORT message. It follows that at most f correct processes did not reach the quorum, whereas $n-f-f$ processes were correct and generated the READY message, which was disseminated by at least $n-3f$ of them in the subsequent round. Given that $n > 5f$, at least $2f+1$ correct processes concurrently disseminate a READY message and thus all correct processes in round r_{b+3} must MBBC-deliver it. This lead to a contradiction and the claim follows. ◀

▶ **Lemma 15.** *The Mobile Byzantine Broadcast Channel problem (MBBC) is solvable in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{FFA}\rangle$ *only if* $n > 5f$*.*

Proof. The claim follows by extending the results proven by Backes and Cachin [3] and by Raynal [25]. The former states that the BRB problem can be solved in a static distributed system where at most t processes may fail-stop, and at most f processes are Byzantine, if and only if $n > 3f + 2t$. Similarly, Raynal proved that the BRB problem can be solved in a static distributed system, where t_l processes may not send messages, and t_s processes may send spurious messages (processes may exhibit both behaviors during the lifetime of the system), if and only if $n > 2t_l + t_s$.

Both scenarios can be simulated by an attacker in our system: the mobile agents can continuously alternate between two disjoint sets P_1 and P_2 of f processes, namely it can turn faulty all processes in P_1 in all rounds $r_j, j \in \mathbb{N}$, and all processes in P_2 in all rounds r_{j+1}, sending spurious messages from process in P_1 and no message from peers in P_2. Therefore, all processes in P_1 send spurious messages (behaving like f Byzantine faulty processes), and all the processes in P_2 send no message (like f fail-stop faulty processes), and the claim follows. ◀

▶ **Theorem 16.** *The Mobile Byzantine Broadcast Channel problem (MBBC) is solvable in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{FFA}\rangle$ *with* $\mathcal{O}_{FFA}$ *if and only if* $n > 5f$*.*

Proof. It follows from Lemmas 14 and 15. ◀

The following Corollary extends the optimality of $\mathcal{P}_{MBBC-RB}$ to the case of slower agents. In other words, even if the mobile agents are slower we are not able to tolerate more agents solving MBBC.

▶ **Corollary 17.** *The Mobile Byzantine Broadcast Channel problem (MBBC) is solvable in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{FFA}\rangle$ *if and only if* $n > 5f$*, for each* $\Delta_s \geq 1$ *round. Furthermore, the actual value of* Δ_s *can be unknown to the processes.*

Note that MBBC and MBBR specifications do not allow processes to be *terminate*, namely to eventually stop propagating messages through the P2P primitive. Intuitively, processes need to continuously relay the messages in order to enforce Δ_c-*Totality/Agreement* and thus allow every temporarily faulty process to eventually deliver a broadcast message. Furthermore, as argued in Section 5, processes are not able to infer if a specific process has delivered a message, and thus conclude if all processes delivered a message when correct. Additional assumptions enabling termination can be considered, such as an upper-bound on the time a process becomes correct when faulty.

7 MBBC with multiple deliveries

The impossibilities identified in Section 5 arise for the general specification we defined. In fact, alternative or weaker specifications could be implementable under weaker assumptions. More in detail, we proved that no protocol can solve the MBBC in $\langle$`SYNC, S-MOB`$^+$`,` $\mathcal{O}_{BFA}\rangle$. We therefore investigate the possibility of a weaker primitive that can be realized when the stringent conditions identified in Theorem 16 are not satisfied.

We start by considering the case where no local failure detector is available, that is, the case of $\mathcal{O}_{NFA}$. The following Theorem show that a weaker MBBC primitive, where the *No duplication* property is not satisfied, is realizable in $\langle$`SYNC, S-MOB`$^+$`,` $\mathcal{O}_{NFA}\rangle$.

▶ **Theorem 18.** *A weaker Mobile Byzantine Broadcast Channel primitive, not guaranteeing the* No duplication *property, is realizable in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{NFA}\rangle$ *if* $\Delta_b = 2$ *rounds,* $\Delta_c = 1$ *round, and* $n > 6f$*.*

Proof. Let us consider the $\mathcal{P}_{MBBC-RB}$ protocol defined in Algorithm 1. Let us ignore the lines that interacts with the local failure detector, namely 7, 8 and 40. Let us substitute all the occurrences of parameter f with $\bar{f} = 2f$ in Algorithm 1.

The difference with respect the setting considered in Lemma 14 is that processes are not aware of being compromised. In particular, they may diffuse messages with P2P-links previously generated by mobile agents. As a matter of fact, the protocol is restored right after the mobile agent left the process.

The proof follows from the same reasoning stated in Lemma 14 except for *No duplication* considering $\bar{f}$ instead of f in Algorithm 1. ◀

The following theorem show that having a slightly better oracle about failures, namely $\mathcal{O}_{BFA}$, permits to withstand more Byzantine agents, for the same weaker problem that does *not* guarantees no duplication.

▶ **Theorem 19.** *A weaker Mobile Byzantine Broadcast Channel primitive, not guaranteeing the* No duplication *property, is realizable in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{BFA}\rangle$ *if* $\Delta_b = 2$ *rounds,* $\Delta_c = 1$ *round, and* $n > 5f$*.*

Abandoning the *No duplication* guarantee, the number of message delivered becomes unbounded: the following theorem shows that it is not possible to bound the number of duplicate messages that are delivered, even assuming an intermediate oracle, namely $\mathcal{O}_{BFA}$.

▶ **Theorem 20.** *Given a constant* $\bar{k} \in \mathbb{N}^+$*, it is not possible to define a weaker Mobile Byzantine Broadcast Channel primitive, not guaranteeing the* No duplication *property, in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{BFA}\rangle$ *where a message* m *MBBC-Broadcast by a process* p_s *is MBBC-Delivered by a process* p_i *at most* $\bar{k}$ *time when correct.*

▶ **Corollary 21.** *Suppose a solution to a weaker Mobile Byzantine Broadcast Channel primitive, not guaranteeing the* No duplication *property, in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{BFA}\rangle$*. If a process* p_i *gets faulty and correct* k *times after the MBBC-Broadcast of a message* m *from* p_s*, then* p_i *MBBC-Delivers* m *from* p_s *at least* k *times.*

▶ **Theorem 22.** *Suppose a solution to a weaker Mobile Byzantine Broadcast Channel primitive, not guaranteeing the* No duplication *property, in* $\langle$*SYNC, S-MOB*$^+$*,* $\mathcal{O}_{NFA}\rangle$*. If a process* p_s *MBBC-Broadcast a message* m*, then every process* p_i *must MBBC-Deliver* m *from* p_s *infinitely often.*

8 Conclusion

We provided a specification for the Byzantine Reliable Broadcast and Byzantine Broadcast Channel problems in distributed systems affected by mobile Byzantine faults. We identified some impossibilities; in particular, we showed that both speed constraints on the mobile agents and timing assumptions on the system evolution are required to solve the problems under investigation, and we proved that the Byzantine Reliable Broadcast cannot be solved even in one of the most constrained mobile Byzantine failure models presented so far. The Byzantine Broadcast Channel problem proved to be solvable, assuming a stronger local failure detector than the ones previously considered in the literature. Lastly, we investigated a weaker Byzantine Broadcast Channel primitive, not guaranteeing the *No duplication* property, in settings equivalent to the ones assumed in related works. Our results characterise the solvability of a fundamental problem in a general dynamic process failure model, and open the path for research on additional important tasks. In particular, to understand the gap that exists between the theoretical model (assumed in this and in related work [4, 6, 17, 24, 31, 27]) and the practical world, investigating the feasibility of the oracles and defining solutions that are as practical as possible. Furthermore, it may be interesting to relax the assumptions of instantaneous fault detection and recovery (of the protocol), to investigate whether the assumption of digitally signed messages has an impact on the solvability of the considered problems, and to analyse the Mobile Byzantine Channel problem assuming the S-MOB agent mobility model (which we have left open for analysis and we conjecture its solvability).

References

1 Ittai Abraham, Ling Ren, and Zhuolun Xiang. Good-case and bad-case latency of unauthenticated byzantine broadcast: A complete categorization. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPIcs*, pages 5:1–5:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.OPODIS.2021.5.

2 Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25–29, 2022*, pages 399–417. ACM, 2022. doi:10.1145/3519270.3538475.

3 Michael Backes and Christian Cachin. Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In *2003 International Conference on Dependable Systems and Networks (DSN 2003), 22-25 June 2003, San Francisco, CA, USA, Proceedings*, pages 37–46. IEEE Computer Society, 2003. doi:10.1109/DSN.2003.1209914.

4 Nazreen Banu, Samia Souissi, Taisuke Izumi, and Koichi Wada. An improved byzantine agreement algorithm for synchronous systems with mobile faults. *International Journal of Computer Applications*, 43(22):1–7, 2012.

5 Romain Boichat and Rachid Guerraoui. Reliable and total order broadcast in the crash-recovery model. *J. Parallel Distributed Comput.*, 65(4):397–413, 2005. doi:10.1016/J.JPDC.2004.10.008.

6 François Bonnet, Xavier Défago, Thanh Dang Nguyen, and Maria Potop-Butucaru. Tight bound on mobile byzantine agreement. *Theor. Comput. Sci.*, 609:361–373, 2016. doi:10.1016/J.TCS.2015.10.019.

7 Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Reliable Broadcast despite Mobile Byzantine Faults, nov 2023. URL: https://hal.science/hal-04277831.

8 Silvia Bonomi, Antonella Del Pozzo, and Maria Potop-Butucaru. Tight self-stabilizing mobile byzantine-tolerant atomic register. In *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*, pages 6:1–6:10. ACM, 2016. doi:10.1145/2833312.2833320.

9 Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Approximate agreement under mobile byzantine faults. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 727–728. IEEE Computer Society, 2016. doi:10.1109/ICDCS.2016.68.

10 Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal mobile byzantine fault tolerant distributed storage: Extended abstract. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 269–278. ACM, 2016. doi:10.1145/2933057.2933100.

11 Silvia Bonomi, Antonella Del Pozzo, Maria Potop-Butucaru, and Sébastien Tixeuil. Optimal storage under unsynchronized mobile byzantine faults. In *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*, pages 154–163. IEEE Computer Society, 2017. `doi:10.1109/SRDS.2017.20`.

12 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987. `doi:10.1016/0890-5401(87)90054-X`.

13 Harry Buhrman, Juan A. Garay, and Jaap-Henk Hoepman. Optimal resiliency against mobile faults. In *Digest of Papers: FTCS-25, The Twenty-Fifth International Symposium on Fault-Tolerant Computing, Pasadena, California, USA, June 27-30, 1995*, pages 83–88. IEEE Computer Society, 1995. `doi:10.1109/FTCS.1995.466995`.

14 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. `doi:10.1007/978-3-642-15260-3`.

15 Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974. `doi:10.1145/361179.361202`.

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

17 Juan A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults (extended abstract). In Gerard Tel and Paul M. B. Vitányi, editors, *Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 – October 1, 1994, Proceedings*, volume 857 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994. `doi:10.1007/BFB0020438`.

18 Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic byzantine reliable broadcast. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.OPODIS.2020.23`.

19 Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. Scalable byzantine reliable broadcast. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPICS.DISC.2019.22`.

20 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. `doi:10.1142/S0129626416500171`.

21 Vasilis P. Koutras and Agapios N. Platis. Chapter 3: Software rejuvenation: Key concepts and granularity. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Coimbra, Portugal, October 12-15, 2020*, pages 321–322. IEEE, 2020. `doi:10.1109/ISSREW51248.2020.00092`.

22 Jing Li, Tianming Yu, Ye Wang, and Roger Wattenhofer. Dynamic byzantine broadcast in asynchronous message-passing systems. *IEEE Access*, 10:91372–91384, 2022. `doi:10.1109/ACCESS.2022.3202627`.

23 Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *J. Netw. Comput. Appl.*, 36(1):16–24, 2013. `doi:10.1016/J.JNCA.2012.09.004`.

24 Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59. ACM, 1991. `doi:10.1145/112600.112605`.

25 Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems – An Algorithmic Approach*. Springer, 2018. `doi:10.1007/978-3-319-94141-7`.

26 Michel Raynal. On the versatility of bracha's byzantine reliable broadcast algorithm. *Parallel Process. Lett.*, 31(3):2150006:1–2150006:9, 2021. doi:10.1142/S0129626421500067.

27 Rüdiger Reischuk. A new solution for the byzantine generals problem. *Inf. Control.*, 64(1-3):23–42, 1985. doi:10.1016/S0019-9958(85)80042-5.

28 Luís E. T. Rodrigues and Michel Raynal. Atomic broadcast in asynchronous crash-recovery distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems, Taipei, Taiwan, April 10-13, 2000*, pages 288–295. IEEE Computer Society, 2000. doi:10.1109/ICDCS.2000.840941.

29 Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*, pages 57–64. IEEE, 2015. doi:10.1109/TRUSTCOM.2015.357.

30 Dimitris Sakavalas and Lewis Tseng. Delivery delay and mobile faults. In *17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge, MA, USA, November 1-3, 2018*, pages 1–8. IEEE, 2018. doi:10.1109/NCA.2018.8548345.

31 Toru Sasaki, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Mobile byzantine agreement on arbitrary network. In Roberto Baldoni, Nicolas Nisse, and Maarten van Steen, editors, *Principles of Distributed Systems – 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings*, volume 8304 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2013. doi:10.1007/978-3-319-03850-6_17.

32 Lewis Tseng. An improved approximate consensus algorithm in the presence of mobile faults. In Paul G. Spirakis and Philippas Tsigas, editors, *Stabilization, Safety, and Security of Distributed Systems – 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*, volume 10616 of *Lecture Notes in Computer Science*, pages 109–125. Springer, 2017. doi:10.1007/978-3-319-69084-1_8.

A The Byzantine Reliable Broadcast and Channel Problems Specification [12, 14]

The Byzantine Reliable Broadcast and the Byzantine Broadcast Channel problems aim at specifying a communication primitive, respectively BRB and BBC, exposing two operations, BRB/BBC-broadcast(m) and BRB/BBC-deliver(s, m), where m is a message and s is a process identifier.

The BRB primitive enables all correct processes of a distributed system to agree on a single message diffused by a (potentially faulty) particular process, the source. The BBC primitive extends BRB allowing all processes to diffuse an arbitrary number of messages so that all correct processes eventually deliver the same set of messages. We say that a process p_i "BRB/BBC-broadcasts a message m" when it invokes BRB/BBC-broadcast(m), and p_i "BRB/BBC-delivers a message m from p_s" when it manage the BRB/BBC-deliver(s, m) event.

We remark that both BRB and BBC primitives assume a *static process failure model* where every process is permanently correct or faulty.

A.1 Byzantine Reliable Broadcast (BRB)

The BRB communication primitive guarantees the following properties:

- *Validity*: If a correct process p_s BRB-broadcasts a message m, then every correct process eventually BRB-delivers m from p_s.
- *No duplication*: Every correct process BRB-delivers at most one message from p_s.
- *Integrity*: If some correct process BRB-delivers a message m from p_s and process p_s is correct, then m was previously BRB-broadcast by p_s.

- *Consistency*: If some correct process BRB-delivers a message m from p_s and another correct process BRB-delivers a message m' from p_s, then $m = m'$.
- *Totality*: If some message is BRB-delivered by any correct process, every correct process eventually BRB-delivers a message.

A.2 Byzantine Broadcast Channel (BBC)

The BBC communication primitive guarantees the following properties:

- *Validity*: If a correct process p_s BBC-broadcasts a message m, then every correct process eventually BBB-delivers m from p_s.
- *No duplication*: No correct process BBC-delivers a message m from p_s more than once.
- *Integrity*: If some correct process BBC-delivers a message m from p_s and process p_s is correct, then m was previously BBC-broadcast by p_s.
- *Agreement*: If some correct process BBC-delivers a message m from p_s then every correct process eventually delivers message m from p_s.

B $\mathcal{P}_{MBBC-RB}$ execution examples

We detail in this Section several execution examples for the $\mathcal{P}_{MBBC-RB}$ protocol defined in Section 6. Given what claimed in Theorem 16, we assume that the correctness conditions for our protocol, i.e. a $\langle$SYNC, S-MOB$^+$, $\mathcal{O}_{FFA}\rangle$ system and $n > 5f$, are satisfied in all of the provided examples. We detail one example where the source is correct and two in which the source is faulty.

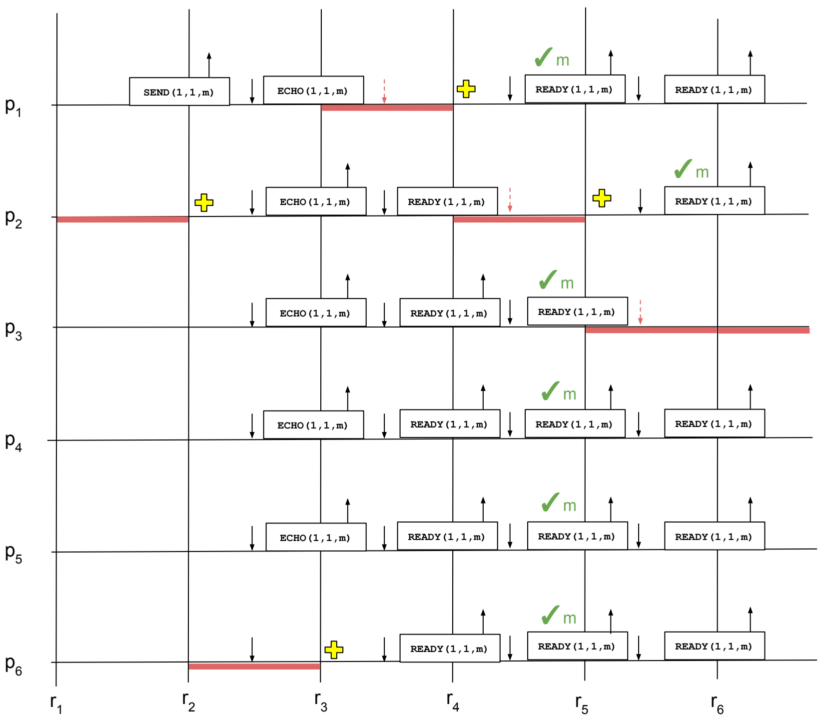


Figure 2 An execution of $\mathcal{P}_{MBBC-RB}$ with a correct source and $f = 1$.

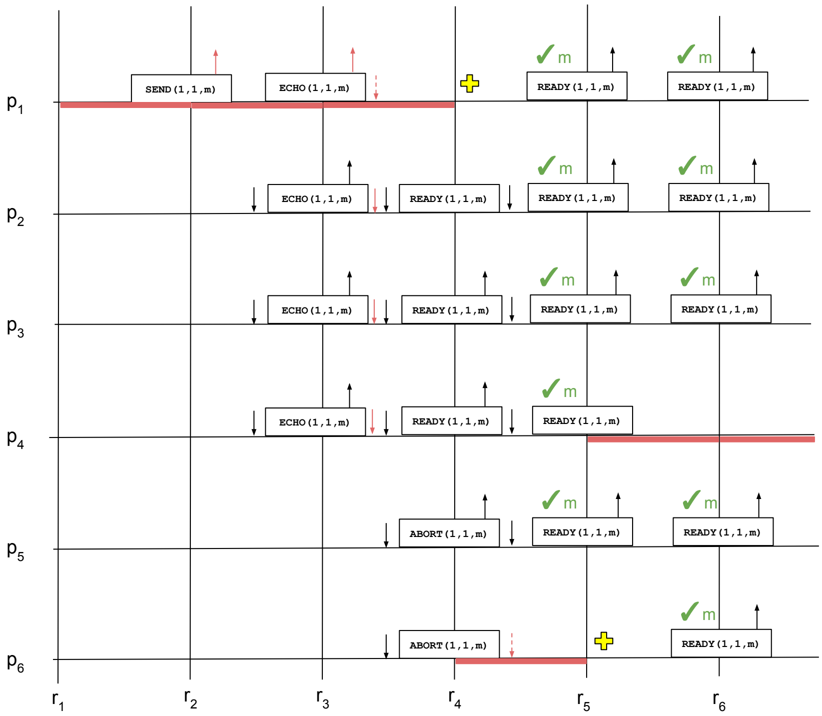


Figure 3 An execution of $\mathcal{P}_{MBBC-RB}$ with a faulty source, $f = 1$ and all infinitely often correct processes delivering.

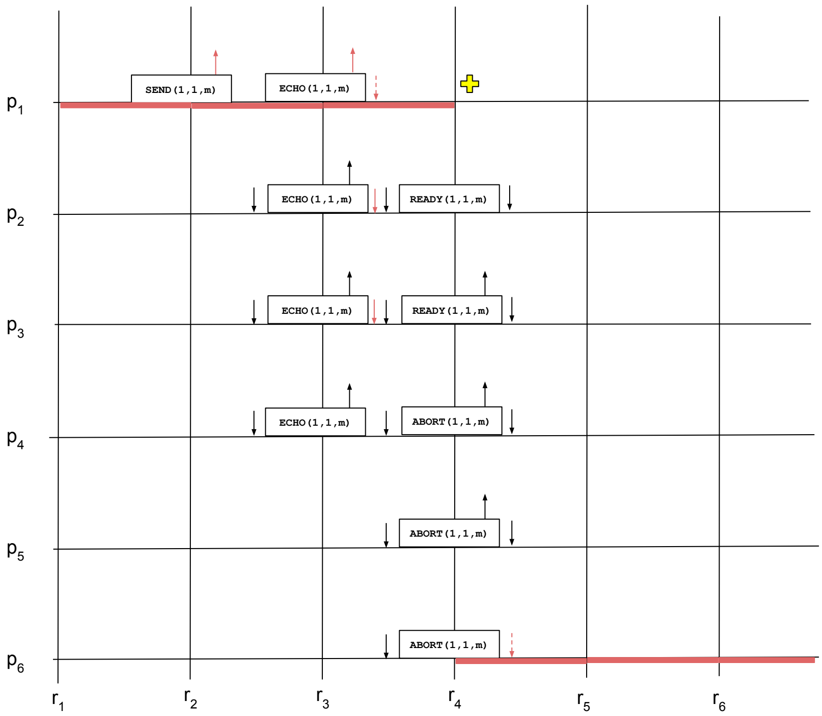


Figure 4 An execution of $\mathcal{P}_{MBBC-RB}$ with a faulty source, $f = 1$ and no infinitely often correct process delivering.

In the execution example in Figure 2, the correct source p_1 starts the MBBC-Broadcast preparing the related SEND message in round r_1, that is P2P-sent to all processes in round r_2 ($\Delta_b = 2$). Process p_2 is faulty in round r_1, then the mobile agent moves to process p_6 in round r_2. All processes but f are correct in round r_2, thus they receive the SEND message from p_1 and generate the related ECHO message. Such message is then P2P-sent to all peers by at least $n - 2f$ processes during the *send* phase in round r_3 (at most f processes could have been faulty in round r_2, p_6 in our example, and at most f processes could become faulty in round r_3, p_1 in our example where the mobile agent moves in round r_3). It follows that $n - f$ processes reach the quorum of ECHO messages generating the related READY message. Again, at least $n - 2f$ processes are correct in round r_4, P2P-send the READY message and deliver the associated payload from p_1, m, during the *compute* phase of the same round. The processes that were faulty in round r_4, p_2 in our example, deliver the message at the first round $r_k > r_4$ they get correct, because all processes that are correct in a round $r_j > r_4$ diffuse the associated READY message.

The only MBBC property that mobile agents may attempt to invalidate in a execution of $\mathcal{P}_{MBBC-RB}$ is the *Agreement* property: the *No duplication* is guaranteed by the *if* statement at line 40 in Algorithm 1 and both *Validity* and *Integrity* consider a correct source. Any source must P2P-send a well-formed SEND message (i.e., with valid source id and round label) to make a correct process proceed in the protocol to deliver a payload m. If the SEND message is P2P-sent to all correct processes, then all Δ_c-infinitely often correct processes will eventually deliver m, as shown in the previous execution, satisfying the MBBC specification. It follows that a Byzantine source must not P2P-send the SEND message to some processes. This behavior has two possible outcomes in our protocol: either all correct processes MBBC-deliver the diffused message or no correct process does it. Let us assume that the mobile agent commands p_1 to P2P-send the SEND message to $\lfloor (n-f)/2 \rfloor - f$ processes, in order to control which ones will proceed in the $\mathcal{P}_{MBBC-RB}$ protocol generating the READY message in round r_4.

In the execution depicted in Figure 3, process p_1 is a faulty source that attempts to prevent the *Agreement* property of MBBC from being satisfied. Specifically, it P2P-sends the ECHO message only to part of the processes, process p_2, p_3, and p_4, that reach the quorum required to generate the READY message. In this case, processes p_5 and p_6 generate the ABORT message but only f of them, namely p_5, P2P-send it, thus blocking no correct process from proceeding in the MBBC-delivery of m from p_1. Nonetheless, in this case more than $2f$ processes are correct and P2P-send the READY message in round r_4. It follows that all Δ_c-infinitely often correct processes eventually deliver the associated payload m.

Differently from the previous example, in the execution in Figure 4 process p_1 sends the ECHO message to processes p_2 and p_3. It follows that all other correct processes, p_4, p_5, and p_6, generate the ABORT message. At most f of them, process p_6 in the example, can be blocked from P2P-sending the ABORT message. It follows that more than f processes diffuse to all correct ones the ABORT message and thus no process delivers the associated payload m. It follows that the specification is not violated in such execution.