Solvability of Byzantine Fault-Tolerant Causal Ordering: Synchronous Systems Case

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

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

ABSTRACT

Causal ordering is widely used in distributed systems to maintain validity and correctness of data across concurrent updates. Previous work has shown that it is impossible to solve the causal ordering problem under the strong safety condition in cryptography-free Byzantine-prone systems. It has also been shown that it is impossible to solve deterministic causal ordering for unicasts/multicasts in asynchronous systems even under a weaker notion of safety called weak safety. However, inherently asynchronous (round-free) protocols solve causal ordering for unicasts/multicasts in synchronous systems under the weak safety condition. In this paper, we first examine the causal ordering problem under the notion of synchronous rounds. We examine whether causal ordering is solvable by simulating rounds in synchronous systems under fault-free, crash-failure and Byzantine failure models. We then provide a round-based synchronous algorithm for causal ordering of unicasts/multicasts/broadcasts under the strong safety condition. Finally, we provide an overall analysis of solvability of causal ordering in synchronous systems for a variety of system model settings.

CCS CONCEPTS

• **Computing methodologies → Distributed algorithms**; • **Security and privacy → Software and application security**; Public key encryption; • **Software and its engineering**;

KEYWORDS

byzantine fault-tolerance, causal order, unicast communication, causal multicast, synchronous message-passing

ACM Reference Format:
Anshuman Misra and Ajay D. Kshemkalyani. 2024. Solvability of Byzantine Fault-Tolerant Causal Ordering: Synchronous Systems Case. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3605098.3636063

1 INTRODUCTION

Causality is an important tool in understanding and reasoning about distributed executions [12]. Causality is defined by the *happens before* [6] relation on the set of events, and by extension, on the set of messages. Causal communication is essential for distributed applications because it provides the ability to concurrently execute operations while protecting correctness of the computation. In the broadcast case, reliable broadcast primitives can provide intrinsic causal ordering in the failure-free and crash failure settings, however, Byzantine processes can break this causal ordering [7]. In the case of atomic broadcast, while it provides a total ordering of messages, it does not preserve causal ordering. Under the Byzantine failure model, causal ordering has recently been studied in [1] for broadcast communication and an algorithm for causal ordering under the weak safety condition was presented. In [8, 10, 11] causal ordering was studied for unicast, multicast, as well as broadcast communication. In these papers, it has been established that causal ordering cannot be solved for multicasts/unicasts deterministically in asynchronous systems under Byzantine failures. Byzantine fault-tolerant deterministic algorithms for causal ordering of multicasts/unicasts in synchronous systems under the weak safety condition were provided in [10, 11]. In [8] it was shown that strong safety of causal ordering cannot be achieved in a cryptography-free setting. Although previous research has examined causal order of messages in both asynchronous and synchronous systems, the possibility of causal ordering under round based communication in synchronous systems has not been examined. In this paper, we establish a relationship between synchronous rounds and causal ordering. Specifically, we make the following contributions.

(1) We investigate the relationship between synchronous rounds and causal ordering. We discover that synchronous rounds inherently provide causal ordering accross messages in a failure-free system (Theorem 1).
(2) We also discover that synchronous rounds provide causal ordering even under crash failures (Theorem 2).
(3) Further, while Byzantine processes can disrupt causal ordering under the strong safety requirement (Theorem 3), synchronous rounds provide causal ordering under the weak safety requirement despite the presence of Byzantine processes (Theorem 4).
(4) We present an algorithm providing strong safety for Byzantine causal multicast using synchronous rounds and cryptography.
(5) We present a table of results on the solvability of BFT causal ordering in synchronous systems across various system settings.

Problem	Model	Weak Safety + Liveness	Weak Safety - Liveness	Strong Safety + Liveness	Strong Safety - Liveness
Unicast	Crypto-free	**Yes** [10, 11] (R-f, D)	**Yes** [8] (R-f, D)	**No**, Theorem 3	**No**, Theorem 3
Unicast	Threshold Crypto.	**Yes** [9] (R-f, D)	**Yes** (R-f, D)	**Yes** [9] (R-f, D)	**Yes** [9] (R-f, D)
Multicast	Crypto-free	**Yes** [10, 11] (R-f, D)	**Yes** [8] (R-f, D)	**No**, Theorem 3	**No**, Theorem 3
Multicast	Threshold Crypto.	**Yes** [9] (R-f, P)	**Yes** (R-f, D)	**Yes**, Algo. 2 (R, D)	**Yes**, Algo. 2 (R, D)
Broadcast	Crypto.-free	**Yes** [1] (R-f, D)	**Yes** [1] (R-f, D)	**No**, Theorem 3	**No**, Theorem 3
Broadcast	Threshold Crypto.	**Yes** [4] (R-f, P) for asynch.; **Yes**, Algo. 2 (R, D)	**Yes** [4] (R-f, P) for asynch.; **Yes**, Algo. 2 (R, D)	**Yes** [4] (R-f, P) for asynch.; **Yes**, Algo. 2 (R, D)	**Yes** [4] (R-f, P) for asynch.; **Yes**, Algo. 2 (R, D)

Legend: R = with rounds, R-f = round-free, D = deterministic, P = probabilistic, crypto. = cryptography, asynch. = asynchronous systems.

Table 1: Solvability of BFT Causal Ordering in Synchronous Systems.

Table 1 summarizes the solvability of causal ordering in synchronous systems under Byzantine failures across a variety of system settings/variables. We consider the solvability in terms of communication model (unicast/multicast/broadcast), cryptography (allowed/not allowed), safety condition (strong vs weak) and whether liveness is a requirement.

2 SYSTEM MODEL

The distributed system is modelled as a complete directed graph $G = (P, C)$. P is the set of processes communicating synchronously over a network. C is the set of communication channels over which processes communicate by message passing over FIFO channels. For a message send event m at time t_1, the corresponding receive event occurs at time $t_2 \in [t_1, t_1 + \delta)$, where δ is a known upper bound on message transmission time. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may exhibit arbitrary behaviour. A Byzantine process cannot impersonate another process or spawn new processes. The execution in a synchronous system may proceed in lock-step in rounds, where messages sent in a round are received in that same round. The local sequence of events in a round contains send events, followed by receive events, followed by internal events. However, simulating rounds has its own overheads and reduces concurrency.

The *happens before* [6] relation, denoted $\longrightarrow$, is an irreflexive, asymmetric, and transitive partial order defined over events in a distributed execution that is used to define causality. Next, we define the $\longrightarrow$ relation on the set of all application-level messages R.

Definition 1. *The happens before relation $\longrightarrow$ on messages consists of the following rules:*

1. *If p_i sent or delivered message m before sending message m', then $m \longrightarrow m'$.*
2. *If $m \longrightarrow m'$ and $m' \longrightarrow m''$, then $m \longrightarrow m''$.*

Definition 2. *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 $\longrightarrow$.*

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 3. *The Byzantine happens before relation $\xrightarrow{B}$ on messages in S consists of the following rules:*

1. *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 $m \xrightarrow{B} m'$.*
2. *If $m \xrightarrow{B} m'$ and $m' \xrightarrow{B} m''$, then $m \xrightarrow{B} m''$.*

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

Definition 4. *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 Byzantine causal order unicast/multicast/broadcast is specified on $(R, \longrightarrow)$ and $(S, \xrightarrow{B})$ as follows.

Definition 5. *A causal ordering algorithm for unicast/multicast/broadcast messages must ensure the following:*

1. ***Strong Safety:** $\forall m' \in CP(m)$ such that m' and m are sent to the same (correct) process, no correct process delivers m before m'.*
2. ***Liveness:** Each message sent by a correct process to another correct process will be eventually delivered.*

Definition 6. *A causal ordering algorithm for unicast/multicast/broadcast messages must ensure the following:*

1. ***Weak Safety:** $\forall m' \in BCP(m)$ such that m' and m are sent to the same (correct) process, no correct process delivers m before m'.*
2. ***Liveness:** Each message sent by a correct process to another correct process will be eventually delivered.*

When $m \xrightarrow{B} m'$, then there exists a causal chain from m to m' along correct processes that sent messages along that chain.

3 BACKGROUND

3.1 Some Cryptographic Basics

We utilize non-interactive threshold cryptography as a means to guarantee strong safety of multicasts [13]. Threshold cryptography consists of an initialization function to generate keys, message encryption, sharing decrypted shares of the message and finally combining the decrypted shares to obtain the original message from ciphertext. The following functions are used in a threshold cryptographic scheme:

Definition 7. *The dealer executes the generate() function to obtain the public key PK, Verification key VK and the private keys SK_0, SK_1, ..., SK_{n-1}.*

The dealer shares private key SK_i with each process p_i while PK and VK are publicly available.

Definition 8. *When process p_i wants to send a message m to p_j, it executes $E(PK, m, L)$ to obtain C_m. Here C_m is the ciphertext corresponding to m, E is the encryption algorithm and L is a label to identify m. p_i then broadcasts C_m to the system of processes.*

Definition 9. *When process p_l receives ciphertext C_m, it executes $D(SK_l, C_m)$ to obtain σ_l^m where D is the decryption share generation algorithm and σ_l^m is p_l's decryption share for message m.*

When process p_j receives a cipher message C_m intended for it, it has to wait for k decryption shares to arrive from the system to obtain m. The value of k depends on the security properties of the system. It derives the message from the ciphertext as follows:

Definition 10. *When process p_j wants to generate the original message m from ciphertext C_m, it executes $C(VK, C_m, S)$ where S is a set of k decryption shares for m and C is the combining algorithm for the k decryption shares.*

The following function is used to verify the authenticity of a decryption share:

Definition 11. *When a decryption share σ is received for message m, the Share Verification Algorithm is used to ascertain whether σ is authentic : $V(VK, C_m, \sigma) = 1$ if σ is authentic, $V(VK, C_m, \sigma) = 0$ if σ is not authentic.*

3.2 Byzantine Causal Multicast via Byzantine Reliable Broadcast

We propose a causal order multicast algorithm for synchronous systems. In a multicast, a message is sent to a subset of processes forming a process group. Different multicast send events can send to different process groups. Byzantine-tolerant causal multicast is invoked as BC_multicast(m, G), where G is the multicast group, and delivers a message through BC_deliver(m).

Definition 12. *Byzantine Causal Multicast satisfies the following properties:*

(1) *(BCM-Validity:) If a correct process p_i* BC_deliver*s message m from $sender(m)$ to group G, then $sender(m)$ must have* BC_multicast *m to G and $p_i \in G$.*
(2) *(BCM-Termination-1:) If a correct process* BC_multicast*s a message m to G, then some correct process in G eventually* BC_deliver*s m.*
(3) *(BCM-Agreement or BCM-Termination-2:) If a correct process* BC_deliver*s a message m from a possibly faulty process, then all correct processes in G will eventually deliver m.*
(4) *(BCM-Integrity:) For any message m, every correct process p_i* BC_deliver*s m at most once.*
(5) *(BCM-Causal-Order:) If $m \rightarrow m'$, then no correct process* BC_deliver*s m' before m.*

BCM-Causal-Order is the Strong Safety property of Definition 5. BCM-Termination-1 and BCM-Agreement imply the liveness property of Definitions 5, 6.

Definition 13. *A Byzantine-tolerant causal multicast algorithm must satisfy BCM-Validity, BCM-Termination-1, BCM-Agreement, BCM-Integrity, and BCM-Causal-Order.*

The Byzantine-tolerant Reliable Broadcast (BRB) [2, 3] is invoked by BR_broadcast and its message is delivered by BR_deliver, and satisfies the properties given below.

Definition 14. *Byzantine-tolerant Reliable Broadcast (BRB) provides the following guarantees [2, 3]:*

(1) *(BRB-Validity:) If a correct process* BR_deliver*s a message m from $sender(m)$, then $sender(m)$ must have* BR_broadcast *m.*
(2) *(BRB-Termination-1:) If a correct process* BR_broadcast*s a message m, then it eventually* BR_deliver*s m.*
(3) *(BRB-Agreement or BRB-Termination-2:) If a correct process* BR_deliver*s a message m from a possibly faulty process, then all correct processes eventually* BR_deliver *m.*
(4) *(BRB-Integrity:) For any message m, every correct process* BR_deliver*s m at most once.*

4 CAUSAL ORDERING AND SYNCHRONOUS ROUNDS

Algorithm 1 serves as a reference point for synchronous round-based communication. Without loss of generality, we assume that all processes send their messages at the beginning of each round, all messages arrive in the same round that they are sent out and messages are delivered at the end of each round.

Algorithm 1: Synchronous round-based message passing protocol

```
Data: Each process locally maintains two FIFO queues $Q_s$
      and $Q_d$ for storing outgoing/incoming messages
      respectively
1 when round $r$ starts:
2     multicast all messages in FIFO order after dequeuing
      from $Q_s$
3 when round $r$ ends:
4     deliver all messages in FIFO order after dequeuing from
      $Q_d$
5 when the application is ready to multicast message $m$ to
  group $G$: ▷ Unicast and broadcast are special
  cases
6     $Q_s.enqueue(\langle m,G \rangle)$
7 when message $m$ arrives:
8     $Q_d.enqueue(m)$
```

Theorem 1 establishes a relationship between causal ordering and synchronous rounds. Theorem 2 establishes the relationship for crash prone systems as well. Theorems 3 and 4 prove that under Byzantine failures synchronous rounds can only guarantee causal ordering under the weak safety condition.

Theorem 1. *In a failure-free setting, synchronous rounds (as demonstrated in Algorithm 1) solve the problem of causal ordering of messages under strong/weak safety.*

Proof. Let $m_1 \rightarrow m_x$, where m_1 and m_x are sent by p_i and p_j, respectively, and p_d is a common destination of both multicasts. There are two cases.

(1) $j = i$. Since p_i sends m_1 before m_x, due to FIFO communication channels, m_1 will always arrive before m_x at a common destination p_d. Therefore m_1 will be enqueued before m_x in the FIFO delivery queue at p_d and ultimately be delivered before m_x. This ensures causal order delivery.
(2) $j \neq i$. As $m_1 \longrightarrow m_x$, there must exist a sequence of (correct) processes delivering and sending messages $m_1, m_2, m_3 \ldots m_q$, ($m_q = m_x$), starting with the send event of m_1 and ending at the send event of m_x. Each correct process along this sequence sends messages at the beginning of a round and delivers messages at the end of a round. Consider the following sequence (where $m_a \xrightarrow{D} m_b$ means that a process delivers m_a and then sends m_b at a later time):
$$m_1 \xrightarrow{D} m_2 \xrightarrow{D} \ldots \xrightarrow{D} m_{q-1} \xrightarrow{D} m_q$$
Since all processes executing send/deliver events across this sequence are correct, the protocol described in Algorithm 1 will be followed. Therefore, any message $m_k, k \in [2, q]$ is sent at least one round after m_{k-1} in the sequence. Therefore, m_x will certainly arrive after m_1 at all common destinations, thereby preserving strong (and weak) safety.

Based on the above, the notion of rounds ensures that any message m such that $m' \longrightarrow m$ gets delivered after m' at all common destinations, thereby ensuring strong/weak safety. Liveness is ensured by the assumption of a failure-free system. □

Theorem 2. *Under crash failures, synchronous rounds (as demonstrated in Algorithm 1) solve the problem of causal ordering of messages under strong/weak safety with a tolerance of $(n - 2)$ failures.*

Proof. Let the number of processes that fail be t, where $t \leq (n - 2)$.

Safety. Any process p_i may fail only during one of the following times:

(1) **At the beginning of a round**: In this case p_i will not be able to send its multicast message m to all members of its intended group G. Let the group of processes that receive message m be G', where $G' \subset G$. The result of Theorem 1 still holds, because it is independent of the size and members of the multicast groups involved in message passing in a system of processes executing Algorithm 1. In this case the safety guarantees of Algorithm 1 apply to any subsequent message m' sent by members of G'. Therefore, any m'' such that $m'' \longrightarrow m'$ will arrive at a common destination before m'.
(2) **During a round**: Here, p_i is neither sending nor delivering any messages, therefore no safety violation can occur after p_i crashes.
(3) **At the end of a round**: Here, p_i fails while delivering messages that have arrived during the round. Any messages delivered from Q_d prior to p_i crashing will be done without violating safety as per Theorem 1. Messages remaining in Q_d after p_i fails will not be delivered, thereby eliminating any chance of a safety violation.

Liveness. By definition, liveness requires that messages to/from correct processes are delivered. Since $t \leq (n - 2)$, two correct processes will always be able to send/deliver messages to/from each other, thereby ensuring liveness. □

Theorem 3. *In the Byzantine failure model, synchronous rounds (as demonstrated in Algorithm 1) cannot solve the problem of causal ordering of messages under strong safety condition in a cryptography-free setting.*

Proof. Let p_i be a Byzantine process in a system of processes executing Algorithm 1. p_i can read a multicast m_1 by p_j prior to the end of round r and based on the contents of m_1, multicast message m_2 during round r. Consider a common destination of m_1 and m_2, p_k. p_k may receive m_2 prior to m_1 depending on network delays. If that is the case, p_k will deliver m_2 before m_1 at the end of round r resulting in a strong safety violation because $m_1 \longrightarrow m_2$ and so m_2 should not have been delivered before m_1. □

Theorem 4. *In the Byzantine failure model, synchronous rounds (as demonstrated in Algorithm 1) solve the problem of causal ordering of messages under weak safety with a tolerance of $(n - 2)$ failures.*

Proof. **Weak Safety.** Given $m_1 \xrightarrow{B} m_x$, there must exist a sequence of correct processes delivering and sending messages, starting with the send event of m_1 and ending at the send event of m_x. The proof of Theorem 1 now applies to this sequence of correct processes, thereby showing that weak safety is preserved.

Liveness. By definition, liveness requires that messages to/from correct processes are delivered. Since $t \leq (n - 2)$, two correct processes will always be able to send/deliver messages to/from each other, thereby ensuring liveness. □

5 SYNCHRONOUS ROUND-BASED BYZANTINE CAUSAL MULTICAST

A synchronous system can assume lock-step execution in rounds. Within a round, a process can send messages, then receive messages, and lastly have internal events; further a message sent in a round is received in the same round at all its destinations. Threshold cryptography in conjunction with the execution in rounds and Byzantine Reliable Broadcast is used to ensure strong safety. A multicast is sent via BR_broadcast to ensure BCM-Validity, BCM-Termination-1, BCM-Termination-2 and BCM-Integrity of the BC_multicast. Let β and γ denote the maximum and minimum number of rounds (sequential steps) respectively in a BRB protocol. For example, Bracha's BRB has $\beta = \infty$, $\gamma = 3$ and requires $n > 3f$ [2, 3] whereas Imbs-Raynal [5] has $\beta = \infty$, $\gamma = 2$ and requires $n > 5f$. However, $\beta = \infty$ is the case when a Byzantine process initiates broadcast and the Byzantine processes do not follow the protocol in its entirety. Whenever a correct process initiates BRB, it is delivered in γ rounds. In the case of a Byzantine broadcaster, the message will either not be delivered or in case it is delivered some correct processes may deliver the message after others as we will show in Lemma 1.

Although a message m sent in a round is delivered after all messages sent in previous rounds, a Byzantine process can peek into the buffer to read m before its receive event and send a causally dependent message m' in the same round to initiate a multicast send via its own BR_broadcast. m' may be BR_delivered before m at some process, thus violating strong safety. To prevent a Byzantine process from peeking into the content of a message in violation of causal ordering, the message is encrypted using threshold encryption. The required number of decryption shares to decrypt a message is $(t+1)$ (the total number of processes is $(3t+1)$). In round $(r+k)$, where $k \geq \gamma$ and the message is BR_broadcasted in round r, each process that has BR_delivered the encrypted message sends its decryption share to the destinations of the multicast. A message cannot get revealed before round $(r+k)$. Any message sent before that cannot be causally dependent on this revealed message m, and the only messages that the process sends that are causally dependent on the above message m can get sent (and hence delivered) only in later rounds. This guarantees strong safety and liveness. Algorithm 2 formalizes the logic. We prove the correctness of Algorithm 2 implementing BC_multicast, including its strong safety and liveness. This algorithm does not use BA_broadcast (atomic broadcast) used in [4, 13].

Algorithm 2: Round-based Synchronous Causal Multicast

```
Data: Each process has access to $PK$ (global public key), $VK$
      (global verification key). Each process $p_i$ has access to
      a local secret key $SK_i$. Each process uses FIFO queues
      $Q_s$, $Q_d$ for outgoing and incoming application
      messages, resp.. All processes in a multicast group $G$
      locally store the group key $K_G$.
1  when round $r$ starts:
2  while $Q_s.head() \neq \phi$ do
3      $\langle C_m, G_{id_m} \rangle = Q_s.pop()$
4      BR_broadcast($C_m, G_{id_m}$)
5  when round $r$ ends:
6  while $Q_d.head()$ is decrypted do
7      $C'_m = Q.pop()$
8      $m = Dec(K_{G_{id_m}}, C'_m)$
9      BC_deliver($m$)
10 when $p_i$ sends $m$ to $G_{id_m}$ via BC_multicast($m, G_{id_m}$) in round $r$:
11 $C'_m = Enc(K_{G_{id_m}}, m)$
12 $C_m = E(PK, C'_m, id_m)$
13 $Q_s.push(\langle C_m, G_{id_m} \rangle)$
14 when $\langle C_m, G_{id_m} \rangle$ is BR_delivered in round $r$:
15 $\sigma_i^m = D(SK_i, C_m)$
16 if $p_i \in G_{id_m}$ then
17     $Q_d.push(C_m)$
18 for $\forall\, p_j \in G_{id_m}$ do
19     send $\sigma_i^m$ to $p_j$ in round $(r+1)$
20 when $p_i$ receives $(t+1)$th valid $\langle \sigma_x^m \rangle$ message by round $r$:
21 Store $(t+1)$ decryption shares in set $S$
22 $C'_m = C(VK, C_m, S)$
23 replace $C_m$ in $Q_d$ with $C'_m$
```

Lemma 1. *In a system following the BRB protocol in [2], if a correct process* BR_deliver*s message m in round r, it will be* BR_deliver*ed at all correct processes at or before round* $(r+1)$.

Proof. Let p_i be the first correct process to BR_deliver m; let it do so in round r. For this to be the case, p_i must have received at least $(2t+1)$ $READY(m)$ messages by round r. At least $(t+1)$ of the $READY(m)$ messages were sent by correct processes. Therefore, at the end of round r, all correct processes will have received at least $(t+1)$ $READY(m)$ messages. At the start of round $(r+1)$, all correct processes will broadcast $READY(m)$ and will receive $(2t+1)$ $READY(m)$ messages before the end of the round. Therefore, all correct processes will BR_deliver m at or before round $(r+1)$. □

Theorem 5. *In the synchronous system model where cryptography is permitted, Algorithm 2 solves the problem of causal ordering of messages under strong safety condition with a tolerance of* $\lfloor (n-1)/3 \rfloor$ *Byzantine failures.*

Proof. **Strong Safety.** Consider messages m_1 and m_2 such that $m_1 \longrightarrow m_2$. Let p_j (possibly Byzantine) be the sender of m_1 and p_k (possibly Byzantine) be the sender of m_2. m_1 is sent at round $r^s_{m_1}$ and m_2 at round $r^s_{m_2}$. p_i (correct process) is a common destination for m_1 and m_2. m_1 (m_2) is BC_delivered at p_i in round $r^d_{m_1}$ ($r^d_{m_2}$). The send event occurs at line 10 of Algorithm 2, and the deliver event occurs at line 9 of Algorithm 2. The following holds in a system of processes executing Algorithm 2:

(1) $r^d_{m_1} - 1 \leq r^s_{m_2}$: Let p_l be the first correct process to BR_deliver m and it does so in round r. As shown by Lemma 1, any correct p_i BR_delivers m_1 at most 1 round after p_l BR_delivers m_1, in round $(r+1)$. Since p_i is a correct process and follows Algorithm 2, it will BC_deliver m_1 in the next round after BR_delivering it, in round $(r+2)$. The earliest that p_k can send m_2 is the round in which it reads m_1. And p_k cannot read m_1 before p_l has BR_delivered m_1, and sent its decryption share in round $(r+1)$. Therefore, p_k can send m_2 at most 1 round before m_1 is BC_delivered at a common destination process p_i.
(2) $r^d_{m_2} \geq r^s_{m_2} + \gamma$: This is the minimum latency to BC_deliver a message.

Combining the equations, we get the following:

$$r^d_{m_2} \geq r^d_{m_1} + \gamma - 1$$

this reduces to: $r^d_{m_2} > r^d_{m_1}$.

Liveness. The termination property of BR_broadcast guarantees that all messages sent by correct processes will be BR_delivered at all other processes as long as $t \leq \lfloor (n-1)/3 \rfloor$. Any message m that arrives at line 14 will eventually be enqueued at line 17 by all valid recipients and all correct processes send their decryption shares σ^m_* at lines 18-19. This ultimately leads to BC_delivery of m by correct processes at line 9, thereby ensuring liveness. □

It can be seen that Algorithm 2 guarantees BCM-Validity, BCM-Termination-1, BCM-Agreement, BCM-Integrity as a direct consequence of using the BR_broadcast primitive on line 4.

Corollary 1. *Algorithm 2 implements Byzantine causal multicast.*

6 DISCUSSION

We established a direct relationship between synchronous rounds and causal ordering. As a consequence of our results, given a fault-free or crash failure prone synchronous rounds-based system, there is no need to execute a causal ordering protocol. This is because synchronous rounds inherently guarantee causal ordering of messages. Even in a system with Byzantine failures, round-based communication ensures causal ordering under the weak safety condition. The only system setting where we anticipate the need of a causal ordering protocol is when the application needs causal ordering under the strong safety condition. Algorithm 2 fulfils this requirement by using cryptography in round-based communication. A important point to note is that although synchronous systems have synchronized clocks, they cannot be used for causal ordering. There are two reasons for this — Byzantine processes can always obfuscate timestamps on their messages and even if the timestamps they provide are correct, only a total-ordering of messages which does not preserve causal relationships can be established with this. Finally, we provided an analysis in Table 1 specifying the solvability of Byzantine fault-tolerant causal ordering under various system settings. Our results are fundamental and significant for causal ordering in synchronous systems, given the impossibility results for causal ordering in asynchronous systems [8, 10].

REFERENCES

[1] Alex Auvolat, Davide Frey, Michel Raynal, and François Taïani. 2021. Byzantine-tolerant causal broadcast. *Theoretical Computer Science* 885 (2021), 55–68.

[2] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[3] Gabriel Bracha and Sam Toueg. 1985. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)* 32, 4 (1985), 824–840.

[4] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.

[5] Damien Imbs and Michel Raynal. 2016. Trading off t-resilience for efficiency in asynchronous Byzantine reliable broadcast. *Parallel Processing Letters* 26, 04 (2016), 1650017.

[6] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21, 7* (1978), 558–565.

[7] Anshuman Misra and Ajay D Kshemkalyani. 2022. Causal Ordering Properties of Byzantine Reliable Broadcast Primitives. In *2022 IEEE 21st International Symposium on Network Computing and Applications (NCA)*, Vol. 21. IEEE, 115–122.

[8] Anshuman Misra and Ajay D Kshemkalyani. 2022. Solvability of byzantine fault-tolerant causal ordering problems. In *International Conference on Networked Systems*. Springer, 87–103.

[9] Anshuman Misra and Ajay D. Kshemkalyani. 2023. Byzantine Fault-Tolerant Causal Order Satisfying Strong Safety. In *25th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. Springer, 111–125.

[10] Anshuman Misra and Ajay D Kshemkalyani. 2023. Byzantine fault-tolerant causal ordering. In *Proceedings of the 24th International Conference on Distributed Computing and Networking*. 100–109.

[11] Anshuman Misra and Ajay D Kshemkalyani. 2023. Causal ordering in the presence of byzantine processes. In *2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 130–138.

[12] Reinhard Schwarz. 1992. Causality in distributed systems. In *Proceedings of the 5th workshop on ACM SIGOPS European workshop: Models and paradigms for distributed systems structuring*. 1–5.

[13] Victor Shoup and Rosario Gennaro. 2002. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology* 15, 2 (2002), 75–96.