Detecting Causality in the Presence of Byzantine Processes: There is No Holy Grail

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

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

***Abstract*—Detecting causality or the happens before relation between events in an asynchronous distributed system is a fundamental building block for 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) In an execution where there exists a causal path between two events passing through only correct processes, the impossibility result for unicasts remains. (5) However, when processes communicate by broadcasting and there exists a causal path between two events passing through only correct processes, it is possible to detect causality between such a pair of events. (6) In an execution where processes communicate by multicasting and there exists a causal path between two events passing through only correct processes, we prove that the impossibility result for multicasts remains.**

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

I. Introduction

Causality is an important tool in understanding and reasoning about distributed systems [1]. In a seminal paper, Lamport formulated the "happens before" or the causality relation between events in an asynchronous distributed system [2]. Applications of causality tracking 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 [3]. The causality relation between events is also used to define causality between messages, and forms the basis of causal ordering of messages [4], which 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 distributed virtual environments.

The causality relation between events can be captured by tracking causality graphs [5], scalar clocks [2], vector clocks [6]–[8], and numerous other variants (such as hierarchical clocks [9], [10], plausible clocks [11], incremental clocks [12], dotted version vectors [13], interval tree clocks [14], logical physical clocks [15], encoded vector clocks [16], and Bloom clocks [17], [18] to mention a few), proposed since Lamport's seminal paper [2]. See [1], [3] or a more recent survey included in [19]. 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 [1], 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 (non-faulty).

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

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. Causal ordering of messages under the Byzantine failure model has recently been examined in [20] for broadcast communication and in [21]–[23] for unicast, multicast, as well as broadcast communication. Causal consistency of replicated data stores under Byzantine failures based on broadcasts has been considered in [24]–[27].

Our main result is that it is impossible to determine the causality or the happens before relation $\rightarrow$ between two events $e1$ and $e2$ when there is even a single Byzantine process in an asynchronous distributed system. In light of this negative result, we investigated whether any positive result can be shown in a system with stronger assumptions. For this, we introduced the Byzantine happens before relation $\xrightarrow{B}$ by which $e1$ is related to $e2$ 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 broadcast communication.

Contributions: 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. This is because both false positives and false negatives can occur.

Mode of communication	Detecting "happens before" $e \rightarrow e'$	Detecting "Byzantine happens before" $e \xrightarrow{B} e'$
Unicasts	Impossible, Theorem 1 FP, FN	Impossible, Theorem 4 $FP_B (\Longrightarrow FP), \overline{FN_B} \wedge FN$
Broadcasts	Impossible, Theorem 2 $\overline{FP}, FN$	Possible, Theorem 5 $\overline{FP} (\Longrightarrow \overline{FP_B}), \overline{FN_B} \wedge FN$
Multicasts	Impossible, Theorem 3 FP, FN	Impossible, Theorem 6 $FP_B (\Longrightarrow FP), \overline{FN_B} \wedge FN$

TABLE I: 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 $\xrightarrow{B}$.

2) We also prove a similar impossibility result when processes communicate by broadcasting. 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. 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, we prove that the impossibility result for unicasts remains. In this case, false positives can occur but false negatives cannot occur.
5) However, when processes communicate by broadcasting and 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. Neither false positives nor false negatives can occur.
6) In an execution where processes communicate by multicasting and there exists a causal path between two events passing through only correct processes, we prove that the impossibility result for multicasts remains. False positives can occur but false negatives cannot occur.

Table I summarizes these results. The structure of our proofs is motivated by the structure of proofs about enforcing causal order in asynchronous systems with Byzantine processes [21], [23].

Roadmap. Section II gives the system model. Section III formulates the problem of detecting causality. Section IV proves the results outlined under "Contributions" above. Section V gives a discussion and concludes.

II. System Model

This paper deals with an asynchronous distributed system having Byzantine processes which are processes that can misbehave [28], [29]. A correct process behaves exactly as specified by the algorithm whereas a Byzantine process may exhibit arbitrary behaviour including crashing at any point during the execution. A Byzantine process cannot impersonate another process or spawn new processes.

The distributed system is modelled 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 (logical) communication links over which processes communicate by message passing. The channels are assumed to be FIFO. G is a complete graph.

The distributed system is assumed to be asynchronous, i.e., there is no known fixed upper bound δ on the message latency, nor any known fixed upper bound ψ on the relative speeds of processors [30].

We do not consider the use of digital signatures or cryptographic techniques in the system model because of their high cost as well as hidden/implicit assumptions such as bounds on message latency which makes them inappropriate for truly asynchronous systems.

Let e_i^x, where $x \geq 1$, denote the x-th 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 $\langle s_i^0, e_i^1, s_i^1, e_i^2, s_i^2 \ldots \rangle$. The *happens before* [2] 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 happens before relation $\rightarrow$ on events consists of the following rules:*

1) ***Program Order**: For the sequence of events $\langle e_i^1, e_i^2, \ldots \rangle$ executed by process p_i, $\forall\ j, k$ such that $j < k$ we have $e_i^j \rightarrow e_i^k$.*
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 in E that causally precede e under $\rightarrow$.*

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

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

1) ***Program Order**: For the sequence of events $\langle e_i^1, e_i^2, \ldots \rangle$ executed by a correct process p_i, $\forall\ j, k$ such that $j < k$ we have $e_i^j \xrightarrow{B} e_i^k$.*

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 $e \xrightarrow{B} e''$.*

The Byzantine causal past of an event is defined as follows:

Definition 4. *The Byzantine causal past of event e, denoted as $BCP(e)$, is defined as the set of events in E^c that causally precede e under $\xrightarrow{B}$.*

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

III. 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 and let $E = \bigcup_i \{E_i\}$. For any causality determination algorithm, let F_i be the execution history at p_i as collected by the algorithm and let $F = \bigcup_i \{F_i\}$. F thus denotes the execution history as collected by the algorithm.

Let $e1 \rightarrow e2|_E$ and $e1 \rightarrow e2|_F$ be the evaluation (1 or 0) of $e1 \rightarrow e2$ 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 $e_h^x \rightarrow e_i^*$. 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 suffix of $CP(e_i^*)$. Byzantine processes may collude as follows.

1) To delete e_h^x from F_h and manipulate F such that $e_h^x \rightarrow e_i^*|_F = 0$, while $e_h^x \rightarrow e_i^*|_E = 1$, or
2) To add a fake event e_h^x in F_h and manipulate F such that $e_h^x \rightarrow e_i^*|_F = 1$, while $e_h^x \rightarrow e_i^*|_E = 0$.

Let $T(E)$ and $T(F)$ denote the set of all events in E and F, respectively. 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.

Observe that for Item 1 above, it suffices to consider only send events as e_h^x because an internal event or a receive event e_h^u can be deleted from F if and only if a subsequent send event e_h^x, where $u < x$, and satisfying $e_h^x \rightarrow e_i^*|_E$ can be deleted from F. For Item 2 above, it suffices to consider only send events as e_h^x because a fake internal event or a fake receive event e_h^u can be added to F if and only if a subsequent fake send event e_h^x, where $u < x$, and satisfying $e_h^x \rightarrow e_i^*|_F$ can be added to F. Therefore, rather than consider $e_h^x \in T(E) \cup T(F)$, it suffices to consider $e_h^x \in S(E) \cup S(F)$, where $S(E)$ and $S(F)$ denote the set of all send events in E and F, respectively.

Definition 5. *The causality determination problem $CD(E, F, e_i^*)$ is to devise an algorithm to collect the execution history E as F and evaluate F at a correct process p_i such that: the problem returns 1 iff $\forall e_h^x, e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F$.*

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 either

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

To determine whether *CD* is solved correctly, we have to evaluate $\forall e_h^x, e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F$ even if $e_h^x \in (S(E) \cup S(F)) \setminus S(E)$ because such an e_h^x is recorded by the algorithm as part of F. The key observation we make is that 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 but also between other E_z and F_z by contaminating F_z via direct and transitive message passing originated at p_b.

IV. Impossibility and Possibility Results

A. Results for "Happens Before"

Theorem 1. *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 by showing:

1) a reduction (denoted $\preceq$) from *Black_Box* to *CD*, where *Black_Box* is defined below,
2) a reduction from the *Consensus* problem (which by the FLP result [31] is unsolvable in the presence of a single Byzantine process) to the *Black_Box* problem.

Specifically, we show how *Consensus* can be solved by invoking a black box that solves *Black_Box*, and how *Black_Box* can be solved by solving *CD*. If *CD* were solvable, *Black_Box* would be solvable, and then *Consensus* would also be solvable but that contradicts the unsolvability of *Consensus*. Therefore, there cannot exist any algorithm to solve *CD*.

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

$$w = \begin{cases} 0 & \text{if each correct } p_j \text{ has } V[j] = 0 \\ 1 & \text{if each correct } p_j \text{ has } V[j] = 1 \\ \bigwedge_{e_h^x} (e_h^x \rightarrow e_i^*|_E = & \\ \quad e_h^x \rightarrow e_i^*|_F) & \text{otherwise} \end{cases}$$

Black_Box is solvable if *CD* at p_i is solvable correctly because solving *CD* requires using the execution histories of potentially Byzantine processes as recorded by the algorithm in F. In order for any algorithm to correctly solve *CD*, it must ensure that F matches E. For this, the following must hold.

- A Byzantine process may attempt to insert a fake entry in F_h and contaminate the reporting of histories in F, leading to a false positive because $S(F) \setminus S(E) \neq \emptyset$.

Therefore, either contamination of F has be prevented or malicious entries have to be filtered out from F within bounded time. But due to unicasting, 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 keep it private. Therefore identification of Byzantine processes, their actual execution histories, and causal chains from them is required.

- Let there be 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 $e_h^x \rightarrow e_i^*$; note that we also have $S(E) \setminus S(F) \neq \emptyset$ because of this. Therefore, either deletion of information from E in F has to be prevented, or such deletions 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 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 (to trace and deal with/resolve the impact of contamination via message passing by the Byzantine processes from those Byzantine sources on the execution histories of other processes)*. Thus, *Black_Box* $\preceq$ *CD*.

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 [28], [29].

- 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.

When *Consensus*($\overline{V}$) is to be solved, the black box is invoked for *Black_Box*($\overline{V}, E, F, e_i^*$). Each correct process outputs as its consensus value the value that it receives from p_i and terminates. Agreement, Validity, and Termination clauses of *Consensus* 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^*). We now have

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

This implies that if the *CD* problem is solvable, then *Consensus* is also solvable. That contradicts the FLP impossibility result when applied to a Byzantine system, hence *CD* is not solvable. □

Digression. It is worth observing that under the crash-failure model, even though *Consensus* $\preceq$ *Black_Box*, we have that *Black_Box* $\not\preceq$ *CD*. This latter relation $\not\preceq$ is because solving *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 to other non-crashed processes. In other words, the execution history of any prefix of an execution can be represented by that execution. Therefore, $S(E) = S(F)$. Hence, it suffices to consider the execution histories E_i of non-crashed processes (that include p_i) to determine $e_h^x \rightarrow e_i^*$ without having to identify the crashed processes.

When the communication pattern is by broadcasts, the proof analyzing the *CD* problem uses Byzantine Reliable Broadcast (BRB) [32], [33] 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 has been defined to satisfy 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).
- Integrity: For any message m, a correct process executes deliver(m) at most once.
- Self-delivery: If a correct process executes broadcast(m), then it eventually executes deliver(m).
- Reliability (or Termination): If a correct process executes deliver(m), then every other correct process also (eventually) executes deliver(m).

Theorem 2. *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 proof structure is similar to that for Theorem 1. We outline the logic that *CD* (Definition 5) cannot be solved for when the underlying send events are broadcasts. We show that F cannot be made to match E.

- By doing broadcasts using the Byzantine Reliable Broadcast (BRB) [32], [33] layer, false positives can be prevented by ensuring no fake events are added to F, whereby $S(F) \setminus S(E) = \emptyset$. If 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, p_g can verify whether or not this insertion is valid as based on the Reliability (or Termination) property of BRB, m must be delivered by the BRB layer at all correct processes including p_g. Only if m is delivered to p_g is authenticity of m verified and the entry about m can be inserted in F_h. Unless that happens, the message m' trying to insert the entry is ignored and is not considered received/delivered. 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 from being inserted in F, ensuring $S(F) \setminus S(E) = \emptyset$.

- 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 by p_g, p_i has no way of knowing about e_h^x and the message sent to it at e_h^x, without waiting indefinitely. Thus, $S(E) \setminus S(F) \neq \emptyset$ and false negatives may occur. To prevent such false negatives, Byzantine processes, their actual execution histories, and causal chains from 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 not considered received/delivered, and hence $e_h^x \not\rightarrow e_i^*$. So this particular scenario does not contribute to a false negative.

Thus, to solve *CD*, it is necessary to identify Byzantine processes, their actual execution histories, and causal chains from them. Therefore *Black_Box* $\preceq$ *CD* and hence *Consensus* $\preceq$ *CD*. As *Consensus* is unsolvable, *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 3. *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 is a special case of multicast where each group is of size 1 (or 2 if the sender is included in the multicast group). From Theorem 1, causality determination in the presence of even a single Byzantine process in an execution with unicast communication is impossible to solve. As a special case of group size 1 (or 2) is not solvable, the general case of multicast is also not solvable. □

B. Results for "Byzantine Happens 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.

Definition 6. *The causality determination problem $CD(E, F, e_i^*)$ is to devise an algorithm to collect the execution history E as F and evaluate F at a correct process p_i such that: the problem returns 1 iff $\forall e_h^x, e_h^x \xrightarrow{B} e_i^*|_E = e_h^x \xrightarrow{B} e_i^*|_F$.*

The problem is solved correctly iff 1 is returned. Observe, $e \xrightarrow{B} e'$ is equivalent to the following: ($e \rightarrow e' \wedge$ *there is a causal path from event e to event 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*). (Likewise for $e \xrightarrow{B} e'|_E$.) Note that evaluating $e \xrightarrow{B} e'|_F$ does not involve determining whether there actually exists the causal path going through correct processes.

Value 0 is returned to CD if either

- $\exists e_h^x$ 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), or
- $\exists e_h^x$ such that $e_h^x \rightarrow e_i^*|_E = 0 \wedge e_h^x \rightarrow e_i^*|_F = 1 \wedge$ *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).

Theorem 4. *It is impossible to solve causality determination (Definition 6) as specified by* CD(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. The proof of Theorem 1 carries identically, subject to the following changes. In the specification of *Black_Box*, the definition $\bigwedge_{e_h^x} (e_h^x \xrightarrow{B} e_i^*|_E = e_h^x \xrightarrow{B} e_i^*|_F)$ instead of $\bigwedge_{e_h^x} (e_h^x \rightarrow e_i^*|_E = e_h^x \rightarrow e_i^*|_F)$ is used.

That *Consensus* $\preceq$ *Black_Box* still holds is self-evident. *Black_Box* $\preceq$ *CD* still holds because solving *CD* correctly still requires using the execution histories of Byzantine processes as recorded by the algorithm in F, similar to the proof for Theorem 1. In order for any algorithm to correctly solve *CD*, it must ensure that F matches E. For this, the following must hold.

- Due to unicasting, a message m from a potentially Byzantine p_h to p_g in F_h, 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 keep it private. Thus, a fake entry may be inserted/injected in F_h by a (some) Byzantine process, even if there exists one causal path going through correct processes from p_h to p_i. This leads to a false positive because $S(F) \setminus S(E) \neq \emptyset$. Thus FP_B holds and it implies FP also holds. Therefore, either contamination of F has to be prevented or malicious entries have to be filtered out from F within bounded time. This requires identifying Byzantine processes, their actual execution histories, and causal message chains from them.
- Let there be 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, as there are no Byzantine processes along some causal path from e_h^x to e_i^*, it is possible to ensure that no Byzantine processes can cause deletion of

information about e_h^x from F_h, thus $(S(E))^B \setminus S(F) = \emptyset$, where $(S(E))^B$ is the set of send events of $S(E)$ from which there exists a path through correct processes to e_i^*. Thus, false negatives (with respect to events in $(S(E))^B$) can be prevented at p_i and hence $\overline{FN_B}$. However, other false negatives can occur and hence FN also holds.

If there were an algorithm to make F match E, it *still requires identifying whether each of the processes that input their execution histories is correct or Byzantine (to trace and deal with/resolve the impact of contamination via message passing by the Byzantine processes from those Byzantine sources on the execution histories of other processes).* Hence *Black_Box* $\preceq$ *CD*. The theorem follows. □

Theorem 5. *It is possible to solve causality determination (Definition 6) as specified by* CD(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 2, 4. We outline the logic that *CD* (Definition 6 with $\rightarrow$ replaced by $\xrightarrow{B}$) can be solved when the underlying send events are broadcasts. We show that F can be made to match E.

- $S(F) \setminus S(E) = \emptyset$, hence false positives cannot occur. Same reasoning as in the first bullet in Theorem 2. Thus $\overline{FP}$ holds and it implies $\overline{FP_B}$.
- $(S(E))^B \setminus S(F) = \emptyset$, hence false negatives cannot occur. Similar reasoning as in the second bullet of Theorem 4. Let a message m be broadcast at e_h^x. During the collection of E_h to p_i for reporting F_h, as $e_h^x \xrightarrow{B} e_i^*$ 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, thus $(S(E))^B \setminus S(F) = \emptyset$. Both $\overline{FN_B}$ and FN hold.

Thus to solve *CD* under broadcasts, it is not necessary to identify whether each process is Byzantine, hence *Black_Box* $\not\preceq$ *CD* and hence *Consensus* $\not\preceq$ *CD*. □

Although Theorem 5 is a positive result, 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 6. *It is impossible to solve causality determination (Definition 6) as specified by* CD(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. Unicast is a special case of multicast where each group is of size 1 (or 2 if the sender is included in the multicast group). From Theorem 4, causality determination in the presence of even a single Byzantine process in an execution with unicast communication is impossible to solve. As a special case of group size 1 (or 2) is not solvable, the general case of multicast is also not solvable. □

1) Algorithm Outline for CD *of Byzantine Happens 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 process p_i.

- Byzantine Causal Broadcast (BCB) [20] is run over Byzantine Reliable Broadcast (BRB) [32], [33]. 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 due to causal order of the BCB layer beneath, 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. 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. Hence this verification by p_k is done.

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:

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

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. Note that a Byzantine process can delete an actual internal event as well as insert a fake internal event.

V. 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 in the presence of even a single Byzantine process when message communication takes place by unicasting and by multicasting are negative results. Only in the case of

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, then causal order 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.

It remains to be explored whether the happens before and the Byzantine happens before relations between events can be determined in the unicast, multicast, and broadcast communication models in a synchronous system.

Detecting causality between a pair of events is a fundamental problem [1]. Other problems that use this problem as a building block include the following:

- detecting causality relation between two "meta-events" [34], each of which spans multiple events across multiple processes [35],
- detecting the interaction type between a pair of intervals at different processes [36],
- detecting the fine-grained modality of a distributed predicate [37], [38], and data-stream based global event monitoring using pairwise interactions between processes [39].

Impossibility results analogous to Theorem 1 (unicast, $\rightarrow$), Theorem 2 (broadcast, $\rightarrow$), Theorem 3 (multicast, $\rightarrow$), Theorem 4 (unicast, $\overset{B}{\rightarrow}$), Theorem 6 (multicast, $\overset{B}{\rightarrow}$) may also hold for these problems. If a reduction from the causality determination problem to each of the above problems can be established, then the impossibility of solving these above problems would directly follow.

REFERENCES

[1] R. Schwarz and F. Mattern, "Detecting causal relationships in distributed computations: In search of the holy grail," *Distributed Comput.*, vol. 7, no. 3, pp. 149–174, 1994. [Online]. Available: https://doi.org/10.1007/BF02277859

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

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

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

[5] E. Elnozahy, "Manetho: Fault tolerance in distributed systems using rollback-recovery and process replication, phd thesis," Tech. Report 93-212, Computer Science Department, Rice University, Tech. Rep., 1993.

[6] C. J. Fidge, "Logical time in distributed computing systems," *IEEE Computer*, vol. 24, no. 8, pp. 28–33, 1991.

[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," *Inf. Process. Lett.*, vol. 39, no. 1, pp. 11–16, 1991. [Online]. Available: https://doi.org/10.1016/0020-0190(91)90055-M

[9] P. A. S. Ward and D. J. Taylor, "Self-organizing hierarchical cluster timestamps," in *Proc. 7th International Euro-Par Conference on Parallel Processing*, 2001, pp. 46–56.

[10] ——, "A hierarchical cluster algorithm for dynamic, centralized timestamps," in *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001)*, 2001, pp. 585–593.

[11] F. J. Torres-Rojas and M. Ahamad, "Plausible clocks: Constant size logical clocks for distributed systems," *Distributed Computing*, vol. 12, no. 4, pp. 179–195, 1999. [Online]. Available: https://doi.org/10.1007/s004460050065

[12] M. Singhal and A. D. Kshemkalyani, "An efficient implementation of vector clocks," *Inf. Process. Lett.*, vol. 43, no. 1, pp. 47–52, 1992. [Online]. Available: https://doi.org/10.1016/0020-0190(92)90028-T

[13] N. M. Preguiça, C. Baquero, P. S. Almeida, V. Fonte, and 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.

[14] P. S. Almeida, C. Baquero, and V. Fonte, "Interval tree clocks," in *Proc. 12th International Conference on Principles of Distributed Systems, OPODIS*, 2008, pp. 259–274.

[15] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, "Logical physical clocks," in *Proc. 18th International Conference on Principles of Distributed Systems, OPODIS*, 2014, pp. 17–32.

[16] A. D. Kshemkalyani, M. Shen, and B. Voleti, "Prime clock: Encoded vector clock to characterize causality in distributed systems," *J. Parallel Distributed Comput.*, vol. 140, pp. 37–51, 2020. [Online]. Available: https://doi.org/10.1016/j.jpdc.2020.02.008

[17] A. D. Kshemkalyani and A. Misra, "The bloom clock to characterize causality in distributed systems," in *Advances in Networked-Based Information Systems - The 23rd International Conference on Network-Based Information Systems, NBiS 2020, Victoria, BC, Canada, 31 August - 2 September 2020*, ser. Advances in Intelligent Systems and Computing, L. Barolli, K. F. Li, T. Enokido, and M. Takizawa, Eds., vol. 1264. Springer, 2020, pp. 269–279. [Online]. Available: https://doi.org/10.1007/978-3-030-57811-4_25

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

[19] T. Pozzetti and A. D. Kshemkalyani, "Resettable encoded vector clock for causality analysis with an application to dynamic race detection," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 4, pp. 772–785, 2021. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3032293

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

[21] A. Misra and A. D. Kshemkalyani, "Solvability of byzantine fault-tolerant causal ordering problems," in *Proc. 10th International Conference on Networked Systems, NETYS, Virtual Event, May 17-19, 2022*, ser. Lecture Notes in Computer Science, M. Koulali and M. Mezini, Eds., vol. 13464. Springer, 2022, pp. 87–103. [Online]. Available: https://doi.org/10.1007/978-3-031-17436-0_7

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

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

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

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

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

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

[28] M. C. Pease, R. E. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, no. 2, pp. 228–234, 1980.

[29] L. Lamport, R. E. Shostak, and M. C. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[30] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[31] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[32] G. Bracha and S. Toueg, "Asynchronous consensus and broadcast protocols," *J. ACM*, vol. 32, no. 4, p. 824–840, Oct. 1985.

[33] G. Bracha, "Asynchronous byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[34] A. D. Kshemkalyani, "A framework for viewing atomic events in distributed computations," *Theor. Comput. Sci.*, vol. 196, no. 1-2, pp. 45–70, 1998. [Online]. Available: https://doi.org/10.1016/S0304-3975(97)00195-3

[35] ——, "Causality and atomicity in distributed computations," *Distributed Comput.*, vol. 11, no. 4, pp. 169–189, 1998. [Online]. Available: https://doi.org/10.1007/s004460050048

[36] ——, "Temporal interactions of intervals in distributed systems," *J. Comput. Syst. Sci.*, vol. 52, no. 2, pp. 287–298, 1996. [Online]. Available: https://doi.org/10.1006/jcss.1996.0022

[37] ——, "A fine-grained modality classification for global predicates," *IEEE Trans. Parallel Distributed Syst.*, vol. 14, no. 8, pp. 807–816, 2003. [Online]. Available: https://doi.org/10.1109/TPDS.2003.1225059

[38] P. Chandra and A. D. Kshemkalyani, "Causality-based predicate detection across space and time," *IEEE Trans. Computers*, vol. 54, no. 11, pp. 1438–1453, 2005. [Online]. Available: https://doi.org/10.1109/TC.2005.176

[39] ——, "Data-stream-based global event monitoring using pairwise interactions," *J. Parallel Distributed Comput.*, vol. 68, no. 6, pp. 729–751, 2008. [Online]. Available: https://doi.org/10.1016/j.jpdc.2008.01.006