Parallel Computing 124 (2025) 103136

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Byzantine-tolerant detection of causality: There is no holy grail ☆

Anshuman Misra [a,b], Ajay D. Kshemkalyani [a,*]

[a] Department of Computer Science, University of Illinois Chicago, Chicago, 60607, IL, USA
[b] Department of Computer Science, Purdue University Fort Wayne, Fort Wayne, 46805, IN, USA

ARTICLE INFO

Keywords:
Byzantine fault-tolerance
Causality
"Happened before" relation
Asynchronous system
Distributed system
Message-passing

ABSTRACT

Detecting causality or the "happened before" relation between events in an asynchronous distributed system is a widely used building block in distributed applications. To the best of our knowledge, this problem has not been examined in a system with Byzantine processes. We prove the following results for an asynchronous system with Byzantine processes. (1) We prove that it is impossible to determine causality between events in the presence of even a single Byzantine process when processes communicate by unicasting. (2) We also prove a similar impossibility result when processes communicate by broadcasting. (3) We also prove a similar impossibility result when processes communicate by multicasting. (4–5) In an execution where there exists a causal path between two events passing through only correct processes, we prove that it is possible to detect causality between such a pair of events when processes communicate by unicasting or broadcasting. (6) However, when processes communicate by multicasting and there exists a causal path between two events passing through only correct processes, we prove that it is impossible to detect causality between such a pair of events. (7–9) Even with the use of cryptography, we prove that the impossibility results of (1–3) for unicasts, broadcasts, and multicasts, respectively, hold. (10–12) With the use of cryptography, when there exists a causal path between two events passing through only correct processes, we prove it is possible to detect causality between such a pair of events, irrespective of whether the communication is by unicasts, broadcasts, or multicasts. Our results are significant because Byzantine systems mirror the real world.

1. Introduction

Causality is an important tool in understanding and reasoning about distributed system executions [2]. In a seminal paper, Lamport formulated the "happened before" or the causality relation, denoted $\rightarrow$, between events in an asynchronous distributed system [3]. Given two events e and e', the *causality determination* problem asks to determine whether $e \rightarrow e'$. Applications of causality determination include determining consistent recovery points in distributed databases, deadlock detection, termination detection, distributed predicate detection, distributed debugging and monitoring, the detection of race conditions and other synchronization errors [4].

The causality relation between events can be captured by tracking causality graphs [5], scalar clocks [3], vector clocks [6–8], matrix and higher-dimensional clocks [9] and numerous other variants (such as hierarchical clocks [10,11], plausible clocks [12], incremental clocks [13], dotted version vectors [14], interval tree clocks [15], logical physical clocks [16], encoded vector clocks [17], and Bloom clocks [18,19] to mention a few), proposed since Lamport's seminal paper [3]. Indirections in knowledge about local logical times can be captured by the abstraction in [9]. See [2,4] or a more recent survey included in [20]. Some of these variants track causality accurately while others introduce approximations and inaccuracies as trade-offs in the interest of savings on the space and/or time and/or message complexity overheads. As enunciated by Schwarz and Mattern [2], the search for the holy grail of the ideal causality tracking mechanism is on. However, all these works in the literature assume that processes are correct.

To the best of our knowledge, there has been no work on detecting the causality relation between events in the presence of Byzantine processes in the system. It is important to solve this problem under the Byzantine failure model as opposed to a failure-free setting because it mirrors the real world.

The related problem of causal ordering of messages asks that if the send event of message m happened before the send event of message m', then m' should not be delivered before m at all common destinations of m and m' [21]. Causal ordering of messages has numerous applications such as in distributed data stores, fair resource allocation, and collaborative applications such as social networks, multiplayer online gaming, group editing of documents, event notification systems, and

☆ An earlier version of this result appeared in Proceedings of IEEE NCA 2022, Misra and Kshemkalyani (2022) [1].

* Corresponding author.
E-mail address: ajay@uic.edu (A.D. Kshemkalyani).

https://doi.org/10.1016/j.parco.2025.103136
Received 22 July 2024; Received in revised form 22 March 2025; Accepted 29 March 2025
Available online 9 April 2025

Table 1
Detecting causality between events under different communication modes. FP is false positive, FN is false negative. FP_B is false positive under $\xrightarrow{B}$. FN_B is false negative under $\xrightarrow{B}$. $\overline{FP}, \overline{FP_B}, \overline{FN_B}$ indicate that the corresponding false positives under $\rightarrow$, false positives under $\xrightarrow{B}$, false negatives under $\xrightarrow{B}$ cannot occur. Next to each result, the relation between t (number of Byzantine processes) and n (total number of processes) is given.

Mode of communication	Detecting "happened before" $e \rightarrow e'$	Detecting "Byzantine happened before" $e \xrightarrow{B} e'$	Detecting "happened before" $e \rightarrow e'$ using cryptography	Detecting "Byzantine happened before" $e \xrightarrow{B} e'$ using cryptography
Unicasts	Impossible, Theorem 3 $FP\,(t > 0),\ FN\,(t > 0)$	Possible, Theorem 6 $\overline{FP_B}\,(t < n),\ \overline{FN_B}\,(t < n)$	Impossible, Theorem 10 $\overline{FP}$[a] $(t < n/3),\ FN\,(t > 0)$	Possible, Theorem 12 $\overline{FP_B}\,(t < n),\ \overline{FN_B}\,(t < n)$
Broadcasts	Impossible, Theorem 4 $\overline{FP}\,(t < n/3),\ FN\,(t > 0)$	Possible, Theorem 7 $\overline{FP_B}\,(t < n),\ \overline{FN_B}\,(t < n)$	Impossible, Theorem 11 $\overline{FP}\ (t < n/3),\ FN\,(t > 0)$	Possible, Theorem 13 $\overline{FP_B}\,(t < n),\ \overline{FN_B}\,(t < n)$
Multicasts	Impossible[b,c], Theorem 5 $FP\,(t > 0),\ FN\,(t > 0)$	Impossible[b], Theorem 8 $FP_B\,(t > 0),\ FN_B\,(t > 0)$	Impossible, Theorem 9 $\overline{FP}$[a] $(t < n/3),\ FN\,(t > 0)$	Possible, Theorem 14 $\overline{FP_B}\,(t < n),\ \overline{FN_B}\,(t < n)$

[a] False positive FP holds if the semantics of the message content in a message dependency matters.
[b] Without using cryptography, it is impossible to implement Byzantine Reliable Multicast (BRM) in multiple groups as this entails identifying the Byzantine processes to satisfy $t_G < |G|/3$, where t_G is the number of Byzantine processes in group G.
[c] Even with access to a black box that implements BRM within groups, FP and FN hold.

distributed virtual environments. Causal ordering of messages under the Byzantine failure model has recently been examined in [22] for broadcast communication and in [23–28] for unicast, multicast, as well as broadcast communication. An algorithm for Byzantine causal order of broadcast messages was given in [22]; this was based on a definition of Byzantine causal order on *broadcast messages*. This definition was modified in several ways to account for the purest (weakest) notion of safety [23], and several possibility and impossibility results about Byzantine causal order of unicast, multicast, and broadcast messages were proved [23]. The algorithm in [22] is an instantiation for one of the many results proved in [23] on causal ordering of messages. Several algorithms for causal ordering of messages under varied settings and assumptions on the system model were given in [24–28]. In contrast, this paper uses a definition of Byzantine causal order on *events* that can be viewed as an adaptation of the definition in [23].

Contributions: Our main result is that it is impossible to (deterministically) determine the causality relation $\rightarrow$ between two events $e1$ and $e2$ when there is even a single Byzantine process in an asynchronous distributed system. False negatives and/or false positives are possible. A false negative means that $e \rightarrow e'$ whereas $e \nrightarrow e'$ is perceived/detected. A false positive means that $e \nrightarrow e'$ whereas $e \rightarrow e'$ is detected. In light of this negative result, we investigate whether any positive result can be shown in a system with stronger assumptions.

- First, we introduce the Byzantine happened before relation $\xrightarrow{B}$, where $e1 \xrightarrow{B} e2$ if $e1 \rightarrow e2$ and there exists a causal path from $e1$ to $e2$ via the transitive closure of the local order of events and the order of message-passing send and corresponding receive events, going through only correct (non-Byzantine) processes. If $e1 \xrightarrow{B} e2$, then we show that causality can be determined for unicasts and broadcasts but not for multicasts.
- Second, we show that even with the use of cryptography, the impossibility results for unicasts, multicasts, and broadcasts remain.
- Third, with the use of cryptography, we show possibility results for unicasts, broadcasts, and multicasts under $\xrightarrow{B}$.

We prove the following results for an asynchronous system with Byzantine processes.

1. We prove that it is impossible to determine causality between events in the presence of even a single Byzantine process when processes communicate by unicasting (Theorem 3). This is because both false positives and false negatives can occur.
2. We also prove a similar impossibility result when processes communicate by broadcasting (Theorem 4). In this case, false positives cannot occur but false negatives can occur.
3. We also prove a similar impossibility result when processes communicate by multicasting (Theorem 5). Both false positives and false negatives can occur.
4. In an execution where there exists a causal path between two events passing through only correct processes, i.e., $\xrightarrow{B}$ holds between the two events, and processes communicate by unicasting, we prove that it is possible to detect causality between such a pair of events (Theorem 6). Neither false positives nor false negatives can occur under $\xrightarrow{B}$. But under the $\rightarrow$ relation, false positives and false negatives can occur.
5. We also prove a similar possibility result when processes communicate by broadcasting and there exists a causal path between two events passing through only correct processes, i.e., $\xrightarrow{B}$ holds between the two events (Theorem 7). Neither false positives nor false negatives can occur under $\xrightarrow{B}$. Under the $\rightarrow$ relation, false positives cannot occur but false negatives can occur.
6. We prove an impossibility result when processes communicate by multicasting and there exists a causal path between two events passing through only correct processes, i.e., $\xrightarrow{B}$ holds between the two events (Theorem 8). False positives and false negatives may occur under $\xrightarrow{B}$ and under the $\rightarrow$ relation.
7. We prove that is impossible to determine causality between events in the presence of even a single Byzantine process when processes communicate by multicasting, even when cryptographic techniques are used (Theorem 9). False positives do not occur (provided the semantics of contents of messages are not accounted for) but false negatives can occur.
8. We also prove that the same above impossibility result holds when processes communicate by unicasting despite the use of cryptography (Theorem 10).
9. We prove an impossibility result for broadcasts allowing the use of cryptography and show it is the same as that without cryptography (Theorem 11) — false positives do not occur but false negatives can occur.
10. With the use of cryptography, when there exists a causal path between two events passing through only correct processes, we prove it is possible to detect causality between such a pair of events, irrespective of whether the communication is by unicasts (Theorem 12), broadcasts (Theorem 13), or multicasts (Theorem 14). Neither false positives nor false negatives can occur under $\xrightarrow{B}$ while no false positives but false negatives can occur under $\rightarrow$.

Table 1 summarizes these results. For each result, the relationship between the number of Byzantine processes t and the total number of processes n is also stated. The key technical difficulty was in identifying and analyzing the various cases, and proving the results. An earlier version of this paper was published as [1]. It has been significantly enhanced and has added details and reworked, formal proofs. New results in a new Section 4.1 and on the solvability of detecting causality using cryptography in a new Section 4.4 are added. A new Section 5 gives certain relationships between the causality determination problem in asynchronous systems and related problems/models.

Roadmap. Section 2 gives the system model. Section 3 formulates the problem of detecting causality. Section 4 proves the results outlined under "Contributions" above. Section 5 proves some relationships between the causality detection problem and other related problems. Section 6 gives a discussion and concludes.

2. System model

This paper deals with an asynchronous distributed system having Byzantine processes which are processes that can misbehave [29,30]. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may deviate arbitrarily from its protocol behavior. A Byzantine process cannot impersonate another process or spawn new processes. Besides distributed systems, Byzantine attacks have been extensively studied in many physical systems and multi-agent systems [31,32].

The distributed system is modeled as an undirected graph $G = (P, C)$. Here P is the set of processes communicating asynchronously in the distributed system. Let $|P| = n$. C is the set of FIFO (logical) communication links over which processes communicate by message passing. G is a complete graph.

The distributed system is assumed to be *asynchronous*, i.e., there is no fixed upper bound δ on the message latency, nor any fixed upper bound ψ on the relative speeds of processors [33]. In contrast, a *synchronous system* has been defined as one in which both δ and ψ exist and are known. [33]. A *partially synchronous* system is an asynchronous system but with periods of synchrony [33].

We first do not consider the use of digital signatures/ cryptography in the system model because of the high cost. Then in a separate section we show results in a model that allows the use of digital signatures/cryptography.

Let e_i^x, where $x \geq 1$, denote the xth event executed by process p_i. An event may be an internal event, a message send event, or a message receive event. Let the state of p_i after e_i^x be denoted s_i^x, where $x \geq 1$, and let s_i^0 be the initial state. The execution at p_i is the sequence of alternating events and resulting states, as $\langle s_i^0, e_i^1, s_i^1, e_i^2, s_i^2 \dots \rangle$. The *execution history* at p_i is the finite execution at p_i up to the current or most recent or specified local state. The *happened before* [3] relation, denoted $\rightarrow$, is an irreflexive, asymmetric, and transitive partial order defined over events in a distributed execution that is used to define causality.

Definition 1. The happened before relation $\rightarrow$ on events consists of the following rules:

1. **Program Order**: For the sequence of events $\langle e_i^1, e_i^2, \dots \rangle$ executed by process p_i, $\forall\, x, y$ such that $x < y$ we have $e_i^x \rightarrow e_i^y$.
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**: If $e \rightarrow e' \wedge e' \rightarrow e''$ then $e \rightarrow e''$.

Definition 2. The *causal past* of an event e is denoted as $CP(e)$ and defined as the set of events that causally precede e under $\rightarrow$.

We require an extension of the happened before relation on events to accommodate the possibility of Byzantine behavior. We present a partial order on messages called *Byzantine happened before*, denoted as $\xrightarrow{B}$, defined on the set of all events at correct processes in P.

Definition 3. The Byzantine happened before relation $\xrightarrow{B}$ on events at correct processes consists of the following rules:

1. **Program Order**: For the sequence of events $\langle e_i^1, e_i^2, \dots \rangle$ executed by a correct process p_i, $\forall\, x, y$ such that $x < y$ we have $e_i^x \xrightarrow{B} e_i^y$.
2. **Message Order**: If event e_i^x is a message send event executed at correct process p_i and e_j^y is the corresponding message receive event at correct process p_j, then $e_i^x \xrightarrow{B} e_j^y$.
3. **Transitive Order**: If $e \xrightarrow{B} e' \wedge e' \xrightarrow{B} e''$ then $e \xrightarrow{B} e''$.

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

Note that the classic happened before relation (Definition 1) applies regardless of the failure model. However, as we prove, detecting it without both false positives and false negatives is impossible in all system model settings considered under the Byzantine failure model. Hence a weaker variant – the Byzantine happened before relation (Definition 3) – has been defined. A version of this definition was first defined on messages [22,23,25]. This definition weakens the causality property to make it detectable in some system model settings under the Byzantine failure model. In essence, the choice between the two relations is an application-level decision rather than an intrinsic property of the Byzantine failure model. Depending on the possibility of detection, the level of causal guarantees required, and costs, an application can select the appropriate relation to meet its requirements.

There are three modes of communication: multicast, unicast, and broadcast. In multicast, a message is sent to a group G of processes corresponding to some subset of P. A unicast is a multicast where $|G| = 1$. A broadcast is a multicast where $G = P$. We specify the multicast definition formally, tailored for Byzantine-tolerant systems.

Definition 4. Byzantine Reliable Multicast (BRM) to group G satisfies the following properties:

1. (Validity:) If a correct process p_i delivers message m from a correct process $sender(m)$ sent to group G, then $sender(m)$ must have executed send(m, G) and $p_i \in G$.
2. (Self-delivery:) If a correct process executes send(m, G), then it eventually delivers m.
3. (Reliability/Termination:) If a correct process delivers a message m from a possibly faulty process, then all correct processes in G will eventually deliver m.
4. (Integrity:) For any message m, a correct process p_i delivers m at most once.
5. (No Information Leakage:) No process outside the group G sees the content of m.

As a unicast has a single destination, the Reliability/Termination property of BRM does not apply to Byzantine Reliable Unicast (BRU). As the destination set of a broadcast is the set of all processes, the No Information Leakage property of BRM does not apply to Byzantine Reliable Broadcast (BRB).

In our analysis of causality detection under $\xrightarrow{B}$, we use Byzantine Causal Broadcast (BCB) with operations `bco_broadcast()` and `bco_deliver()`, where BCB is defined as BRB + a property BCO-Causality based on the Byzantine happened before relation $\xrightarrow{B}$ on messages sent by and delivered at correct processes [23,25,28], similar to [22].

- BCO-Causality: for messages m, m' sent at events e, e', respectively, if $e \xrightarrow{B} e'$ then no correct process `bco_delivers` m' before m.

3. Problem formulation

An algorithm to solve the causality determination problem collects the execution history of each process in the system and derives causal relations from it. Let E_i denote the *actual* execution history at p_i (which was defined as the sequence of alternating events and resulting states) and let $E = \bigcup_i \{E_i\}$. (Here the range of i is the set of process ids in P; here and subsequently the range of variables should be clear from context and is omitted to avoid excessive cluttering.) For any causality determination algorithm, let F_i be the execution history at p_i as perceived and collected by the algorithm and let $F = \bigcup_i \{F_i\}$. F thus denotes the execution history as collected by the algorithm. Let $T(E)$ and $T(F)$ denote the sets of all events in E and F, respectively. Analogous to Definitions 1 and 3, we can define the *happened before* and *Byzantine happened before* relations on $T(F)$ instead of on $T(E)$.

Let $e1 \to e2|_E$ and $e1 \to e2|_F$ be the evaluation (1 or 0) of $e1 \to e2$ using E and F, respectively. Byzantine processes may corrupt the collection of F to make it different from E. We assume that a correct process p_i needs to determine whether $e_h^x \to e_i^*$ holds and e_i^* is an event in $T(E)$. If $e_h^x \notin T(E)$ then $e_h^x \to e_i^*|_E$ evaluates to *false*; note that $e_h^x \to e_i^*|_E$ may evaluate to *false* even if $e_h^x \in T(E)$. If $e_h^x \notin T(F)$ (or $e_i^* \notin T(F)$) then $e_h^x \to e_i^*|_F$ evaluates to *false*. We assume an oracle that is used for determining correctness of the causality determination algorithm; this oracle has access to E which can be any execution history such that $T(E) \supseteq CP(e_i^*)$. Byzantine processes may collude as follows.

1. To delete e_h^x from F_h or in general, record F as any alteration of E such that $e_h^x \to e_i^*|_F = 0$, while $e_h^x \to e_i^*|_E = 1$, or
2. To add a fake event e_h^x in F_h or in general, record F as any alteration of E such that $e_h^x \to e_i^*|_F = 1$, while $e_h^x \to e_i^*|_E = 0$.

Without loss of generality, we have that $e_h^x \in T(E) \cup T(F)$. Note that e_h^x belongs to $T(F) \setminus T(E)$ when it is a fake event in F.

Definition 5. The causality determination problem $CD(E, F, e_i^*)$ for any event $e_i^* \in T(E)$ at a correct process p_i is to devise an algorithm to collect the execution history E as F at p_i such that $valid(F) = 1$, where

$$valid(F) = \begin{cases} 1 & \text{if } \forall e_h^x, e_h^x \to e_i^*|_E = e_h^x \to e_i^*|_F \\ 0 & \text{otherwise} \end{cases}$$

When 1 is returned, the algorithm output matches God's truth and solves *CD* correctly. Thus, returning 1 indicates that the problem has been solved correctly by the algorithm using F. 0 is returned if one of the following two cases holds.

- $\exists e_h^x$ such that $e_h^x \to e_i^*|_E = 1 \wedge e_h^x \to e_i^*|_F = 0$ (denoting a false negative, abbreviated FN).
- $\exists e_h^x$ such that $e_h^x \to e_i^*|_E = 0 \wedge e_h^x \to e_i^*|_F = 1$ (denoting a false positive, abbreviated FP).

In order to determine whether *CD* is solved correctly, we have to evaluate $\forall e_h^x, e_h^x \to e_i^*|_E = e_h^x \to e_i^*|_F$ even if $e_h^x \in (T(E) \cup T(F)) \setminus T(E)$ because such an e_h^x is recorded by the algorithm as part of F. We make the following crucial observation: in *CD*, a single Byzantine process p_b can cause F (as recorded by the algorithm) to be different from E. This is not just a mismatch between E_b and F_b at p_b but also at other processes, and also a mismatch between other E_a and F_a at processes p_c, by contaminating F_b and/or F_a via direct and transitive message passing (across different messages) originated at or passing through p_b.

- A FN arises because a send-receive event pair (e_f^u, e_g^v) of E in a causal chain from e_h^x to e_i^* is missing as per F. In addition, a FN may arise if $e_h^x \in T(E) \setminus T(F)$.
- A FP arises because a non-existent send-receive message pair (e_f^u, e_g^v) in E appears in a causal chain from e_h^x to e_i^* as per F. In addition, a FP may arise if $e_h^x \in T(F) \setminus T(E)$.

Byzantine processes are an integral part of the system. The occurrence of an event at such a process, and its correct order with respect to other events locally, matters to correct processes because it can impact the causality relation among events at correct processes. Let p_{c1} and p_{c2} be correct processes and let p_b be a Byzantine process. Let message $m1$ sent at e_s_{c1} be received at e_r_b. Let message $m2$ sent at e_s_b be received at e_r_{c2}. Consider the following scenarios.

1. In E, we have $e\_s_{c1} \to e\_r_b \to e\_s_b \to e\_r_{c2}$. If F at the correct processes does not match this (specifically, $e\_r_b \nrightarrow e\_s_b$ due to p_b lying), a causality detection algorithm fails to recognize $e\_s_{c1} \to e\_r_{c2}$, resulting in a false negative.
2. In E, we have $e\_s_{c1} \to e\_r_b$, $e\_s_b \to e\_r_{c2}$, and $e\_s_b \to e\_r_b$. If F at the correct processes does not match this and reflects $e\_r_b \to e\_s_b$ (due to p_b lying), a causality detection algorithm wrongly detects $e\_s_{c1} \to e\_r_{c2}$, resulting in a false positive.

Or let p_{b1} and p_{b2} be Byzantine processes. Let message $m1$ sent at e_s_{c1} be received at e_r_{b1}. Let message $m2$ sent at e_s_{b2} be received at e_r_{c2}. Consider the following scenarios.

1. In E, we have $e\_s_{c1} \to e\_r_{b1} \to e\_s_{b1} \to e\_r_{b2} \to e\_s_{b2} \to e\_r_{c2}$. If F at the correct processes does not match this (specifically, e_s_{b1} and e_r_{b2} are not revealed due to p_{b1} and p_{b2} lying), a causality detection algorithm fails to recognize $e\_s_{c1} \to e\_r_{c2}$, resulting in a false negative.
2. In E, we have $e\_s_{c1} \to e\_r_{b1}$, $e\_s_{b2} \to e\_r_{c2}$. If F at the correct processes does not match this and reflects $e\_s_{c1} \to e\_r_{b1} \to e\_s_{b1} \to e\_r_{b2} \to e\_s_{b2} \to e\_r_{c2}$ (due to p_{b1} and p_{b2} lying), a causality detection algorithm wrongly detects $e\_s_{c1} \to e\_r_{c2}$, resulting in a false positive.

Therefore it not sufficient for the correct processes to agree mutually on a F that differs from E in what happened in E at the Byzantine processes; their F_j must also agree with E_j at all processes p_j.

4. Impossibility and possibility results

4.1. Two basic results

Theorem 1. *It is impossible to prevent false negatives in solving the causality determination problem (Definition 5) as specified by CD(E, F, e_i^*) in an asynchronous unicast/multicast/broadcast-based message passing system with one or more Byzantine processes.*

Proof. In the determination of $e_h^x \to e_i^*$, a false negative may arise when a send-receive event pair (e_f^u, e_g^v) in a causal chain from e_h^x to e_i^* is missing as per F. The causal chain has the following subsequence: $\langle \cdots e_f^u, e_g^v, e_g^{v'} \cdots \rangle$, where $e_g^{v'}$ is a send event at p_g. Both e_g^v and $e_g^{v'}$ are local to p_g. A Byzantine p_g can suppress letting the rest of the system know of the occurrence of e_g^v or swap the order of occurrence of e_g^v and $e_g^{v'}$ in what it lets the rest of the system know about the occurrence of the two local events. Both actions have the effect of breaking the causality chain from e_h^x to e_i^* which can give rise to a false negative. □

Another reason why false negatives cannot be prevented is that a Byzantine process pair may collude in performing *out-of-band communication* with each other. Such an out-of-band send-receive event pair (e_f^u, e_g^v) does establish a causal chain from e_h^x to e_i^* by the composition of $e_h^x \to e_f^u$, $e_f^u \to e_g^v$, and $e_g^v \to e_i^*$, that is missing as per F because $e_f^u \to e_g^v$ is not recorded by the algorithm — it is not detectable outside the subsystem of the Byzantine processes if they choose not to disclose it.

We also have the following result about internal events at a process.

Theorem 2. *For an internal event e_h^x, it is impossible to prevent false negatives or false positives in determining $e_h^x \to e_i^*$ at a correct process p_i in an asynchronous message passing system with one or more Byzantine processes.*

Proof. There may be no other event in the rest of the system to corroborate the occurrence of an internal event at a process. A Byzantine process p_h can choose to not reveal about an internal event e_h^x to the rest of the system, leading to a false negative that cannot be prevented. It may also choose to add a fake internal event e_h^x in what it reveals to the rest of the system, leading to a false positive that cannot be prevented. □

In light of Theorem 2, we implicitly prove our impossibility or possibility results on the CD problem considering only send and receive events in E and F.

4.2. Results for "happened before"

A main contribution in our results is relating the causality determination problem to the well-known *Consensus* problem [29,30]. In the *Consensus* problem, each process has an initial value and all correct processes must agree on a single value. The solution needs to satisfy the following three conditions.[1]

- Agreement: All non-faulty processes must agree on the same single value.
- Validity: If all non-faulty processes have the same initial value, then the agreed-on value by all the non-faulty processes must be that same value.
- Termination: Each non-faulty process must eventually decide on a value.

According to the FLP impossibility result [35], it is impossible to solve *Consensus* in an asynchronous message-passing system with even a single crash failure prone process. We show a reduction from *Consensus* to *CD* in Byzantine systems to prove the impossibility of solving *CD* in a system with Byzantine failures. This proves that *CD* is at least as hard as *Consensus* in Byzantine systems. Of related but orthogonal interest, we also later show that *CD* does not reduce to *Consensus* in a Byzantine system, i.e., a solution to *Consensus* cannot be used to solve *CD* in such a system, thus establishing that *CD* is harder to solve than *Consensus*.

Theorem 3. *It is impossible to solve causality determination (Definition 5) as specified by $CD(E, F, e_i^*)$ in an asynchronous unicast-based message passing system with one or more Byzantine processes.*

Proof. We prove the impossibility of solving the *CD* problem in two steps.

1. We give a reduction (denoted $\preceq$) from *Black_Box* to *CD*, where *Black_Box* is defined below.
2. We also give a reduction from the *Consensus* problem (which by the FLP result [35] is unsolvable in the presence of a single Byzantine process) to the *Black_Box* problem.

We then transitively compose these two reductions. In more detail, after showing how *Consensus* can be solved by invoking a black box that solves *Black_Box*, and how *Black_Box* can be solved by solving *CD*, we argue as follows. If *CD* were solvable, *Black_Box* would be solvable, and then *Consensus* would also be solvable; however, that contradicts the unsolvability of *Consensus*. Therefore, there cannot exist any algorithm to solve *CD*.

The definition of the *Black_Box* problem is as follows. *Black_Box*$(\overline{V}, E, F, e_i^*)$ executed at p_i takes as input a vector $\overline{V}$ of initial boolean values, one per process, E, F, and local event e_i^* at a process p_i. *Black_Box* invoked at p_i acts as follows. The correct process p_i broadcasts the value w where:

$$w = \begin{cases} 0 & \text{if each correct } p_j \text{ has } V[j] = 0 \\ 1 & \text{if each correct } p_j \text{ has } V[j] = 1 \\ CD(E, F, e_i^*) & \text{otherwise} \end{cases}$$

and locally returns L, a list of ids of correct processes. Solving *Black_Box* requires identifying the set of correct processes; we do not claim *Black_Box* is solvable.

[1] In some literature, Conservation, which requires the sum of the initial values to equal the sum of the final values, is co-specified [34]. However, in general *Consensus* is independent of Conservation.

Solving *Black_Box* at p_i requires identifying the set of correct processes and solving CD. In order for any algorithm to correctly solve *CD* (Definition 5), it must ensure that the collected execution history F (containing execution histories of potentially Byzantine processes) matches E, and this requires identifying all Byzantine processes as we prove next. From this it will follow that *Black_Box* $\preceq$ *CD*.

For F to match E, the following must hold.

- (*Managing false positives:*) A Byzantine process may attempt to insert a fake entry in F_h (based on a fake send-receive event pair) and contaminate the reporting of histories in F, leading to a false positive. Therefore, there needs to be a mechanism to prevent contamination of F or filter out the malicious entries from F within bounded time. However, due to unicasting, message privacy needs to be maintained. Note that we are not considering the use of cryptography. Hence a message send event in F_h from a potentially Byzantine p_h to a potentially Byzantine p_g cannot be verified within bounded time by other processes while collecting the reported execution history as the message itself cannot be broadcast or communicated to any process other than p_g to maintain its privacy. No bound on the time period exists because p_h may be Byzantine and because there is no upper bound on the message latency. (After event e_i^*, process p_i can try to verify with p_h whether the entry in F_h is genuine or fake but this may not conclude in bounded time; if treated as genuine, a fake send-receive event pair introduces a false positive and if treated as fake, a genuine send-receive event pair introduces a false negative. Furthermore, if both p_h and p_g are Byzantine, a fake event in F_h can appear as genuine to p_i despite any verification attempts.) Therefore identification of Byzantine processes, their actual execution histories, and causal chains from and through them is required.
- (*Managing false negatives:*) Consider a message m sent at e_h^x from p_h to p_g in E_h. During the collection of E_h to p_i for reporting F_h, Byzantine processes may delete information about e_h^x and m from F_h, leading to a false negative when $e_h^x \rightarrow e_i^*$. Further by Theorem 1, a false negative may occur if the receive event of m by p_g is not disclosed to the rest of the system by a Byzantine p_g or if the receive event is swapped after a subsequent send event by p_g that is part of the causality chain $e_h^x \rightarrow e_i^*$, in what p_g discloses to the rest of the system thereby breaking the causality chain of E in F. Therefore, either deletion of information from E in F or alteration of E in F has to be prevented, or such deletions and alterations from E when presented with F have to be recognized within bounded time. This requires identification of the Byzantine processes, their actual execution histories, and causal chains from and through them.

If there were an algorithm to make F match E, it *requires identifying whether each of the processes that input their execution histories is correct or Byzantine, and tracing and dealing with/resolving the impact of contamination via message passing by the Byzantine processes from and through those Byzantine processes on the execution histories of processes at other processes*. Thus, *Black_Box* $\preceq$ *CD*.

Now we give the reduction from *Consensus* to *Black_Box*. To solve *Consensus*$(\overline{V})$ at (a correct process) p_i, we invoke *Black_Box*$(\overline{V}, E, F, e_i^*)$ locally (and likewise to solve *Consensus*$(\overline{V})$ at (each process) p_j, invoke *Black_Box*$(\overline{V}, E, F, e_j^*)$ at each p_j). Each correct process computes $\min(L)$ from the locally returned list L and outputs as its consensus value the broadcast value that it receives from $p_{\min(L)}$ and terminates. The conditions of *Consensus* – Agreement, Validity, and Termination – can be seen to be satisfied. So *Consensus* $\preceq$ *Black_Box*.

If *CD* is (correctly) solvable, it returns 1 for $\forall e_h^x, e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F$, (and implicitly for all e_i^*). This gives:

$Consensus \preceq Black\_Box \preceq CD$.

Transitivity of reductions implies that if the *CD* problem is solvable, then *Consensus* is also solvable. However, that contradicts the FLP impossibility result [35] when applied to a Byzantine system, hence *CD* cannot be solvable. □

When the communication pattern is by broadcasts, the proof analyzing the *CD* problem uses Byzantine Reliable Broadcast (BRB) [36,37] as a layer beneath the broadcast invocation. Without loss of generality, this proof considers the strongest form of broadcast that gives the highest resilience to Byzantine behavior, namely BRB. BRB requires that the number of Byzantine processes t be such that $n > 3t$. BRB is an instantiation of Definition 4 having $G = P$ and without the No Information Leakage Property, and satisfies the following properties.

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

Theorem 4. *It is impossible to solve causality determination (Definition 5) as specified by CD(E, F, e_i^*) in an asynchronous broadcast-based message passing system with one or more Byzantine processes.*

Proof. The overall structure of the proof is along the lines of that for Theorem 3. We outline the logic that *CD* (Definition 5) cannot be solved for when the underlying send events are broadcasts. Specifically, we show that F cannot be made to match E.

- (*Managing false positives:*) By doing broadcasts using the Byzantine Reliable Broadcast (BRB) [36,37] layer, false positives can be prevented by ensuring no fake events/ causal dependencies are added to F. Consider the case that a Byzantine process p_b attempts to insert a fake entry about broadcast of m by p_h in F_h (whether $h = b$ or $h \neq b$) at a correct process p_g via a message m' sent to p_g. As broadcasts are sent over the underlying BRB, p_g can verify whether or not this insertion is valid — based on the Reliability (or Termination) property of BRB, m must get delivered by the BRB layer at all correct processes including p_g if the insertion is valid. Only if m is delivered to p_g is authenticity of m verified and the entry about m can be inserted in F_h. Now in particular, p_g may be p_i because it is correct. Therefore, correct processes including p_i have a mechanism to prevent fake send events (and their corresponding fake receive events) from being inserted in F, ensuring no false positives.
  A fake receive event r_h for message m cannot be inserted in F_h at p_g, using the mechanism outlined next. r_h is included as a causal dependency on the next broadcast by p_h and only on the receipt of such a broadcast by p_g is p_g allowed to include r_h in F_h at p_g. This inclusion is done only if the send event of m can be verified by p_g using BRB. p_h could learn of r_h by receiving m (or information of m from a colluding Byzantine p_k that has received m). If multiple identical r_h are reported to p_g, the first is included in F_h.
- (*Managing false negatives:*) However, a Byzantine process p_g can delete from F_g information about a broadcast of m by p_h at e_h^x that it has received, despite doing broadcasts using the BRB layer. Even if $e_h^x \rightarrow e_i^*$ where the causality chain passes through a message broadcast event subsequently at p_g after receiving m, p_i has no way of knowing about this chain or about the receive event of m at p_g if p_g so chooses. Further by Theorem 1, a false negative may occur if the receive event of m by p_g is not disclosed to the rest of the system by a Byzantine p_g or if the receive event is swapped after a subsequent broadcast event by p_g that is part of the causality chain $e_h^x \rightarrow e_i^*$, in what p_g discloses to the rest of the system thereby breaking the causality chain of E in F. To prevent such false negatives, Byzantine processes, their actual execution histories, and causal chains from and through such processes need to be identified.
  Note that in the bullet above regarding prevention of false positives, if m is not delivered to p_g within the time to report F, the entry about sending of m is not added to F_h even though m might have been sent. However the message m' carrying information about sending of m is considered received/delivered, and hence $e_h^x \rightarrow e_i^*$. So this scenario contributes to a false negative.

Thus, to solve *CD*, it is necessary to identify Byzantine processes, their actual execution histories, and causal chains from and through them. So we have *Black_Box* ≤ *CD* and, as *Consensus* ≤ *Black_Box*, hence *Consensus* ≤ *CD*. As *Consensus* is unsolvable in a system with Byzantine processes, *CD* is also unsolvable. □

When processes communicate by multicasting, each send event sends a message to a group G consisting of processes in a subset of P. Different send events can send to different subsets of processes in P. The number of possible groups is $2^{|P|} - 1$. Communicating via unicasts and communicating via broadcasts are special cases of multicasting.

Theorem 5. *It is impossible to solve causality determination (Definition 5) as specified by CD(E, F, e_i^*) in an asynchronous multicast-based message passing system with one or more Byzantine processes.*

Proof. Unicast mode of communication is a special case of multicast where each group is of size 1 (or 2 if the sender is included in the multicast group). Theorem 3 proved that causality determination in the presence of even a single Byzantine process under unicast communication is impossible to solve. As the special case of group size 1 (or 2) is not solvable, the general case of multicast is also not solvable. □

4.3. Results for "Byzantine happened before"

The *CD* problem (Definition 5) defined in terms of the $\rightarrow$ relation is now redefined in terms of the $\xrightarrow{B}$ relation for the correctness criteria for causality determination.

From Definition 3, we have that $e \xrightarrow{B} e'$ is equivalent to ($e \rightarrow e' \wedge$ *there is a causal path from event e to event e′ going through correct processes in the execution*). We define $e \xrightarrow{B} e'|_E$ and $e \xrightarrow{B} e'|_F$ as follows. $e \xrightarrow{B} e'|_E$ is defined as ($e \rightarrow e'|_E \wedge$ *there is a causal path from e to e′ going through correct processes in the execution*). $e \xrightarrow{B} e'|_F$ is defined as ($e \rightarrow e'|_F \wedge$ *there is a causal path from e to e′ going through correct processes in the execution*). Note that evaluating $e \xrightarrow{B} e'|_F$ does not involve determining whether there actually exists the causal path going through correct processes; also the process at which e' occurs does not know that the process at which e occurs is correct.

Definition 6. The causality determination problem $CD\_B(E, F, e_i^*)$ for any event $e_i^* \in T(E)$ at a correct process p_i is to devise an algorithm to collect the execution history E as F at p_i such that $valid\_B(F) = 1$, where

$$valid\_B(F) = \begin{cases} 1 & \text{if } \forall e_h^x, e_h^x \xrightarrow{B} e_i^*|_E = e_h^x \xrightarrow{B} e_i^*|_F \\ 0 & \text{otherwise} \end{cases}$$

The problem is solved correctly iff 1 is returned. Value 0 is returned if one of the following two cases holds.

- $\exists e_h^x$ such that $e_h^x \rightarrow e_i^*|_E = 1 \wedge e_h^x \rightarrow e_i^*|_F = 0 \wedge$ *there exists a causal path from e_h^x to e_i^* going through correct processes* (denoting a false negative under $\xrightarrow{B}$, abbreviated FN_B).

- $\exists e_h^x$ such that $e_h^x \to e_i^*|_E = 0 \wedge e_h^x \to e_i^*|_F = 1 \wedge$ *there exists a causal path from* e_h^x *to* e_i^* *going through correct processes* (denoting a false positive under $\xrightarrow{B}$, abbreviated FP_B). This case cannot occur as the first and third terms cannot both be true. Hence FP_B cannot occur.

Theorem 6. *It is possible to solve causality determination (Definition 6) as specified by CD_B(*E, F, e_i^**), now defined in terms of the* $\xrightarrow{B}$ *relation, in an asynchronous unicast-based message passing system with one or more Byzantine processes.*

Proof. A process sends a unicast via a point-to-point message, satisfying the properties of BRU.

- (*Managing false positives:*) Let each process p_j be responsible for adding its local history in the F_j at other processes. This can be achieved by doing a broadcast, simulated as point-to-point messages, after an application unicast send event, of control information about the application unicast send event's ID and other local (internal and receive) events' IDs since the previous such simulated broadcast. The local histories of the correct processes will be correctly recorded in F at p_i; no Byzantine processes can cause deletion of or addition to this information. Thus, if there are no Byzantine processes along some causal path from e_h^x to e_i^*, $e_h^x \xrightarrow{B} e_i^*$ will be correctly detected at p_i. Thus, false positives under $\xrightarrow{B}$ can be prevented and hence $\overline{FP_B}$.
- (*Managing false negatives:*) Consider a message m from correct process p_h to p_g sent at e_h^x in E_h. During the collection of E_h to p_i for reporting F_h, if there are no Byzantine processes along some causal path from e_h^x to e_i^*, it is possible to ensure by faithful propagation of causal dependency information along that path that no Byzantine processes can cause deletion of information about e_h^x from F_h or about other events in F that can negate $e_h^x \xrightarrow{B} e_i^*$. Thus, false negatives under $\xrightarrow{B}$ can be prevented and hence $\overline{FN_B}$.

The theorem follows. □

Theorem 7. *It is possible to solve causality determination (Definition 6) as specified by CD_B(*E, F, e_i^**), now defined in terms of the* $\xrightarrow{B}$ *relation, in an asynchronous broadcast-based message passing system with one or more Byzantine processes.*

Proof. The proof structure is similar to that of Theorems 4, 6. Similar to Theorem 4, we assume that a broadcast is sent via BRB. We outline the logic that *CD_B* (Definition 6 with $\to$ replaced by $\xrightarrow{B}$) can be solved when the underlying send events are broadcasts.

- (*Managing false positives:*) False positives cannot occur. Same reasoning as in the first bullet in Theorem 4 (thus $\overline{FP}$ holds and it implies $\overline{FP_B}$) or similar to that in the first bullet of Theorem 6. In fact, by the second bullet after Definition 6, $\overline{FP_B}$.
- (*Managing false negatives:*) False negatives cannot occur. Similar reasoning as in the second bullet of Theorem 6. Let a message m be broadcast at e_h^x. During the collection of E_h to p_i for reporting F_h, if $e_h^x \xrightarrow{B} e_i^*$ there are no Byzantine processes along some causal path from e_h^x to e_i^*, hence it is possible to ensure that no Byzantine process can cause deletion of information of e_h^x from F_h or of other events in F that can negate $e_h^x \xrightarrow{B} e_i^*$. Both $\overline{FN_B}$ and FN hold. □

It follows that to solve *CD_B* under unicasts and broadcasts, it is not necessary to identify whether each process is Byzantine. As a result, *Black_Box* $\npreceq$ *CD_B* and hence *Consensus* $\npreceq$ *CD_B*.

Although Theorems 6, 7 are positive results, in practice it is not possible to know whether the $\xrightarrow{B}$ relation holds between e_h^x and e_i^* because knowing it requires identifying each process as being either Byzantine or non-Byzantine. All it can be used for is to guarantee that if the $\xrightarrow{B}$ relation holds, then it is possible to determine causality between the corresponding two events.

Theorem 8. *It is impossible to solve causality determination (Definition 6) as specified by CD_B(*E, F, e_i^**), now defined in terms of the* $\xrightarrow{B}$ *relation, in an asynchronous multicast-based message passing system with one or more Byzantine processes.*

Proof. The properties of BRM cannot be satisfied without doing BRB within multicast group G, where $t_G < |G|/3$ and t_G is the number of Byzantine processes within group G. However to satisfy this condition for (multiple) groups implicitly requires identifying the Byzantine processes, which is not possible. Therefore, since BRM is impossible to achieve (without cryptography), detecting causality over Byzantine Reliable Multicast is also impossible to achieve. Hence we say technically that false positives and false negatives can occur under $\xrightarrow{B}$, even though, by definition, false positives cannot occur as discussed earlier.

However, if a multicast is done via point-to-point messages, without satisfying BRM properties, then FP_B and FN_B can be prevented based on Theorem 6. □

Section 4.3.2 gives an algorithm for $\overline{FP_B}$ and $\overline{FN_B}$ for unicasts satisfying BRU and for multicasts without satisfying BRM.

4.3.1. Algorithm outline for CD_B of Byzantine happened before under broadcasts

Each process p_i maintains $F_z(\forall z)$ in which it tracks p_z's execution history. The goal is to make F_z match E_z for correct p_z, at each p_i.

- Byzantine Causal Broadcast (BCB) [22] defined using the $\xrightarrow{B}$ relation on messages sent by correct processes [23,25,28], is run over Byzantine Reliable Broadcast (BRB) [36,37]. The ath broadcast by p_i of message m is denoted (m, i, a) and is done by invoking BCB(m, i, a, inc_hist) where inc_hist is the local incremental history since its last broadcast $(a-1)$. For the delivery event of a message m' in inc_hist, p_i also includes entry (m', j, b), where m' was delivered locally by the BCB layer at p_i and it was the bth broadcast by p_j.
- When p_k BCB-delivers message (m, i, a, inc_hist), p_k verifies whether each (m', j, b) corresponding to a delivery event in the received inc_hist has already been locally BCB-delivered. It should have been delivered by the causal order property of the BCB layer via a previously executed BCB-deliver, if it is not a fake entry in inc_hist; if it has not been BCB-delivered locally, p_i is a Byzantine process trying to enter a fake entry (about a receive event of message (m', j, b)) which is to be ignored. This prevents false positives. (Any event executed by a Byzantine p_i can be ignored because it is not considered by the definition of $\xrightarrow{B}$.) For each (m', j, b) that has been BCB-delivered locally the corresponding receive/deliver event at p_i and internal events at p_i up to the send event for (m, i, a) in inc_hist at p_i and the send event for (m, i, a) are inserted in F_i at p_k. Note that the BCB layer delivers a message (m, i, a, inc_hist) only when all the causal dependencies in its causal barrier have been BCB-delivered (as they must be delivered by the BRB layer at p_k if they are not fake) but inc_hist sent by p_i may contain a fake entry about an older delivery event for (m', j, b) that has dropped out of the causal barrier [22]. Hence this verification by p_k is done.
- False negatives while determining $e_h^x \xrightarrow{B} e_i^*$ at p_i cannot occur as Byzantine processes cannot modify/delete events from the causal histories reported by correct processes via broadcasting.

The above logic can be seen to be correct due to the properties of the BRB layer, on top of which the BCB layer is run and invoked while doing an application-layer broadcast. We now have that for a correct process p_i:

$e_h^x \xrightarrow{B} e_i^* \iff e_h^x$ exists in F_h at p_i.

Additionally, e_h^x in F_h at p_i implies $e_h^x \rightarrow e_i^*$ when e_h^x is a send or receive event. This is because a Byzantine process p_b cannot insert fake send and receive events e_b^y in F_b at a correct process p_i (follows from Theorem 4). Note that a Byzantine process can delete an actual internal event as well as insert a fake internal event (follows from Theorem 2).

4.3.2. Algorithm outline for CD_B of Byzantine happened before under unicasts (and under multicasts without satisfying BRM)

Unicasts are sent point-to-point, satisfying BRU. (Multicasts are sent point-to-point without satisfying BRM.) After sending the unicast/multicast, the sender does BCB($e_h^x\_id, i, a, G, inc\_hist$) over BRB of control information only. The BCB over BRB is like in Section 4.3.1; however (a) the event identifier $e_h^x\_id$ of the send event is used instead of the message m that was sent via BRB to G, (b) inc_hist also specifies event identifiers, not actual events or messages, and (c) for the receive event ID $e_i^z\_id$ in inc_hist, the event ID of the corresponding send event e_j^w is used as $(e_j^w\_id, j, b)$. On BCB-delivery of $(e_h^x\_id, i, a, G, inc\_hist)$ at p_k, p_k inserts inc_hist in F_i at p_k; each process p_i is responsible for including its inc_hists in F_i at each other process.

If there exists a causal path $e_h^x \xrightarrow{B} e_i^*$ through correct processes, there will be no false positive under $\xrightarrow{B}$ detected at p_i. Similarly, there will be no false negative under $\xrightarrow{B}$ as correct processes along the path would have caused the insertion of the actual local histories in F at p_i. Thus $\overline{FP_B}$ and $\overline{FN_B}$.

If the BCB over BRB is replaced by a regular (point-to-point) broadcast, still $\overline{FP_B}$ and $\overline{FN_B}$ once the broadcasts by the correct processes are delivered; here $t < n$ instead of $t < n/3$ as required by BRB.

4.4. Results for "happened before" allowing cryptography

4.4.1. Use of group encryption

Theorem 9. *It is impossible to solve causality determination (Definition 5) as specified by $CD(E, F, e_i^*)$ in an asynchronous multicast-based message passing system with one or more Byzantine processes even when using cryptography.*

Proof. The proof structure is similar to that for Theorem 3 but combines elements from Theorem 4. We outline the logic that CD (Definition 5) cannot be solved for when the underlying send events are multicasts. In particular, we show that the collected execution history F cannot be made to match E.

A send-receive dependency induced by a multicast send event e_h^x by p_h when it sends message m to multicast group G needs to be verified by other processes before insertion in F_h while not disclosing contents of m to processes outside G for confidentiality. So the ciphertext C_m of m signed by the group key K_G is created (to maintain confidentiality) and sent via Byzantine Reliable Broadcast (BRB) so that other processes can verify that the message was indeed sent. It is assumed that each multicast group shares a unique symmetric key for encryption and decryption of messages intended for processes in that group. (G, C_m) are the parameters of a BRB broadcast. On arrival of (G, C_m) at p_b, there are 2 cases.

1. $p_b \in G$: p_b decrypts C_m using the group key K_G. (a) If the decrypted m is valid (and p_b is non-Byzantine), a receive event e_b^y occurs and p_b can include the send event e_h^x and receive event e_b^y of C_m (in the form of control information, $\langle e_h^x, e_b^y, (G, C_m)\rangle$) on the next message it sends to help build causal chains. (b) If C_m is encrypted by a group key other than that for G (by a Byzantine sender p_h) and hence the decrypted m is garbage, a correct p_b ignores it, but a Byzantine p_b may still include the send event of m (i.e., of C_m) on a later message m' (i.e., $C_{m'}$) it sends. We treat the dependency of C_m preceding $C_{m'}$ at p_b as valid or true if we ignore the semantics of the content of m which is private to the sender p_h of m and members of G. (If the semantics of the message matters, this is a fake dependency being introduced by p_b.) This dependency is never valid and a non-Byzantine p_b will never include this dependency.
2. $p_b \notin G$: (a) If p_b is correct, there is no receive event at the application and p_b does not include the send event of C_m on any later message. (b) A Byzantine p_b may include the send event of C_m on a later message m' it sends in order to introduce a fake causal dependency of C_m preceding $C_{m'}$ but as other (correct) processes learn via the BRB of (G, C_m) that $p_b \notin G$, they will not be tricked into adding this fake dependency of C_m before $C_{m'}$. (c) If both sender p_h and receiver p_b are Byzantine and p_h shared the group key with p_b even though $p_b \notin G$, C_m is decryptable by p_b and there is a dependency from p_h to p_b. As $p_b \notin G$, no other correct process will be able to verify this dependency, resulting in a false negative. The dependency and the resulting false negative is due to *out-of-band communication*.

Thus we have the following.

- (*Managing false positives:*) By doing broadcasts using the Byzantine Reliable Broadcast (BRB) [36,37] layer, false positives can be prevented by ensuring no fake events are added to F. If a Byzantine process p_b attempts to insert a fake entry (e_h^x, G, C_m) about multicast send event of m by p_h in F_h (whether $h = b$ or $h \neq b$) at a correct process p_g via a message $(G', C_{m'})$ sent to p_g, p_g can verify whether or not this insertion is valid as based on the Reliability (or Termination) property of BRB, (G, C_m) must be delivered by the BRB layer at all correct processes including p_g. Only if (G, C_m) is delivered to p_g and $p_b \in G$ is authenticity of C_m verified (similarly for other messages in the control information of $(G', C_{m'})$).
  Once $(G', C_{m'})$ is BRB delivered, the sequence of receive events like e_b^y of (G, C_m) (and other similar messages) at p_b up to the send event of $(G', C_{m'})$, whose authenticity is verified, and that send event of $(G', C_{m'})$, can be inserted in F_b. The entry about the send event of m (that is, of C_m, to G) would be inserted in F_h because/when C_m is BRB delivered at p_g (whether or not $p_g \in G$). The receive event of $(G', C_{m'})$ occurs and is added to F_g only if $p_g \in G'$. Now in particular, p_g may be p_i because it is correct. Therefore, correct processes including p_i have a mechanism to prevent fake send events (and fake receive events) from being inserted in F, ensuring no false positives.
  Note that there is no false positive only in the sense that the dependency of C_m before $C_{m'}$ is valid and not fake, i.e., C_m was sent to p_b and was delivered to p_b before p_b sent $C_{m'}$. However, a correct process p_g can never know whether the content of C_m is decryptable by the group key K_G and is semantically sound. If semantic validity is a requirement, then this dependency of C_m before $C_{m'}$ may be fake and false positives cannot be prevented even with cryptography.
- (*Managing false negatives:*) Two types of false negatives can occur.

  1. A Byzantine process p_g can delete from F_g information about a multicast of m (i.e., of (G, C_m)) by p_h at e_h^x that it has received and such that $p_g \in G$, despite doing broadcasts using the BRB layer. Even if $e_h^x \rightarrow e_i^*$ where the causality chain passes through a message multicast event subsequently at p_g after receiving m, p_i has no way to know about this causality chain if p_g chooses not to disclose it. Further by Theorem 1, a false negative may occur if the

receive event of m, i.e., of (G, C_m), by p_g where $p_g \in G$, is not disclosed to the rest of the system by a Byzantine p_g or if the receive event is swapped after a subsequent multicast event by p_g that is part of the causality chain $e_h^x \to e_i^*$, in what p_g discloses to the rest of the system thereby breaking the causality chain of E in F.

2. A false negative may occur due to *out-of-band communication*, one form of which is as follows. If both sender p_h and receiver p_b are Byzantine and p_h shared the group key with p_b even though $p_b \notin G$, C_m is decryptable by p_b and there is a dependency from p_h to p_b. As $p_b \notin G$, no other correct process will be able to verify this dependency, resulting in a false negative.

To prevent all such false negatives, Byzantine processes, their actual execution histories, and causal chains from and through such processes need to be identified.

Note that in the bullet above regarding prevention of false positives, if C_m is not delivered to p_g within the time to report F, the entry about sending of C_m is not added to F_h even though C_m might have been sent. However the message $C_{m'}$ carrying information about sending of C_m is considered received/delivered, and hence $e_h^x \to e_i^*$. So this particular scenario contributes to a false negative.

Thus, to solve *CD*, it is necessary to identify Byzantine processes, their actual execution histories, and causal chains from and through them. Therefore *Black_Box* $\preceq$ *CD* and, as *Consensus* $\preceq$ *Black_Box*, hence *Consensus* $\preceq$ *CD*. As *Consensus* is unsolvable, *CD* is also unsolvable. □

In the proof of Theorem 9, the number of Byzantine processes $t < n/3$ due to BRB of Bracha [37].

Theorem 10. *It is impossible to solve causality determination (Definition 5) as specified by $CD(E, F, e_i^*)$ in an asynchronous unicast-based message passing system with one or more Byzantine processes even when using cryptography.*

Proof. The proof of Theorem 9 carries over identically where each multicast group consists of two processes — the sender and the receiver. False positives can be prevented only if the semantics of the message content of a message do not matter. Otherwise false positives cannot be prevented. False negatives cannot be prevented. □

Theorem 11. *It is impossible to solve causality determination (Definition 5) as specified by $CD(E, F, e_i^*)$ in an asynchronous broadcast-based message passing system with one or more Byzantine processes even when using cryptography.*

Proof. The proof of Theorem 4 which is for broadcast mode of message passing without cryptography carries over mostly identically with the two observations that

1. (*Managing false positives:*) False positives can be prevented even without cryptography, and
2. (*Managing false negatives:*) False negatives cannot be prevented due to Theorem 1 whose proof is independent of whether or not cryptography is used. □

4.4.2. Use of recursive hash histories

In an alternate cryptography-based approach, we can use event hashes, message hashes, and recursive hash histories (hash taken over the current event and the current state) [38–41] to provide a proof of the causal past in F. However, solutions based on this approach conform to the results shown in Section 4.4.1. This is because the hashes reported by Byzantine processes and those reported by correct processes impacted by the Byzantine processes may be in conformity to the events, messages and execution histories reported in F and not match E. In particular, while matching F,

- hashes over fake events can lead to FPs,
- hashes over a swapped order of (local) events can lead to FNs, and
- hashes not taken/reported over actual events that occurred can lead to FNs,

because they corroborate F and do not help to prevent contamination of F or filter out malicious entries in F.

Consider the encoding of the causal past using recursive hash histories as follows. Let H be a (cryptographic) collision-resistant hash function such that computationally it is not feasible to find y such that $H(y) = H(x)$. Let $\hat{s}_i^x$ denote the hash associated with state s_i^x.

- *Initialize:* $\hat{s}_i^0 = H(\langle s_i^0 \rangle)$.
- *At internal event* e_i^x*:* $\hat{s}_i^x = H(\langle \hat{s}_i^{x-1}, e_i^x \rangle)$.
- *At send event* e_i^x *of* m *to* p_j*:* $\hat{s}_i^x = H(\langle \hat{s}_i^{x-1}, m \rangle)$; send $(m, \hat{s}_i^x)$ to p_j.
- *At receive event* e_i^x *of* $(m, \hat{s}_j^w)$ *from* p_j*:* process m; $\hat{s}_i^x = H(\langle \hat{s}_i^{x-1}, \hat{s}_j^w, m \rangle)$.

The second component $\hat{s}_i^x$ sent with a message acts as a (recursive) proof. For a process p_g to claim that it has received a message (sent at a send event e_h^x that did not occur) it has to create the proof that encodes the causal history (causal past) up to the supposed send event. This is checked against the proof of e_h^x that p_h provides and the hashes of the events in the causal past of e_h^x. The Byzantine processes can collude to create an alternate execution history/causal past that they have agreed on among themselves and executed, and corresponding event hashes and recursive hash histories that support the alternate reality of p_g. If $n > 2t$, where t is the upper bound on the number of Byzantine processes, such Byzantine behavior can be detected by taking a majority view. Thus such ensuing false positives can be prevented. However, false negatives will still occur because hashes taken over a swapped order of events and hashes not taken/reported over actual events that occurred, with a matching F, break causal chains of E in F.

4.5. Results for "Byzantine happened before" allowing cryptography

To detect $e \xrightarrow{B} e'$, from Theorems 6, 7, false positives and false negatives can be prevented for unicasts and broadcasts even without cryptography. For multicasts, from the proof of Theorem 9, Section 4.3.2, and the fact that there is a causal path through only correct processes from e to e', only all true causal dependencies are faithfully transmitted and hence $\overline{FN_B}$ and $\overline{FP_B}$. For $\overline{FP_B}$, note that semantic validity is also guaranteed when $e \xrightarrow{B} e'$. This gives the following results.

Theorem 12. *It is possible to solve causality determination (Definition 6) as specified by $CD\_B(E, F, e_i^*)$, now defined in terms of the $\xrightarrow{B}$ relation, in an asynchronous unicast-based message passing system with one or more Byzantine processes when using cryptography.*

Theorem 13. *It is possible to solve causality determination (Definition 6) as specified by $CD\_B(E, F, e_i^*)$, now defined in terms of the $\xrightarrow{B}$ relation, in an asynchronous broadcast-based message passing system with one or more Byzantine processes when using cryptography.*

Theorem 14. *It is possible to solve causality determination (Definition 6) as specified by $CD\_B(E, F, e_i^*)$, now defined in terms of the $\xrightarrow{B}$ relation, in an asynchronous multicast-based message passing system with one or more Byzantine processes when using cryptography.*

5. Auxiliary results on relationships of *CD* to other problems

5.1. Relationship to consensus

We show two auxiliary results about the relationship of *CD* to *Consensus* in this section. First, in asynchronous systems with Byzantine failures, we show that *CD* is harder than *Consensus* by (i) showing that even if *Consensus* were solvable, *CD* cannot be solved, i.e., $CD \npreceq Consensus$, and (ii) combining with Theorems 3–5 and Theorems 9–11 which showed that *Consensus* $\preceq CD$ under Byzantine failures. Second, we show that under crash failures in asynchronous systems, *CD* is solvable but *Consensus* is not solvable by the FLP result [35]; thus $Consensus \npreceq CD$ and $CD \preceq Consensus$.

Theorem 15. *In an asynchronous system with Byzantine failures,* $CD \npreceq Consensus$ *and the CD problem is harder than Consensus.*

Proof. Let there exist an oracle accessible to each process that identifies each other process as being either correct or Byzantine. This allows each correct process to know the identity of all other correct processes. It now broadcasts its initial value of the *Consensus* problem and waits for the corresponding broadcasts from the set of other correct processes. After obtaining the initial values of all other correct processes, a correct process runs a local algorithm to decide on the consensus output — if the initial values are all the same, output that value, otherwise output a default value. This satisfies Agreement, Validity, and Termination clauses of *Consensus*. Thus, knowing the identities of all Byzantine processes, *Consensus* can be solved. The oracle is a sufficient condition to solve *Consensus*.

Let us revisit the proof of Theorem 1. In the determination of $e_h^x \to e_i^*$, a false negative may arise when a send-receive event pair (e_f^u, e_g^v) in a causal chain from e_h^x to e_i^* is missing as per F. The causal chain has the following subsequence: $\langle \cdots e_f^u, e_g^v, e_g^{v'} \cdots \rangle$, where $e_g^{v'}$ is a send event at p_g. Both e_g^v and $e_g^{v'}$ are local to p_g. A Byzantine p_g can suppress letting the rest of the system know of the occurrence of e_g^v or swap the order of occurrence of e_g^v and $e_g^{v'}$ in what it lets the rest of the system know about the occurrence of the two local events. Both actions have the effect of breaking the causality chain from e_h^x to e_i^* which can give rise to a false negative. With the oracle we assumed (a sufficient condition for solving *Consensus*), p_i can know whether or not p_g is Byzantine. Knowing that p_g is Byzantine when the causality chain is not detected by p_i as per F does not help in knowing whether p_g has broken the causality chain from e_h^x to e_i^*, or whether the causality does not exist in E. Thus, a false negative can occur if p_i infers from F and this argument holds for unicasts, multicasts, and broadcasts. If p_i wrongly guesses that the causality chain exists in E but was broken by p_g in F and assumes the causality chain existed, then a false positive can occur. Also observe that if a causality chain does not exist in E but is observed in F, a false positive can occur. This argument holds for unicasts and multicasts but not for broadcasts. Refer to the results in Table 1 and Theorems 3, 4, 5. In whichever mode of communication, false negatives and possibly false positives can occur and therefore knowing the identity of the Byzantine processes does not help to solve *CD*. Thus *CD* $\npreceq$ *Consensus*. Combining this with Theorems 3–5 and Theorems 9–11 that showed that *Consensus* $\preceq CD$, it follows that *CD* is harder than *Consensus*. □

Theorem 16. *In an asynchronous system with crash failures, CD is solvable but Consensus is not solvable; thus* $Consensus \npreceq CD$ *and* $CD \preceq Consensus$.

Proof. To solve *CD* does not require identifying the crashed processes; their (correct) execution histories can be faithfully transmitted to other processes (transitively) via the execution messages sent in the execution history itself as it grows and be present at the other (correct) processes' execution histories and in in-transit messages. The execution histories of senders that might crash can transitively propagate beginning via messages they sent before their crash to other non-crashed processes. By this logic, for a process p_h that crashes, $e_h^x \to e_i^*|_F$ is equal to $e_h^x \to e_i^*|_E$ for correct process p_i. So it suffices to consider the execution histories E_j of non-crashed processes (that include p_i) to determine $e_h^x \to e_i^*$ and solve *CD* without having to identify the crashed processes. However, solving *Consensus* in the crash failure model requires identifying the crashed processes in asynchronous systems — which is impossible by the FLP impossibility result [35]. Hence, solving *Consensus* is impossible while solving *CD* is possible under crash failures.

It follows that *Consensus* $\npreceq$ *CD* and *CD* $\preceq$ *Consensus* under crash failures. □

5.2. Causal ordering of messages (CO)

We consider the relationship of *CD* to the problem of causal ordering of messages *CO*.

Definition 7. The happened before relation → on (application-level) messages consists of the following rules:

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

Definition 8. The *causal past* of message m is denoted as $CP(m)$ and defined as the set of messages that causally precede message m under →.

The *CO* problem is specified as follows.

Definition 9. 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.

When correct process p_r receives m_2, it needs to correctly determine whether to deliver m_2 before a message m_1 or to wait for m_1 before delivery of m_2. To formulate this, we rephrase the causal ordering problem (Definition 9) as *CO*(E, F, m_2) as follows [23,25].

Definition 10. The causal ordering problem $CO(E, F, m_2)$ for a message m_2 received by a correct process p_r is to devise an algorithm to collect the execution history E as F at p_r such that $CO\_Deliv(m_2) = 1$, where

$$CO\_Deliv(m_2) = \begin{cases} 1 & \text{if } \forall m_1, m_1 \to m_2|_E = m_1 \to m_2|_F \\ 0 & \text{otherwise} \end{cases}$$

$CO\_Deliv(m_2)$ returns 1 iff $\forall m_1, m_1 \to m_2|_E = m_1 \to m_2|_F$. When 1 is returned, the algorithm output matches the actual truth and solves *CO* correctly. Thus, returning 1 indicates that the problem has been solved correctly by the algorithm using F. 0 is returned if either

1. $\exists m_1$ such that $m_1 \to m_2|_E = 1$ and $m_1 \to m_2|_F = 0$, denoting a strong safety violation because p_r will not wait for m_1 before delivery of m_2, or
2. $\exists m_1$ such that $m_1 \to m_2|_E = 0$ and $m_1 \to m_2|_F = 1$, denoting a liveness violation because p_r may continue waiting indefinitely for a fake m_1 to arrive before delivering the arrived m_2.

Theorem 17. *In an asynchronous system with Byzantine processes,* $CO \preceq CD \wedge CD \preceq CO$.

Proof. We first show $CO \preceq CD$. For instance $CO(E, F, m_2)$, let m_2 be received at p_r at event e_r^*. Invoke $CD(E, F, e_r^*)$. If this can be solved, then it is straightforward to observe that $CO(E, F, m_2)$ is solved as p_r now knows which non-fake messages m_1 to wait for before delivering m_2. Thus $CO \preceq CD$.

Next we show $CD \preceq CO$. Consider instance $CD(E, F, e_i^*)$. Let $m_2(j)$ $(\forall j \in P)$ be the most recent message received from p_j at $e_i^{h_j}$

at or before e_i^*. If $CO(E, F, m)$ can be solved, all $(n-1)$ instances $CO(E, F, m_2(j))$ can be solved. This implies that all (and only all) send and receive events causally preceding $e_i^{h_j}$ and hence e_i^* can be correctly identified. Thus there are no false negatives and no false positives, and $CD(E, F, e_i^*)$ can be solved. Thus $CD \preceq CO$. □

Theorem 18 (*[42]*). *In a system with even one Byzantine process, the CO problem is subject to the same limitations (exposure to false positives and false negatives) as the CD problem, resulting in liveness and safety violations.*

Proof. Let e_h^x be a send event of a message m_1 to p_i, e_j^z be an event where p_j sends a message m_2 to p_i, ϕ be a predicate on when/whether p_i can safely deliver m_2 sent at e_j^z to itself (i.e., has received and determines it is safe to give m_2 with respect to all other messages (like m_1) sent to itself in the execution to the application), $e_i^{\hat{y}}$ be an event where p_i delivers the message m_2 from p_j, and $\rightarrow$ be defined on application messages.

The formula $\Phi_{CO}(e_i^{\hat{y}})$ needs to be satisfied in order to solve CO, where:

$$\Phi_{CO}(e_i^{\hat{y}}) \stackrel{def}{=} \bigwedge_{e_h^x \in E \cup F} (e_h^x \rightarrow e_j^z|_E = e_h^x \rightarrow e_j^z|_F) \wedge \phi(e_i^{\hat{y}}).$$

As e_h^x is a send event, detecting $e_h^x \rightarrow e_j^z$ is susceptible to false positives and/or false negatives (refer Table 1). Thus it cannot be guaranteed that the predicate $e_h^x \rightarrow e_j^z|_E = e_h^x \rightarrow e_j^z|_F$ in the formula Φ_{CO} can be satisfied. Hence, CO cannot be solved.

A false positive of the CD problem can result in a liveness violation – waiting indefinitely at p_i for the delivery of m_2 until the prior delivery of m_1 that was never sent by p_h – in the CO problem. A false negative of the CD problem is a safety violation – not waiting for the delivery of m_1 that was sent by p_h at e_h^x to p_i – in the CO problem. □

6. Discussion and conclusions

We proved the results about possibility or impossibility of determining causality between events in the presence of Byzantine processes using executions, independent of specific implementations such as causality graphs, vector clocks and their variants, and other clock systems. The impossibility of being able to determine causal order between a pair of events under the $\rightarrow$ relation in the presence of even a single Byzantine process when message communication takes place by unicasting or by multicasting or by broadcasting are negative results. Only in the cases of unicasting and broadcasting can there be a weak positive result in that if there exists a causal path going through events at only correct processes between the two events, i.e., the $\xrightarrow{B}$ relation holds, then the causality relation can be determined correctly. However, it is impossible to ascertain whether such a path going through events at non-Byzantine processes exists, so this result is of questionable practical use. This is also an expensive operation because each broadcast must be done via Byzantine Reliable Broadcast which requires $O(n)$ control message broadcasts per application message broadcast and an increased latency that depends on the particular implementation of BRB used. We also showed that the impossibility results under the $\rightarrow$ relation remain despite allowing the use of cryptography. However in contrast, we showed possibility results under the $\xrightarrow{B}$ relation for unicasts, broadcasts, and multicasts using cryptography.

One way out of the impossibility results then is to either assume the "Byzantine" process or a parallel process can run in a OS-controlled (user) library not subject to Byzantine influence or use trusted components (such as hardware) or assume a Trusted Execution Environment (TEE) to curtail the Byzantine behavior of processes [43]. For example, trusted components (TC) can issue timestamps to events correctly — however the assumption that the processes are Byzantine is negated. Such assumptions about a *Validator* can solve problems such as Byzantine Agreement (BA) and Consensus that are known to be unsolvable in asynchronous systems. More specifically, the correctness of such approaches hinges on the following assumption.

- *Assumption:* What the process in the library/TEE does is outside Byzantine influence (and reigns in/negates actions by Byzantine processes by forcing them to do exactly as the process in the library/TEE dictates or be ignored).

If the assumption can be justified one could trivially solve problems known to be unsolvable. For example,

- *Claim:* It is possible to solve BA deterministically, with Byzantine failures bounded at $(n-2)$.
- *Reasoning:* The Validator is assumed to be reliable. Its backend executes in a TEE and can act as a correct process whose correctness is known to all processes. This Validator can receive the initial value from the initiator process and broadcast it to all processes, solving BA in $O(1)$ time (2 message hops). Any values sent by processes that contradict (only Byzantine processes' values may contradict) the values of the Validator are ignored.
- *Contradiction:* This contradicts FLP impossibility [35] because BA cannot be solved deterministically in an asynchronous fault-prone system. It also contradicts the bound (a maximum of t Byzantine processes in a system of at least $3t+1$ processes) for synchronous systems [30].
- *Conclusion:* The assumption about a reliable Validator reduces a Byzantine-prone system into a sequential system with a reliable and centralized oracle — the Validator.

Detecting causality between a pair of events is a fundamental problem [2]. Other problems that use this as a building block include the following:

- detecting causality relation between two "meta-events" [44], each of which spans multiple events across multiple processes [45,46],
- detecting the interaction type between a pair of intervals at different processes [47],
- detecting the fine-grained modality of a distributed predicate [48–50], and data-stream based global event monitoring using pairwise interactions between processes [51].

Impossibility results analogous to the ones we have shown also hold for these problems. A reduction from CD to each of the above problems can be established; the impossibility of solving these above problems would directly follow.

In light of the impossibility results in asynchronous systems, we showed that the *CD* problem can be solved in synchronous systems [52,53] using the replicated state machine (RSM) based approach [54]. As RSMs can be implemented deterministically even in partially synchronous systems, the solvability of *CD* also extends to such systems.

Based on the impossibility of solving *CD* in asynchronous systems, several other Byzantine-tolerant state observation, synchronization, and graph computation problems in asynchronous distributed systems have also been shown impossible to solve using distributed algorithms [42].

CRediT authorship contribution statement

Anshuman Misra: Writing – review & editing, Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. **Ajay D. Kshemkalyani:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] A. Misra, A.D. Kshemkalyani, Detecting causality in the presence of Byzantine processes: There is no holy grail, in: 21st IEEE International Symposium on Network Computing and Applications, NCA, 2022, pp. 73–80, http://dx.doi.org/10.1109/NCA57778.2022.10013644.

[2] R. Schwarz, F. Mattern, Detecting causal relationships in distributed computations: In search of the holy grail, Distrib. Comput. 7 (3) (1994) 149–174, http://dx.doi.org/10.1007/BF02277859.

[3] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565, http://dx.doi.org/10.1145/359545.359563.

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

[5] E. Elnozahy, Manetho: Fault Tolerance in Distributed Systems Using Rollback-Recovery and Process Replication, PhD Thesis, Tech. Rep., Tech. Report 93-212, Computer Science Department, Rice University, 1993.

[6] C.J. Fidge, Logical time in distributed computing systems, IEEE Comput. 24 (8) (1991) 28–33, http://dx.doi.org/10.1109/2.84874.

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

[8] B. Charron-Bost, Concerning the size of logical clocks in distributed systems, Inform. Process. Lett. 39 (1) (1991) 11–16, http://dx.doi.org/10.1016/0020-0190(91)90055-M.

[9] A.D. Kshemkalyani, The power of logical clock abstractions, Distrib. Comput. 17 (2) (2004) 131–150, http://dx.doi.org/10.1007/s00446-003-0105-9.

[10] R. Prakash, M. Singhal, Dependency sequences and hierarchical clocks: Efficient alternatives to vector clocks for mobile computing systems, Wirel. Netw. 3 (5) (1997) 349–360, http://dx.doi.org/10.1023/A:1019134007206.

[11] P.A.S. Ward, D.J. Taylor, Self-organizing hierarchical cluster timestamps, in: Proc. 7th International Euro-Par Conference on Parallel Processing, 2001, pp. 46–56, http://dx.doi.org/10.1007/3-540-44681-8_8.

[12] F.J. Torres-Rojas, M. Ahamad, Plausible clocks: Constant size logical clocks for distributed systems, Distrib. Comput. 12 (4) (1999) 179–195, http://dx.doi.org/10.1007/s004460050065.

[13] M. Singhal, A.D. Kshemkalyani, An efficient implementation of vector clocks, Inform. Process. Lett. 43 (1) (1992) 47–52, http://dx.doi.org/10.1016/0020-0190(92)90028-T.

[14] N.M. Preguiça, C. Baquero, P.S. Almeida, V. Fonte, R. Gonçalves, Brief announcement: efficient causality tracking in distributed storage systems with dotted version vectors, in: ACM Symposium on Principles of Distributed Computing, PODC, 2012, pp. 335–336, http://dx.doi.org/10.1145/2332432.2332497.

[15] P.S. Almeida, C. Baquero, V. Fonte, Interval tree clocks, in: Proc. 12th International Conference on Principles of Distributed Systems, OPODIS, 2008, pp. 259–274, http://dx.doi.org/10.1007/978-3-540-92221-6_18.

[16] S.S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, M. Leone, Logical physical clocks, in: Proc. 18th International Conference on Principles of Distributed Systems, OPODIS, 2014, pp. 17–32, http://dx.doi.org/10.1007/978-3-319-14472-6_2.

[17] A.D. Kshemkalyani, M. Shen, B. Voleti, Prime clock: Encoded vector clock to characterize causality in distributed systems, J. Parallel Distrib. Comput. 140 (2020) 37–51, http://dx.doi.org/10.1016/j.jpdc.2020.02.008.

[18] A.D. Kshemkalyani, A. Misra, The bloom clock to characterize causality in distributed systems, in: The 23rd International Conference on Network-Based Information Systems, NBiS 2020, in: Advances in Intelligent Systems and Computing, vol. 1264, Springer, 2020, pp. 269–279, http://dx.doi.org/10.1007/978-3-030-57811-4_25.

[19] A. Misra, A.D. Kshemkalyani, The bloom clock for causality testing, in: D. Goswami, T.A. Hoang (Eds.), Proc. 17th International Conference on Distributed Computing and Internet Technology, in: Lecture Notes in Computer Science, vol. 12582, Springer, 2021, pp. 3–23, http://dx.doi.org/10.1007/978-3-030-65621-8_1.

[20] T. Pozzetti, A.D. Kshemkalyani, Resettable encoded vector clock for causality analysis with an application to dynamic race detection, IEEE Trans. Parallel Distrib. Syst. 32 (4) (2021) 772–785, http://dx.doi.org/10.1109/TPDS.2020.3032293.

[21] K.P. Birman, T.A. Joseph, Reliable communication in the presence of failures, ACM Trans. Comput. Syst. 5 (1) (1987) 47–76, http://dx.doi.org/10.1145/7351.7478.

[22] A. Auvolat, D. Frey, M. Raynal, F. Taïani, Byzantine-tolerant causal broadcast, Theoret. Comput. Sci. 885 (2021) 55–68, http://dx.doi.org/10.1016/J.TCS.2021.06.021.

[23] A. Misra, A.D. Kshemkalyani, Solvability of Byzantine fault-tolerant causal ordering problems, in: M. Koulali, M. Mezini (Eds.), Proc. 10th International Conference on Networked Systems, NETYS, Virtual Event, May 17-19, 2022, in: Lecture Notes in Computer Science, vol. 13464, Springer, 2022, pp. 87–103, http://dx.doi.org/10.1007/978-3-031-17436-0_7.

[24] A. Misra, A.D. Kshemkalyani, Causal ordering in the presence of Byzantine processes, in: 28th IEEE International Conference on Parallel and Distributed Systems, ICPADS, 2022, pp. 130–138, http://dx.doi.org/10.1109/ICPADS56603.2022.00025.

[25] A. Misra, A.D. Kshemkalyani, Byzantine fault-tolerant causal ordering, in: 24th International Conference on Distributed Computing and Networking, ICDCN, 2023, pp. 100–109, http://dx.doi.org/10.1145/3571306.3571395.

[26] A. Misra, A.D. Kshemkalyani, Byzantine fault-tolerant causal order satisfying strong safety, in: S. Dolev, B. Schieber (Eds.), Stabilization, Safety, and Security of Distributed Systems - 25th International Symposium, in: Lecture Notes in Computer Science, vol. 14310, Springer, 2023, pp. 111–125, http://dx.doi.org/10.1007/978-3-031-44274-2_10.

[27] A. Misra, A.D. Kshemkalyani, Solvability of Byzantine fault-tolerant causal ordering: Synchronous systems case, in: J. Hong, J.W. Park (Eds.), Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, April 8-12, 2024, ACM, 2024, pp. 251–256, http://dx.doi.org/10.1145/3605098.3636063.

[28] A. Misra, A.D. Kshemkalyani, Byzantine-tolerant causal ordering for unicasts, multicasts, and broadcasts, IEEE Trans. Parallel Distrib. Syst. 35 (5) (2024) 814–828, http://dx.doi.org/10.1109/TPDS.2024.3368280.

[29] M.C. Pease, R.E. Shostak, L. Lamport, Reaching agreement in the presence of faults, J. ACM 27 (2) (1980) 228–234, http://dx.doi.org/10.1145/322186.322188.

[30] L. Lamport, R.E. Shostak, M.C. Pease, The Byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401, http://dx.doi.org/10.1145/357172.357176, URL http://doi.acm.org/10.1145/357172.357176.

[31] Y. Shang, Consensus of hybrid multi-agent systems with malicious nodes, IEEE Trans. Circuits Syst. II Express Briefs 67-II (4) (2020) 685–689, http://dx.doi.org/10.1109/TCSII.2019.2918752.

[32] Y. Shang, Resilient cluster consensus of multiagent systems, IEEE Trans. Syst. Man Cybern. Syst. 52 (1) (2022) 346–356, http://dx.doi.org/10.1109/TSMC.2020.2997855.

[33] C. Dwork, N.A. Lynch, L.J. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323, http://dx.doi.org/10.1145/42282.42283, URL http://doi.acm.org/10.1145/42282.42283.

[34] Y. Shang, A system model of three-body interactions in complex networks: consensus and conservation, Proc. R. Soc. A 478 (2022) 20210564, URL https://doi.org/10.1098/rspa.2021.0564.

[35] M.J. Fischer, N.A. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, J. ACM 32 (2) (1985) 374–382, http://dx.doi.org/10.1145/3149.214121.

[36] G. Bracha, S. Toueg, Asynchronous consensus and broadcast protocols, J. ACM 32 (4) (1985) 824–840, http://dx.doi.org/10.1145/4221.214134.

[37] G. Bracha, Asynchronous Byzantine agreement protocols, Inform. and Comput. 75 (2) (1987) 130–143, http://dx.doi.org/10.1016/0890-5401(87)90054-X.

[38] J.D. Cohen, Recursive hashing functions for n-grams, ACM Trans. Inf. Syst. 15 (3) (1997) 291–320, http://dx.doi.org/10.1145/256163.256168.

[39] B. Kang, R. Wilensky, J. Kubiatowicz, The hash history approach for reconciling mutual inconsistency, in: 23rd International Conference on Distributed Computing Systems, 2003, pp. 670–677, http://dx.doi.org/10.1109/ICDCS.2003.1203518.

[40] M. Kleppmann, H. Howard, Byzantine eventual consistency and the fundamental limits of peer-to-peer databases, 2020, CoRR abs/2012.00472. arXiv:2012.00472. URL https://arxiv.org/abs/2012.00472.

[41] F. Jacob, H. Hartenstein, Logical clocks and monotonicity for Byzantine-tolerant replicated data types, in: Proceedings of the 11th Workshop on Principles and Practice of Consistency for Distributed Data, PaPoC 2024, ACM, 2024, pp. 37–43, http://dx.doi.org/10.1145/3642976.3653034.

[42] A.D. Kshemkalyani, A. Misra, Impossibility results for Byzantine-tolerant state observation, synchronization, and graph computation problems, Algorithms 18 (2025) 26, http://dx.doi.org/10.3390/a18010026.

[43] G. Sun, T. Tao, Y. Guo, M.H. Yiqing, J. Li, Building a verifiable logical clock for P2P networks, 2024, http://dx.doi.org/10.48550/ARXIV.2405.13349, CoRR abs/2405.13349. arXiv:2405.13349.

[44] A.D. Kshemkalyani, A framework for viewing atomic events in distributed computations, Theoret. Comput. Sci. 196 (1–2) (1998) 45–70, http://dx.doi.org/10.1016/S0304-3975(97)00195-3.

[45] A.D. Kshemkalyani, Causality and atomicity in distributed computations, Distrib. Comput. 11 (4) (1998) 169–189, http://dx.doi.org/10.1007/s004460050048.

[46] A.D. Kshemkalyani, Reasoning about causality between distributed nonatomic events, Artificial Intelligence 92 (1–2) (1997) 301–315, http://dx.doi.org/10.1016/S0004-3702(97)00004-0.

[47] A.D. Kshemkalyani, Temporal interactions of intervals in distributed systems, J. Comput. System Sci. 52 (2) (1996) 287–298, http://dx.doi.org/10.1006/jcss.1996.0022.

[48] A.D. Kshemkalyani, R. Kamath, Orthogonal relations for reasoning about posets, Int. J. Intell. Syst. 17 (12) (2002) 1101–1110, http://dx.doi.org/10.1002/int.10062.

[49] P. Chandra, A.D. Kshemkalyani, Causality-based predicate detection across space and time, IEEE Trans. Comput. 54 (11) (2005) 1438–1453, http://dx.doi.org/10.1109/TC.2005.176.

[50] A.D. Kshemkalyani, A fine-grained modality classification for global predicates, IEEE Trans. Parallel Distrib. Syst. 14 (8) (2003) 807–816, http://dx.doi.org/10.1109/TPDS.2003.1225059.

[51] P. Chandra, A.D. Kshemkalyani, Data-stream-based global event monitoring using pairwise interactions, J. Parallel Distrib. Comput. 68 (6) (2008) 729–751, http://dx.doi.org/10.1016/j.jpdc.2008.01.006.

[52] A. Misra, A.D. Kshemkalyani, Detecting causality in the presence of Byzantine processes: The synchronous systems case, in: 30th International Symposium on Temporal Representation and Reasoning, TIME, in: LIPIcs, vol. 278, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 11:1–11:14, http://dx.doi.org/10.4230/LIPICS.TIME.2023.11.

[53] A. Misra, A.D. Kshemkalyani, Detecting causality in the presence of Byzantine processes: The case of synchronous systems, Inform. and Comput. 301 (2024) 105212, http://dx.doi.org/10.1016/J.IC.2024.105212.

[54] F.B. Schneider, Implementing fault-tolerant services using the state machine approach: A tutorial, ACM Comput. Surv. 22 (4) (1990) 299–319, http://dx.doi.org/10.1145/98163.98167.