Total Order Multicast to Multiple Groups[*][†]

Rachid Guerraoui André Schiper
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland

Abstract

We present a fault-tolerant algorithm that ensures total order *delivery of messages sent to multiple groups of processes. Our algorithm is a multiple group "genuine" multicast algorithm in the sense that (1) any process can send a message to any set of process groups, and (2) only the sender and the receivers of a message take part in the algorithm needed to deliver the message. The correctness of our algorithm does not require reliable failure detectors, but requires causal order delivery of messages. This establishes a new and interesting link between causal order delivery and fault-tolerance with unreliable failure detectors.*

1 Introduction

We present in the paper a fault-tolerant total order multicast algorithm that allows multicasting of messages to multiple groups of processes, in an asynchronous system with process crash failures and (unreliable) failure detectors. Before discussing the algorithm, we start by clarifying some issues related to total order algorithms.

[†]Partially supported by OFES under contract number 95.0830, as part of the ESPRIT BROADCAST-WG (number 22455)

1.1 Total order multicast vs total order broadcast

A *total order broadcast* primitive[1] enables to send messages to all the processes in a system, with the guarantee that (1) all correct processes (those that do not crash) agree on either to deliver or to not deliver a message, and (2) no two correct processes deliver any two messages in a different order.

Contrary to a broadcast that is targeted to the set of all the processes in a system, a *multicast* can be targeted to a subset of the processes. Apart from this, similarly to total order broadcast, total order multicast ensures that (1) the correct addressees of every message agree either to deliver or to not deliver the message, and (2) no two correct processes deliver any two messages in a different order.

1.2 Genuine multicast vs feigned multicast

At first glance, it might seem that implementing total order broadcast and total order multicast are equivalent problems, as each algorithm that solves one of these problems, can be transformed to solve the other problem. Indeed, let Ω denote the set of all the processes in the system, and let $\mathsf{Dst}(m) \subset \Omega$ denote the subset of the processes to which the message m is multicast. Firstly, by defining $\mathsf{Dst}(m) = \Omega$ for every message m, a total order multicast algorithm can directly be used to implement a total order broadcast algorithm. Secondly, a naive total order multicast algorithm can be obtained from any total order broadcast algorithm as follows: consider a message m to be multicast to $\mathsf{Dst}(m)$: (1) the pair $(m, \mathsf{Dst}(m))$ is broadcast to all the processes in Ω, using the total order broadcast primitive; (2) a process $p_i \in \Omega$ delivers m only if $p_i \in \mathsf{Dst}(m)$.

[1]Also sometimes called *atomic* broadcast.

The above transformation of total order broadcast into total order multicast leads however to a *feigned* multicast algorithm. With this transformation, and for $\mathsf{Dst}(m) \subset \Omega$, a multicast is as costly as a broadcast, since all the processes in the system are involved in the algorithm, *even those that are not concerned with the message m.*

We require from a *"genuine"* multicast algorithm that it satisfies the following *Minimality* property: *the execution of the algorithm implementing the total order multicast of a message m to a destination set $\mathsf{Dst}(m)$ involves only the sender process, and the processes in $\mathsf{Dst}(m)$*[2]. The naive implementation above – using total order broadcast – does obviously not satisfy the *Minimality* property, as every process in the system, even if not concerned by a message, is involved in the protocol needed to deliver the message. We define a *genuine* total order multicast algorithm to be a total order multicast algorithm that satisfies the *Minimality* property.

1.3 Multicast to multiple groups

In [11] we consider a very general model, where a process can multicast a message m to any subset of processes, and show that in this model, implementation of genuine total order multicast requires a perfect failure detector (i.e., a failure detector that makes no incorrect failure suspicion).

In this paper, we restrict the general system model by considering total order multicasts to process groups (rather than to individual processes), namely the destination of a message m is a set $\mathsf{Dst}(m) = \{g_1, \ldots, g_n\}$, where g_i are process groups. Assuming a majority of correct processes within each group, we present an algorithm named *MTO* (Multicast Total Order), which implements the total order multicast primitive to multiple groups. Multicast to multiple groups can typically be used to implement a specific class of transactions on replicated objects (one replicated object = one group) [18].

MTO is a fault-tolerant extension of Skeen's total order multicast algorithm [19] [3]. MTO requires an unreliable failure detector of the class $\diamond\mathcal{S}$ [6], and causal delivery of messages. Failure detectors of the class $\diamond\mathcal{S}$ are those which ensure that (i) there is a time after which every process that crashes is permanently suspected by all correct processes, and (ii) there is a time after which some correct process is never suspected by any correct process. Such failure detectors are unreliable in the sense that they can make an infinity of false failure suspicions. It has been shown that such failure detectors are the weakest ones that allow solving consensus and total order broadcast problems [5]. Causal delivery of messages is actually not needed by our MTO algorithm if failure detectors are reliable. This establishes a new and interesting link between causal order delivery and fault-tolerance with unreliable failure detectors.

The rest of the paper is structured as follows. Section 2 discusses related work. Section 3 gives the system model and defines total order multicast. Section 4 recalls the original Skeen's algorithm, and shows why it is not fault-tolerant. Section 5 describes the MTO algorithm. Section 6 analyses the cost of MTO, and discusses the tradeoff between total order broadcast and genuine total order multicast. Section 7 concludes the paper.

2 Related work

We mention here only total order algorithms for asynchronous systems. Many of these algorithms are broadcast algorithms, e.g. the Chandra-Toueg algorithm [6], the token based algorithms [7, 1], and the sequencer based algorithm given in [15]. Total order multicast algorithms that have been published, either (1) implement a weaker ordering property called *local total order* [13][4], (2) are not genuine multicast algorithms, (3) require perfect failure detectors (i.e., failure detectors that do not make false suspicions), or (4) rely on a membership service:

1. The current Isis system [4] implements only local total order.

2. The algorithms based on Lamport's total order algorithm [16] do not satisfy the Minimality property of a genuine multicast, because the messages are delivered according to the order defined by the timestamps initially assigned to messages at multicast time [8, 1, 9]. Hence, a process p can deliver a message m, timestamped $ts(m)$, only once p knows that it will receive no further message m' such that $ts(m') < ts(m)$. This might require p to interact with all the processes that can send a message to p. In the case where p can receive a message from any process in the system, p might need to interact with the whole system.

3. The algorithm presented by Garcia-Molina and Spauster [10], and its extension described in [14], are both total order genuine multicast algorithms, but both require reliable failure detectors.

[2] The *Minimality* property reflects the scalability of a multicast algorithm.

[3] Skeen's original algorithm is not fault-tolerant.

[4] Local total order does not prevent processes in intersecting destination sets to deliver messages in different orders.

4. The first Isis ABCAST algorithm [3] is based on Skeen's algorithm [19], and therefore has similarities with our MTO algorithm described in the paper. However, fault-tolerance is ensured in the Isis algorithm by a membership service, whereas MTO does not require a membership service. The membership service can incorrectly suspect a correct process, and force it to crash. This does not occur with the MTO algorithm.

3 System model and definitions

3.1 Asynchronous system

We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Communication is by message passing, asynchronous and eventually reliable. Asynchrony means that there is no bound on communication delays, nor on process relative speeds. An eventual reliable channel ensures that a message sent by a process p_i to a process p_j is eventually received by p_j, if p_i and p_j are correct (i.e., do not fail)[5]. An eventual reliable channel can be implemented by retransmitting lost or corrupt messages. Processes fail by crashing (we do not consider Byzantine failures). A correct process is a process that does not crash in an infinite run.

A process $p_i \in \Omega$ may (1) send a message to another process (*send* event), (2) receive a message sent by another process (*receive* event), (3) perform some local computation, or (4) crash.

3.2 Failure detectors

As in [6], we augment the asynchronous system with failure detectors. A failure detector can be viewed as a distributed oracle. Each process p_i has access to a local failure detector module D_i. This module maintains a list of processes that it currently *suspects* to have crashed.

Failure detectors are abstractly characterized by their *completeness* and *accuracy* properties. In the following, we consider failure detectors in class $\Diamond\mathcal{S}$ (*Eventually Strong*) [6]. These failure detectors guarantee (1) *strong completeness* (there is a time after which every process that crashes is permanently suspected by every correct process), and (2) *eventual weak accuracy* (there is a time after which some correct process is never suspected by any correct process). Notice that failure detectors in $\Diamond\mathcal{S}$ are unreliable: they can make mistakes, i.e., false failure suspicions. The fundamental characteristic of $\Diamond\mathcal{S}$ is that it is the *weakest* class of failure detectors that allow to solve the consensus and the atomic broadcast problems [6].

A failure detector can be implemented based on time-outs. The choice of the time-out value is system dependent (i.e., network dependent). The completeness property (property 1) of $\Diamond\mathcal{S}$ is trivially satisfied with time-outs. If a time-out is chosen too small for a given system, then the probability that the accuracy property (property 2) of $\Diamond\mathcal{S}$ is satisfied is close to 0. If the time-out value is chosen adequately, the probability that the property of $\Diamond\mathcal{S}$ is satisfied can be arbitrary close to 1.

[5]This does not exclude link failures, if we require that any link failure is eventually repaired.

3.3 Total order multicast to multiple groups

We assume that the set Ω of processes is partitioned into (non-intersecting) groups g_i, and messages are multicast to groups. Given a message m, we denote by $\mathsf{Dst}(m)$ the set of process groups to which m is multicast. Total order multicast is defined on the set Ω by the primitives (1) *TO-multicast*$(m, \mathsf{Dst}(m))$ which issues m to $\mathsf{Dst}(m)$, and (2) *TO-deliver(m)* which is the corresponding delivery of m. When a process p_i executes *TO-multicast*$(m, \mathsf{Dst}(m))$ (resp *TO-deliver(m)*), we say that p_i *"TO-multicasts m"* (resp *"TO-delivers m"*). The primitive *TO-multicast* is defined by the following three properties [13]:

TO1. Uniform Agreement. Consider *TO-multicast*$(m, \mathsf{Dst}(m))$. If a process in $\mathsf{Dst}(m)$ (correct or not) has *TO-delivered*(m), then every correct process in $\mathsf{Dst}(m)$ eventually *TO-delivers*(m).

TO2. Termination. If a correct process *TO-multicasts*$(m, \mathsf{Dst}(m))$, then every correct process in $\mathsf{Dst}(m)$ eventually *TO-delivers*(m).

TO3. Uniform Global Order. Let m_1 and m_2 be two messages that are *TO-multicast*. We have $m_1 < m_2$ if and only if a process (correct or not) *TO-delivers* m_1 before m_2. Uniform Global Order ensures that the relation $<$ is acyclic.

4 Skeen's original algorithm

As MTO can be viewed as a fault-tolerant extension of Skeen's algorithm [19], we first recall the latter. Skeen's algorithm is based on Lamport's logical clocks [16]. Given an event e, we denote $ts(e)$ its timestamp using Lamport's clocks [6]. Figure 1 shows the three parts of the algorithm on a simple example of a process p_i, *TO-multicasting* a message m to two groups g_1 and g_2, i.e., $\mathsf{Dst}(m) = \{g_1, g_2\}$. Skeen's algorithm works as follows:

TO-multicast. When *TO-multicast*$(m, \mathsf{Dst}(m))$ is executed by process p_i, the message m is sent to every process in $\mathsf{Dst}(m)$ (part 1 in Fig. 1).

TO-delivery. The TO-delivery of m occurs as the result of the following sequence of actions:

- when $p_j \in \mathsf{Dst}(m)$ receives m, it sends $ts(receive_j(m))$ (the timestamp of the $receive_j(m)$ event) back to p_i (part 2 in Fig. 1). Process p_j then stores the pair $(m, ts(receive_j(m))$ in a *"pending"* buffer (Fig. 2);
- the initiator p_i waits to obtain the timestamp $ts(receive_j(m))$ from every process p_j in $\mathsf{Dst}(m)$, and defines $sn(m)$ as the maximum of all timestamps. Process p_i then sends $sn(m)$ to $\mathsf{Dst}(m)$ (part 3 in Fig. 1);
- when $p_j \in \mathsf{Dst}(m)$ receives $sn(m)$, the pair $(m, ts(receive_j(m))$ is removed from p_j's pending buffer, and $(m, sn(m))$ is inserted into p_j's *"TO-delivery"* queue (Fig. 2), ordered according to $sn(m)$ (i.e., the pair $(m, sn(m))$ with the smallest $sn(m)$ is at the head of the queue). The message m at the head of the TO-delivery queue is TO-delivered by p_j if and only if there is no pair $(m', ts(receive_j(m'))$ in p_j's pending buffer, such that $ts(receive_j(m')) < sn(m)$.

This algorithm ensures the Minimality property of a genuine multicast, and the properties TO1 (Uniform Agreement) and TO3 (Uniform Global Order) of a total order multicast. Property TO2 (Termination) is however only guaranteed if no process crashes. Firstly, if a process in $\mathsf{Dst}(m)$ crashes before sending its timestamp, the initiator p_i would wait forever (to obtain the timestamp from *every* process in $\mathsf{Dst}(m)$). Note that if the failure detector is reliable, p_i can simply wait to obtain the timestamp from every *non suspected* process in $\mathsf{Dst}(m)$. The issue is more tricky if the failure detector is unreliable. Secondly, if the initiator p_i (which computes $sn(m)$) crashes before $sn(m)$ is known to the processes in $\mathsf{Dst}(m)$, the processes in $\mathsf{Dst}(m)$ are blocked, waiting for p_i to recover.

[6] As usual, we consider that the process ID is added as a suffix to the logical clock values, to ensure that all the timestamps are unique.

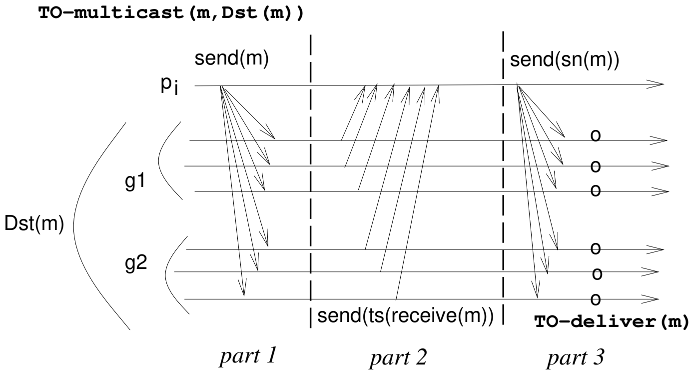

Figure 1: Communication in Skeen's algorithm

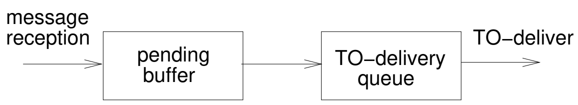

Figure 2: The pending and the delivery queue in Skeen's algorithm

5 The MTO algorithm

5.1 The structure of the MTO algorithm

The principle of the MTO algorithm is similar to that of Skeen's algorithm. Both algorithms order messages according to a sequence number that is computed after the message has been sent, and both algorithms ensure the Minimality property of a genuine multicast.

The MTO algorithm solves however the blocking problems that occur with Skeen's algorithm when processes can crash, i.e., MTO ensures the properties TO1, TO2, and TO3, even if processes crash. Fault-tolerance is achieved in MTO using three key elements: (1) a reliable multicast used to issue the message (*R-multicast*, see Fig. 3), (2) a consensus-like protocol, named *QM-unif-consensus (Qualified Majority Uniform Consensus*), used by a specific *MTO-SN* function, to compute the sequence number (Fig. 3), and (3) causal delivery of messages.

5.2 Implementation model

Before describing the MTO algorithm, we first introduce the key notions used by the algorithm.

Causal order delivery. We assume causal order delivery for all the messages sent in the system, and we denote $deliver_i(m)$ the delivery of message m at process p_i according to causal order. Given two events

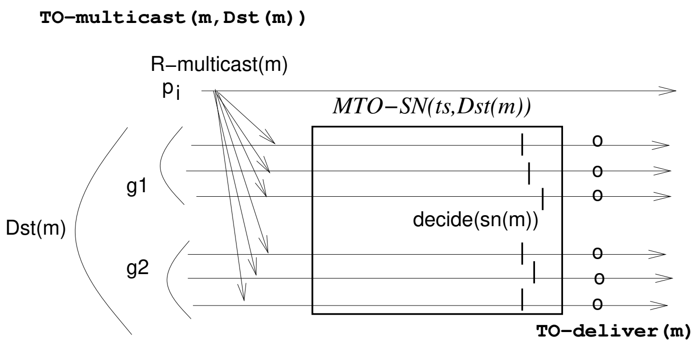

Figure 3: Principle of the MTO algorithm

e_1 and e_2, we denote $e_1 \rightarrow e_2$ the happened-before relation defined by Lamport [16]. Causal order delivery ensures that given two messages m and m' sent to a common destination p_i, if $send(m) \rightarrow send(m')$, then $deliver_i(m) \rightarrow deliver_i(m')$. We assume an implementation of causal order delivery as in [17]. Such implementation does not generate additional messages (it just leads to larger messages), and hence does not violate the Minimality property that we require from our MTO algorithm.

Reliable multicast. We assume the existence of a *reliable* multicast primitive, denoted *R-multicast(m,*$\mathsf{Dst}(m)$*)*, and the corresponding reliable delivery primitive, denoted *R-deliver(m)*. The primitive *R-multicast(m,*$\mathsf{Dst}(m)$*)* is defined by the following two properties: (1) if a process in $\mathsf{Dst}(m)$ (correct or not) has *R-delivered*(m), then every correct process in $\mathsf{Dst}(m)$ eventually *R-delivers*(m); (2) if a correct process *R-multicasts(m,*$\mathsf{Dst}(m)$*)*, then every correct process in $\mathsf{Dst}(m)$ eventually *R-delivers*(m) [13]. An example of a simple implementation of reliable multicast is given in [6]: *R-multicast(m,*$\mathsf{Dst}(m)$*)* is implemented by having each process in $\mathsf{Dst}(m)$ forwarding m to $\mathsf{Dst}(m)$ before delivering m [6]. We consider reliable multicast implemented on top of the causal communication layer defined above. Thus reliable multicast satisfies causal delivery order. Moreover, we assume that causal order delivery is also ensured between (1) messages issued with the *send* primitive, and (2) messages issued with the *R-multicast* primitive.

Qualified majority. Let f_i be the maximum number of processes of group g_i that can crash, and we assume $f_i < |g_i|/2$ for every group g_i (i.e., we assume a majority of correct processes in each group). Given *TO-multicast*$(m, \mathsf{Dst}(m))$, we define a *qualified majority* of $\mathsf{Dst}(m)$ as any subset of $\mathsf{Dst}(m)$ that contains a majority of processes of every group g_i in $\mathsf{Dst}(m)$. Notation: $QM(\mathsf{Dst}(m))$. As an example, if $\mathsf{Dst}(m) = \{g_1, g_2\}$, $g_1 = \{p_1, p_2, p_3\}$ and $g_2 = \{p_4, p_5, p_6\}$, then $\{p_1, p_2, p_4, p_5\}$ is a qualified majority of $\mathsf{Dst}(m)$, whereas $\{p_1, p_2, p_3, p_4\}$ is not.

QM-unif-consensus (QM = Qualified Majority). We assume the existence of a *QM-unif-consensus* function. This function solves a simple variation of the uniform consensus problem[7]. Consider uniform consensus to be solved among the set $\mathsf{Dst}(m)$ of groups of processes, and let $decide_i(v)$ be the event by which a process p_i *decides* on a value v. *We define a "QM-unif-consensus" algorithm to be a uniform consensus algorithm that satisfies the following additional "QM-causal" property: for every process p_i that decides v, there is a "qualified majority" of* $\mathsf{Dst}(m)$*, denoted QM(m), such that for every process $p_j \in QM(m)$, we have $deliver_j(\langle v \rangle) \rightarrow decide_i(v)$*[8]. A QM-unif-consensus algorithm is trivially obtained from the Chandra-Toueg uniform consensus algorithm using $\diamond\mathcal{S}$ [6]. The transformation consists in replacing every "majority" condition in the algorithm by a "qualified majority" condition.

Summary of notations. To summarize, the following notations are used in the paper: (1) *send(m)* and the corresponding *deliver(m)*, which satisfies causal order delivery, (2) *R-multicast*$(m, \mathsf{Dst}(m))$ and the corresponding *R-deliver(m)*, which also satisfies causal order delivery, and (3) *TO-multicast*$(m, \mathsf{Dst}(m))$ and the corresponding *TO-deliver(m)*.

5.3 Description of MTO

The MTO algorithm works as follows:

TO-multicast. When *TO-multicast*$(m, \mathsf{Dst}(m))$ is executed by a process p_i, the message m is *R-multicast* to $\mathsf{Dst}(m)$.

TO-delivery. The TO-delivery of m occurs as the result of the following sequence of actions:

1. Consider $p_j \in \mathsf{Dst}(m)$ and let $ts(R\text{-}deliver_j(m))$ be the timestamp of the $R\text{-}deliver_j(m)$ event.

[7] The uniform consensus problem differs from the consensus problem by requiring "Uniform" Agreement (all processes, *correct or not*, decide the same value) instead of Agreement (all *correct* processes decide the same value). The Chandra-Toueg algorithm, using a failure detector of class $\diamond\mathcal{S}$, solves the uniform consensus problem [6].

[8] We denote $\langle v \rangle$ a message carrying the value v.

```
1 function MTO-SN (ts_j, Dst(m))
    /*  ts_j = ts(R-deliver_j(m))  */

2       send timestamp ts_j to Dst(m) ;
3       wait until [delivered timestamps from
                    a qualified majority of Dst(m)] ;
4       prop_j ← max of all timestamps delivered ;
5       res_j ← QM-unif-consensus (prop_j, Dst(m)) ;
6       return res_j ;
```

Figure 4: Computing the sequence number $sn(m)$: code of process p_j

Process p_j stores the pair $(m, ts(R\text{-}deliver_j(m))$ in its *pending* buffer;

2. Process p_j then computes $sn(m)$ by calling the function *MTO-SN* with $ts(R\text{-}deliver_j(m))$ and $\mathsf{Dst}(m))$ as parameters. The function MTO-SN computing $sn(m)$ is given in Figure 4. At line 5, MTO-SN calls the *QM-unif-consensus* function to compute the sequence number $sn(m)$. When MTO-SN returns (line 6), $sn(m)$ is "*decided*" by p_j.

3. When the call to *MTO-SN* terminates, $sn(m)$ is known by p_j. From here on, MTO is identical to Skeen's algorithm. The pair $(m, ts(R\text{-}deliver_j(m))$ is removed from p_j's pending buffer and is inserted into p_j's TO-delivery queue, ordered according to $sn(m)$. The message m at the head of the TO-delivery queue is finally TO-delivered by p_j, if and only if there is no pair $(m', ts(R\text{-}deliver_j(m'))$ in p_j's pending buffer, such that $ts(R\text{-}deliver_j(m')) < sn(m)$.

5.4 Correctness of MTO

We sketch now the proof of correctness of MTO. The complete proofs can be found in [12].

Proposition TO1. *MTO satisfies the Uniform Agreement property (TO1).*

Proof (Sketch).
Consider *TO-multicast*$(m, \mathsf{Dst}(m))$ and a process $p_k \in \mathsf{Dst}(m)$ that has TO-delivered m. As p_k must have decided $sn(m)$ first, then a qualified majority of $\mathsf{Dst}(m)$ has R-delivered m. By the assumption of a qualified majority of correct processes, at least one correct process has R-delivered m. By the assumption of a reliable multicast, every correct process R-delivers m, and thus every correct process starts QM-unif-consensus (see Sect. 5.2). As we assume a qualified majority of correct processes and a failure detector of class $\diamond\mathcal{S}$, every correct process eventually decides $sn(m)$. We then show by induction on $x = sn(m)$ that every correct process eventually TO-delivers m. □

Proposition TO2. *MTO satisfies the Termination property (TO2).*

Proof (Sketch). Consider the execution of *TO-multicast*$(m, \mathsf{Dst}(m))$ by a correct process p_i. By the property of reliable multicast, every correct process R-delivers(m). Similarly to the proof of TO1, we then show that every correct process eventually TO-delivers m. □

Proposition TO3. *MTO satisfies the Uniform Global Order property TO3.*

Proof (Sketch). We show that messages are TO-delivered in the order defined by their sequence number. Consider TO-multicast$(m_1, \mathsf{Dst}(m_1))$, TO-multicast$(m_2, \mathsf{Dst}(m_2))$, and $sn(m_1) < sn(m_2)$. Using the property of Lamport's clocks and the QM-causal property of QM-unif-consensus, we first show that there is a process p_x in $\mathsf{Dst}(m_1) \cap \mathsf{Dst}(m_2)$, such that for every process p_i in $\mathsf{Dst}(m_1) \cap \mathsf{Dst}(m_2)$ which *TO-delivers* m_2, $R\text{-}deliver_x(m_1) \rightarrow TO\text{-}deliver_i(m_2)$. We then show, using the causal order message delivery assumption, that p_i R-delivers m_1 before TO-delivering m_2. Hence, when p_i TO-delivers m_2, m_1 is either (1) in p_i's pending buffer, or (2) in p_i's delivery buffer, or (3) already TO-delivered. As $sn(m_1) < sn(m_2)$, (1) and (2) are excluded by the TO-delivery condition of MTO (no message with smaller sequence number in the pending buffer or in the delivery queue, when m is delivered). Thus only case (3) is possible, i.e., p_i has already TO-delivered m_1 when it TO-delivers m_2. □

Proposition "Minimality". *MTO satisfies the Minimality property of a genuine multicast algorithm.*

Proof. We distinguish the two parts of MTO (Fig. 3): (1) the initial R-multicast, and (2) the MTO-SN function.
(1) Implementation of R-multicast$(m, \mathsf{Dst}(m))$ only involves the sender and the processes in $\mathsf{Dst}(m)$ [6].
(2) Uniform consensus among a set $\mathsf{Dst}(m)$, and thus QM-uniform consensus, can be solved involving only processes in $\mathsf{Dst}(m)$ [6]. Thus the function MTO-SN only involves processes in $\mathsf{Dst}(m)$.
Finally, implementation of causal order delivery does not introduce any new message [17]. Altogether, MTO satisfies the Minimality property of a genuine multicast algorithm. □

5.5 On the necessity of the assumptions

The correctness proofs of MTO are based on the following assumptions: (1) the failure detector is of class $\diamond\mathcal{S}$ [6], (2) in each group there is a majority of correct processes, and (3) causal order delivery is ensured. We discuss here the necessity of these assumptions.

It has been shown that, (i) failure detectors of class $\diamond\mathcal{S}$ are the weakest that allow solving consensus and total order broadcast problems [5], and (ii) consensus and total order broadcast problems [6] cannot be solved with failure detectors of class $\diamond\mathcal{S}$ if less than a majority of processes can crash. As one can implement total order broadcast using total order multicast, it is obvious that failure detectors of class $\diamond\mathcal{S}$ are the weakest that allow solving total order multicast. Furthermore, since the destination set $\mathsf{Dst}(m)$ of a message m can be a single group g_i, solving total order in g_i with $\diamond\mathcal{S}$ requires a majority of correct processes.

To illustrate the need for causal order delivery in MTO, consider *TO-multicast*$(m_1, \mathsf{Dst}(m_1))$, *TO-multicast*$(m_2, \mathsf{Dst}(m_2))$, and p_i in $\mathsf{Dst}(m_1) \cap \mathsf{Dst}(m_2)$. Assume that all processes in $\mathsf{Dst}(m_1) \cap \mathsf{Dst}(m_2)$, except some process p_i also in $\mathsf{Dst}(m_1) \cap \mathsf{Dst}(m_2)$, R-delivers m_1 before knowing about $sn(m_2)$, i.e., before delivering $\langle sn(m_2)\rangle$[9]. Without causal delivery, we can have $sn(m_1) < sn(m_2)$, and p_i TO-delivering m_2 before R-delivering m_1. In this case, either (1) p_i does never TO-deliver m_1 and the Uniform Agreement property TO1 is violated, or (2) p_i TO-delivers m_1 after m_2, and the Uniform Order property TO3 is violated.

6 Cost analysis

In this section, we analyse the cost of the MTO algorithm in terms of number of messages and number of communication steps needed to TO-deliver a message. We compare these numbers with those of the Chandra-Toueg total order broadcast algorithm. The two algorithms are comparable in that both are based on unreliable failure detectors of class $\diamond\mathcal{S}$ [6], and none relies on a membership service. We consider in our analysis only the best case scenario, i.e., runs with no failure and no failure suspicion. This is the most frequent case in practice.

Number of communication steps. [10] The MTO algorithm consists of (1) R-multicast of message m to $\mathsf{Dst}(m)$ (R-multicast in Fig. 3), then (2) for every process p_j, the sending of ts_j to $\mathsf{Dst}(m)$ (line 2 in Fig. 4), and finally (3) the consensus. Let C be the number of communication steps needed to solve the consensus problem in the best case scenario. The MTO algorithm requires thus $2 + C$ communication steps. By comparison, the Chandra-Toueg total order broadcast algorithm consists of (1) R-broadcast of message m, followed by (2) the consensus. This leads to $1 + C$ communication steps.

For the Chandra-Toueg consensus algorithm based on $\diamond\mathcal{S}$, we have $C = 3$, which leads to 4 communication steps for the Chandra-Toueg total order broadcast algorithm in the best case, and 5 for the MTO algorithm.

	Communication steps	Messages	Causal order
Chandra–Toueg (broadcast / unreliable FD)	4	$2(n-1)^2 + 2(n-1)$	no
MTO (multicast / unreliable FD)	5	$3(d-1)^2 + 4(d-1) + 1$	yes

Figure 5: Trade off between total order broadcast and (genuine) multicast ($d \leq n$)

Number of messages. Let $n = |\Omega|$ be the number of processes in the system, let m be a message TO-multicast to $\mathsf{Dst}(m)$ with $d = |\mathsf{Dst}(m)|$, and let M be the number of messages needed to solve the consensus problem in the best case scenario.

Consider first the MTO algorithm. If we assume that the initiator p_i of TO-multicast is not in $\mathsf{Dst}(m)$, then the initial R-multicast of m to $\mathsf{Dst}(m)$ costs $d + (d-1)^2$ messages (causal order delivery does not require any additional messages, but leads to larger messages [17]). For every process in $\mathsf{Dst}(m)$, sending ts_j to $\mathsf{Dst}(m)$ costs $d(d-1)$ messages. Thus MTO costs $M + 2(d-1)^2 + 2(d-1) + 1$ messages. By comparison, the Chandra-Toueg total order broadcast algorithm costs $(n-1)^2 + M$ messages (R-broadcast and consensus).

For Chandra-Toueg's consensus algorithm based on $\diamond\mathcal{S}$ among x processes, we have $M = 2(x-1) + (x-1)^2$ in the best case. This leads to a total of $3(d-1)^2 + 4(d-1) + 1$ messages for MTO in the best case, and $2(n-1)^2 + 2(n-1)$ messages for Chandra-Toueg's broadcast algorithm in the best case.

[9] $\langle sn(m_2)\rangle$ is a message carrying the value $sn(m_2)$.

[10] This measure is less ambiguous than the usual number of "phases". To give an example, the classical two phase commit protocol (2PC) has 3 communication steps [2]: (1) *vote request* sent from the coordinator to the participants, (2) *reply* of the participants sent to the coordinator, and (3) *decision* sent by the coordinator to the participants.

Trade-off. Figure 5 summarizes the numbers given above, for the case of a multicast sent to d processes in a system of n processes ($d \leq n$). All n processes take part to the protocol in the case of a broadcast, whereas only d processes take part to the protocol in the case of a multicast. The figure illustrates the interesting trade offs (1) between *broadcast* with 4 communication steps (which does not require causal order delivery), where all n processes take part to the protocol, and (2) *multicast* with 5 communication steps, where only the d processes in $\mathsf{Dst}(m)$ take part to the protocol ($d \leq n$), but with the cost of causal delivery (larger messages).

7 Concluding remarks

The paper has presented a fault-tolerant total order multicast algorithm, named MTO, that allows multicasting of messages to multiple groups of processes. MTO is a genuine multicast algorithm in the sense that, given *TO-multicast*$(m, \mathsf{Dst}(m))$, only the sender and the processes in $\mathsf{Dst}(m)$ take part in the protocol needed to TO-deliver m. The only previous genuine total order multicast algorithms that we know of, either do not tolerate failures (e.g [19]), or assume reliable failure detectors (e.g. [10, 14]).

MTO has a modular structure as it is based on a reduction to a consensus algorithm. This means that, given *TO-multicast*$(m, \mathsf{Dst}(m))$ and using the results established by Chandra-Toueg [6], MTO requires only (1) (unreliable) failure detectors of the class $\diamond\mathcal{S}$, and (2) within $\mathsf{Dst}(m)$, a qualified majority of correct processes. Interestingly, the correctness of MTO requires causal order delivery of messages. This does not mean that any total order genuine multicast algorithm, based on a failure detector of class $\diamond\mathcal{S}$, requires causal order delivery. We suspect, however, that causal order delivery is necessary, but this has still to be formally proved, and constitutes an interesting open question.

When considering performance issues, there is an interesting trade-off between MTO which is a total order "genuine multicast", and the Chandra-Toueg's total order "broadcast" algorithm (which also relies on failure detectors of class $\diamond\mathcal{S}$). Roughly speaking, MTO requires less (but larger) messages, and more communication steps.

Finally, whereas many published total order algorithms are based on a membership service, neither Chandra-Toueg's algorithm, nor MTO, require a membership service. Hence, there is no need with these algorithms to force incorrectly suspected processes to crash.

References

[1] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P.Ciarfella. The Totem Single-Ring Ordering and Membership Protocol. *ACM Trans. on Computer Systems*, 13(4):311–342, November 1995.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Distributed Database Systems.* Addison-Wesley, 1987.

[3] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, February 1987.

[4] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, August 1991.

[5] T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of ACM*, 43(4):685–722, 1996.

[6] T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2):225–267, 1996.

[7] J. M. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Trans. on Computer Systems*, 2(3):251–273, August 1984.

[8] D. Dolev, S. Kramer, and D. Malki. Early Delivery Totally Ordered Broadcast in Asynchronous Environments. In *IEEE 23rd Int Symp on Fault-Tolerant Computing (FTCS-23)*, pages 544–553, June 1993.

[9] P. Ezhilchelvan, R. Macedo, and S. Shrivastava. Newtop: A Fault-Tolerant Group Communication Protocol. In *IEEE 15th Intl. Conf. Distributed Computing Systems*, pages 296–306, May 1995.

[10] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Trans. on Computer Systems*, 9(3):242–271, August 1991.

[11] R. Guerraoui and A. Schiper. Atomic Multicast harder than Atomic Broadcast. Technical Report 96/193, EPFL, Dept d'Informatique, May 1996.

[12] R. Guerraoui and A. Schiper. Total Order Multicast to Multiple Groups. Technical Report 96/183, EPFL, Dept d'Informatique, April 1996.

[13] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[14] X. Jia. A Total Ordering Multicast Protocol Using Propagation Trees. *IEEE Trans. Parallel & Distributed Syst.*, 6(6):617–627, June 95.

[15] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[16] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, July 1978.

[17] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *3rd Intl. Workshop on Distributed Algorithms (WDAG-3)*, pages 219–232. Springer Verlag, LNCS 392, September 1989.

[18] A. Schiper and M. Raynal. From Group Communication to Transactions in Distributed Systems. *Comm. ACM*, 39(4):84–87, April 1996.

[19] D. Skeen. Unpublished communication. Feb 1985. Referenced in *K. Birman and T.Joseph. Reliable Communication in the Presence of Failures. ACM Trans. on Computer Systems, 5(1):47-76, February 1987.*