Causal Ordering Properties of Byzantine Reliable Broadcast Primitives

Anshuman Misra
University of Illinois at Chicago, USA
amisra7@uic.edu

Ajay D. Kshemkalyani
University of Illinois at Chicago, USA
ajay@uic.edu

***Abstract*—In this paper, we examine the inherent properties of the Byzantine Reliable Broadcast (BRB) primitive as pertain to the ability to provide causal ordering. We prove the following results. First, we analyze Bracha's BRB algorithm and show that under the failure-free model, safety is guaranteed across broadcasts. Second, we also prove that Bracha's BRB algorithm guarantees safety across broadcasts under the crash failure model tolerating any number of crash failures. Third, we prove that Bracha's BRB algorithm cannot provide weak or strong safety under the Byzantine failure model. Fourth, we prove that neither the Imbs-Raynal BRB protocol nor any (2,*)-round BRB protocol can provide causal order even if all processes are correct, and they must incur additional latency to causally order messages at a higher layer. The inherent causal ordering properties of Bracha's BRB can be of use under favourable circumstances in practical applications, given the widespread adoption of the protocol.**

***Index Terms*—Byzantine Reliable Broadcast, Byzantine fault-tolerance, Happens Before, Causality, Asynchronous message-passing**

I. Introduction

Causality is a critical tool for designing and reasoning in distributed systems [1]. Theoretically causality is defined by the *happens before* [2] relation on the events in a distributed execution. In practice, logical clocks [3] are used to timestamp events and messages, to track causality. If message $m1$ causally precedes $m2$ then $m1$ must be delivered before $m2$ at all common destinations to enforce *causal order* [4]. Causal ordering ensures that concurrent updates do not render data invalid in distributed systems. Applications of causal ordering include implementing distributed shared memory over message passing, distributed databases, collaborative applications, social networks, multiplayer online gaming, group editing of documents, event notification systems, and distributed virtual environments.

Causal ordering of messages under the Byzantine failure model is studied in [5], which considered causal broadcasts, and in [6]–[8] which considered unicasts, multicasts, and broadcasts. The works in [9]–[12] relied on broadcasts for causal consistency of replicated data under Byzantine behavior. The notions of *strong safety* and *weak safety* for causal ordering of messages were defined in [6]. Weak safety is essentially the weakest possible form of safety that can exist under the Byzantine failure model. It only considers messages that are both sent by and delivered to correct processes. Strong safety is the regular safety property that considers all messages regardless of whether they are sent/delivered by Byzantine processes. It was proved in [6] that an algorithm to causally order broadcast messages in the presence of Byzantine processes under the strong safety condition cannot exist. Therefore, the notion of weak safety becomes critical in identifying the merits of any Byzantine Causal Broadcast primitive. In [5], a Byzantine Causal Broadcast primitive that ensures weak safety (defined in section II) was developed as a layer above Bracha's Byzantine Reliable Broadcast (BRB) abstraction [13], [14].

BRB is a fundamental underlying abstraction in designing distributed software systems. BRB essentially guarantees that all processes deliver the exact same set of messages in the presence of Byzantine adversaries. Bracha's BRB primitive has been widely used as a fundamental building block in distributed protocols [15], [16]. Recent work has shown the utility of Bracha's BRB for implementing cryptocurrencies [17]–[19]. Bracha's BRB also serves as a reference point for further theoretical work on reliable broadcast, such as the recent work under the message adversarial model where Byzantine channels were considered in addition to Byzantine processes [20]. Despite its popularity, Bracha's BRB has not been analyzed for its intrinsic properties related to causal ordering of the reliable broadcasts. Another BRB protocol is by Imbs-Raynal [21], and yet another is that recently introduced by Abraham et al. [22].

The contributions of this paper are as follows:

1) Given the widespread practical and theoretical adoption of Bracha's BRB protocol, it is important to study the causal ordering properties intrinsically provided by it. We analyze Bracha's BRB protocol under the fault-free, the crash failure, and the Byzantine fault models.
    a) We prove that Bracha's BRB protocol implicitly causally orders broadcast messages under the strong and weak safety conditions in the presence of correct processes.
    b) We prove that Bracha's BRB protocol causally orders broadcast messages under strong safety and weak safety in the crash failure model with any number of crashed processes.
    c) We prove that a single Byzantine process can prevent Bracha's BRB protocol from causally ordering broadcast messages, thereby attacking weak safety.

979-8-3503-9730-7/22/$31.00 ©2022 IEEE

2) We also prove that neither the Imbs-Raynal BRB protocol [21] nor any (2,*)-round latency BRB protocol, defined as one that has a good-case (where the broadcaster is a correct process) latency of 2 message hops (for correct processes to deliver a message) [22], can provide causal order even if all processes are correct, and must incur additional latency to causally order messages at a higher layer.

The inherent causal ordering properties of Bracha's BRB will be of use under favourable circumstances in practical applications, given the widespread adoption of the protocol. In particular, when during periods of execution the system does not exhibit Byzantine behavior, no extra delays will be incurred at a higher layer above Bracha's BRB to provide strong and weak safety of Byzantine-tolerant causal broadcast. In contrast, even during periods of execution when the system exhibits fault-free behavior, the Raynal-Imbs algorithm and any (2,*)-round latency BRB algorithm will incur delays at the layer above to provide strong or weak safety of Byzantine-tolerant causal broadcast.

Roadmap. Section II gives the system model. Section III gives the properties to be satisfied by Byzantine Reliable Broadcast (BRB) and reviews Bracha's famous algorithm for BRB. Section IV gives our results on whether Bracha's BRB satisfies safety of causal order if there are no failures, only crash failures, and in the presence of Byzantine adversaries. Sections V and VI analyze the causal ordering properties of the Imbs-Raynal BRB algorithm and (2,*)-round latency BRB algorithms. Section VII gives the conclusions.

II. System Model

The distributed system is modelled as an undirected graph $G = (P, C)$. Here P is the set of processes communicating asynchronously over a geographically dispersed network. Let n be $|P|$. C is the set of communication channels over which processes communicate by message passing. The channels are assumed to be FIFO. G is a complete graph. For a message send event at time t_1, the corresponding receive event occurs at time $t_2 \in [t_1, \infty)$. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may exhibit arbitrary behaviour including crashing at any point during the execution. A Byzantine process cannot impersonate another process or spawn new processes.

Let e_i^x, where $x \geq 0$, denote the x-th event executed by process p_i. In order to deliver messages in causal order, we require a framework that captures causality as a partial order on a distributed execution. The *happens before* [2] relation, denoted $\rightarrow$, is an irreflexive, asymmetric, and transitive partial order defined over events in a distributed execution that captures causality. We do not allow the usage of cryptographic primitives due to their substantial overhead and due to assumptions about message latencies in existing protocols based on them.

Definition 1. *The happens before relation on events consists of the following rules:*

1) ***Program Order***: *For the sequence of events $\langle e_i^1, e_i^2, \ldots \rangle$ executed by process p_i, $\forall\ j, k$ such that $j < k$ we have $e_i^k \rightarrow e_i^j$.*
2) ***Message Order***: *If event e_i^x is a message send event executed at process p_i and e_j^y is the corresponding message receive event at process p_j, then $e_i^x \rightarrow e_j^y$.*
3) ***Transitive Order***: *Given events e and e'' in execution trace α, if $\exists\ e' \in \alpha$ such that $e \rightarrow e' \wedge e' \rightarrow e''$ then $e \rightarrow e''$.*

Next, we define the happens before relation $\rightarrow$ on the set of all application-level messages R.

Definition 2. *The happens before relation on messages consists of the following rules:*

1) *The set of messages delivered from any $p_i \in P$ by a process is totally ordered by $\rightarrow$.*
2) *If p_i sent or delivered message m before sending message m', then $m \rightarrow m'$.*
3) *If $m \rightarrow m' \wedge m' \rightarrow m''$ then $m \rightarrow m''$.*

Definition 3. *The causal past of message m is denoted as $CP(m)$ and defined as the set of messages in R that causally precede message m under $\rightarrow$.*

We require an extension of the happens before relation on messages to accommodate the possibility of Byzantine behaviour. We present a partial order on messages called *Byzantine happens before*, denoted as $\xrightarrow{B}$, defined on S, the set of all application-level messages that are both sent by and delivered at correct processes in P.

Definition 4. *The Byzantine happens before relation consists of the following rules:*

1) *The set of messages delivered from any correct process $p_i \in P$ by any correct process is totally ordered by $\xrightarrow{B}$.*
2) *If p_i is a correct process and p_i sent or delivered message m (to/from another correct process) before sending message m' to a correct process, then $m \xrightarrow{B} m'$.*
3) *If $m \xrightarrow{B} m' \wedge m' \xrightarrow{B} m''$ then $m \xrightarrow{B} m''$.*

The Byzantine causal past of a message is defined as follows:

Definition 5. *The Byzantine causal past of message m, denoted as $BCP(m)$, is defined as the set of messages in S that causally precede message m under $\xrightarrow{B}$.*

The correctness of a Byzantine causal order broadcast is specified on $(R, \rightarrow)$ and $(S, \xrightarrow{B})$. We now define the correctness criteria that a causal ordering algorithm must satisfy. Ideally, strong safety and liveness should be satisfied because as shown in [6], strong safety is desirable over weak safety for application semantics.

Definition 6. ***Weak Safety [6]:*** *$\forall m' \in BCP(m)$ such that m' and m are sent to the same correct process(es), no correct process delivers m before m'.*

Definition 7. ***Strong Safety [6]:*** *$\forall m' \in CP(m)$ such that m' and m are sent to the same correct process(es), no correct process delivers m before m'.*

Definition 8. ***Liveness:*** *Each message sent by a correct process to another correct process will be eventually delivered.*

When $m \xrightarrow{B} m'$, then all processes that sent messages along the causal chain from m to m' are correct processes. This definition is different from $m \rightarrow_M m'$ [5], where M was defined as the set of all application-level messages delivered at correct processes, and $MCP(m')$ could be defined as the set of messages in M that causally precede m'. When $m \rightarrow_M m'$, then all processes, *except the first*, that sent messages along the causal chain from m to m' are correct processes. The definition of $\xrightarrow{B}$ (Definition 4) allows for the purest notion of safety – weak safety. Our definition of $\xrightarrow{B}$ and $\rightarrow_M$ [5] both make the assumption that from the second to the last process that send messages along the causal chain from m to m', are correct processes.

III. Byzantine Reliable Broadcast

Byzantine Reliable Broadcast (BRB) has traditionally been defined based on Bracha's Byzantine Reliable Broadcast (BRB) [13], [14]. For this algorithm to work, the number of Byzantine processes, t must less than one third of the overall number of processes, n ($t \leq \lfloor (n-1)/3 \rfloor$). When a process does a broadcast, it invokes `br_broadcast()` and when it is to deliver such a message, it executes `br_deliver()`. In the discussion below, it is implicitly assumed that a message is uniquely identified by a (sender ID, sequence number) tuple. BRB satisfies the following properties.

- Validity: If a correct process `br_delivers` a message m from a correct process p_s, then p_s must have executed `br_broadcast(m)`.
- Integrity: For any message m, a correct process executes `br_deliver` at most once.
- Self-delivery: If a correct process executes `br_broadcast(m)`, then it eventually executes `br_deliver(m)`.
- Reliability (or Termination): If a correct process executes `br_deliver(m)`, then every other correct process also (eventually) executes `br_deliver(m)`.

As causal broadcast is an application layer property, it runs on top of the BRB layer. Byzantine Causal Broadcast (BCB) is invoked as `bc_broadcast`(m) which in turn invokes `br_broadcast`(m') to the BRB layer. Here, m' is m plus some control information appended by the BCB layer. A `br_deliver`(m') from the BRB layer is given to the BCB layer which delivers the message m to the application via `bc_deliver(m)` after the processing in the BCB layer. The control information is abstracted by the *causal barrier* [5], [23] which tracks the immediate or direct dependencies and is bounded by $O(n)$. In addition to the BCB-layer counterparts of the properties satisfied by BRB, BCB must satisfy safety and liveness. Liveness and weak safety can be satisfied as given by the protocol in [5]. The BRB protocol by Bracha is given in Algorithm 1. Actually, Algorithm 1 gives a minor variant of Bracha's protocol that enhances the original with local sequence numbers added to broadcasts. We prove in Lemma 3 and Theorem 3 that this BRB broadcast does not provide causal ordering guarantees under the Byzantine failure model.

Algorithm 1: Bracha's Byzantine Reliable Broadcast Protocol, $t \leq \lfloor (n-1)/3 \rfloor$

```
Data: Each process $p_i$ maintains an integer $seq_i$
      initialized to 0. $seq_i$ is used to provide sequence
      numbers to outgoing messages.
1  when the application is ready to BRB broadcast
     message $m$:
2  broadcast INIT($p_i$,$seq_i$,$m$)
3  $seq_i = seq_i + 1$
4  when INIT($p_j$,$seq_j$,$m$) is received for the first time:
5  broadcast ECHO($p_j$,$seq_j$,$m$)
6  when ECHO($p_j$,$seq_j$,$m$) is received and
     READY($p_j$,$seq_j$,$m$) not yet broadcast:
7  if ECHO($p_j$,$seq_j$,$m$) received from more than
     $(n+t)/2$ different processes then
8      broadcast READY($p_j$,$seq_j$,$m$)
9  when READY($p_j$,$seq_j$,$m$) is received:
10 if READY($p_j$,$seq_j$,$m$) received from $(t+1)$ different
     processes $\wedge$ READY($p_j$,$seq_j$,$m$) not yet broadcast
     then
11     broadcast READY($p_j$,$seq_j$,$m$)
12 if READY($p_j$,$seq_j$,$m$) received from $(2t+1)$ different
     processes then
13     br_deliver(m)
```

IV. Causal Ordering Properties of Bracha's BRB Protocol

In this section we formally analyze the causal ordering properties of Bracha's BRB. Lemma 1 and Theorem 1 prove that Bracha's BRB causally orders broadcast messages under strong/weak safety (in the absence of Byzantine processes strong safety is the same as weak safety) in the fault-free setting. Lemma 2 and Theorem 2 prove that Bracha's BRB provides a causal ordering guarantee in the crash failure model as well. Lemma 3 and Theorem 3 prove that Bracha's BRB cannot guarantee causal order under the weak safety condition across broadcast messages in the presence of a single Byzantine process. (In [6], it was proved that no algorithm can order broadcast messages under strong safety in the presence of even one Byzantine process). However, since Bracha's BRB protocol intrinsically provides causal order in the absence of Byzantine processes as we show, the Byzantine processes will have to launch an attack on weak safety in order to prevent this causal ordering.

A. Behaviour in a Fault-free Setting

Lemma 1. *For correct processes p_i, p_j and p_k executing Algorithm 1: If p_i executes* `br_broadcast(`m_1`)` *which is delivered at p_j, then p_j executes* `br_broadcast(`m_2`)`*, p_k will deliver m_1 before m_2 in a fault-free setting.*

Proof. In order for p_j to deliver m_1, it must receive at least $(2t+1)$ READY(m_1) messages. Let C_1 be the set of processes that have broadcast READY(m_1). $|C_1| = (2t + 1 + x)$, $0 \leq x \leq t$. Let the remaining set of processes be in set C_2. $|C_2| = (t - x)$. Now, p_j BRB broadcasts m_2. Since all processes in C_1 have broadcasted READY(m_1) before p_j BRB broadcasted m_2, due to the FIFO property, all outgoing channels from processes in C_1 will have READY(m_1) ahead of READY(m_2). Due to FIFO channels, all channels from processes in C_1 to p_k will deliver READY(m_1) before READY(m_2) at p_k. Processes in C_2 can broadcast READY(m_2) messages when they receive:

1) Either $(2t + 1)$ ECHO(m_2) messages
2) Or $(t + 1)$ READY(m_2) messages

We prove that all processes in C_2 broadcast READY(m_1) before READY(m_2), by induction on τ, where p_τ is the τ^{th} process in C_2 to broadcast READY(m_2). The induction hypothesis is that p_τ broadcasts READY(m_1) before broadcasting READY(m_2).

1) The first process $p_l \in C_2$ to broadcast READY(m_2), will need to receive either at least $(t+1)$ ECHO(m_2) or $(t+1)$ READY(m_2) messages from processes in C_1 (p_l may receive at most t ECHO(m_2) messages from within C_2). However, since all processes in C_1 have broadcasted READY(m_1) before p_j BRB broadcasted m_2, due to the FIFO property of channels, p_l will receive READY(m_1) before READY(m_2) or ECHO(m_2) from each process in C_1. This means that p_l is guaranteed to receive $(t+1)$ READY(m_1) messages before $(t+1)$ READY(m_2) messages and $(2t+1)$ ECHO(m_2) messages. Therefore p_l will broadcast READY(m_1) before READY(m_2).

2) Assume the induction hypothesis that p_τ, the τ^{th} process in C_2 to broadcast READY(m_2), broadcasts READY(m_1) before broadcasting READY(m_2).

3) $p_{\tau+1}$ is the $(\tau + 1)^{th}$ process in C_2 to broadcast READY(m_2). It needs $(2t + 1)$ ECHO(m_2) or $(t + 1)$ READY(m_2) messages in order to broadcast READY(m_2). All channels within the system directed to $p_{\tau+1}$ have the one of the following in transit:

   a) READY(m_1) ahead of READY(m_2) (from the induction hypothesis, this is true for τ channels from processes in C_2 to p_r). These channels may or may not have ECHO(m_2) ahead of READY(m_1). Therefore, from these channels, τ ECHO(m_2) messages may arrive before any READY(m_1) messages arrive and at most $(\tau-1)$ READY(m_2) messages may arrive before τ READY(m_1) messages.
   b) ECHO(m_1) and READY(m_1) ahead of ECHO(m_2) and READY(m_2) ($(2t + 1 + x)$ channels from processes in C_1 to p_r). Therefore, from these channels, t ECHO(m_2) messages may arrive before any $(t + 1)$ READY(m_1) messages arrive and at most t READY(m_2) messages may arrive before $(t + 1)$ READY(m_1) messages.
   c) ECHO(m_2) in transit ($(t - x - \tau)$ channels from processes in C_2 to p_r). These are the processes that are yet to broadcast READY(m_2). (At this point in time, τ processes have broadcast READY(m_2)).

   Combining (a), (b) and (c), it becomes clear that $p_{\tau+1}$ can receive at most t READY(m_2) and $(\tau + (t) + (t - x - \tau)) = (2t - x)$ ECHO(m_2) messages before receiving $(t + 1)$ READY(m_1) messages. Therefore, $p_{\tau+1}$ broadcasts READY(m_1) before READY(m_2).

In order for p_k to deliver m_2 before m_1, it must receive $(2t + 1)$ READY(m_2) messages before $(2t + 1)$ READY(m_1) messages. Since, all processes in the system broadcast READY(m_1) before READY(m_2), p_k can receive at most $2t$ READY(m_2) messages before it receives $(2t + 1)$ READY(m_1) messages. Therefore, p_k is guaranteed to deliver m_1 before m_2. The lemma follows. ☐

Theorem 1. *Algorithm 1 ensures causal order (strong/weak safety) across broadcast messages in the absence of Byzantine processes.*

Proof. From Lemma 1, if p_i executes `br_broadcast(`m_1`)` which is delivered at p_j, then p_j executes `br_broadcast(`m_2`)`, p_k will deliver m_1 before m_2. Similarly, by Lemma 1 if a process broadcasts m_3 after delivering m_2, it will be delivered after m_2 at all other processes. Using transitivity, given any causal chain of messages $m_0 \rightarrow m_1 \rightarrow ... \rightarrow m_q$, m_0 is guaranteed to be delivered before m_q at all processes. Therefore, Algorithm 1 ensures causal order across broadcasts in the absence of Byzantine processes. This proves the theorem. ☐

B. Behaviour under Crash Failures

Lemma 2. *For correct processes p_i, p_j and p_k executing Algorithm 1: If p_i executes* `br_broadcast(`m_1`)` *which is delivered at p_j, then p_j executes* `br_broadcast(`m_2`)`*, p_k will deliver m_1 before m_2 in the crash failure fault model if less than or equal to $(n - 1)$ processes crash.*

Proof. Let p_i broadcast m_1 which is delivered to p_j which then broadcasts m_2. We show that no process delivers m_2 before m_1 in the face of any number of crash failures.

When p_i's message m_1 is delivered to p_j, it has received at least $(2t + 1)$ READY(m_1) messages. Based on this, the strongest assumptions that are guaranteed to hold are:

- at least $(2t + 1)$ processes received either $(2t + 1)$ ECHO(m_1) messages or at least $(t + 1)$ READY(m_1) messages, and reached the step of broadcast READY(m_1).

- Of these at least $(2t+1)$ processes, if READY(m_1) is not sent on a outgoing channel, then neither READY(m_2) nor ECHO(m_2) will be sent on that outgoing channel as the process has crashed.

At the time of delivery to p_j, let C_1 be defined as the set of processes that have reached the step of broadcasting READY(m_1). Let $|C_1| = 2t+1+x$, where $t \geq x \geq 0$. Let C_2 be the set of the remaining $(t-x)$ processes. We show that all processes in C_1 and in C_2 are guaranteed to broadcast READY(m_1) before READY($m2$). Based on the above mentioned assumptions, all processes in C_1 (including processes that may have crashed) start broadcasting READY(m_1) prior to p_j broadcasting INIT(m_2). Therefore, due to the FIFO property of channels, all outgoing channels from processes in C_1 (to processes in C_1 and in C_2) will contain either READY(m_1) before READY(m_2) or READY(m_1) and not READY(m_2) (this may happen in the event of a crash failure), or neither. So all processes will receive READY(m_1) before receiving READY(m_2) from a process in C_1.

All $(t-x)$ processes in C_2 may receive INIT(m_2) before INIT(m_1) and broadcast ECHO(m_2) before ECHO(m_1). No process in C_2 can broadcast READY(m_2) until it receives $(2t+1)$ ECHO(m_2) messages or $(t+1)$ READY(m_2) messages. $(2t+1)-(t-x) = (t+1+x)$ ECHO(m_2) are required from processes in C_1 by the first process in C_2 in order to broadcast READY(m_2). On $(t+1+x)$ (C_1,C_2) channels, when $(t+1+x)$ ECHO(m_2) messages arrive, READY(m_1) messages have arrived earlier. This is because all processes in C_2 arrive at the step of broadcasting READY(m_1) before p_j broadcasts INIT(m_2). So the first process in C_2 to broadcast READY(m_2) would first broadcast READY(m_1). We now show by induction that the α^{th} process in C_2 to broadcast READY(m_2) will broadcast READY(m_1) before READY(m_2). Having shown the induction hypothesis is true for the base case $\alpha = 1$, we now show for $\alpha > 1$. Assume the hypothesis is true for α. First note that for any value of α, as in the case for $\alpha = 1$, it is impossible for any process in C_2 to obtain the required number of $(t+1+x)$ ECHO(m_2) messages from processes in C_1 to broadcast READY(m_2) before getting at least $(t+1)$ READY(m_1) messages from processes in C_1. The $(\alpha+1)^{th}$ process (for $\alpha \geq 1$) to broadcast READY(m_2) can get α READY(m_2) messages from processes in C_2, however, by the induction hypothesis, it would have received α READY(m_1) messages due to FIFO channels. It will also require $t+1-\alpha$ READY(m_2) messages from C_1. However, on these channels from C_1, READY(m_1) will be received before before READY(m_2) as shown earlier. Therefore, the $(\alpha+1)^{th}$ process will also broadcast READY(m_1) before READY(m_2).

As all processes broadcast READY(m_1) before READY(m_2), FIFO channels guarantee that each process receives $(2t+1)$ READY(m_1) messages before $(2t+1)$ READY(m_2) messages. Therefore, p_k will deliver m_1 before m_2. □

Theorem 2. *Algorithm 1 ensures causal order (strong/weak safety) across broadcast messages in the crash failure model as long as less than $(n-1)$ processes crash.*

Proof. From Lemma 2, if p_i executes `br_broadcast(`m_1`)` which is delivered at p_j, then p_j executes `br_broadcast(`m_2`)`, p_k will deliver m_1 before m_2. Similarly, by Lemma 2 if a process broadcasts m_3 after delivering m_2, it will be delivered after m_2 at all other processes. Based on this, given any causal chain of messages $m_0 \to m_1 \to ... \to m_q$, m_0 is guaranteed to be delivered before m_q at all processes. Therefore, Algorithm 1 ensures causal order across broadcasts in the presence of crash failures. This proves the theorem. □

Corollary 1. *Bracha's BRB protocol will guarantee the safety condition under upto $(n-1)$ crash failures, however if the number of crash failures exceed $\lfloor (n-1)/3 \rfloor$, liveness may not be guaranteed because some (or all) correct processes may not receive the required number of READY messages to deliver the application message that has been broadcasted.*

C. Behaviour under Byzantine Failure Model

Lemma 3. *For correct processes p_i, p_j and p_k executing Algorithm 1: If p_i executes `br_broadcast(`m_1`)` which is delivered at p_j, then p_j executes `br_broadcast(`m_2`)`, p_k may deliver m_2 before m_1 if a single Byzantine process is present.*

Proof. This is a proof by counter-example. Let p_l be the single Byzantine process. There are $3t$ correct processes in the system. Consider the following (ordered) sequence of events:

1) p_i BRB broadcasts m_1.
2) p_j receives $(2t+1)$ READY(m_1) messages. At least $2t$ correct processes must have broadcast READY(m_1) for this to be possible. Let these $2t$ correct processes be in set C_1 of correct processes that have broadcast READY(m_1) by the time p_j receives $2t{+}1$ READY(m_1) messages, where $|C_1| = (2t+x)$ $(0 \leq x \leq t)$. Let the remaining correct processes (those who have not broadcast READY(m_1)) be in set C_2, $|C_2| = (t-x)$. $|C_1| + |C_2| = 3t$. The Byzantine process p_l has sent ECHO(m_1) and READY(m_1) messages only to processes in C_1 (which includes p_j).
3) p_j delivers m_1.
4) p_j BRB broadcasts m_2.
5) p_l, the Byzantine process had previously sent ECHO(m_1) and READY(m_1) only to processes in C_1; now it broadcasts ECHO(m_2) and READY(m_2).
6) All processes in C_2 receive INIT(m_2) (whereas INIT(m_1) is yet to arrive), broadcast ECHO(m_2).
7) All processes in C_1 broadcast ECHO(m_2).
8) All processes in C_1 receive $(2t+1)$ ECHO(m_2) messages and broadcast READY(m_2).
9) All channels from processes in C_1 to processes in C_2 now have in left to right order the following messages : [ECHO(m_1), READY(m_1), ECHO(m_2), READY(m_2)]. These messages will be delivered in left to right order due to FIFO channels.

10) Let t channels get flushed from all processes in C_1 to all processes in C_2. All other messages in channels from C_1 to C_2 are in transit. All processes in C_2 have now received $(t+1)$ READY(m_2) (t messages from correct processes and 1 message from Byzantine process p_l) messages and t READY(m_1) messages (all from correct processes). All processes in C_2 broadcast READY(m_2).
11) All channels from processes in C_2 to processes in C_2 ($(t-x)$ channels per process) now have in left to right order: [ECHO(m_2), READY(m_2)].
12) Let all the $(t-x)$ channels from processes in C_2 to $p_k \in C_2$ now get flushed.
13) p_k now has $(2t+1-x)$ READY(m_2) messages ($(t-x)$ READY(m_2) messages from processes in C_2, t READY(m_2) messages from processes in C_1 and 1 READY(m_2) message from Byzantine p_l) and t READY(m_1) messages (All from processes in C_1). When $x = 0$, p_k has received exactly $(2t+1)$ READY(m_2) messages and will deliver m_2.
14) Due to the Reliability property of BRB broadcast p_k eventually delivers m_1.

The lemma follows. □

Theorem 3. *Algorithm 1 does not ensure causal order (under strong or weak safety) across broadcast messages in the presence of Byzantine processes.*

Proof. From Lemma 3, if p_i executes `br_broadcast`(m_1) which is delivered at p_j, then p_j executes `br_broadcast`(m_2), p_k may deliver m_2 before m_1. Clearly, $m_1 \xrightarrow{B} m_2$. Therefore, Algorithm 1 does not guarantee causal order across broadcast messages in the presence of Byzantine processes. □

V. Causal Ordering Properties of Imbs-Raynal BRB Protocol

Here, we analyze and show that the BRB protocol proposed by Imbs and Raynal in [21] does not provide causal ordering even in the fault-free setting. The Imbs-Raynal protocol differs from Bracha's protocol in the following aspects:

1) Imbs-Raynal operates in two phases whereas Bracha's protocol is a three phase protocol. Therefore, there is reduced latency in message delivery.
2) Imbs-Raynal protocol has (n^2-1) protocol messages per broadcast as opposed to $(2n^2-n-1)$ protocol messages per broadcast in Bracha's protocol.
3) Imbs-Raynal protocol ($t \leq \lfloor(n-1)/5\rfloor$) is less resilient than Bracha's protocol ($t < \lfloor(n-1)/3\rfloor$).

The Imbs-Raynal protocol is described in Algorithm 2, and formal proof regarding its causal ordering properties are provided in Lemma 4 and Theorem 4.

Lemma 4. *For correct processes p_i, p_j and p_k executing Algorithm 2: If p_i executes* `br_broadcast`(m_1) *which is delivered at p_j, then p_j executes* `br_broadcast`(m_2), *p_k may not deliver m_1 before m_2 in a fault-free setting.*

Algorithm 2: Imbs-Raynal Byzantine Reliable Broadcast Protocol, $t \leq \lfloor(n-1)/5\rfloor$

```
Data: Each process $p_i$ maintains an integer $seq_i$
      initialized to 0. $seq_i$ is used to provide sequence
      numbers to outgoing messages.

1  when the application is ready to BRB broadcast
     message $m$:
2    broadcast INIT($p_i$,$seq_i$,$m$)
3    $seq_i = seq_i + 1$

4  when INIT($p_j$,$seq_j$,$m$) is received for the first time
     and WITNESS($p_j$,$seq_j$,$-$) not yet broadcast:
5    broadcast WITNESS($p_j$,$seq_j$,$m$)

6  when WITNESS($p_j$,$seq_j$,$m$) is received:
7    if WITNESS($p_j$,$seq_j$,$m$) received from at least $(n-2t)$
       different processes and WITNESS($p_j$,$seq_j$,$m$) not yet
       broadcast then
8        broadcast WITNESS($p_j$,$seq_j$,$m$)
9    if WITNESS($p_j$,$seq_j$,$m$) received from $(n-t)$ different
       processes then
10       br_deliver(m)
```

Proof. This is a proof by counter-example. There are $n = 5t+1$ processes in the system. All processes are correct and follow Algorithm 2. Consider the following (ordered) sequence of events:

1) p_i BRB broadcasts m_1.
2) All processes except p_l and p_k receive INIT(m_1) and broadcast WITNESS(m_1).
3) p_j delivers m_1.
4) p_j BRB broadcasts m_2.
5) p_l receives INIT(m_2) from p_j and broadcasts WITNESS(m_2). At this point in time INIT(m_1) and all WITNESS(m_1) messages are still in transit on all incoming channels at p_l (therefore p_l has not yet broadcasted WITNESS(m_1)).
6) At this point in time no messages have arrived at p_k.
7) All processes except p_k deliver m_1 and m_2.
8) All channels except the channel from p_l to p_k now have WITNESS(m_1) followed by WITNESS(m_2) in transit.
9) WITNESS(m_2) gets delivered at p_k from p_l
10) $(n-t-1)$ incoming channels get flushed at p_k. This results in $(n-t)$ WITNESS(m_2) and $(n-t-1)$ WITNESS(m_1) messages getting delivered at p_k.
11) p_k delivers m_2.
12) Due to the BRB reliability property, p_k eventually delivers m_1

Therefore p_k may deliver m_2 before m_1. □

Theorem 4. *Algorithm 2 does not guarantee causal order (strong/weak safety) across broadcast messages in the failure-free setting.*

Proof. From Lemma 4, if p_i executes `br_broadcast`(m_1) which is delivered at p_j, then p_j executes `br_broadcast`(m_2), p_k may deliver m_2 before m_1. Clearly, $m_1 \rightarrow m_2$. Therefore, Algorithm 2 does not guarantee causal order across broadcast messages in the failure-free setting. □

Corollary 2. *Algorithm 2 cannot guarantee causal ordering in both the crash failure and Byzantine failure models.*

VI. Results for Good-Case Latency=2 BRB Algorithms

Good-case (bad-case) latency R_g (R_b) of a BRB algorithm is defined as the number of message hops or rounds it takes for the correct processes to deliver a message when the broadcaster is a correct (not necessarily correct) process [22]. A (R_g, R_b)-round BRB alogrithm has good case latency R_g and bad-case latency R_b. Bracha's BRB is a (3,4)-round algorithm where $n \geq 3t+1$ whereas the Imbs-Raynal BRB is a (2,3)-round algorithm where $n \geq 5t+1$. Abraham et al. [22] also proposed a (2,4)-round BRB algorithm where $n \geq 4t$.

Here we make a case that any (2,*)-BRB algorithm does not satisfy causal order even if the broadcasters are correct. Observe that *any conceivable* (2,*)-round BRB algorithm operates as follows in the good case. First, the broadcaster sends an INIT message to all processes. Then on receiving the first INIT message, a process sends a WITNESS message to all processes. Finally on the good path, a process BRB-delivers a message when it receives WITNESS messages from $n-t$ different processes, (to also accommodate the bad-case). The counter-example we gave for the Imbs-Raynal algorithm in Section V in Lemma 4 also applies to such a (2,*)-round BRB alogrithm by considering the good-case path. This leads to the following results.

Lemma 5. *For correct processes p_i, p_j and p_k executing any (2,*)-round BRB algorithm: If p_i executes* `br_broadcast`(m_1) *which is delivered at p_j, then p_j executes* `br_broadcast`(m_2), *p_k may not deliver m_1 before m_2 in a fault-free setting.*

Theorem 5. *Any (2,*)-round BRB algorithm does not guarantee causal order (strong/weak safety) across broadcast messages in the failure-free setting.*

Corollary 3. *A (2,*)-round BRB algorithm cannot guarantee causal ordering in both the crash failure and Byzantine failure models.*

Significance: Although a (2,*)-round BRB algorithm has lower latency than Bracha's (3,4)-round BRB algorithm in the good-case, it cannot provide causal ordering even if all processes are correct. Thus, a higher-layer protocol is needed for Byzantine causal ordering and this will introduce additional variable latency. In contrast, Bracha's (3,4)-round BRB protocol which has higher good-case latency is guaranteed to incur no additional delays to enfocrce causal order when the processes are correct or even under the crash-failure model.

VII. Conclusion

This paper analyzed Bracha's BRB protocol which is one of the most popular protocols for reliable broadcast under the Byzantine failure model for satisfiability of causal ordering. We discovered that Bracha's BRB protocol has an inherent capability to causally order broadcasts when there are no failures. Further, this paper proved that the BRB protocol can also causally order broadcasts under the crash failure model with any number of crash failures. However, we showed that a single Byzantine process can mount an attack that prevents Bracha's BRB from providing causal order. We also proved that neither the Imbs-Raynal BRB protocol nor any (2,*)-round latency BRB protocol can provide causal order even if all processes are correct, and must incur additional latency to causally order messages at a higher layer.

The inherent causal ordering properties of Bracha's BRB will be of use under favourable circumstances in practical applications, given the widespread adoption of the protocol. In particular, when during periods of execution the system does not exhibit Byzantine behavior, no extra delays will be incurred at a higher layer above Bracha's BRB to provide weak safety of Byzantine-tolerant causal broadcast. In contrast, even during periods of execution when the system exhibits fault-free behavior, the Raynal-Imbs algorithm and any (2,*)-round latency BRB algorithm will incur delays at the layer above to provide weak safety of Byzantine-tolerant causal broadcast.

References

[1] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2011. [Online]. Available: https://doi.org/10.1017/CBO9780511805318

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM 21, 7*, pp. 558–565, 1978.

[3] F. Mattern, "Virtual time and global states of distributed systems," in *Parallel and Distributed Algorithms*. North-Holland, 1988, pp. 215–226.

[4] K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," *ACM Transactions on Computer Systems (TOCS)*, vol. 5, no. 1, pp. 47–76, 1987.

[5] A. Auvolat, D. Frey, M. Raynal, and F. Taïani, "Byzantine-tolerant causal broadcast," *Theoretical Computer Science*, vol. 885, pp. 55–68, 2021.

[6] A. Misra and A. D. Kshemkalyani, "Solvability of byzantine fault-tolerant causal ordering problems," in *Proc. 10th International Conference on Networked Systems, NETYS 2022, Virtual Event, 2022*, ser. Lecture Notes in Computer Science, M. Koulali and M. Mezini, Eds., vol. 13464. Springer, 2022, pp. 87–103. [Online]. Available: https://doi.org/10.1007/978-3-031-17436-0_7

[7] ——, "Causal ordering in the presence of byzantine processes," in *28th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2022.

[8] ——, "Byzantine fault-tolerant causal ordering," in *24th International Conference on Distributed Computing and Networking (ICDCN)*, 2023.

[9] K. Huang, H. Wei, Y. Huang, H. Li, and A. Pan, "Byz-gentlerain: An efficient byzantine-tolerant causal consistency protocol," *arXiv preprint arXiv:2109.14189*, 2021.

[10] L. Tseng, Z. Wang, Y. Zhao, and H. Pan, "Distributed causal memory in the presence of byzantine servers," in *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, 2019, pp. 1–8.

[11] A. van der Linde, J. Leitão, and N. M. Preguiça, "Practical client-side replication: Weak consistency semantics for insecure settings," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2590–2605, 2020.

[12] M. Kleppmann and H. Howard, "Byzantine eventual consistency and the fundamental limits of peer-to-peer databases," *arXiv preprint arXiv:2012.00472*, 2020.

[13] G. Bracha and S. Toueg, “Asynchronous consensus and broadcast protocols,” *J. ACM*, vol. 32, no. 4, p. 824–840, Oct. 1985. [Online]. Available: https://doi.org/10.1145/4221.214134

[14] G. Bracha, “Asynchronous byzantine agreement protocols,” *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987. [Online]. Available: https://doi.org/10.1016/0890-5401(87)90054-X

[15] L. R. Christian Cachin, Rachid Guerraoui, *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

[16] M. Raynal, *Fault-Tolerant Message-Passing Distributed Systems: An Algorithmic Approach*. Springer, 2018.

[17] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi, “The consensus number of a cryptocurrency,” in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 307–316.

[18] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y.-A. Pignolet, D.-A. Seredinschi, A. Tonkikh, and A. Xygkis, “Online payments by merely broadcasting messages,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 26–38.

[19] A. Auvolat, D. Frey, M. Raynal, and F. Taïani, “Money transfer made simple: a specification, a generic algorithm, and its proof,” *arXiv preprint arXiv:2006.12276*, 2020.

[20] T. Albouy, D. Frey, M. Raynal, and F. Taïani, “Asynchronous byzantine reliable broadcast with a message adversary,” *arXiv preprint arXiv:2205.09992*, 2022.

[21] D. Imbs and M. Raynal, “Trading off t-resilience for efficiency in asynchronous byzantine reliable broadcast,” *Parallel Processing Letters*, vol. 26, no. 04, p. 1650017, 2016.

[22] I. Abraham, L. Ren, and Z. Xiang, “Good-case and bad-case latency of unauthenticated byzantine broadcast: A complete categorization,” in *25th International Conference on Principles of Distributed Systems, OPODIS*, ser. LIPIcs, Q. Bramas, V. Gramoli, and A. Milani, Eds., vol. 217, 2021, pp. 5:1–5:20. [Online]. Available: https://doi.org/10.4230/LIPIcs.OPODIS.2021.5

[23] V. Hadzilacos and S. Toueg, “A modular approach to fault-tolerant broadcasts and related problems,” *Tech Report 94-1425, Cornell University*, p. 83 pages, 1994.