Distrib. Comput. (1998) 11: 91–111

DISTRIBUTED COMPUTING



# Necessary and sufficient conditions on information for causal message ordering and their optimal implementation ⋆

**Ajay D. Kshemkalyani**[1], **Mukesh Singhal**[2]

[1] ECECS Department, P.O. Box 210030, University of Cincinnati, Cincinnati, OH 45221-0030, USA
(e-mail: ajayk@ececs.uc.edu)
[2] Department of Computer and Information Science, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210, USA
(e-mail: singhal@cis.ohio-state.edu)

Received: January 1996 / Accepted: February 1998

**Summary.** This paper formulates necessary and sufficient conditions on the information required for enforcing causal ordering in a distributed system with asynchronous communication. The paper then presents an algorithm for enforcing causal message ordering. The algorithm allows a process to multicast to arbitrary and dynamically changing process groups. We show that the algorithm is optimal in the space complexity of the overhead of control information in both messages and message logs. The algorithm achieves optimality by transmitting the bare minimum causal dependency information specified by the necessity conditions, and using an encoding scheme to represent and transmit this information. We show that, in general, the space complexity of causal message ordering in an asynchronous system is $\Omega(n^2)$, where $n$ is the number of nodes in the system. Although the upper bound on space complexity of the overhead of control information in the algorithm is $O(n^2)$, the overhead is likely to be much smaller on the average, and is always the least possible.

**Key words:** Causal message ordering – Distributed systems – Synchronization – Concurrency

## 1 Background and previous work

A distributed system consists of a collection of geographically dispersed autonomous sites connected by a communication network. The sites do not share any memory and communicate solely by message passing. Message propagation delay is finite but unpredictable, and between a pair of sites, messages may be delivered out of order. There is no common physical clock.

The execution of a process at a site is modeled by three types of events, namely, message send, message delivery[1], and internal events. An internal event represents a local computation at the process, whereas message send and delivery events establish cause and effect relationships among the processes. The cause and effect relationship between the events of a distributed execution is captured by the *happened before* or *causality* relation ($\longrightarrow$) [14] which defines a partial order on the events.

There exist several paradigms for ordered delivery of messages in a distributed system (see Fig. 1). Synchronous communication between processes, that tantamounts to instantaneous message delivery, simplifies the design, verification, and analysis of distributed applications. However, it results in a loss in concurrency within the distributed application because each message exchange requires a handshake between the sender and the receiver. FIFO and non-FIFO communication on each channel are asynchronous and provide much more concurrency to the distributed application, but the asynchronous execution of processes and unpredictable communication delays create nondeterminism in distributed systems that complicates the design, verification, and analysis of distributed applications. To simplify the design and development of distributed applications, while retaining much of the concurrency provided by the asynchronous communication, the idea of *causal message ordering* (CO) was introduced by Birman and Joseph [4]. Causal message ordering guarantees that if the send events for two messages are ordered by causality and the messages have a common destination, then the messages are delivered to that destination in the causal order of their send events. In Fig. 1, message $M3$ violates causal ordering but message $M4$ does not. Birman and Joseph defined several group communication primitives, of which ABCAST is relevant to causal ordering [4]. ABCAST guarantees that for any two multicasts, all common destinations see the two multicasts in some common order, and this order need not be CO even if the multicast send events are related by causality. ABCAST is weaker than synchronous communication, and does not offer the full concurrency of FIFO communication. ABCAST provides a total ordering of messages, and is not quite com-

⋆ The results of this paper appear in [10] and a brief announcement of the optimal implementation of the results appears in [11].
*Correspondence to:* A.D. Kshemkalyani

[1] It is important to distinguish between the arrival of a message and its delivery. The arrival of a message signifies that the communication network has placed the message in the buffer of the receiving process. The delivery of a message means that the process has taken up the message for processing.

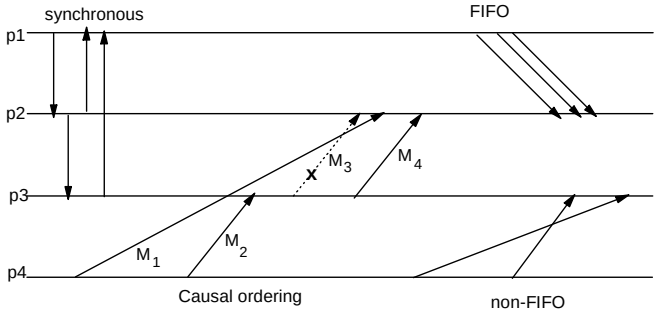


**Fig. 1.** Message ordering paradigms

parable to CO. The total ordering property of ABCAST is useful for a different and smaller range of applications than the causal ordering property.

Causal ordering provides a built-in message synchronization and reduces the nondeterminism in a distributed computation. Causal ordering provides an equivalent of the FIFO channel property at a global (communication network) level where sequential send of messages along a communication channel is replaced by causally related sends of messages going to the same destination over a communication network. Note that the causal ordering property is strictly stronger than the FIFO property. Also note that synchronous communication trivially guarantees CO without any overhead. A formal study of the relation between synchronous communication, causal ordering, and asynchronous communication – FIFO and non-FIFO, is presented in [6]. The concept of causal ordering is of considerable interest to the design of distributed systems and finds applications in several domains such as updates of replicated data [8], global state collection [1], distributed shared memory [3], teleconferencing [19], multimedia systems [2], and fair resource allocation [14].

In a system with FIFO or non-FIFO communication, enforcing causal message ordering requires appending some control information with each message to enforce the causal order. The recipient process of a message uses this information to determine if there are undelivered messages which were sent causally before this message was sent, and delays the delivery of this message until all such messages have been delivered.

*Previous work*

In the first ISIS system implementation of CO [4], a message carries a history of all the messages that causally precede it. Due to redundant information, this scheme is resilient to processor crashes; however, a complex mechanism is required to prevent unbounded growth of the control information. In any case, the volume of control information can be huge. The CO algorithm in [18] is similar to [4] but carries message-ids rather than entire messages in the control information. Furthermore, unnecessary control information is not sent if the sending host had sent it before.

The control information in the Schiper-Eggli-Sandoz causal ordering algorithm [22] consists of $n$ vectors of length up to $n$ each, where $n$ is the number of processes in the system. This information represents messages sent in the causal past that are not known to be delivered. The receiving site uses vector time [7, 15, 23] to determine whether messages represented in the control vectors need to be delivered before the current message is delivered. The causal ordering algorithm of Raynal-Schiper-Toueg [20] attaches a matrix $SENT$ of size $n \times n$ with every message. $SENT[i, j]$ indicates the number of messages that are known to be sent by $i$ to $j$. Each process also uses an array $DELIV$ of size $n$, where $DELIV[i]$ is the number of messages from node $i$ that have been delivered to the process. Clearly, the overhead of control information in messages and in storage for both these algorithms [20, 22] is O($n^2$) integers. This overhead can be reduced under restricted multicasting environments or when some nice properties about the underlying communication medium are assumed. The implementation of these nice properties results in the exchange of additional messages and typically incurs additional delays in the delivery of messages. In the causal multicast in overlapping groups implementation of ISIS [5], every process maintains a vector for every group whether it belongs to that group or not. A vector for a group informs the process of the number of messages multicast by the various members of the group. When a process sends a message, it appends all of its vectors to the message. Clearly, this method can get expensive if there are several groups with large sizes. In particular, the maximum number of groups is $2^n - 1$.

In the causal multicast in overlapping groups algorithm of Mostefaoui and Raynal [17], a process keeps only one scalar for every group and appends only one vector (with the size equal to the number of groups) to every message; however, the algorithm assumes synchronous model of distributed execution. That is, the execution proceeds in synchronized phases and it requires additional resynchronization messages. Moreover, unlike other causal ordering algorithms, this algorithm is not optimal with respect to message delivery time; a message may have to wait for resynchronization messages. Nonetheless, this algorithm is desirable in situations where additional message traffic and delays can be tolerated for much reduced overhead of control information in messages.

Rodrigues and Verissimo [21] exploit the topology of the underlying communication network to reduce the size of control information transferred in messages. Unlike [21], the algorithm proposed here does not require the knowledge of the topology of the underlying network; instead, it uses the dynamic communication pattern and structure of the computation to reduce the overhead of control information in messages.

The algorithm in [9] tracks direct predecessors of a message $M$, rather than all the predecessors of $M$. This results in a savings in the control information overhead in messages and makes it more efficient than previous ones. However, this algorithm is designed for group communication (where groups are fixed a priori) and the reduction in the overhead of control information in messages is partly because all messages are sent to within the group. This implementation [9] tracks direct predecessors inefficiently and consequently tracks indirect predecessors also.

The above algorithms have higher overhead of control information in messages and local storage than is required for an optimal algorithm. Though some of the algorithms attempt to save on the control information overhead in mes-

sages and in storage by assuming certain topologies or communication patterns, they still have control information overhead that is not really needed to enforce causal ordering, and that can have an avalanche effect.

Mattern and Fünfrocken presented a CO algorithm in a system that allows only unicasts, i.e., point-to-point message delivery [16]. Their algorithm requires transport layer acknowledgements between FIFO buffers at the sending and receiving processes. Each process has a dedicated input buffer and a dedicated output buffer. The output buffer of the sender process blocks i.e., cannot send further messages, until it receives an acknowledgement from the input buffer of the destination process of the previously sent message. This is a clever algorithm that does not force the sender application process to wait – thus the application is nonblocking – even though there may be blocking at the transport layer. In this specialized framework, the algorithm does not require any control information in messages, but has the following limitations: (i) it does not work with message multicasts and (ii) although the application is nonblocking, it may involve blocking synchronization at the transport layer. The algorithm essentially uses synchronous communication with input/output buffers to yield causal order. The blocking provides built-in synchronization which simplifies the algorithm design and eliminates the overhead of control information in messages.

The overhead due to the control information in existing causal ordering algorithms that make no simplifying assumptions about the system, O($n^2$) or higher, can be prohibitively large if the number of processes is large and limits the scalability of these algorithms.

In this paper, we discuss the optimality conditions for causal ordering algorithms in terms of the size of control information appended to messages and stored in local logs for enforcing causal order among message multicasts in a generalized framework, *viz.*, the algorithm is nonblocking, is completely decentralized, and does not use any *a priori* knowledge about the topology or communication pattern. We present an optimal causal ordering algorithm that appends to each message and stores in local logs the least amount of control information necessary to enforce CO, as per the optimality conditions. Instead of maintaining and transmitting an $n \times n$ matrix, the algorithm stores and transmits the bare minimum information required to enforce CO, and uses an efficient encoding scheme to represent this information. Although the upper bound on space complexity of the overhead of control information in the algorithm is $O(n^2)$, the overhead is likely to be much smaller on the average, and is always the least possible.

This paper is organized as follows: Sect. 2 presents the system model, notations, and definitions. Section 3 presents constraints on the propagation of information to achieve optimality and presents the necessary and sufficient conditions on the information for causal ordering. Section 4 presents an optimal causal ordering algorithm. Section 5 proves the correctness of the algorithm. Section 6 proves the optimality of the algorithm. In Sect. 7, some results on the complexity of the causal ordering problem are presented. Finally, Sect. 8 concludes.

## 2 Preliminaries

### *2.1 System model*

A distributed system consists of a collection of geographically dispersed sites connected by a communication network. There is a logical communication channel between each pair of sites. The sites do not share any memory and communicate solely by *asynchronous message passing*, i.e., the message propagation delay is finite but unpredictable, and the computation at a site that sends a message does not halt waiting for an acknowledgement that the message is delivered. We assume that for any pair of sites, messages are delivered in order (FIFO delivery) because most known communication networks provide FIFO support. There is no common physical clock. The underlying communication medium is reliable and does not alter, generate, or consume messages.

Without loss of generality, we assume that a single process runs on a single site. Henceforth, a process will also be referred to as a node. A process that fails does so in a fail-stop manner and Byzantine behavior is not allowed. A process can be dynamically created and can dynamically exit. Let $N$ be the set of all the processes that existed in the system and let $n$ be the number of processes running in the system at any time. The execution at a process is modeled by three types of events, namely, *message send*, *message delivery*, and *internal* events. An internal event represents a local computation at the process, whereas message send and delivery events establish cause and effect relationships among the processes. At a send event, the message that is sent is first placed in the output buffer of the process and is subsequently sent to its destination by the underlying networking protocol. When a message arrives at a process along a communication channel, it is placed in an input buffer. The event at which a message arrives and is placed in the input buffer of a process is a receive event that we treat as an internal event because it does not establish any cause and effect relationship among processes.

Let $E_i$ denote the set of events at process $i$. The execution of events at process $i$ is a linear order on $E_i$. The set of events in the *distributed computation* $E$ is the set of events $\bigcup_{\forall i \in N} E_i$. Each process $i$ maintains a scalar counter, $clock_i$, that is incremented on the occurrence of at least each local send event. $clock_i$ need not be a dedicated counter of the messages sent, which was used in [20]. Let $(i, a)$ denote the event at local time $a$ at process $i$. A message $M$ sent by $i$ at local time $a$ is denoted as $M_{i,a}$. The subscripts of $M_{i,a}$ are dropped only if there is no ambiguity. The send and the delivery events of a message $M$ are respectively denoted by $Send(M)$ and $Delivery(M)$. $Delivery_d(M)$ is the event at which $M$ is delivered to process $d$. A boolean function $Delivered_d(M_{i,a})$ is true iff $M_{i,a}$ has been delivered at $d$. The phrase "receive a message" means the arrival and (eventual) delivery of a message.

The cause and effect relationship between the events of a distributed execution is captured by Lamport's "happened before" or causality relation ($\longrightarrow$) [14], which defines a partial order $(E, \longrightarrow)$.

**Definition 1** *The "happens before" relation, also called the "causality" relation and denoted by $\longrightarrow$, is defined by the following three conditions:*

1. $(i,a) \longrightarrow (j,b)$ *if* $i = j \bigwedge a < b$.
2. $(i,a) \longrightarrow (j,b)$ *if* $(i,a)$ *is the send of a message and* $(j,b)$ *is the delivery of that message.*
3. $(i,a) \longrightarrow (j,b)$ *if* $\exists (k,c) \mid ((i,a) \longrightarrow (k,c) \bigwedge (k,c) \longrightarrow (j,b))$

**Definition 2 (Causal ordering (CO)).** *If messages* $M$ *and* $M'$ *have the same destination* $d$ *and* $Send(M) \longrightarrow Send(M')$*, then causal ordering ensures that* $Delivery_d(M) \longrightarrow Delivery_d(M')$*.*

Causal ordering simplifies the development of distributed programs and applications by providing a built-in message synchronization and reducing the nondeterminism in a distributed computation.

**Causal multicast.** *Multicast* is a communication paradigm that enables a process to send a message to a group of processes in a single send event. The set of destinations of a multicast message $M_{i,a}$ is denoted by $M_{i,a}.Dests$. In causal multicast, message delivery events satisfy causal ordering. In our model, a process can be a member of multiple overlapping groups. Successive multicasts by a process can be to different overlapping groups. Also, a group of processes to which a process can multicast messages can vary and such groups need not be formed *a priori*. The ability of a process to multicast to arbitrary and dynamically changing groups is useful in many applications such as updating replicas of different variables stored at different sets of sites.

Further notations and definitions

We define a directed graph, referred to as the *computation graph*, of a distributed computation as follows: there is one-to-one mapping between set of vertices in the graph and the set of events in the computation, $E$, and there is a directed edge between two vertices iff either these vertices correspond to two consecutive events at a process or correspond to message send and delivery events of the same message, respectively. A vertex in the computation graph will be referred to by the event to which it maps. An edge in the computation graph is either (i) a *message dependency edge* if it corresponds to a *causal message dependency* between the send-delivery pair of events of a message, or (ii) a *local dependency edge* if it corresponds to a *causal local dependency* between two consecutive events at a process. A *maximal chain* in a distributed computation is a linearly ordered subset of $E$ such that any two adjacent events in this linear order are either consecutive events at a process or the send-delivery pair of events of a message. Henceforth, each reference to a chain assumes it is a maximal chain. A *path* (also termed a *causal path*) in the computation graph is a sequence of directed edges such that for two adjacent edges in the sequence, the second edge is an out-edge at the event (vertex) at which the first edge is an in-edge.

We define a function $S\_len_d(Send(M_{i,a}), Send(M_{j,b}))$ that returns the maximum number of $Send$ events among the paths between $Send(M_{i,a})$ and $Send(M_{j,b})$, including $Send(M_{j,b})$ but excluding $Send(M_{i,a})$, that have $d$ as a destination. Also define a function $length(path, e)$, where $e$ is an event on the path $path$, that returns the number of send events preceding $e$ on the path.

The relation $\stackrel{=}{\longrightarrow}$ on two events is as follows: $(i,a) \stackrel{=}{\longrightarrow} (j,b)$ iff $(i,a) = (j,b)$ or $(i,a) \longrightarrow (j,b)$. The *causal past* of an event $(i,a)$ is the set of all events $(k,c)$ such that $(k,c) \longrightarrow (i,a)$. The *causal future* of an event $(i,a)$ is the set of all events $(k,c)$ such that $(i,a) \longrightarrow (k,c)$.

Table 1 summarizes the important notations we have defined in this section.

### *2.2 Objectives of the paper*

It is a recognized fact that in an asynchronous system, certain information about messages sent in the past must be stored and propagated to enforce CO. The overhead due to the control information in existing causal ordering algorithms that make no simplifying assumptions about the system, O($n^2$) or higher, can be prohibitively large when the number of processes is large. In this paper, we address the following fundamental question:

**Problem 1** *What is the minimum amount of information regarding messages that were sent in the causal past that is necessary to be propagated and stored by any protocol to enforce causal ordering under the following framework in our system model?*

- *The protocol must be a* nonblocking protocol*, i.e., a protocol in which a process can send messages without waiting for messages that it had sent earlier to be delivered to their respective destinations or even to be copied out of the output buffers.*
- *The protocol does not use any a priori knowledge about the topology or communication.*
- *The protocol should be deterministic, completely decentralized, and the role of each process should be completely symmetric. Thus, the use of a coordinator or even a hierarchical organization of processes is ruled out.*

Under the framework of Problem 1, the system cannot make any assumptions such as "all messages are broadcast", or "all messages are unicast", or "there is a synchronous message exchange at the application layer or the transport layer".

We answer Problem 1 by formulating necessary and sufficient conditions on this information. This is the first characterization of the sufficient and the necessary conditions on this information. We discuss the optimality conditions for causal ordering algorithms in terms of the size of control information that is appended to messages and stored in local logs for enforcing CO in the above framework.

We present an optimal causal ordering algorithm that appends the least amount of control information to each message and stores the least amount of control information in logs, as per the optimality conditions we formulate. The algorithm stores and transmits the bare minimum information required to enforce CO, and uses an encoding scheme to represent this information. The proposed algorithm does not maintain and transmit an $n \times n$ matrix or $n$ vectors of length $n$ to enforce causal ordering, unlike other algorithms. Although the upper bound on space complexity of the overhead of control information in the algorithm is $O(n^2)$, the overhead is likely to be much smaller on the average, and is always the least possible.

**Table 1.** Explanation of the notations used

| Notations | Brief explanation or definition of the notations |
|---|---|
| $clock_i$ | Scalar event counter or clock at process $i$. |
| $(i, a)$ | Event at local time $a$ at process $i$. |
| $M_{i,a}$ | Message sent by process $i$ at $(i, a)$. The subscripts can be dropped only if there is no ambiguity. |
| $Send(M)$ | Event at which message $M$ is sent. |
| $Delivery(M)$ | Event at which message $M$ is delivered. |
| $Delivery_d(M)$ | Event at which message $M$ is delivered to process $d$. |
| $Delivered_d(M_{i,a})$ | Boolean function; true iff $M_{i,a}$ has been delivered to $d$. |
| $M_{i,a}.Dests$ | Set of destinations of multicast message $M_{i,a}$. |
| $S\_len_d(Send(M_{i,a}), Send(M_{j,b}))$ | Integer function; returns the max number of $Send$ events among the paths between $Send(M_{i,a})$ and $Send(M_{j,b})$, including $Send(M_{j,b})$ but excluding $Send(M_{i,a})$, that have $d$ as a destination. |
| $length(path, e)$ | Integer function; given event $e$ on the path $path$, returns the number of send events preceding $e$ on the path. |

## 3 Achieving optimality

We show in Sect. 7 that the overhead of control information in messages and in local storage to enforce CO in the framework of Problem 1 is $\Omega(n^2)$. Enforcing CO by having $O(n^2)$ overhead of control information in messages and in local storage is well-understood – simply use an $n \times n$ matrix to do explicit (source, destination, timestamp) tracking of messages already delivered or sent, maintain this information in local storage and transmit it in messages.

We achieve optimality by identifying constraints on the propagation of control information, called the *propagation constraints*, to curtail the propagation of redundant information at the earliest instant. The Propagation Constraints specify the earliest events along causal paths *only* up to which control information needs to travel to ensure causal ordering. A node stores and transfers only the bare minimum information in accordance with the Propagation Constraints, and uses an encoding to represent such information.

### *3.1 Delivery condition*

In order to ensure the safety of a CO algorithm, no message should be delivered to a node unless all messages sent causally before it have been delivered to that node. Causal ordering algorithms achieve this by having each message $M'$ carry a list of messages $M$ sent causally before $M'$ was sent. $M'$ is delivered to the receiving node only after the messages $M$ that were also destined for this receiving node have been delivered. We state this condition as the *Delivery Condition*.

**Delivery condition.** A message $M'$ that carries information "$d \in M.Dests$" about a message $M$ sent to $d$ in the causal past, is not delivered to $d$ until $M$ has been delivered to $d$.

### *3.2 Fixed points of information propagation*

We next identify events in causal paths, called *Fixed Points*, up to which information about messages sent in their causal past needs to be propagated to enforce causal ordering. For optimality, information about messages sent in the causal past needs to travel only up to these events. We present two observations that lead to the definition of fixed points. These fixed points form the basis for Propagation Constraints that specify how information about messages sent in the causal past must be stored and propagated to enforce CO optimally.

The information "$d \in M_{i,a}.Dests$" should not be stored or propagated in the causal future of $Delivery_d(M_{i,a})$ for optimality, as $M_{i,a}$ has already been delivered to $d$ in CO with respect to any message sent to $d$ in the causal future of $Delivery_d(M_{i,a})$. This observation leads to the first fixed point, FP1.

We also make the following observation. Let $d \in M_{i,a}.Dests$. Consider any event $(j, b)$ such that $(i, a) \longrightarrow (j, b)$, $d \in M_{j,b}.Dests$, and there does not exist $(k, c)$, where $(i, a) \longrightarrow (k, c) \longrightarrow (j, b)$, such that $d \in M_{k,c}.Dests$. If it is not known at $(j, b)$ that $M_{i,a}$ has been delivered to $d$, information "$d \in M_{i,a}.Dests$" must be propagated up to $(j, b)$ to enforce CO delivery of $M_{i,a}$ to $d$ with respect to the delivery of $M_{j,b}$ to $d$. Also, information "$d \in M_{i,a}.Dests$" should not be stored or propagated in the causal future of $Send(M_{j,b})$, except on the message $M_{j,b}$ sent to $d$, for optimality. This is because $M_{i,a}$ gets delivered to $d$ in CO with respect to $M_{j,b}$ due to the Delivery Condition when $M_{j,b}$ is delivered; for any message $M'$ sent to $d$ in the causal future of $(j, b)$, if we ensure that $M_{j,b}$ is delivered to $d$ in CO with respect to $M'$, then it is ensured transitively that $M_{i,a}$ is delivered to $d$ in CO with respect to $M'$. Thus, we say that in the causal future of $(j, b)$, "$M_{i,a}$ is *guaranteed to be delivered* to $d$ in CO w.r.t. $M'$ sent to $d$" and $M'$ need not carry any information about "$d \in M_{i,a}.Dests$". (Note that CO between $M_{j,b}$ and $M'$ is enforced similarly.) This observation leads to the second fixed point, FP2.

**Definition 3** ***("Fixed Points" of information propagation)****: A fixed point of propagation of information "$d \in M_{i,a}.Dests$" is an event $(j, b)$ such that* **either**

**FP1:** $(i, a) \longrightarrow (j, b) \bigwedge M_{i,a}$ *is delivered to* $d$ *at* $(j, b)$ *(= event* $Delivery_d(M_{i,a})$*),* **or**

**FP2:** $(i, a) \longrightarrow (j, b) \bigwedge d \in M_{j,b}.Dests \bigwedge \nexists (k, c) \mid ((i, a) \longrightarrow (k, c) \longrightarrow (j, b) \bigwedge (d \in M_{k,c}.Dests \bigvee (k, c)$ *is the event* $Delivery_d(M_{i,a}))$*).*

An event $(j, b)$ which is the fixed point FP1 of information "$d \in M_{i,a}.Dests$" denotes that message $M_{i,a}$ is known to have been delivered at event $(j, b)$ to destination $d$ (=$j$). $M_{i,a}$ is already delivered in causal order to $d$ with respect to any multicasts to $d$ in the future of $(j, b)$. Thus, any future multicasts to destination $d$ do not have to carry information "$d \in M_{i,a}.Dests$".

An event $(j, b)$ which is a fixed point FP2 of information "$d \in M_{i,a}.Dests$" denotes that message $M_{i,a}$ is guaranteed to be delivered in causal order with respect to message $M_{j,b}$ to the common destination $d$. The causal delivery of $M_{j,b}$

to $d$ with respect to any future multicasts to $d$ automatically ensures the causal delivery of $M_{i,a}$ to $d$ with respect to any multicasts in the future of $(j, b)$ to $d$. Thus, any future multicasts to destination $d$ do not have to carry information "$d \in M_{i,a}.Dests$".

A significance of a fixed point of propagation of information "$d \in M_{i,a}.Dests$" is that the information traverses only up to the fixed point. Nowhere in the causal future of the fixed point does this information need to exist! Multiple fixed points of this information, if any, are concurrent. All events in the past of a fixed point as well as all events that are concurrent with all fixed points must have information "$d \in M_{i,a}.Dests$". (From Definition 3, note that if $(j, b)$ is a fixed point FP2, then $Delivery_d(M_{i,a})$ (which is an FP1 event) $\not\longrightarrow (j, b)$. If $(j, b)$ is a fixed point FP1, then $\nexists\ M_{k,c} \mid (i, a) \longrightarrow (k, c) \longrightarrow (j, b)$ (which is the event $Delivery_d(M_{i,a})$) $\bigwedge\ d \in M_{k,c}.Dests$, because of the Delivery Condition.)

### *3.3 Necessary and sufficient conditions*

Based on the identification of fixed points, we formulate necessary and sufficient conditions on the propagation of information about messages sent in the causal past. In a correct CO algorithm, the information must propagate at least "up to" its fixed points – this identifies the sufficiency conditions on information propagation. In an optimal algorithm, the information must propagate "only up to" fixed points – this identifies the necessity conditions on information propagation. It follows that in a nonoptimal algorithm, the information may propagate beyond the fixed points.

**Sufficient conditions.** Information "$d \in M_{i,a}.Dests$" must propagate from event $(j, b)$ on its outward edges in the computation graph if **both**:

1. $Delivery_d(M_{i,a}) \stackrel{=}{\not\longrightarrow} (j, b)$, **and**
2. $\nexists (k, c) \mid (i, a) \longrightarrow (k, c) \longrightarrow (j, b) \bigwedge d \in M_{k,c}.Dests$.
   If such a $(k, c)$ does not exist and $(j, b)$ is a multicast send event such that $d \in M_{j,b}.Dests$, then the information "$d \in M_{i,a}.Dests$" is sent *only* on the message to $d$.

If the first condition above were false, then there is no need to carry the information "$d \in M_{i,a}.Dests$" because $M_{i,a}$ is guaranteed to be delivered in causal order with respect to all messages sent in the causal future of the current event $(j, b)$. If the second condition above were false, then the causal delivery of the present message with respect to $M_{k,c}$ would ensure the causal delivery of the present message with respect to $M_{i,a}$ if $M_{k,c}$ is causally delivered with respect to $M_{i,a}$. Information "$d \in M_{i,a}.Dests$" need not be carried or stored beyond the current event.

However, when both the conditions above are true, it is not known if $M_{i,a}$ has been delivered or if it is guaranteed to be delivered without violating CO, and therefore, the information must be carried futher to enforce CO.

**Necessary conditions.** Information about "$d \in M_{i,a}.Dests$" must not be propagated from $(j, b)$ on any outgoing edge in the computation graph if **either**:

1. $Delivery_d(M_{i,a}) \stackrel{=}{\longrightarrow} (j, b)$, **or**
2. $\exists (k, c) \mid (i, a) \longrightarrow (k, c) \longrightarrow (j, b) \bigwedge d \in M_{k,c}.Dests$.
   If such a $(k, c)$ does not exist and $(j, b)$ is a multicast send event such that $d \in M_{j,b}.Dests$, then the information "$d \in M_{i,a}.Dests$" is sent *only* on the message to $d$.

The absence of information "$d \in M_{i,a}.Dests$" when either of the two conditions holds does not lead to a violation of CO because $M_{i,a}$ is already delivered to $d$ or is guaranteed to be delivered to $d$ without violating CO. Necessary conditions state that information must not be currently carried in messages or stored at nodes if a fixed point of that information belongs to the causal past. An optimal CO algorithm must satisfy these conditions so that no redundant information is propagated.

### *3.4 Propagation constraints*

We now combine the necessary and sufficient conditions, on information about messages sent in the past that must be stored and propagated to enforce CO optimally, in the form of *propagation constraints*.

**Propagation constraints.** The information "$d \in M_{i,a}.Dests$" about a message $M_{i,a}$ sent to $d$ must propagate along all causal paths starting at event $(i, a)$ up to and only up to the earliest events $(j, b)$ on any such path such that **either:**

**PC1:** $Delivery_d(M_{i,a}) \stackrel{=}{\longrightarrow} (j, b)$, i.e., it is known that $M_{i,a}$ has been delivered, **or**

**PC2:** $\exists\ (k, c) \mid ((i, a) \longrightarrow (k, c) \longrightarrow (j, b) \bigwedge d \in M_{k,c}.Dests)$, i.e., it is guaranteed that the message will be delivered in CO.
If such a $(k, c)$ does not exist and $(j, b)$ is a multicast send event such that $d \in M_{j,b}.Dests$, then the information "$d \in M_{i,a}.Dests$" is sent *only* on the message to $d$.

The "up to" part of the Propagation Constraints specifies the sufficient condition on the information propagation and must be satisfied for the correctness of a CO algorithm. The "only up to" part of the Propagation Constraints specifies the necessary condition on the information propagation and must be satisfied for the optimality of a CO algorithm. A proof of these properties is given in Sect. 3.6.

Figure 2 illustrates the Propagation Constraints for a $Send(M_{i,a})$ event (shown by a shaded circle). The rectangle shows the delivery event of message $M_{i,a}$ at process $d$. Clearly, all send events in the future of this event (e.g., $e3$, $e4$, and $e6$) need not have access to information "$d \in M_{i,a}.Dests$". Each unshaded circle denotes a message send event at which a message is sent to $d$ and there is no such event on any causal path between $(i, a)$ and this event. From PC2, no message sent in the future of such a send event (e.g., $e4$, $e5$, $e6$, $e7$, and $e8$) needs to have access to information "$d \in M_{i,a}.Dests$" to enforce CO. However, all send events not in the future of events marked by either the rectangle or the unshaded circles (e.g., $e1$ and $e2$) must have access to information "$d \in M_{i,a}.Dests$" to enforce CO, as per PC1 and PC2.

Thus, the Propagation Constraints specify two different ways to identify events $(j, b)$ which are the events "up to and only up to" which information "$d \in M_{i,a}.Dests$" must

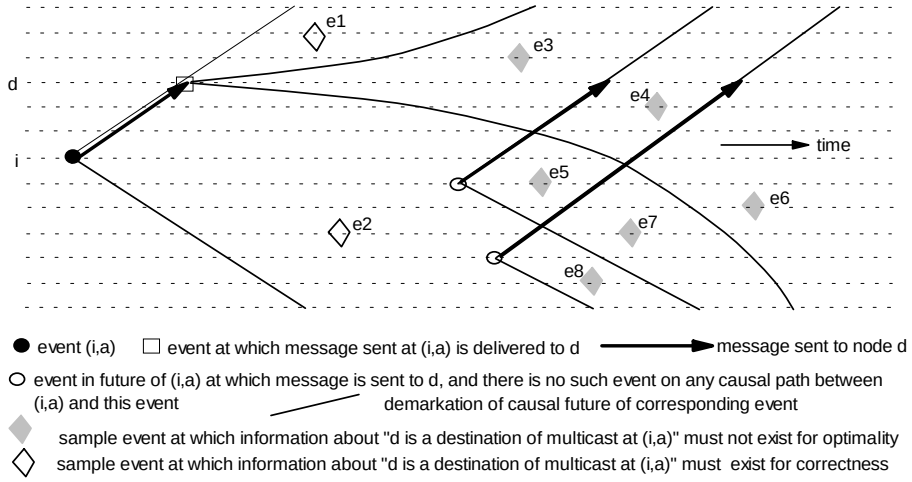


**Fig. 2.** Propagation Constraints

propagate, for any $d$ and $M_{i,a}$. Such events $(j, b)$ are identified for every path originating at $(i, a)$ and are termed *Constraint Propagation Endpoints (CPEs)* of information "$d \in M_{i,a}.Dests$".

**Definition 4 (Constraint Propagation Endpoints (CPEs)):** *A CPE of "$d \in M_{i,a}.Dests$" is either a fixed point of "$d \in M_{i,a}.Dests$" or the earliest event on each path originating at $(i, a)$ such that it is in the causal future of a fixed point of "$d \in M_{i,a}.Dests$".*

The CPEs are the earliest events on all causal paths from $(i, a)$ at which it is known that $M_{i,a}$ is delivered to $d$ or is guaranteed to be delivered to $d$ in CO. The information "$d \in M_{i,a}.Dests$" must propagate into the future of $(i, a)$ up to and only up to the earliest events in the future of some fixed point of "$d \in M_{i,a}.Dests$", on a per path basis. Note that all CPEs of "$d \in M_{i,a}.Dests$" need not be causally concurrent. Two or more CPEs may be causally related and may occur at the same process even if they lie in the causal future of the same fixed point because they represent information that has traversed along different paths from $(i, a)$ into the future of the fixed point. We now define "redundant information" for an optimal CO algorithm.

**Definition 5** *If information is propagated as per the Propagation Constraints, information "$d \in M_{i,a}.Dests$" that traverses beyond a CPE for "$d \in M_{i,a}.Dests$" is redundant information.*

Also note that although PC1 and PC2 specify that the CPEs are the events up to and only up to which information "$d \in M_{i,a}.Dests$" propagates, PC2 also requires that if event $(j, b)$ is a CPE by PC2, then $(j, b)$ must also send information "$d \in M_{i,a}.Dests$" *only* on $M_{j,b}$ sent to $d$. This requirement is implicit henceforth whenever it is stated that this information propagates up to and only up to the CPEs.

### *3.5 Other constraints on information propagation*

Besides the Propagation Constraints presented in the previous section, there are some general, commonsense constraints on information propagation. These constraints are common to many distributed algorithms and we mention them in this section for completeness.

**No transmission of "spurious information".** In addition to the Propagation Constraints, we require that an algorithm that enforces CO optimally should not store or propagate any other information. Specifically, it should not store and propagate information about messages that have already been delivered or that are guaranteed to be delivered in CO.

**No transmission of "duplicate information".** The Propagation Constraints require that information about a message sent in the causal past should be stored and propagated as long as and only as long as the knowledge that "the message is delivered" or "the message is guaranteed to be delivered in CO" is not locally available. As communication channels are FIFO, this information need not be resent over the same channel, nor locally stored if it has been sent once on all the outgoing channels. This motivates the following principle of "nonduplicate information transmission and storage of information" to achieve optimality: "A node must not transmit the same information more than once to the same node and not store information that has been sent once on each outgoing channel". A similar notion has been previously used in [24] to efficiently implement vector clocks [7, 15].

Enforcement of "no transmission of duplicate information" is a generic constraint common to many distributed algorithms that assume FIFO communication. Therefore, it is of secondary concern to our problem. It also requires some additional data structures at each process to keep track of what information has been sent to what processes. Moreover, it may require additional computation at each process to update the data structures and to process the control information received in messages. Thus, there is a trade-off and we will not be elaborating on this constraint in the paper.

### *3.6 Correctness of propagation constraints*

Theorem 1 proves that if information propagates as defined by the Propagation Constraints, and the Delivery Condition is satisfied, then CO is never violated.

**Theorem 1** *The Propagation Constraints and the Delivery Condition together ensure CO.*

*Proof.* Let $d \in M_{i,a}.Dests$. Let there exist event $Send(M_{j,b})$ such that $d \in M_{j,b}.Dests$ and $Send(M_{i,a}) \longrightarrow Send(M_{j,b})$. There are two possibilities based on the relation between a fixed point of "$d \in M_{i,a}.Dests$" and $Send(M_{j,b})$.

1. If event $Send(M_{j,b})$ is not in the causal future of a fixed point of information "$d \in M_{i,a}.Dests$" and is itself not such a fixed point, then "$d \in M_{i,a}.Dests$" is sent with $M_{j,b}$ by the Propagation Constraints PC1 and PC2. Along with the Delivery Condition, this guarantees that $M_{i,a}$ is delivered to $d$ before $M_{j,b}$ is.
2. If event $Send(M_{j,b})$ is in the causal future of a fixed point of "$d \in M_{i,a}.Dests$" or is itself such a fixed point, then we use induction on variable $S\_len_d(Send(M_{i,a}), Send(M_{j,b}))$ to show that $M_{i,a}$ is delivered to $d$ before $M_{j,b}$ is.
   $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) = 1$**:** In this base case, if event $Send(M_{j,b})$ is in the causal future of a fixed point FP1, then $M_{i,a}$ is already delivered to $d$ before $M_{j,b}$ is. Otherwise event $Send(M_{j,b})$ is itself a fixed point FP2 of information "$d \in M_{i,a}.Dests$". Along with the Delivery Condition, PC2 guarantees that $M_{i,a}$ is delivered to $d$ in causal order with respect to $M_{j,b}$.
   $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) = x, x > 1$**:** Assume that the hypothesis "If $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) = x$, then $M_{i,a}$ is delivered to $d$ before $M_{j,b}$ is" holds.
   $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) = x + 1, x > 1$**:** There must exist $Send(M_{k,c}) \mid Send(M_{i,a}) \longrightarrow Send(M_{k,c}) \longrightarrow Send(M_{j,b}) \bigwedge d \in M_{k,c}.Dests \bigwedge S\_len_d(Send(M_{i,a}), Send(M_{k,c})) = x$. By the induction assumption, $M_{i,a}$ is delivered to $d$ or is guaranteed to be delivered to $d$ before $M_{k,c}$ is. Causality is transitive, hence the burden of the proof is to show that $M_{k,c}$ is delivered to $d$ or is guaranteed to be delivered to $d$ before $M_{j,b}$ is.
   Note that $S\_len_d(Send(M_{k,c}), Send(M_{j,b})) = 1$. We have already shown that when $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) = 1$, $M_{i,a}$ is delivered to $d$ before $M_{j,b}$. Therefore, $M_{k,c}$ is delivered to $d$ before $M_{j,b}$ is. Hence, the theorem. □

We now show that if information does not propagate as per Propagation Constraints or if the Delivery Condition does not hold, then causal ordering is violated.

**Theorem 2** *The Propagation Constraints and the Delivery Condition are necessary to ensure CO.*

*Proof.* Consider an event $(j, b)$ such that:

– (Assumption 1:) $\nexists (k, c), (i, a) \longrightarrow (k, c) \longrightarrow (j, b)$ where $(k, c)$ is an FP1 or FP2 event of "$d \in M_{i,a}.Dests$". Moreover, event $(j, b)$ is not the FP1 event of "$d \in M_{i,a}.Dests$".

The assumption implies that (i) neither at $(j, b)$ nor at any event in the causal past of $(j, b)$ is $M_{i,a}$ delivered to $d$, and (ii) no message $M_{k,c}$ sent in the causal future of $Send(M_{i,a})$ and in the causal past of $(j, b)$ has $d$ as a destination. As per Propagation Constraints, information "$d \in M_{i,a}.Dests$" must reach $(j, b)$.

We also make the following assumption about $(j, b)$ in order to show that if one of the Propagation Constraints is violated, then CO is violated.

– (Assumption 2:) Let $(j, b)$ be a send event and $d \in M_{j,b}.Dests$.

Based on the assumptions on $(j, b)$, we make the following assertions.

– (Assertion 1:) If the "up to" part of the propagation of any one of the Propagation Constraints PC1 and PC2 is violated, then information about "$d \in M_{i,a}.Dests$" does not reach $(j, b)$, from Assumption (1).
– It follows from assumptions (1) and (2) on event $(j, b)$ that $(j, b)$ is event FP2, and from the Propagation Constraints, "$d \in M_{i,a}.Dests$" must be sent with $M_{j,b}$ to $d$.
  (Assertion 2:) If the part of PC2 that specifies that the information about "$d \in M_{i,a}.Dests$" must be sent on $M_{j,b}$ to $d$ is violated, the information "$d \in M_{i,a}.Dests$" is not carried by $M_{j,b}$ to $d$, from Assumption (2).

Assume that $M_{i,a}$ has not arrived at $d$ before $M_{j,b}$ arrives at $d$. Proof of the theorem has two parts.

1. We show that if any part of the Propagation Constraints does not hold, then despite the Delivery Condition, CO is violated.
   If any part of the Propagation Constraints does not hold, from Assertions (1) and (2), we have that information "$d \in M_{i,a}.Dests$" does not propagate up to $(j, b)$, or $(j, b)$ does not propagate this information to $d$. In either case, $M_{j,b}$ sent to $d$ is not accompanied by this information "$d \in M_{i,a}.Dests$". Therefore, $M_{j,b}$ is delivered to $d$ before $M_{i,a}$ is.
2. We show that if the Delivery Condition does not hold, then even if the Propagation Constraints hold, CO is violated.
   $M_{j,b}$ sent to $d$ is accompanied by control information "$d \in M_{i,a}.Dests$". Despite this information, delivery of $M_{j,b}$ to $d$ on arrival at $d$ is not prevented if the Delivery Condition is violated. Therefore, $M_{j,b}$ is delivered to $d$ before $M_{i,a}$ is.

In both cases, CO is violated. Therefore, the Propagation Constraints and the Delivery Condition are necessary for enforcing CO. □

## 4 An optimal CO algorithm

In this section, we describe an information encoding scheme to realize the Propagation Constraints and present an optimal CO algorithm, followed by an explanation of the various steps of the algorithm. The algorithm is optimal in the sense that it maintains and transfers bare minimum information that is necessary to enforce causal ordering, as per the Propagation Constraints. Although the upper bound on the overhead of control information in messages and in local storage in this algorithm is $O(n^2)$, the overheads are likely to be much smaller on the average, and are always the least possible.

### 4.1 Notations

Information about a message $M_{i,a}$ that is carried in a later message $M'$ is denoted as $o_{i,a}$ = $(i, a, Dests)$, where $o_{i,a}.Dests \subseteq M_{i,a}.Dests$; $o_{i,a}.Dests$ denotes the set of destinations of $M_{i,a}$ for which (i) it is not known that $M_{i,a}$ has been delivered, and (ii) it is not guaranteed that $M_{i,a}$ will be delivered in causal order, as far as the sender of $M'$ can discern. Multiple $o_{i,a}$ that are collectively carried in a message $M$ are denoted by a set $O_M$. Subscripts of $o_{i,a}$ are dropped only when they are not necessary or are clear from the context.

Similarly, information about certain messages sent causally before any local event at a node $j$ is stored in a local log, denoted by $LOG_j$, so that it can be sent on future messages to enforce CO. Information about $M_{i,a}$ that is stored at node $j$ in $LOG_j$ is denoted by $l_{i,a}$ = $(i, a, Dests)$, where $l_{i,a}.Dests \subseteq M_{i,a}.Dests$; $l_{i,a}.Dests$ denotes the set of destinations of $M_{i,a}$ for which (i) it is not known that $M_{i,a}$ has been delivered, and (ii) it is not guaranteed that $M_{i,a}$ will be delivered in causal order, as far as node $j$ can presently discern. Subscripts of $l_{i,a}$ are dropped only when they are not necessary or are clear from the context.

### 4.2 Information encoding

Enforcing CO requires information about messages sent in the past and the Propagation Constraints dictate how much of this information needs to be carried in messages up to an event in the computation. We now discuss how we encode this information efficiently using explicit and implicit tracking of messages sent in the past.

Information about messages (i) not known to be delivered and (ii) not guaranteed to be delivered in CO, is *explicitly* tracked by the algorithm using (source, destination, timestamp) information. The information must be deleted as soon as either (i) or (ii) becomes false. The key problem in designing an optimal CO algorithm is to identify the events at which (i) or (ii) becomes false. Information about messages already delivered and messages guaranteed to be delivered in CO is *implicitly* tracked without storing or propagating it, and is derived from the explicit information. Such implicit information is used for determining when (i) or (ii) becomes false for the explicit information being stored or carried in messages.

*1. Explicit tracking.* Tracking of (source, timestamp, destination) information for messages (i) not known to be delivered and (ii) not guaranteed to be delivered in CO, is done explicitly using the $l.Dests$ field of entries in local logs at nodes and $o.Dests$ field of entries in messages. Sets $l_{i,a}.Dests$ and $o_{i,a}.Dests$ contain explicit information of destinations to which $M_{i,a}$ is not guaranteed to be delivered in CO and is not known to be delivered. The information about "$d \in M_{i,a}.Dests$" is propagated up to and only up to its CPEs, viz., the earliest events on all causal paths from $(i, a)$ at which it is known that $M_{i,a}$ is delivered to $d$ or is guaranteed to be delivered to $d$ in CO. Thus, redundant information (Definition 5) is never stored in logs or propagated in messages.

*2. Implicit tracking.* Tracking of messages that are either (i) already delivered, or (ii) guaranteed to be delivered in CO, is performed implicitly.

The information about messages (i) already delivered or (ii) guaranteed to be delivered in CO, is deleted at the CPEs of this information and not propagated because it is redundant as far as enforcing CO is concerned. However, the semantics of information is useful in determining what information that is being carried in other messages and that is being stored in logs at other nodes has become redundant and thus can be purged. This semantics is implicitly stored and propagated.

We track messages that are (i) already delivered or (ii) guaranteed to be delivered in CO, without explicitly storing such information! Rather, we derive it from the existing explicit information about messages (i) not known to be delivered and (ii) not guaranteed to be delivered in CO, by examining only $o_{i,a}.Dests$ or $l_{i,a}.Dests$, which is a part of the explicit information.

We use two types of implicit trackings: First, the absence of a node id from destination information – i.e., $\exists d \in M_{i,a}.Dests \mid d \notin l_{i,a}.Dests \bigvee d \notin o_{i,a}.Dests$ – implicitly contains information that the message has been already delivered or is guaranteed to be delivered in CO to $d$. Clearly, $l_{i,a}.Dests = \emptyset$ or $o_{i,a}.Dests = \emptyset$ implies that message $M_{i,a}$ has been delivered or is guaranteed to be delivered in CO to *all* destinations in $M_{i,a}.Dests$.

An entry whose $.Dests = \emptyset$ is maintained because the implicit information in it, viz., that of known delivery or guaranteed CO delivery to all destinations of the multicast, is useful to purge redundant information at CPEs as per the Propagation Constraints. Note that if $l_{i,a}.Dests$ is $\emptyset$ at $(k, c)$, then $\forall d \in M_{i,a}.Dests$, a fixed point of "$d \in M_{i,a}.Dests$" must lie in the past of $(k, c)$. However, this does not preclude a CPE of the information being in the causal future of $(k, c)$. To identify such CPEs, $l_{i,a}.Dests$ must be maintained and propagated.

Note that as the distributed computation evolves, several entries $l_{i,a_1}$, $l_{i,a_2}$, ... such that $\forall p$, $l_{i,a_p}.Dests = \emptyset$ may exist in a node's log and a message may be carrying several entries $o_{i,a_1}$, $o_{i,a_2}$, ... such that $\forall p$, $o_{i,a_p}.Dests = \emptyset$. The second implicit tracking uses a mechanism to prevent the proliferation of such entries. The mechanism is based on the following observation:

"For any two multicasts $M_{i,a_1}$, $M_{i,a_2}$ such that $a_1 < a_2$, if $l_{i,a_2} \in LOG_j$, then $l_{i,a_1} \in LOG_j$. (Likewise for any message.)"

Therefore, if $l_{i,a_1}.Dests$ becomes $\emptyset$ at a node $j$, then it can be deleted from $LOG_j$ provided $\exists\ l_{i,a_2} \in LOG_j$ such that $a_1 < a_2$. The presence of such $l_{i,a_1}$s in $LOG_j$ is automatically implied by the presence of entry $l_{i,a_2}$ in $LOG_j$. Thus, for a multicast $M_{i,z}$, if $l_{i,z}$ does not exist in $LOG_j$, then $l_{i,z}.Dests = \emptyset$ implicitly exists in $LOG_j$ iff $\exists\ l_{i,a} \in LOG_j \mid a > z$.

As a result of the second implicit tracking mechanism, a node does not keep (and a message does not carry) entries of type $l_{i,a}.Dests = \emptyset$ in its log. However, note that a node must always keep at least one entry of type $l_{i,a}$ (the one with the highest timestamp) in its log for each sender node $i$. The same holds for messages.

The information tracked implicitly is "live" and propagates wherever the explicit information in $o_{i,a}$ in some $O_{M'}$ propagates; it is useful in purging information explicitly carried in other $O_{M''}$s and stored in $LOG$ entries about "yet to be delivered to" destinations for the same message $M_{i,a}$ as well as for messages $M_{i,a'}$, where $a' < a$. Thus, whenever $o_{i,a}$ in some $O_{M'}$ propagates to node $j$, (i) the implicit information in $o_{i,a}.Dests$ is used to eliminate redundant information in $l_{i,a}.Dests \in LOG_j$; (ii) the implicit information in $l_{i,a}.Dests \in LOG_j$ is used to eliminate redundant information in $o_{i,a}.Dests$; (iii) the implicit information in $o_{i,a}$ is used to eliminate redundant information $l_{i,a'} \in LOG_j$ if $\nexists$ $o_{i,a'} \in O_{M'}$ and $a' < a$; (iv) the implicit information in $l_{i,a}$ is used to eliminate redundant information $o_{i,a'} \in O_{M'}$ if $\nexists$ $l_{i,a'} \in LOG_j$ and $a' < a$; (v) only nonredundant information remains in $O_{M'}$ and $LOG_j$; this is merged together into an updated $LOG_j$.

### *4.3 The algorithm*

The algorithm uses the symbol $\longleftarrow$ to denote an assignment. Procedure SND is executed atomically. Procedure RCV is executed atomically except for a possible interruption in step RCV(1) where a nonblocking wait is required to meet the Delivery Condition.

**Data Structures**:

$clock_j \longleftarrow 0$; /* local counter clock at node $j$ */
$SR_j[1...n] \longleftarrow \overline{0}$; /* $SR_j[i]$ is the timestamp of last msg. from $i$ delivered to $j$; */
$LOG_j = \{(i, clock_i, Dests)\} \longleftarrow \{\forall i, (i, 0, \emptyset)\}$;
/* Each entry denotes a message sent in the causal */
/* past, by $i$ at $clock_i$. $Dests$ is the set of */
/* remaining destinations for which it is not */
/* known that $M_{i,clock_i}$ (i) has been delivered, */
/* or (ii) is guaranteed to be delivered in CO. */

**SND: $j$ sends a message M to Dests**:

1. $clock_j \longleftarrow clock_j + 1$;
2. **for all** $d \in M.Dests$ **do**:
   $O_M \longleftarrow LOG_j$; /* $O_M$ denotes $O_{M_{j,clock_j}}$ */
   **for all** $o \in O_M$, modify $o.Dests$ as follows:
   **if** $d \notin o.Dests$ **then**
   $o.Dests \longleftarrow (o.Dests \setminus Dests)$;
   **if** $d \in o.Dests$ **then**
   $o.Dests \longleftarrow (o.Dests \setminus Dests) \bigcup \{d\}$;
   /* Do not propagate information about */
   /* indirect dependencies that are */
   /* guaranteed to be transitively satisfied */
   /* when dependencies of $M$ are satisfied. */
   **for all** $o_{s,t} \in O_M$ **do**
   **if** $o_{s,t}.Dests = \emptyset \bigwedge (\exists o'_{s,t'} \in O_M \mid t < t')$
   **then** $O_M \longleftarrow O_M \setminus \{o_{s,t}\}$;
   /* do not propagate older entries */
   /* for which $Dests$ field is $\emptyset$ */
   **send** $(j, clock_j, M, Dests, O_M)$ to $d$;
3. **for all** $l \in LOG_j$ **do** $l.Dests \longleftarrow l.Dests \setminus Dests$;
   /* Do not store information about indirect */
   /* dependencies that are guaranteed */
   /* to be transitively satisfied when */
   /* dependencies of $M$ are satisfied. */
4. $LOG_j \longleftarrow LOG_j \bigcup \{(j, clock_j, Dests)\}$;
5. /* Purge older entries $l$ for which $l.Dests = \emptyset$ */
   $PURGE\_NULL\_ENTRIES(LOG_j)$.

**RCV: $j$ receives a message $(k, t_k, M, Dests, O_M)$ from k:**

1. **for all** $o_{m,t_m} \in O_M$ **do**
   **if** $j \in o_{m,t_m}.Dests$ **wait until** $t_m \leq SR_j[m]$;
   /* Delivery Condition; ensure that messages */
   /* sent causally before M are delivered. */
2. Deliver M; $SR_j[k] \longleftarrow t_k$;
3. $O_M \longleftarrow \{(k, t_k, Dests)\} \bigcup O_M$;
   **for all** $o_{m,t_m} \in O_M$ **do**
   $o_{m,t_m}.Dests \longleftarrow o_{m,t_m}.Dests \setminus \{j\}$;
   /* delete the now redundant dependency */
   /* of message represented by $o_{m,t_m}$ sent to $j$ */
4. /* Merge $O_M$ and $LOG_j$ by eliminating */
   /* all redundant entries. */
   /* Implicitly track "already delivered" & "guaranteed */
   /* to be delivered in CO" messages.*/
   **for all** $o_{m,t} \in O_M$ **and** $l_{s,t'} \in LOG_j$ **such that** $s = m$ **do**
   **if** $t < t' \bigwedge l_{s,t} \notin LOG_j$ **then** mark $o_{m,t}$ for deletion;
   /* $l_{s,t}$ had been deleted or never inserted, */
   /* as $l_{s,t}.Dests = \emptyset$ in the causal past */
   **if** $t' < t \bigwedge o_{m,t'} \notin O_M$ **then** mark $l_{s,t'}$ for deletion;
   /* $o_{m,t'} \notin O_M$ because $l_{s,t'}$ had become $\emptyset$ */
   /* at another process in the causal past */
   Delete all elements marked for deletion in $O_M$ and $LOG_j$ ;
   /* delete entires that represent */
   /* redundant information */
   **for all** $l_{s,t'} \in LOG_j$ **and** $o_{m,t} \in O_M$, **such that** $s = m \bigwedge t' = t$ **do**
   $l_{s,t'}.Dests \longleftarrow l_{s,t'}.Dests \bigcap o_{m,t}.Dests$;
   /* delete destinations for which */
   /* Delivery Condition is satisfied or */
   /* guaranteed to be satisfied as per $o_{m,t}$ */
   Delete $o_{m,t}$ from $O_M$;
   /* information has been incorporated in $l_{s,t'}$ */
   $LOG_j \longleftarrow LOG_j \bigcup O_M$;
   /* merge nonredundant information */
   /* of $O_M$ into $LOG_j$ */
5. /* Purge older entries $l$ for which $l.Dests = \emptyset$ */
   $PURGE\_NULL\_ENTRIES(LOG_j)$.

**PURGE_NULL_ENTRIES(L):**

/* Purge older entries $l$ in $L$, for which $l.Dests = \emptyset$ */
/* and which can be implicitly inferred */
$L$: $LOG$ of a process; /* local log of any process */
**for all** $l_{s,t} \in L$ **do**
**if** $l_{s,t}.Dests = \emptyset \bigwedge (\exists l'_{s,t'} \in L \mid t < t')$ **then**
$L \longleftarrow L \setminus \{l_{s,t}\}$.

### *4.4 An explanation of the algorithm*

This section explains the steps of the algorithm and illustrates the major steps using examples.

**SEND $M_{j,b}$:**

The SND procedure enforces propagation constraint PC2.

**Example Scenario:** Let $l_{i,a}.Dests = \{2, 3, 4, 6, 8\}$, $l_{i,a} \in LOG_j$. Consider event $Send(M_{j,b})$, where $M_{j,b}.Dests = \{3, 4, 7, 8, 11\}$.

1. **SND(1):** The local clock is incremented at a multicast send event.
2. **SND(2):** The only new information available at event $Send(M_{j,b})$ with respect to $M_{i,a}$ is that $\forall\, x \in (l_{i,a}.Dests \bigcap M_{j,b}.Dests)$, $(j, b)$ is Fixed Point FP2 of information "$x \in M_{i,a}.Dests$". As per PC2, information that such $x$ are destinations of $M_{i,a}$ must be suppressed on all outgoing edges of the computation graph from $(j, b)$, with the exception that when sending $M_{j,b}$ to any particular $x \in (l_{i,a}.Dests \bigcap M_{j,b}.Dests)$, the information "$x \in M_{i,a}.Dests$" must be sent. All other information in $l_{i,a}.Dests$, namely, $l_{i,a}.Dests \setminus M_{j,b}.Dests$, must be propagated.
   Note that before propagating $O_M$, if $\exists o_{i,a}, o_{i,a'} \in O_M \mid o_{i,a}.Dests = \emptyset$, and $a < a'$, then delete $o_{i,a}$ because it represents redundant information. This prevents the transmission of entries of the form $o_{i,*}$, where $o_{i,*}.Dests = \emptyset$, whenever there is another entry for a message sent later by $i$. As discussed in Sect. 4.2, the use of information encoding to meet the Propagation Constraints must prevent the proliferation of such entries. See SND(5) and RCV(5) for further explanation.
   **Example SND(2):** $l_{i,a}.Dests \setminus M_{j,b}.Dests = \{2, 6\}$; information "$\{2, 6\} \in M_{i,a}.Dests$" must be propagated on all outgoing edges of the computation graph from $(j, b)$.
   $l_{i,a}.Dests \bigcap M_{j,b}.Dests = \{3, 4, 8\}$; As per PC2, $\forall\, x \in (l_{i,a}.Dests \bigcap M_{j,b}.Dests)$, "$x \in M_{i,a}.Dests$" must be suppressed on all outgoing edges of the computation graph from $(j, b)$, with the exception of the message dependency edge to $x$. Hence, $o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{j,b}}$, is as follows:
   - To Node 3: $o_{i,a}.Dests = \{2, 3, 6\}$
   - To Node 4: $o_{i,a}.Dests = \{2, 4, 6\}$
   - To Node 8: $o_{i,a}.Dests = \{2, 6, 8\}$
   - To Nodes 7 and 11: $o_{i,a}.Dests = \{2, 6\}$
3. **SND(3):** Step SND(2) suppressed the propagation of information "$x \in M_{i,a}.Dests$", $\forall\, x \in (l_{i,a}.Dests \bigcap M_{j,b}.Dests)$, on the outgoing message edges of $(j, b)$ in the computation graph to enforce PC2. SND(3) suppresses the propagation of the above information on the outgoing local edge, viz., deletes it from the local log.
   **Example SND(3):** $l_{i,a}.Dests$ is updated to $\{2, 6\}$ by deleting $\{ 3,4,8 \}$.
   The idea of deleting older dependencies that will get transitively satisfied was used in [9] which tracked direct dependencies in the context of system-wide broadcasts. However, the contribution of the proposed step is that it deduces and maintains implicit information from the bare minimum information and does not use $n \times n$ arrays to store and transfer information.
4. **SND(4):** At event $Send(M_{j,b})$, a new causal dependency to each destination is created. As per the Propagation Constraints, this information must be propagated on the outgoing edges of the computation graph, including the local log. Therefore, $l_{j,b}$, where $l_{j,b}.Dests = M_{j,b}.Dests$, is inserted in $LOG_j$ to help ensure that $M_{j,b}$ is guaranteed to be delivered in causal order with respect to any message sent in the causal future of $Send(M_{j,b})$. (The information was propagated to each destination of $M_{j,b}$ as a parameter in step SND(2)).
   **Example SND(4):** $l_{j,b}$, where $l_{j,b}.Dests = \{ 3, 4, 7, 8, 11 \}$, is inserted in $LOG_j$.
5. **SND(5):** The invocated procedure PURGE_NULL_ENTRIES deletes redundant entries that can be inferred using implicit tracking. See explanation of PURGE_NULL_ENTRIES and also of RCV(5) which invokes this procedure.

**RECEIVE $M$ at node $j$:**

The RCV procedure infers and manipulates information about messages that (i) are known to be delivered or (ii) are guaranteed to be delivered in CO, in an implicit manner from the information about messages neither (i) known to be delivered nor (ii) guaranteed to be delivered in CO. It then uses the inferred information to enforce propagation constraints PC1 and PC2.

**Example Scenario:** Let $M$ arrive at $j$. Entries of the form $o_{i,a}$ in $O_M$ are: $\{ o_{i,7}, o_{i,9}, o_{i,12}, o_{i,20} \}$. Entries of the form $l_{i,a}$ in $LOG_j$ just before $M$ is received are: $\{l_{i,9}, l_{i,10}, l_{i,14}, l_{i,20}, l_{i,21}\}$.

1. **RCV(1):** When message $M$ arrives at $j$, the processing of $M$ is delayed until the Delivery Condition is satisfied, i.e., until those messages that have been sent to $j$ as per the information in $O_M$ have been delivered to $j$.
2. **RCV(2):** Message $M$ is delivered and the local data structure is updated to reflect the timestamp of this latest delivered message from its specific sender.
3. **RCV(3):** Information about messages identified in $O_M$ as having been sent to $j$ causally before $M$ was sent is deleted from $O_M$ because it served its purpose of enforcing the Delivery Condition in RCV(1).
4. **RCV(4):** This step deduces the implicit information in $LOG_j$ and $O_M$, uses this implicit information to detect and delete redundant explicit information in $LOG_j$ and $O_M$, and combines the nonredundant explicit information in $LOG_j$ and $O_M$ to update $LOG_j$. This step enforces the information encoding described in Sect. 4.2 to satisfy the Propagation Constraints. The two loops in this step achieve the following.
   - Let $M$ be delivered to $j$ and let $max\{x \mid o_{i,x} \in O_M\} = a$. All messages in the set $\{M_{i,a'} \mid a' < a \bigwedge o_{i,a'} \notin O_M\}$ are known to be delivered or guaranteed to be delivered in CO to all their respective destinations. At the time $M$ is delivered to $j$, all $l_{i,a'} \in LOG_j$, where $a' < a$ and $o_{i,a'} \notin O_M$, can be deleted. The value $a$ represents the greatest lower bound on timestamps of messages sent by $i$, except for $a''| (a'' < a \bigwedge o_{i,a''} \in O_M)$, that do not have to be tracked any further to enforce CO by the sender of the message $M$.
     Similarly, at the time $M$ is delivered to $j$, let $max\{x \mid l_{i,x} \in LOG_j\} = a$. All messages in the set $\{M_{i,a'} \mid a' < a \bigwedge l_{i,a'} \notin LOG_j\}$ are known to be delivered or guaranteed to be delivered in CO to all their respective destinations. All $o_{i,a'} \in O_M$, where $a' < a$ and $l_{i,a'} \notin LOG_j$, can be deleted. The value $a$ represents the greatest lower bound on timestamps of messages sent by $i$, except for $a''| (a'' < a \bigwedge l_{i,a''}$

$\in LOG_j$), that do not have to be tracked any further to enforce CO, as known to $j$.
All the information identified above is deleted in the first loop.
**Example RCV(4.loop 1):** The following implicit information in $O_M$ and $LOG_j$ is deduced. Of all messages sent by node $i$ before 20, only the dependencies given in $o_{i,7}$, $o_{i,9}$, and $o_{i,12}$ remain to be satisfied. Of all messages sent by node $i$ before 21, only the dependencies given in $l_{i,9}$, $l_{i,10}$, $l_{i,14}$, and $l_{i,20}$ remain to be satisfied.
Therefore, delete the entries $o_{i,7}$, $o_{i,12}$, $l_{i,10}$, and $l_{i,14}$. (The absence of $l_{i,7}$, $l_{i,12}$, $o_{i,10}$, $o_{i,14}$ indicates that $M_{i,7}$, $M_{i,12}$, $M_{i,10}$, $M_{i,14}$ have all been delivered or are guaranteed to be delivered in CO to all their respective destinations.)

– For each pair of the form $l_{i,a}$ and $o_{i,a}$ that exists in $LOG_j$ and $O_M$, respectively, it is inferred that $M_{i,a}$ has been delivered or is guaranteed to be delivered in CO to destinations in $(M_{i,a}.Dests \setminus l_{i,a}.Dests) \bigcup (M_{i,a}.Dests \setminus o_{i,a}.Dests)$. Therefore, no constraint remains to be satisfied by the delivery of $M_{i,a}$ to the above nodes, viz., nodes not in $o_{i,a}.Dests$ or not in $l_{i,a}.Dests$. Thus, constraints on the delivery of $M_{i,a}$ need to be satisfied only for the delivery to nodes in $o_{i,a}.Dests \bigcap l_{i,a}.Dests$.
After the information from the $o_{i,a}$ entries identified above is captured in the corresponding $l_{i,a}$ entries by doing the set intersection, the $o_{i,a}$ entries used in this step are deleted.
**Example RCV(4.loop 2):** Both $LOG_j$ and $O_M$ contain entries about $M_{i,9}$ and $M_{i,20}$. Common entries in $LOG_j$ and $O_M$ about the same message are processed as follows:

  – If $o_{i,9}.Dests = \{7, 11\}$ and $l_{i,9}.Dests = \{2, 7, 13\}$, then update $l_{i,9}.Dests \longleftarrow \{7\}$. (From $o_{i,9}$, node $j$ infers that $M_{i,9}$ has been delivered or is guaranteed to be delivered in CO to $\{2, 13\}$. From $l_{i,9}$, node $j$ infers that $M_{i,9}$ has been delivered or is guaranteed to be delivered in CO to $\{11\}$.)

  – If $o_{i,20}.Dests = \{15, 16\}$ and $l_{i,20}.Dests = \{3\}$, then update $l_{i,20}.Dests \longleftarrow \emptyset$. (From $o_{i,20}$, node $j$ infers that $M_{i,20}$ has been delivered or is guaranteed to be delivered in CO to $\{3\}$. From $l_{i,20}$, node $j$ infers that $M_{i,20}$ has been delivered or is guaranteed to be delivered in CO to $\{15, 16\}$.)

  As the explicit and implicit information in $o_{i,9}$ and $o_{i,20}$ is captured in $l_{i,9}$ and $l_{i,20}$, respectively, $o_{i,9}$ and $o_{i,20}$ are deleted.

The remaining information in $O_M$ and $LOG_j$ is about messages that are not known to be delivered and not guaranteed to be delivered in CO to their respective remaining destinations. This information must be stored and propagated. The information is merged and consolidated in $LOG_j$ by doing a union of the current states of $LOG_j$ and $O_M$.

5. **RCV(5):** This step executes procedure PRUGE_NULL_ENTRIES which prevents the proliferation of entries of the form $l_{i,*}$, where $l_{i,*}.Dests = \emptyset$. See explanation of PURGE_NULL_ENTRIES.
**Example RCV(5):** Delete $l_{i,20}$ from $LOG_j$. Its presence is implied despite its subsequent absence, by the presence of $l_{i,21}$ in $LOG_j$.

**PURGE_NULL_ENTRIES(LOG):**
This procedure prevents the proliferation of entries of the form $l_{i,*}$, where $l_{i,*}.Dests = \emptyset$ and $l \in LOG$, where $LOG$ is the log at some process. As discussed in Sect. 4.2, the use of information encoding to meet the Propagation Constraints must prevent the proliferation of such entries. If $\exists l_{i,a}, l_{i,a'} \in LOG \mid l_{i,a}.Dests = \emptyset$, and $a < a'$, then delete $l_{i,a}$ because it represents redundant information. However, entry $l_{i,a''}$, where $a'' = max\{\ a' \mid l_{i,a'} \in LOG\}$, is retained because it is required in RCV(4) to represent the implicit greatest lower bound on timestamps of messages sent by $i$ that do not have to be tracked.
**Example PURGE_NULL_ENTRIES:** See RCV(5) which invokes this procedure and continues the running example of procedure RCV.

### *4.5 Some notes on the algorithm*

We now make some notes on the algorithm.

– The form in which the algorithm is presented requires that during a multicast, the message sent to the various destinations is different. This appears to preclude a hardware multicast. However, the algorithm can be readily modified to enable hardware multicast as follows.
When a node $j$ sends a message $M$ to $Dests$, in Step SND(2), the node $j$ sends $(j, clock_j, M, Dests, O_M)$ to each $d \in Dests$ using hardware multicast. The logic in step SND(2) can be performed at the start of step RCV(3). This results in extra overhead only on the multicast message; at each destination of the multicast, the logic of SND(2) is performed and hence there is no avalanche effect of such overhead.
– When a message $M$ is received by process $j$ but the message cannot be delivered, process $j$ can perform a lookahead and execute RCV(4) and RCV(5) using $O_M$. This lookahead eliminates redundant information from $LOG_j$, thus preventing its propagation on multicasts that occur after $M$ arrives and before $M$ gets delivered. Subsequently, when $M$ is delivered, these steps are repeated. Thus, the algorithm steps could be reordered so that steps RCV(4) and RCV(5) also occur before RCV(1).
– Standard programming techniques can be used to improve the computation overhead of SND and RCV. For example, the log at a process can be kept sorted by the sender identifier and the sender's clock value. Each $l_{i,a}.Dests$ can be kept sorted on the destination identifier. Similarly, within each $O_M$ that is sent from node $k$ to node $j$, the information about all earlier messages sent to $j$, that is contained in $O_M$, can be isolated so that the search in RCV(1) and RCV(3) is simplified. It is evident that in SND(2), a separate $O_M$ is computed from $LOG$ for each destination $d \in M.Dests$ even though these computations are almost identical, viz., deleting $M.Dests$ from $o.Dests$, $\forall\ o \in O_M$, and the same as

in SND(3–5). Instead, a single common computation of deleting $M.Dests$ from each log entry can be done, concurrently with which individual $O_M$s to each node $d$ can be adjusted by identifying log entries for which $d$ is a destination. The presentation of the algorithm avoids such details of programming techniques and data structures so as to simplify the conceptual presentation and minimize the complexity of the proof procedure.
- Processes can be dynamically added and deleted. A new process starts with its local clock set to 0. A departed process appears to other processes as though it never sends messages, until it is removed from their view. The issue of how a process (departed process) is included in (excluded from) the view of other processes is a network management issue and an application-specific issue beyond the scope of the CO problem.

## 5 Correctness proof

In Sect. 5.1, we prove certain properties of the algorithm that are used in its correctness proof and optimality proof. Theorem 3 in Sect. 5.2 shows that the algorithm satisfies the sufficient conditions on the information that must be stored and transmitted for enforcing CO, as per the Propagation Constraints. In Sect. 5.3, Theorem 3 in conjunction with the Delivery Condition is used to show the safety and the liveness of the algorithm.

### *5.1 Algorithm properties*

We identify properties of the algorithm that will be used in the correctness proof and the optimality proof.

Lemma 1 states that for every sender node $i$, a destination node $d$ belongs to $l_{i,*}.Dests$ for at most one entry $l_{i,*}$ in the log at any node. Moreover, for every sender node $i$, a destination node $d$ belongs to $o_{i,*}.Dests$ for at most one entry $o_{i,*}$ in the control information sent in any message.

**Lemma 1** *$\forall i \forall j \forall d \forall a \forall a'$, $d \in l_{i,a}.Dests$ and $d \in l_{i,a'}.Dests$, where $l_{i,a}, l_{i,a'} \in LOG_j$, implies that $a = a'$. (Likewise for entries in $O_{M_{j,b}}$.)*

*Proof.* We first prove the result for log entries. The proof uses contradiction.
(Assumption 1:) $\exists i \exists j \exists d \exists a \exists a'$ such that $d \in l_{i,a}.Dests$, $d \in l_{i,a'}.Dests$ immediately following some event $(j, b)$, where $l_{i,a} \in LOG_j$, $l_{i,a'} \in LOG_j$, and $a' < a$. Without loss of generality, also assume that no event that causally precedes $(j, b)$ satisfies this property.

There are two cases to consider: $(j, b)$ is a send event or a delivery event.
$(j, b)$ is a SEND event: In SND, only $l_{j,b}$ is added to $LOG_j$, hence $i = j$. Also, we must have $b = a > a'$. From SND(3), if $d \in l_{j,a}.Dests$ for the current multicast at $(j, a)$ and $\exists d \in l_{j,a'}.Dests$, where $a' < a$ and $l_{j,a'} \in LOG_j$, then $d$ is deleted from $l_{j,a'}.Dests$. This contradicts Assumption(1) and hence such an event $(j, b)$ cannot exist.
$(j, b)$ is a DELIVERY event for $M_{j',c}$: Just before event $(j', c)$, the following did not hold from Assumption(1): "$d \in l_{i,a}.Dests$ and $d \in l_{i,a'}.Dests$, where $l_{i,a} \in LOG_{j'}$ and $l_{i,a'} \in LOG_{j'}$." We then have from SND(2) that "$d \in o_{i,a}.Dests$ and $d \in o_{i,a'}.Dests$, where $o_{i,a} \in O_{M_{j',c}}$ and $o_{i,a'} \in O_{M_{j',c}}$" does not hold when $O_{M_{j',c}}$ is processed by RCV at $(j, b)$ . . . . . (Claim 1).

(Note that if $j' = i$, $d = j$, $c = a$, $d \in M_{j',c}.Dests$, and $d \in l_{i,a'}.Dests$, where $l_{i,a'} \in LOG_{j'}$, just before $(j', c)$, then $d \in o_{i,a'}.Dests$, where $o_{i,a'} \in O_{M_{j',c}}$. In RCV(3), $O_{M_{j',c}}$ is modified as follows: $d$ is deleted from $o_{i,a'}.Dests$ and $o_{i,a}$, where $d \in o_{i,a}.Dests$, is added to $O_{M_{j',c}}$.)

From Assumption(1), we have that before $(j, b)$, $LOG_j$ does not contain both $l_{i,a}$ and $l_{i,a'}$, where $d \in l_{i,a}.Dests$ and $d \in l_{i,a'}.Dests$. . . . . (Claim 2).

From Claims(1 and 2), assume without loss of generality that $l_{i,a} \in LOG_j$, where $d \in l_{i,a}.Dests$, and that $o_{i,a'} \in O_{M_{j',c}}$, where $d \in o_{i,a'}.Dests$. (The reasoning in the case that $l_{i,a'} \in LOG_j$, where $d \in l_{i,a'}.Dests$, and $o_{i,a} \in O_{M_{j',c}}$, where $d \in o_{i,a}.Dests$ is similar). In RCV(4), we have the following:

- If $l_{i,a'} \in LOG_j$, then $d \notin l_{i,a'}.Dests$ (Claim 2) and $d$ is deleted from $o_{i,a'}.Dests$.
- If $l_{i,a'} \notin LOG_j$, then $o_{i,a'}$ is deleted from $O_{M_{j',c}}$.

In either case, "$d \in l_{i,a}.Dests$, $d \in l_{i,a'}.Dests$ where $l_{i,a} \in LOG_j$, $l_{i,a'} \in LOG_j$" does not hold after the execution of procedure RCV for the delivery event. Hence, Assumption(1) is contradicted and such an event $(j, b)$ cannot exist.

As Assumption(1) is contradicted in all cases, the lemma holds for log entries. Observe from SND(2) that $O_{M_{j,b}}$ is a subset of $LOG_j$ just before the event $(j, b)$. Hence, the lemma holds for entries in the control information in messages. □

We now define a function on the timestamps of messages.

**Definition 6** *The function $TS$ is as follows:*

- $TS(i,LOG_j) = max \{ x \mid l_{i,x} \in LOG_j \}$.
- $TS(i,O_M) = max \{ x \mid o_{i,x} \in O_M \}$.

$TS(i,LOG_j)$ is the highest value of the timestamp of messages multicast by $i$ as known from the information in $LOG_j$. $TS(i,O_M)$ is the highest value of the timestamp of messages multicast by $i$ as known from the information in $O_M$. $TS(i,LOG_j)$ in the duration between two consecutive events at node $j$ is associated with the local dependency edge between the events in the computation graph. Similarly, $TS(i,O_M)$ for a message $M$ is associated with the message dependency edge between the send-delivery pair of events of $M$ in the computation graph. The timestamp $TS(i,LOG_j)$ or $TS(i,O_M)$ is useful for detecting and deleting explicitly stored redundant information about messages multicast by $i$ at or before the event represented by the timestamp.

The significance of $l_{i,x}$, where $x = TS(i, LOG_j)$, is as follows. All dependencies originating at node $i$ before $(i, x)$ that are not explicitly included in some $l_{i,*}$ are implicitly included in $l_{i,x}$, with the semantics that these dependencies have been satisfied. Term $x$ represents implicitly the greatest lower bound on timestamps of all dependencies originating at $i$ before $(i, x)$ that have been satisfied or are guaranteed to be satisfied. Only unsatisfied dependencies are explicitly expressed in some $l_{i,*}$. By using a single value $x$, this scheme

allows the implicit representation of information about all dependencies originated by a given $i$ that have been satisfied or are guaranteed to be satisfied. The significance of $TS(i, O_M)$ is the same.

Lemma 2 states that for each sender node $i$, and any other node $j$, timestamp $TS(i, LOG_j)$ is always defined and never decreases as a function of time.

**Lemma 2** *$\forall i \forall j$, $TS(i,LOG_j)$ is a monotonic continuous nondecreasing function of time.*

*Proof.* Initially, $\exists\ l_{i,0} \in LOG_j$; hence $TS(i, LOG_j) = 0$. Let the value of $TS(i, LOG_j)$ presently be $a$. Only the following scenarios at $j$ could potentially trigger a change to $TS(i, LOG_j)$ :

- Node $j$ sends a message at $clock_j$. In SND(3), one or more $l_{i,a'}.Dests$ may become $\emptyset$. In addition, if $i = j$, then in SND(4), $TS(i, LOG_j)$ gets incremented to a higher value, namely, $clock_j$. By SND(5), the value of $TS(i, LOG_j)$ is unchanged.
- Message $M$ is delivered. In RCV(4), when $O_M$ and $LOG_j$ entries are merged, it is seen that $TS(i, LOG_j)$ gets updated to the greater of its current value and $TS(i, O_M)$. RCV(5) does not change this value.

It follows that $TS(i, LOG_j)$ is monotonic nondecreasing and is continuous. □

Lemma 3 is a counterpart of Lemma 2. It states that for any path in the computation graph (corresponding to a chain in the distributed computation), the timestamps $TS(i, O_M)$ and $TS(i, LOG)$ associated with the edges of the path never decrease along the path.

**Lemma 3** *For any path in the computation graph, the timestamp $TS(i, O_M)$ or $TS(i, LOG)$ associated with the edges of the path is a monotonic continuous nondecreasing function along the path.*

*Proof.* Given any path in the computation graph, the result holds for each of its subpaths that contain dependency edges on the same process, by Lemma 2. It remains to show that if $M_{j,b}$ is sent to $k$ and delivered at $(k, c)$, "$TS(i, LOG_j)$ just before $(j, b)$" $\leq TS(i, O_{M_{j,b}}) \leq$ "$TS(i, LOG_k)$ just after $(k, c)$".

Let $a = TS(i, LOG_j)$ just before $(j, b)$ and let $a' = TS(i, LOG_k)$ just before $(k, c)$. (If $j = i$, then $b > a$, and at the time $M_{j,b}$ is being delivered in RCV(3), $o_{j,b} \in O_{M_{j,b}}$ and $TS(i, O_{M_{j,b}}) = b\ (> a)$). From the above case and SND(2), we have that when $M_{j,b}$ is delivered, $TS(i, O_{M_{j,b}}) \geq a$. Step RCV(4) sets $TS(i, LOG_k)$ to the maximum of $a'$ and $TS(i, O_{M_{j,b}})\ (\geq a)$. In RCV(5), this value is unchanged. The result follows. □

**Definition 7** *A set is monotonic nonincreasing if an element is never added to the set after it is initialized.*

Each set $l_{i,a}.Dests$ in $LOG_j$ in the duration between two consecutive events at node $j$ is associated with the local dependency edge between the events in the computation graph. Similarly, each set $o_{i,a}.Dests$ in $O_M$ for a message $M$ is associated with the message dependency edge between the send-delivery pair of events of $M$ in the computation graph.

Lemma 4 states that the contents of any $l_{i,a}.Dests$ are monotonically nonincreasing as a function of time in the log at any node after its first insertion in the local log.

**Lemma 4** *The set $l_{i,a}.Dests$, where $l_{i,a} \in LOG_j$, and $j$ is any node, is monotonic nonincreasing after initial insertion.*

*Proof.* Node $j$ never inserts $l_{i,a}$ into $LOG_j$ at an internal event. $l_{i,a}$ at node $j$ changes, including being added and deleted, only in the following situations:

**Procedure SND at $j$:** In SND(3), the destinations of the current multicast are deleted from $l_{i,a}.Dests$ if $l_{i,a} \in LOG_j$. In SND(5), $l_{i,a}$ is deleted from $LOG_j$ if $l_{i,a}.Dests = \emptyset$ and $\exists\ l_{i,a'} \in LOG_j$, where $a' > a$. Elements are never added to $l_{i,a}.Dests$ in procedure SND except that if $j = i$, then $l_{i,a}$ is inserted in $LOG_i$ for the first time when step SND(4) is executed for event $Send(M_{i,a})$ (the initial insertion case).

**Procedure RCV at $j$:** Whenever a message $M$ arrives at $j$, these scenarios exist:

1. $\exists l_{i,a} \in LOG_j$. Three possibilities determine the actions in RCV.
   - $o_{i,a} \in O_M$. In Step RCV(4), $l_{i,a}.Dests \longleftarrow l_{i,a}.Dests \bigcap o_{i,a}.Dests$.
   - $o_{i,a} \notin O_M$ and $\exists o_{i,a'} \in O_M$, where $a' > a$. In Step RCV(4), $l_{i,a}$ is deleted from $LOG_k$
   - $o_{i,a} \notin O_M$ and $\nexists o_{i,a'} \in O_M$, where $a' > a$. In Step RCV(4), $l_{i,a}$ is unmodified.

   Subsequently, in procedure RCV, elements are never added to $l_{i,a}.Dests$.
2. $\nexists l_{i,a} \in LOG_j$. $o_{i,a}$ is inserted in $LOG_j$ only if $(\nexists l_{i,a'} \in LOG_j \mid a' > a)$ in RCV(4). Therefore, if $o_{i,a}$ is inserted in $LOG_j$ (implying that $\nexists l_{i,a'} \in LOG_j \mid a' > a$), it remains to be shown that this is an initial insertion, i.e., $l_{i,a}$ had not existed in $LOG_j$ before.

   We prove by contradiction. Assume that $l_{i,a}$ had existed in $LOG_j$ before and was deleted. This could have happened only in the following ways, all of which lead to contradiction:
   - in RCV(4), when a message $M''$ was delivered, $\nexists o_{i,a} \in O_{M''} \bigwedge (\exists o_{i,a''} \in O_{M''} \mid a'' > a)$. When $O_{M''}$ and $LOG_j$ were merged in RCV(4), $TS(i, LOG_j)$ is $a''\ (> a)$. From Lemma 2, it follows that currently $TS(i,LOG_j) > a$. This contradicts the current nonexistence of $l_{i,a'}$.
   - in RCV(5), at an event when $l_{i,a}.Dests = \emptyset$ and $(\exists l_{i,a''} \mid a'' > a)$. From Lemma 2, it follows that currently $TS(i,LOG_j) \geq a'' > a$. This contradicts the current nonexistence of $l_{i,a'}$.

   Hence, $l_{i,a}$ had not existed in $LOG_j$ before.

In all cases, the lemma holds. □

Lemma 5 is the counterpart of Lemma 4 and states that for any path in the computation graph (corresponding to a chain in the distributed computation) originating at $(i, a)$, the contents of $o_{i,a}.Dests$ or $l_{i,a}.Dests$ associated with the edges of the path are monotonically nonincreasing along the path.

**Lemma 5** *For any path in the computation graph originating at $(i, a)$, the sets $o_{i,a}.Dests$ or $l_{i,a}.Dests$ associated with the edges of the path are monotonically nonincreasing along the path.*

*Proof.* From Lemma 4, $l_{i,a}.Dests$ is monotonic nonincreasing in $LOG_j$, for any node $j$. It remains to show that if $M_{j,b}$ is sent to $k$, where $k$ is any node, and delivered at $(k,c)$, then "$l_{i,a}.Dests$ just before $(j,b)$" is a superset of $o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{j,b}}$, and this $o_{i,a}.Dests$ is a superset of "$l_{i,a}.Dests$ just after $(k,c)$".

In the base case when $i = j$ and $a = b$, $o_{j,b}$ is inserted in $O_{M_{j,b}}$ on delivery in RCV(3). In all other cases, from SND(2), $M_{j,b}$ will contain $l_{i,a}.Dests$ modified in a nonincreasing manner as $o_{i,a}$ in $O_{M_{j,b}}$. When $M_{j,b}$ gets delivered, $o_{i,a}$ may be reduced in RCV(3). In RCV(4), there are three possibilities.

- $l_{i,a} \in LOG_k$. In Step RCV(4), $l_{i,a}.Dests \longleftarrow l_{i,a}.Dests \bigcap o_{i,a}.Dests$.
- $l_{i,a} \notin LOG_k$ and $\exists l_{i,a'} \in LOG_k$, where $a' > a$. In Step RCV(4), $l_{i,a}$ is not added to $LOG_k$, hence $l_{i,a} \notin LOG_k$.
- $l_{i,a} \notin LOG_k$ and $\nexists l_{i,a'} \in LOG_k$, where $a' > a$. In Step RCV(4), $o_{i,a}$ is inserted into $LOG_k$ in unmodified form.

Subsequently, in procedure RCV, elements are never added to $l_{i,a}.Dests$. The lemma follows. □

Lemma 6 deals with the existence of information "$d \in M_{i,a}.Dests$" in the local log at any event in the causal future of $(i,a)$, which is the earliest event that does not propagate the information on some outgoing edge in the computation graph.

**Lemma 6** *Consider any event $(k,c)$, other than the event $Delivery_d(M_{i,a})$, such that*

- *$(i,a) \longrightarrow (k,c)$ and information "$d \in M_{i,a}.Dests$" is not propagated from $(k,c)$ on at least one outgoing edges in the computation graph, and*
- *$\forall\ (k',c') \mid (i,a) \longrightarrow (k',c') \longrightarrow (k,c)$, information "$d \in M_{i,a}.Dests$" is propagated from $(k',c')$ on all outgoing edges in the computation graph.*

*Then information "$d \in M_{i,a}.Dests$" exists in $LOG_k$ just before $(k,c)$.*

*Proof.* We first show that for the two cases: $k = i$ and $k \neq i$, the information "$d \in M_{i,a}.Dests$" exists in $LOG_k$ at some event before $(k,c)$.

- $\underline{k = i.}$ In SND(4) at $(i,a)$, $l_{i,a}$, where $d \in l_{i,a}.Dests$, is inserted in $LOG_k$.
- $\underline{k \neq i.}$ We prove using contradiction. As $(i,a) \longrightarrow (k,c)$, at least one message sent at or causally after $(i,a)$ must have been delivered to $k$ before $(k,c)$. From our definition of $(k,c)$, each such message $M'$ must have contained $d \in o_{i,a}.Dests$, where $o_{i,a} \in O_{M'}$. Without loss of generality, assume $M'$ is the first such message delivered to $k$. It follows that $l_{i,a}$ could not have existed in $LOG_k$ before delivery of $M'$.
  Let us assume that $d \in o_{i,a}.Dests$, where $o_{i,a} \in O_{M'}$, and this $o_{i,a}$ was not inserted into $LOG_k$ when $M'$ was delivered. The only reason this could have happened is that at the time of delivery of $M'$, $TS(i,LOG_k) = a'$ $(> a)$ (see RCV(4)). This implies that $l_{i,a'}$ must have already been inserted in $LOG_k$ when a message $M''$ was delivered earlier. Clearly, $(i,a) \longrightarrow Send(M'') \longrightarrow (k,c)$ and from the definition of $(k,c)$, $d \in o_{i,a}.Dests$, where $o_{i,a} \in O_{M''}$. This contradicts the assumption that $M'$ was the first message delivered to $k$ such that $d \in o_{i,a}.Dests$, where $o_{i,a} \in O_{M'}$. Therefore, when $M'$ was delivered before $(k,c)$, $l_{i,a}$ was inserted in $LOG_k$, where $d \in l_{i,a}.Dests$.

The only way the information "$d \in M_{i,a}.Dests$" can be deleted after it is inserted in $LOG_k$ is on the delivery of some message $M_{l,f}$, such that information "$d \in M_{i,a}.Dests$" is not contained in $O_{M_{l,f}}$, i.e., (i) $d \notin o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{l,f}}$, or (ii) $o_{i,a} \notin O_{M_{l,f}}$ and $\exists\ o_{i,a'} \in O_{M_{l,f}}$, where $a' > a$. But then event $(l,f)$ did not propagate this information on its outgoing edge in the computation graph, which contradicts the definition of $(k,c)$. Hence, $M_{l,f}$ cannot exist, and $l_{i,a} \in LOG_k$, where $d \in l_{i,a}.Dests$, just before $(k,c)$. □

### *5.2 Sufficiency of information*

Theorem 3 shows that the algorithm satisfies the sufficiency portion of the Propagation Constraints. Observe that Theorem 3 is a restatement of the sufficiency condition. In the proof, we identify events with respect to information "$d \in M_{i,a}.Dests$" such that this information propagates on all paths from $(i,a)$ up to such events. Then we show that any such event implies the existence of a fixed point of "$d \in M_{i,a}.Dests$" at the event or in its causal past. In Sect. 5.3, we show the liveness and safety of the algorithm using Theorem 3 and the Delivery Condition.

**Theorem 3** *If information "$d \in M_{i,a}.Dests$" is not propagated from $(j,b)$, where $(i,a) \longrightarrow (j,b)$, on some outgoing edge in the computation graph, then* **either**

- *$Delivery_d(M_{i,a}) \stackrel{=}{\longrightarrow} (j,b)$,* **or**
- *$\exists\ (k,c) \mid (i,a) \longrightarrow (k,c) \stackrel{=}{\longrightarrow} (j,b) \bigwedge d \in M_{k,c}.Dests \bigwedge$ information "$d \in M_{i,a}.Dests$" is propagated from $(k,c)$ to $d$.*

*Proof.* Consider any event $(k,c)$ that satisfies the following constraints:

1. $(i,a) \longrightarrow (k,c) \stackrel{=}{\longrightarrow} (j,b)$ and information "$d \in M_{i,a}.Dests$" is not propagated from $(k,c)$ on at least one outgoing edge in the computation graph, and
2. $\forall\ (p,f) \mid (i,a) \longrightarrow (p,f) \longrightarrow (k,c)$, information "$d \in M_{i,a}.Dests$" is propagated from $(p,f)$ on all outgoing edges in the computation graph.

Such an event $(k,c)$ must exist by Lemma 5 because initially $d \in l_{i,a}.Dests$, where $l_{i,a} \in LOG_i$ right after $(i,a)$, and the information "$d \in M_{i,a}.Dests$" is propagated on each $M_{i,a}$ that was multicast. If $Delivery_d(M_{i,a})$ satisfies the constraints on $(k,c)$ (this amounts to whether $Delivery_d(M_{i,a}) \longrightarrow (j,b)$), then the theorem stands proved. Otherwise, the proof proceeds as follows.

Observe that at events $Delivery_k(M_{i,a})$, $k \neq d$, information "$d \in M_{i,a}.Dests$" is received in $O_{M_{i,a}}$ and stored in the local logs, i.e., propagated on all outgoing edges of the computation graph from event $Delivery_k(M_{i,a})$, $k \neq d$. Hence, such delivery events do not satisfy the first constraint. (Observation A)

Observe from RCV(4) that no other event $Delivery_k(M)$, where $M \neq M_{i,a}$, will satisfy the constraints

on $(k, c)$ because the only way a node does not propagate information "$d \in M_{i,a}.Dests$" from event $Delivery_k(M)$ along the outgoing edges in the computation graph is if

- the information is not contained in $LOG_k$ and not in $O_M$, or
- the information is contained in $LOG_k$ but not in $O_M$, and $\exists\ o_{i,a'} \in O_M$, where $a' > a$, or
- the information is contained in $O_M$ but not in $LOG_k$, and $\exists\ l_{i,a'} \in LOG_k$, where $a' > a$.

For all three cases, the second constraint on $(k, c)$ is not satisfied by events $Delivery_k(M)$, where $M \neq M_{i,a}$. (Observation B)

From Observations (B) and (A), if $Delivery_d(M_{i,a})$ does not satisfy the constraints on $(k, c)$, then $(k, c)$ must be an event $Send(M_{k,c})$. It remains to show that $d \in M_{k,c}.Dests$ and "$d \in M_{i,a}.Dests$" is propagated on $O_{M_{k,c}}$ to $d$.

From Lemma 6, information "$d \in l_{i,a}.Dests$" must have existed in $LOG_k$ just before $(k, c)$. We have the following: If $d \notin M_{k,c}.Dests$, then from SND(2) it follows that the information "$d \in l_{i,a}.Dests$" is sent on all outgoing edges in the computation graph from $Send(M_{k,c})$, thus violating our assumption about $(k, c)$. Therefore, $d \in M_{k,c}.Dests$. Observe from SND(2) that $M_{k,c}$ that is sent to $d \in (l_{i,a}.Dests \bigcap M_{k,c}.Dests)$ does contain $d$ in $o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{k,c}}$. The theorem follows. □

**Corollary 1** *The information "$d \in M_{i,a}.Dests$" travels to the fixed points of the information.*

*Proof.* The information implicitly reaches FP1, the event $Delivery_d(M_{i,a})$, when $M_{i,a}$ is delivered to $d$.

We prove that the information reaches each event FP2 using contradiction. Assume that the information does not reach $Send(M_{j,b})$, an FP2 event. Let this event be the event $Send(M_{j,b})$ in Theorem 3. From the theorem, there must exist $Send(M_{k,c})$, where $(i, a) \longrightarrow (k, c) \stackrel{=}{\longrightarrow} (j, b)$, that sends the information to $d$. As $(j, b)$ is an FP2 event, $(k, c) \not\rightarrow (j, b)$. Also, $(k, c) \neq (j, b)$ because the information does reach $(k, c)$. Hence, the event $(k, c)$ does not exist, a contradiction. The result follows. □

### *5.3 Correctness of the algorithm*

We prove the correctness of the algorithm by proving its liveness and safety. Proving the safety entails showing that information propagated as per Theorem 3 is used correctly to regulate message delivery to ensure CO. Proving liveness entails showing that each message is eventually delivered.

**Theorem 4 (Safety)***:* $d \in M_{i,a}.Dests \bigwedge d \in M_{j,b}.Dests \bigwedge Send(M_{i,a}) \longrightarrow Send(M_{j,b}) \bigwedge Delivered_d(M_{j,b}) \Longrightarrow Delivered_d(M_{i,a})$.

*Proof.* We prove the theorem by contradiction.

(**Assumption A:**) Let $M_{j,b}$ be delivered to $d$ without $M_{i,a}$ having been delivered.

(**Assumption B:**) Without loss of generality, assume that $M_{i,a}$ is the message such that $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) \leq S\_len_d(Send(M''), Send(M_{j,b}))$, $\forall M''|$

- $Send(M'') \longrightarrow Send(M_{j,b})$, and
- $d \in M''.Dests$ and $\overline{Delivered_d(M'')}$ at the time $M_{j,b}$ is delivered.

This assumption states that $Send(M_{i,a})$ is such that the length of the longest path (defined by function $S\_len_d$) from some $Send(M'')$ to $Send(M_{j,b})$, where $M''$ has not been delivered to $d$ before $M_{j,b}$ is delivered, is least when $M''$ is $M_{i,a}$.

From Assumption A and the Delivery Condition, we must have that $d \notin o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{j,b}}$, or $o_{i,a} \notin O_{M_{j,b}}$ on $M_{j,b}$ sent to $d$. By Theorem 3, then $\exists$ $(k, c)$, $(i, a) \longrightarrow (k, c) \stackrel{=}{\longrightarrow} (j, b)$, such that (i) $(k, c) = Delivery_d(M_{i,a})$, or (ii) $d \in M_{k,c}.Dests \bigwedge d \in o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{k,c}}$ sent to $d$. Case (i) contradicts Assumption A. Therefore, we consider (ii) only. From Assumption B on $M_{i,a}$, any such $M_{k,c}$ must have been delivered to $d$ at the time $M_{j,b}$ is delivered because $S\_len_d(Send(M_{i,a}), Send(M_{j,b})) > S\_len_d(Send(M_{k,c}), Send(M_{j,b}))$. But such an $M_{k,c}$ could have been delivered to $d$ only after $M_{i,a}$ is delivered to $d$ because of the Delivery Condition that is enforced by RCV(1) which is executed when $M_{k,c}$ is delivered. This contradicts Assumption A that $M_{i,a}$ has not been delivered at the time $M_{j,b}$ is delivered. □

**Theorem 5 (Liveness)***: Each message is eventually delivered to all its destinations.*

*Proof.* The proof is by contradiction. Message delivery is reliable and message transmission time is arbitrary but finite. Let there exist $M_{j,b}$ and some $d$ such that $d \in M_{j,b}.Dests$ and every message sent causally before $(j, b)$ to $d$ is eventually delivered to $d$ but $M_{j,b}$ is never delivered to $d$. $O_{M_{j,b}}$ may contain information only about messages sent in the causal past of $(j, b)$. Eventual delivery of $M_{j,b}$ is prevented only by the permanent nondelivery of those messages identified in $O_{M_{j,b}}$ which have $d$ as a destination; however, note that $M_{j,b}$ is such that all such messages are eventually delivered to $d$. Therefore, $M_{j,b}$ is delivered. A contradiction. □

## 6 Optimality of the algorithm

### *6.1 Proof of optimality - necessary conditions for CO*

We now show that the algorithm is optimal in both the control information overhead in messages as well as in local logs in the sense that redundant information (Definition 5) is not carried in messages or stored at nodes. The proof entails showing that the algorithm satisfies the necessary portion of the Propagation Constraints that constitutes the condition for optimality. The proof first identifies fixed points of information "$d \in M_{i,a}.Dests$" and shows that (i) all messages sent at or in the future of the fixed points do not carry this information beyond the fixed points, and (ii) the local logs at any events in the causal future of the fixed points do not store this information. Thus, the "only up to" part of the Propagation Constraints is satisfied.

**Theorem 6** *Information "$d \in M_{i,a}.Dests$" is not propagated on any outgoing edge in the computation graph from any event $(j, b)$ if* **either**

– $Delivery_d(M_{i,a}) \xrightarrow{=} (j,b)$, **or**
– $\exists\ (k,c) \mid (i,a) \longrightarrow (k,c) \longrightarrow (j,b) \bigwedge d \in M_{k,c}.Dests$. *If such a* $(k,c)$ *does not exist and* $(j,b)$ *is a multicast send event such that* $d \in M_{j,b}.Dests$*, then the information "*$d \in M_{i,a}.Dests$*" is sent* only *on the message to* $d$.

*Proof.* From Corollary 1, information "$d \in M_{i,a}.Dests$" traverses up to the fixed points. We prove the result by induction on the $length$ of events $(j,b)$ on any path (defined by function $length$, see Sect. 2.1) from a fixed point FP1 or FP2 of "$d \in M_{i,a}.Dests$".

**Induction hypothesis:** An event $(j,b)$ at $length\ x$ on a given path from a fixed point of information "$d \in M_{i,a}.Dests$" does not propagate "$d \in M_{i,a}.Dests$" on its outgoing edges in the computation graph.

**Event** $(j,b)$ **is at** $length = 0$**:** Two cases exist based on whether the path begins at FP1 or FP2.

(Case I – Paths starting from FP1): Event $(j,b)$ can be any event from FP1, which is the event $Delivery_d(M_{i,a})$, up to the first send event following FP1. Thus, $j = d$.

When $M_{i,a}$ is delivered to $d$ (=$j$) at FP1, $d$ is deleted from $o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{i,a}}$ (RCV(3)) and thus when $l_{i,a}$ is inserted in $LOG_j$, $d \notin l_{i,a}.Dests$ (RCV(4)). From Lemma 4, $l_{i,a}.Dests$ is nonincreasing in $LOG_j$. So from SND(2), it follows that node $j$ never sends information "$d \in M_{i,a}.Dests$" from event $(j,b)$ on any outgoing edge in the computation graph.

(Case II – Paths starting from FP2): Event $(j,b)$ can only be the FP2 event because $(j,b)$ is at $length$ 0 and the FP2 event is a send event. Thus, $(j,b)$ is a fixed point FP2 of "$d \in M_{i,a}.Dests$".

During the sending of $M_{j,b}$, in SND(2), $d \notin o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{j,b}}$ sent to any node except node $d$. However, in this exception case, when $M_{j,b}$ is delivered to node $d$, in RCV(3) and subsequent processing, $d \notin o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{j,b}}$ because it has served its purpose of enforcing the Delivery Condition; this is equivalent to the FP2 event not sending "$d \in M_{i,a}.Dests$" to $d$, as far as optimality is concerned. Moreover, in SND(3), $d$ is deleted from $l_{i,a}.Dests$, where $l_{i,a} \in LOG_j$. It follows that node event $(j,b)$ does not send information "$d \in M_{i,a}.Dests$" from $(j,b)$ on its outgoing edges in the computation graph.

**Event** $(j,b)$ **is at** $length = x, x > 0$**:** Assume that the induction hypothesis holds, i.e., an event $(j,b)$ at $length\ x$ on the given path from a fixed point of information "$d \in M_{i,a}.Dests$" does not propagate this information on its outgoing edges in the computation graph.

**Event** $(j,b)$ **is at** $length = x+1, x > 0$ **:** Let $(j,b)$ be the send event at $length\ x+1$ on the path. If $(j,b)$ does not exist, there are no further send events on the path, implying there are no events on the path at $length$ greater than $x+1$; the proof is done.

If the send events at $length\ x$ and $x+1$ on the path are on the same node, then from the induction hypothesis and Lemma 4, it follows that (i) a message sent at event $(j,b)$ at $length\ x+1$ on the given path does not contain information "$d \in M_{i,a}.Dests$" in $O_{M_{j,b}}$, and (ii) this information does not exist in $LOG_j$ after $(j,b)$. For all delivery events at $length\ x+1$ on this path, this information does not exist in $LOG_j$ after the delivery event.

If the send events at $length\ x$ and $x$+1 on the path are on different nodes, we use the following reasoning. Let $(h,f)$ be the send event at $length\ x$, and let $M_{h,f}$ be delivered at $(j,b')$ along the path under consideration from a fixed point to $(j,b)$. Only the following possibilities exist assuming the induction hypothesis for $length\ x$.

1. $d \notin o_{i,a}.Dests$, where $o_{i,a} \in O_{M_{h,f}}$. The following scenarios exist at $j$ just before $(j,b')$.
   a) $l_{i,a} \in LOG_j$. Step RCV(4) ensures that $d \notin l_{i,a}.Dests$ after the delivery of $M_{h,f}$ at $(j,b')$. From Lemma 4, it follows that $l_{i,a}.Dests$, where $l_{i,a} \in LOG_j$, is monotonic nonincreasing. Hence, $d \notin l_{i,a}.Dests$, where $l_{i,a} \in LOG_j$, at any time after $(j,b')$.
   b) $l_{i,a} \notin LOG_j$. Let $TS(i,LOG_j) = a'$. We now have the following:
      – If $a' > a$, then $o_{i,a}$ is not added into $LOG_j$. From Lemma 2 and RCV(4), it follows that $o_{i,a}$ is not added into $LOG_j$ at any time after $(j,b')$.
      – Otherwise, $a' < a$. $o_{i,a}$ is inserted in $LOG_j$ in RCV(4). From Lemma 4, $d$ will never belong to $l_{i,a}.Dests$, where $l_{i,a} \in LOG_j$, at any time after $(j,b')$.
2. $o_{i,a} \notin O_{M_{h,f}}$. It follows from the induction hypothesis, Lemmas 3 and 5, RCV(4), and RCV(5) that $TS(i,LOG_h) > a$ at event $(h,f)$ and $TS(i,O_{M_{h,f}}) = TS(i,LOG_h)$. The following scenarios exist at $j$ just before $(j,b')$.
   a) $l_{i,a} \in LOG_j$. At $(j,b')$, from RCV(4), it follows that $l_{i,a}$ is deleted because $TS(i,O_{M_{h,f}}) > a$. The value of $TS(i,LOG_j)$ gets updated to the maximum of $TS(i,LOG_j)$ and $TS(i,O_{M_{h,f}})$ in RCV(4). RCV(5) does not alter this value which is greater than $a$. It follows from Lemma 2 and RCV(4) that $o_{i,a}$ is not added into $LOG_j$ after $(j,b')$.
   b) $l_{i,a} \notin LOG_j$. At $(j,b')$, the value of $TS(i,LOG_j)$ gets updated to the maximum of $TS(i,LOG_j)$ and $TS(i,O_{M_{h,f}})$ in RCV(4). RCV(5) does not alter this value which is greater than $a$. When delivery of $M_{h,f}$ at $(j,b')$ is complete, $TS(i,LOG_j) > a$. It follows from Lemma 2 and RCV(4) that $o_{i,a}$ is not added into $LOG_j$ after $(j,b')$.

In each of the above scenarios 1a, 1b, 2a, and 2b, once the delivery of $M_{h,f}$ at $(j,b')$ is completed, information "$d \in M_{i,a}.Dests$" is never added into $LOG_j$. In particular, for all delivery events at $length\ x+1$ along the path at node $j$ (such events lie between $(j,b')$ and $(j,b)$), information "$d \in M_{i,a}.Dests$" does not exist in $LOG_j$ after the event. From procedure SND, event $(j,b)$ will not send information "$d \in M_{i,a}.Dests$" on any of its outgoing edges in the computation graph.

Thus, in all cases, an event in the causal future of a fixed point of "$d \in M_{i,a}.Dests$" does not propagate information "$d \in M_{i,a}.Dests$" on any of its outgoing edges in the computation graph. The theorem follows. □

The proof also follows from Corollary 1, Lemma 5, and the fact that redundant information is deleted at the fixed points (from the argument in the $length$ = 0 case of the proof of Theorem 6).

Lemma 7 states that for every other node $i$, at most one entry $l_{i,a}$ in the local log has its $Dests$ field as $\emptyset$, and

the timestamp of the message it represents is greater than the timestamps of the messages sent by $i$ as represented by other entries $l_{i,*}$ in the local log.

**Lemma 7** *$\forall i \forall j$, $l_{i,a}.Dests = \emptyset$, where $l_{i,a} \in LOG_j \Longrightarrow \forall l_{i,a'} \in LOG_j$, $a \geq a'$.*
*(Likewise for $o_{i,a} \in O_M$).*

*Proof.* From SND(5) and RCV(5), the lemma is seen to hold for log entires after each send and delivery event. From SND(2), the lemma is seen to hold for entries in $O_M$. □

$o_{i,a}$ may be transmitted or stored even if $\forall\ d \in M_{i,a}.Dests$, $M_{i,a}$ is known to be delivered or is guaranteed to be delivered in CO – however, $o_{i,a}.Dests$ must be $\emptyset$ and $\nexists\ o_{i,a'} \mid a' > a$. The sole purpose of transmitting or storing $o_{i,a}$ is to maintain the implicit greatest lower bound on timestamps of messages from node $i$ that have been delivered or are guaranteed to be delivered in CO to all their destinations.

**Theorem 7** *The algorithm in Sect. 4.3 is optimal.*

*Proof.* From Theorem 6, information "$d \in M_{i,a,}.Dests$" is not stored in logs or sent in any messages sent in the causal future of its fixed points, with the exception that a fixed point FP2 propagates the information only to $d$. Hence, information is propagated as per the Propagation Constraints. In addition, no spurious information other than what is specified by the Propagation Constraints is stored or transmitted. Therefore, the algorithm is optimal. □

Thus, the proposed algorithm is optimal with respect to the Propagation Constraints. Enforcement of "no transmission of duplicate information" is a generic constraint common to many distributed algorithms that assume FIFO communication. Therefore, it is of secondary concern to our problem. It requires some additional data structures at each process to keep track of what information has been sent to what processes. Moreover, it may require additional computation at each process to update the data structures and to process the control information received in messages. Thus, there is a trade-off which is another reason why we do not elaborate on this constraint in the paper.

### *6.2 Performance*

We consider the control information overhead and the processing overhead. The control information overhead is important because although networks are becoming faster and provide higher bandwidths, faster computers and parallel processing technology are likely to make the network a bottleneck. Moreover, the demand for computer networks is rising at a faster rate than the communication bandwidth of computer networks; therefore, the volume of information pushed through computer networks will always be an important performance concern.

#### Control information overhead

We compute the size of the control information sent as $O_M$ in a message $M$ and stored in local logs. For every message sent in the past of $Send(M)$ that is not yet known to be delivered or is not guaranteed to be delivered in CO to all its destinations, there is an entry $o_{i,a}$ in $O_M$. Let $o_{i,t_1}$, $o_{i,t_2}$, ..., $o_{i,t_{n_i}}$ be the entries in $O_M$ corresponding to such messages sent by node $i$. Let $\Phi_i$=$o_{i,t_1}.Dests \bigcup o_{i,t_2}.Dests \bigcup .... \bigcup o_{i,t_{n_i}}.Dests$. $\Phi_i$ denotes the destinations to which the messages from node $i$ are not yet known to be delivered or are not guaranteed to be delivered in CO. From Lemma 1 and procedure SND, the number of entries in $\Phi_i$ as well as the number of corresponding entries in any log are bounded by $n$; i.e., there is at most one entry for each other node. (Observe that at any node, if some $l_{i,a}.Dests = \emptyset$, then $M_{i,a}$ is known to be delivered or guaranteed to be delivered in CO to at least one destination, say $d$. From Lemma 7 and the correctness of the algorithm (Theorem 4), $\nexists\ l_{i,a'} \mid d \in l_{i,a'}.Dests$. Hence, $|\Phi_i| < n$ in this case.)

Let $nd_i = |\Phi_i|$. ($nd_j$=0 if there is no entry for a message from node $j$.) Clearly, $0 \leq nd_i \leq n$, $1 \leq i \leq n$.

The overhead in terms of size of control information in messages is equal to

$$\sum_{i=1}^{i=n} nd_i$$

The above expression gives the number of independent units of information of type "$d \in M_{i,a}.Dests$" that are required by the algorithm. The data structures used in the algorithm were selected for power of expression, not for efficiency, and could be optimized in an implementation. For example, instead of transmitting multiple $o_{i,*}$ entries, where the identifier $i$ is replicated, an implementation could store a single instance of identifier $i$, and then associate the various values of $*$ and their associated $Dests$ fields with it. Alternately, a sparse representation of an $n \times n$ array of type *timestamp* could be used. The following encoding can further reduce the overhead. In an entry $o_{i,a} \in O_M$ carried in a message $M$, field $Dests$ can be substituted by $PROC - Dests$ if $|PROC - Dests| < |Dests|$. ($PROC$ is the set of node identifiers.) This optimization ensures the following bound: $0 \leq nd_i \leq n/2$, $1 \leq i \leq n$. We do not discuss any such optimizations because they are orthogonal to the problem of determining the bare minimum pieces of information for enforcing CO, which is the problem addressed in this paper.

The upper bound on the overhead of control information in messages and in local logs is $n^2$. However, in a real-life computation, we expect that the size of these overheads will be much smaller on the average, and in every case, it is always the least possible.

#### Processing overhead

The processing overhead of procedure RCV is $O(n^2)$. The processing overhead of procedure SND is $O(n^3)$ because SND(2) has $O(n^3)$ overhead, even though other steps in SND have $O(n^2)$ or better overhead. In Step SND(2), the inner **for all** loop has $O(n^2)$ overhead in the worst case because it has to process $n^2$ entries in $O_M$ in the worst case. The discussion on programming techniques in Sect. 4.5 shows how to reduce the overhead. Note from Lemma 1 that the number of entries in $O_M$ is likely to be much smaller on the average. Furthermore, note that the outer loop in SND(2) can be executed in parallel. Therefore, the processing time overhead of the algorithm as presented in Sect. 4.3 is effectively $O(n^2)$.

In practice, the algorithm is implemented using hardware multicast, as noted in Sect. 4.5. To adapt the algorithm to the use of hardware multicast as explained in Sect. 4.5, the logic in step SND(2) is performed by the recipients of the multicast and the outer loop of SND(2) gets distributed among the recipients' RCV code. Therefore, SND(2) has $O(n^2)$ complexity, SND has $O(n^2)$ complexity, and RCV also has $O(n^2)$ complexity, comparable to that of existing algorithms.

## 7 The complexity of the CO problem

**Theorem 8** *For the system model and problem description of Sect. 2, enforcing CO over causal dependency chains of length greater than two has $\Omega(n^2)$ overhead of control information in messages [2].*

*Proof.* For each $(source,\ destination)$ pair, there is a latest message that has been sent. In general, there are $n^2$ such instances. We prove by contradiction that if all these $n^2$ instances are not included in the control information in messages and message logs, then CO over causal dependency chains of length 3 is violated.

Let $M_{m,t_m}.Dests$=$\{x, d\}$. Suppose after $M_{m,t_m}$ has been delivered to $x$, $x$ unicasts a message to node $y$ at time $t_x$. Let "$d \in M_{m,t_m}.Dests$" be the information instance that is not contained in the control information. Then $d \notin o_{m,t_m}.Dests$, where $o_{m,t_m} \in O_{M_{x,t_x}}$. After $M_{x,t_x}$ has been delivered to $y$, $d \notin l_{m,t_m}.Dests$, where $l_{m,t_m} \in LOG_y$, because $d \notin o_{m,t_m}.Dests$, where $o_{m,t_m} \in O_{M_{x,t_x}}$. Now if $y$ unicasts $M_{y,t_y}$ to node $d$, delivery of $M_{y,t_y}$ to node $d$ will not wait for the delivery of $M_{m,t_m}$ because $d \notin o_{m,t_m}.Dests$, where $o_{m,t_m} \in O_{M_{y,t_y}}$. This results in a CO violation for the dependency chain of length 3 ($m$ to $x$, $x$ to $y$, $y$ to $d$). As nodes $d$ and $m$ can be any nodes in the system, in general, all $n^2$ identifiers must be transmitted on any message to enforce CO. □

Corollary 2 shows that enforcing CO over chains of length 3 has at least as much overhead as enforcing CO for chains of arbitrary length.

**Corollary 2** *For the system model and problem description of Sect. 2, the space complexity of the overhead of control information in messages and in local logs to enforce CO is $\Omega(n^2)$.*

*Proof.* Follows from Theorem 8. □

Although, in general, the complexity of the overhead of control information in messages and in local storage is $\Omega(n^2)$ under the framework of Problem 1 (Sect. 2.2), redundant information is carried in messages and stored at nodes by existing algorithms. Our algorithm employs techniques to eliminate the flow of redundant information. The information stored is only about "to be delivered" messages. The key of the algorithm is to use a representation to store the information such that the information on already delivered messages and messages guaranteed to be delivered in CO is derived from the "to be delivered" information !

### *7.1 Special cases*

The complexity of the CO problem may be reduced by making simplifying assumptions and considering special cases that fall outside the scope of Problem 1. In this section, we discuss some such special cases.

#### Broadcast case

When each message is broadcast to all other nodes, the $Dests$ field of the entries carried in messages and stored in local logs always contains the ids of all nodes and thus, can be eliminated. Thus, the control information overhead in messages and in the logs needs to contain only $O(n)$ entries at any time – one entry $clock_j$ that uniquely identifies the latest broadcast of each node $j$ that occurred at $clock_j$.

Observe that at any node $i$, the values of $clock_i$ ordered by 'precedes locally' and the values of $T_i[i]$ ordered by 'precedes locally', where $T_i$ is the vector timestamp of an event [7, 15], are isomorphic. Therefore, maintaining an entry $clock_j$ for the latest broadcast by each node $j$ as part of the control information is equivalent to maintaining a vector clock and vector timestamps as part of the control information. This latter approach that uses vector clocks to enforce CO was first suggested in [5].

#### Serialized broadcast case

If broadcasts are serialized (by using some mutual exclusion algorithm), then it is sufficient to track (sender, timestamp) for the most recent broadcast in the system. Thus the overhead of control information in messages and in local logs is one entry only.

#### Serialized multicasts to arbitrary process groups

If multicast *Sends* to arbitrary and dynamically changing process groups are serialized (by using some mutual exclusion algorithm), then it is sufficient to track only the latest multicast on a per destination basis. Thus, for each node $d$, it suffices to store only one entry (sender, timestamp) that gives the sender identifier and the timestamp of the most recent *Send* to that node $d$. The overhead of control information is $n$ entries. If the current multicast has $s$ destinations, then after sending the multicast and on delivery, the corresponding $s$ old entries in the local logs are replaced by $s$ entries for the $s$ new dependencies. Thus, at most $n$ (sender, timestamp) pairs are required, resulting in $O(n)$ space overhead.

#### Unicasts with synchronous communication

Unicasts with synchronous communication automatically guarantee CO without any overhead of control information [6]. Let $Send(M)$ and $Send(M')$ be two consecutive sends on any causal chain without any intervening sends. $Send(M')$ can occur only after $Delivery(M)$ because of synchronous communication. It follows that CO can never be violated.

Unicasts with synchronization at transport layer

The algorithm by Mattern and Fünfrocken [16], outlined in Sect. 1, provides CO for unicasts by using a built-in synchronization provided by transport layer acknowledgements between the sender's output buffer and the receiver's input buffer. Let $Send(M)$ and $Send(M')$ be two consecutive sends on any causal chain without any intervening sends. $M$ is placed in the receiver's input FIFO buffer before $M'$ is sent from the sender's output FIFO buffer. It follows that $Delivery(M) \longrightarrow Send(M')$ as the computation is equivalent to a synchronous computation. Thus, CO is never violated and there is no overhead of control information in messages in the algorithm.

Enforcing CO for dependency chains of length two

**Lemma 8** *For the system model and problem description of Sect. 2, CO over causal dependency chains of length two can be enforced with $O(n)$ overhead of control information in messages [2].*

*Proof.* When sending a message from node $i$ to node $j$, $i$ sends a message identifier for (1) the latest message it has sent to each other node ($n$ identifiers) and for (2) the latest message from each other node to $j$ that it is aware of ($n$ identifiers). When the message is delivered at $j$, $j$ updates its information with (1) and (2). Suppose $j$ now sends a message to a randomly chosen node $k$. $j$ has the information of the last message sent by every node, including $i$, to $k$. When $j$ sends information about the latest message from each other node to node $k$ (including $i$ to $k$), $k$ can enforce CO for paths of length two. □

Enforcing CO for dependency chains of length $>2$

Theorem 8 showed that enforcing CO over chains of length 3 has $\Omega(n^2)$ overhead of control information in messages.

Asymmetric/hierarchical organization

If we allow asymmetric protocols to enforce CO, we get a family of algorithms. At one extreme, we get a centralized algorithm wherein a process sends its message to be multicast to a centralized process, which serially multicasts all such requests. At the other extreme, there is the completely hierarchical structure to address the scalability problem. Processes are grouped together based on having a common process to represent the processes at the higher level in the hierarchy. A process that wishes to multicast a message sends it to its designated representative at a higher level in the hierarchy. The message gets broadcast/multicast between representative processes at the higher level in the hierarchy. When such a process receives a broadcast, it multicasts it to the set of processes for which it is the representative process. This logic can be extended to a multi-level hierarchy. Such protocols tend to cut down on the message complexity by using smaller $n$, at the cost of additional delay in the delivery of messages. We do not consider such protocols because they are asymmetric. However, to make a fair comparison, the use of our proposed algorithm for multicasts at any higher level in the hierarchy gives better efficiency than the use of traditional protocols for multicasts at those higher levels in the hierarchy.

Enforcing CO given non-FIFO channels

The algorithm was presented assuming FIFO channels because most known communication networks provide FIFO support. The algorithm can be easily converted into one for non-FIFO channels by requiring sequence numbers in messages on a per channel basis, and a vector at each node to represent the sequence number of the latest in-sequence message delivered to it from every other node.

## 8 Conclusions

Asynchronous execution of processes and unpredictable communication delays create nondeterminism in distributed systems that complicates the design, verification, and analysis of distributed programs. The concept of "causal message ordering" was introduced to simplify the design and development of distributed applications while avoiding the long latencies and loss of parallelism inherent in synchronous communication and total ordering, and while retaining much of the parallelism of asynchronous communication. Causal ordering provides a built-in message synchronization, reduces the nondeterminism in a distributed computation, and is of considerable interest to the design of distributed systems.

The concept of causal ordering is useful in several domains such as updates of replicated data, global state collection, distributed shared memory, teleconferencing, multimedia systems, and fair resource allocation. Recently, the problem of causal ordering has attracted much attention and a number of algorithms have been proposed for causal ordering in distributed systems under a variety of assumptions regarding the underlying communication medium and process communication patterns. The overheads of control information in messages and in process logs for the algorithms in the framework of Problem 1 is O($n^2$) or higher, which limits their scalability, preventing them from meeting the growing demands of future computing environments.

In this paper, we addressed the following fundamental question regarding the efficiency of CO implementations in asynchronous distributed systems under the system model of Sect. 2.1: "What is the minimum amount of information regarding messages sent in the causal past that is necessary to be propagated and stored to enforce causal ordering by an algorithm under the following framework: The protocol is nonblocking, completely decentralized, deterministic, and does not use *a priori* knowledge about the topology or communication pattern?" We answered the question by formulating necessary and necessity conditions on the information required for enforcing causal ordering. The necessity conditions provide the optimality conditions for enforcing causal ordering in terms of the size of control information in messages and in local logs. The necessary and sufficient

conditions were used to formulate two Propagation Constraints that govern the propagation of CO related information through the network.

We used the developed characterization and framework to design an algorithm for enforcing causal message ordering. The algorithm allows a process to multicast to arbitrary and dynamically changing process groups. We proved the correctness of the algorithm and showed that it satisfies the necessity conditions proving that it is optimal in the size of control information in a message and in the size of local storage. The algorithm achieves optimality by using the Propagation Constraints to curtail the propagation of redundant information at the earliest instants and by employing an encoding scheme to represent and pass in messages only the necessary causal dependency information. The encoding scheme allows deduction of implicit information about already delivered messages from the explicit information about messages yet to be delivered in order to satisfy causal ordering. We showed that the space complexity of causal message ordering for any algorithm under the framework of Problem 1 is $\Omega(n^2)$. Although the upper bound on space complexity of the overhead of control information in our algorithm is $O(n^2)$, we expect that the overhead is likely to be much smaller on the average, and in every case, it is always the least possible. We also discussed how the algorithm can be adapted to various special situations outside the scope of Problem 1. In the face of network failures, techniques from [25] can be adapted for log management to maintain causal consistency.

Modeling nonatomic events is useful for event abstraction in reasoning about related groups of events [12, 13]. A nonatomic event is a collection of more basic atomic events. A future research problem is to optimally enforce causal ordering only among messages that are sent in different nonatomic events.

*Acknowledgments.* The authors thank the anonymous referees for their very useful comments on an earlier version of the paper.

## References

1. Alagar A, Venkatesan S: An Optimal Algorithm for Distributed Snapshots with Causal Message Ordering, Inf Process Lett 50: 311–316 (1994)
2. Adelstein F, Singhal M: Real-Time Causal Message Ordering in Multimedia Systems, 15th IEEE Intl. Conf. on Distributed Computing Systems, pp 36–43, May 1995
3. Ahamad M, Hutto P, John R: Implementing and Programming Causal Distributed Memory, 11th IEEE Intl. Conf. on Distributed Computing Systems, pp 274–281, 1991
4. Birman K, Joseph T: Reliable Communication in Presence of Failures, ACM TOCS 5(1): 47–76 (1987)
5. Birman K, Schiper A, Stephenson P: Lightweight Causal and Atomic Group Multicast, ACM TOCS 9(3): 272–314 (1991)
6. Charron-Bost B, Tel G, Mattern F: Synchronous, Asynchronous, and Causally Ordered Communication, Distrib Comput 9(4): 173–191 (1996)
7. Fidge CA: Timestamps in Message-Passing Systems That Preserve Partial Ordering, Aust Comput Sci Commun 10(1): 56–66 (1988)
8. Joseph T, Birman K: Low Cost Management of Replicated Data in Fault-Tolerant Distributed Systems, ACM TOCS 4(1): 54–70 (1986)
9. Kim J, Kim C: An Efficient Causal Ordering Protocol in Group Communications, 10th Int. Conf. on Information Networking, pp 121–128, January 1996
10. Kshemkalyani A, Singhal M: Necessary and Sufficient Conditions on the Information for Causal Message Ordering and Their Optimal Implementation, TR29.2040, IBM, July 1995 (Also available as Tech. Report CISRC-7/95-TR33, July 1995, The Ohio State University, ftp.cis.ohio-state.edu/pub/tech-report/1995/TR33.ps.gz.)
11. Kshemkalyani A, Singhal M: An Optimal Algorithm for Generalized Causal Message Ordering, 15th ACM Symp. on Principles of Distributed Computing, pp 87, May 1996
12. Kshemkalyani A: Temporal Interactions of Intervals in Distributed Systems, J Comput Syst Sci 52(2): 287–298 (1996)
13. Kshemkalyani A: Framework for Viewing Atomic Events in Distributed Computations, Theor Comput Sci 196(1-2): 45–70 (1998) (Abstract appears in Proc. EuroPar'96, L. Bouge, P. Fraigniaud, A. Mignotte, Y. Robert (eds.) LNCS 1123, pp 495–505, Berlin Heidelberg New York: Springer 1996)
14. Lamport L: Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM **21(7)**: 558–565 (1978)
15. Mattern F: Virtual Time and Global States of Distributed Systems, Parallel and Distributed Algorithms, pp 215–226, Amsterdam: North-Holland 1989
16. Mattern F, Fünfrocken S: A Nonblocking Lightweight Implementation of Causal Order Message Delivery, In: KP Birman, F Mattern, A Schiper (eds) Theory and Practice in Distributed Systems, LNCS 938, pp 197–213, Berlin Heidelberg New York: Springer 1995
17. Mostefaoui A, Raynal, M: Causal Multicasts in Overlapping Groups: Towards a Low Cost Approach, IEEE Workshop on Future Trends of Distributed Computer Systems, pp 136–142, September 1993
18. Peterson LL, Buchholz NC, Schlichting RD: Preserving and Using Context Information in Interprocess Communication, ACM TOCS 7: 217–246 (1989)
19. Ravindran K, Prasad B: Communication Structures and Paradigms for Distributed Conferencing Applications, 12th IEEE Int. Conf. on Distributed Computing Systems, May 1992
20. Raynal M, Schiper A, Toueg S: The Causal Ordering Abstraction and a Simple Way to Implement It, Inf Process Lett 39(6): 343–350 (1991)
21. Rodrigues L, Verissimo P: Causal Separators for Large-Scale Multicast Communication, 15th IEEE Int. Conf. on Distributed Computing Systems, pp 83–91, May 1995
22. Schiper A, Eggli J, Sandoz A: A New Algorithm to Implement Causal Ordering, In: J-C Bermond, M Raynal (eds) Proc. 3rd Int. Workshop on Distributed Algorithms, LNCS 392, pp 219–232, Berlin Heidelberg New York: Springer 1989
23. Schwarz R, Mattern F: Detecting Causal Relations in Distributed Computing: in Search of the Holy Grail, Distrib Comput 7(3): 149–174 (1994)
24. Singhal M, Kshemkalyani AD: An Efficient Implementation of Vector Clocks, Inf Process Lett 43: 47–53 (1992)
25. Wuu G, Bernstein A: Efficient Solutions to the Replicated Log and Dictionary Problems, 3rd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, pp 233–242, Vancouver, Canada, August 1984