Atomic Multicast harder than Atomic Broadcast

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

Abstract

Atomic broadcast and atomic multicast are communication primitives which ensure that processes agree on the *sequence* of messages they deliver. Whereas atomic broadcast is used to send messages to *the set of all* the processes in the system, atomic multicast sends messages to specific *subsets* of processes. We say that an atomic multicast algorithm is *"genuine"*, if only the sender and the destination processes of a message take part in the protocol needed to deliver the message.

We formally define the genuine atomic multicast problem, and we show that this problem turns out to be harder than the atomic broadcast problem. Whereas atomic broadcast can be solved with unreliable failure detectors that can make false failure suspicions, we show that atomic multicast cannot.

Keywords. Atomic multicast, atomic broadcast, asynchronous system, crash failure, reliable channel, failure detector.

1 Introduction

The motivation of this work was to find out whether the results stated by Chandra and Toueg [1], on solving the *atomic broadcast* problem in distributed asynchronous systems with unreliable failure detectors [1], also apply to the *atomic multicast* problem. Whereas the two problems might appear to be equivalent, it turns out that atomic multicast is harder than atomic broadcast, as atomic multicast cannot be solved using unreliable failure detectors.

1.1 Atomic multicast vs atomic broadcast

An *atomic broadcast* primitive enables to send messages to all the processes in a system, with the guarantee that all correct processes (those that do not crash) agree on the *sequence* of messages they deliver. This primitive guarantees the agreement both on (1) the *set* of messages delivered, and (2) the *order* according to which the messages are delivered.

Contrary to a broadcast that is targeted to the set of all the processes in a system, a *multicast* can be targeted exclusively to a subset of the processes. Apart from this, similarly to atomic broadcast, atomic 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 atomic broadcast and atomic 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, an atomic multicast algorithm can directly be used to implement an atomic broadcast algorithm. Secondly, a naive atomic multicast algorithm can be obtained from any atomic broadcast algorithm as follows: consider a message m to be multicast to $\mathsf{Dst}(m)$: (1) m is broadcast, together with the information $\mathsf{Dst}(m)$, to all the processes in Ω; (2) a process $p_i \in \Omega$ only delivers m if $p_i \in \mathsf{Dst}(m)$.

The above transformation of atomic broadcast into atomic multicast, leads however to a *feigned* multicast algorithm. Indeed, with this transformation and for $\mathsf{Dst}(m) \subset \Omega$, a multicast

[1] We consider here reliable channels and process crash failures. We call unreliable failure detector, one that can make an infinite number of false failure suspicions.

is as costly as a broadcast, and 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 a specific *Minimality* property. Roughly speaking, this property requires that only the sender and the addressees of a message be involved in the protocol needed to deliver the message [2]. The naive implementation above using atomic 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"* atomic multicast algorithm to be an atomic multicast algorithm that satisfies the *Minimality* property.

1.3 Atomic multicast harder than atomic broadcast

In this paper, we show that genuine atomic multicast is harder than atomic broadcast. Indeed, whereas atomic broadcast can be solved with unreliable failure detectors [1] (which can make an infinite number of mistakes, i.e. false failure suspicions), we show that in a system with n processes ($n \geq 3$), among which one can crash, there exists no genuine atomic multicast algorithm using unreliable failure detectors. This impossibility result is not in contradiction with genuine atomic multicast algorithms that have been proposed in the literature. Either these algorithms assume reliable failure detection (e.g. [2]), or they assume failure free systems (e.g. [6]).

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 gives a precise definition of the notion of "genuine atomic multicast". In Section 4, we state and prove our impossibility result. We discuss in Section 5 different ways of circumventing the impossibility result. Finally, Section 6 summarizes the paper contributions.

2 Model and background

Our model of asynchronous computation with failure detection is similar to the one described in [1]. In the following, we recall some definitions that are needed to prove our result. Then we recall some results on solving atomic broadcast with unreliable failure detectors (also from [1]).

[2]Hence if a message m is multicast to $\mathsf{Dst}(m) \subset \Omega$, the complexity of the algorithm needed to deliver m depends on $|\mathsf{Dst}(m)|$ and not on $|\Omega|$.

2.1 Failures

A discrete global clock is assumed, and Φ, the range of the clock's ticks, is the set of natural numbers. Processes do not have access to the global clock. The distributed system consists of a set Ω of processes. Processes fail by *crashing*, and failures are permanent. A correct process is a process that does not crash. A *failure pattern* is a function F from Φ to 2^{Ω}, where $F(t)$ denotes the set of processes that have crashed through time t. We assume, as in [1], that in any failure pattern, there is at least one correct process. A *failure detector history* is a function from $\Omega \times \Phi$ to 2^{Ω}, where $H(p,t)$ denotes the set of processes suspected by process p at time t. A *failure detector* is a function $\mathcal{D}$ that maps each failure pattern F to a set of failure detector histories.

2.2 Algorithms

An *algorithm* is a collection A of n deterministic automata $A(p)$ (one per process p). In each step of an algorithm A, a process p performs atomically the following phases: (1) p receives a message from q, or a "null" message λ; (2) p queries and receives a value d from its failure detector module (d is said to be *seen* by p); (3) p changes its state and sends a message (possibly null) to some process. This third phase is performed according to (a) the automaton $A(p)$, (b) the state of p at the beginning of the step, (c) the message received in phase 1, and (d) the value d seen by p in phase 2. The message received by a process is chosen non-deterministically among the messages in the message buffer destined to p, and the null message λ. A *configuration* is a pair (I, M) where I is a function mapping each process p to its local state, and M is a set of messages currently in the message buffer. A configuration (I, M) is an initial configuration if $M = \emptyset$. A step of an algorithm A is a tuple $e = (p, m, d, A)$, uniquely defined by the algorithm A, the identity of the process p that takes the step, the message m received by p, and the failure detector value d seen by p during the step. A step $e = (p, m, d, A)$ is *applicable to a configuration* (I, M) if and only if $m \in M \cup \{\lambda\}$. The *unique* configuration that results from applying e to $C = (I, M)$, is noted $e(C)$.

2.3 Schedules and runs

A *schedule* of an algorithm A is a (possibly infinite) sequence $S = S[1]; S[2]; \ldots S[k]; \ldots$ of steps of A. A schedule is applicable to a configuration C if (1) S is the empty schedule, or (2) $S[1]$ is applicable to C, $S[2]$ is applicable to $S[1](C)$, etc. Given any schedule S, we note $P(S)$ the set of the processes that have at least one step in S.

A *partial run* of A using a failure detector $\mathcal{D}$, is a tuple $R =< F, H, C, S, T >$ where, F is a failure pattern, H is a failure detector history and $H \in \mathcal{D}(F)$, C is an initial configuration of A, T is a finite sequence of increasing time values, and S is a finite schedule of A such that: (1) $|S| = |T|$, (2) S is applicable to C, and (3) for all $i \leq |S|$ where $S[i] = (p, m, d, A)$, we have $p \notin F(T[i])$ and $d = H(p, T[i])$.

A *run* of an algorithm A using a failure detector $\mathcal{D}$, is a tuple $R =< F, H, C, S, T >$ where F is a failure pattern, H is a failure detector history and $H \in \mathcal{D}(F)$, C is an initial configuration of A, S is an infinite schedule of A, T is an infinite sequence of increasing time values, and in addition to the conditions above of a partial run ((1), (2) and (3)), the two following conditions are satisfied: (4) every correct process takes an infinite number of steps, (5) every message sent to a correct process q is eventually received by q.

Let $R =< F, H, C, S, T >$ be a partial run of some algorithm A. We say that $R' =< F', H', C', S', T' >$ is *an extension of* R, if R' is either a run or a partial run of A, and $F' = F$, $H' = H$, $C' = C$, $\forall i$ s.t. $T[1] \leq i \leq T[|T|]$, $S'[i] = S[i]$, $T'[i] = T[i]$.

2.4 Failure detector properties

Failure detectors are abstractly characterized by completeness and accuracy properties. We consider here the *Strong Completeness* property (introduced in [1]): eventually every process that crashes is permanently suspected by every correct process. The following accuracy properties are defined in [1]: (1) *Strong Accuracy*: no process is suspected before it crashes; (2) *Weak Accuracy*: some correct process is never suspected; (3) *Eventual Strong Accuracy*: eventually no correct process is suspected by any correct process, and (4) *Eventual Weak Accuracy*: eventually some correct process is never suspected by any correct process.

	Accuracy			
Completeness	**Strong**	**Weak**	**$\Diamond$Strong**	**$\Diamond$Weak**
Strong	*P (Perfect)*	*S (Strong)*	$\Diamond P$	$\Diamond S$

Figure 1: Failure detector classes

A failure detector class is a set of failure detectors defined by some accuracy and some completeness property. Figure 1 shows some of the notations introduced in [1]. For example, the class $\Diamond\mathcal{S}$ contains failure detectors that satisfy *Strong Completeness* and *Eventual Weak Accuracy* properties. The class $\mathcal{P}$ contains failure detectors that never make any mistake (false failure suspicion). That is, if a failure detector of class $\mathcal{P}$ suspects some process q, then q must have crashed. Such failure detectors are said to be *reliable*. An algorithm using a reliable failure detector can rely on the suspicions provided by the failure detector. The classes $\mathcal{S}$, $\diamond\mathcal{S}$, and $\diamond\mathcal{P}$, contain failure detectors that can make an infinite number of mistakes. An algorithm using an unreliable failure detector can never rely on the suspicions provided by the failure detector. In Section 4, we formally characterize algorithms that use unreliable failure detectors.

2.5 Solving atomic broadcast with unreliable failure detectors

Chandra and Toueg have shown that atomic broadcast algorithms that use unreliable failure detectors do exist [1]. More precisely, (1) if a majority or processes are correct, there is an atomic broadcast algorithm that can use any failure detector of class $\diamond\mathcal{S}$ [3], and (2) if at least one process is correct, there is an atomic broadcast algorithm that can use any failure detector of class $\mathcal{S}$.

In Section 4, we show that these results do not apply to genuine atomic multicast. More precisely, we define a specific unreliable failure detector which belongs to all the classes $\diamond\mathcal{P}$, $\diamond\mathcal{S}$ and $\mathcal{S}$, and we show that in a system with at least three processes among which at least one can crash, there exists no genuine atomic multicast algorithm which can use that failure detector.

3 Definition of genuine atomic multicast

In this section, we define what it means for an algorithm to be a "genuine atomic multicast" algorithm. We assume here that the state of each process contains, among other things, an output buffer, named *multicast-buffer*, and an input buffer, named *delivery-buffer*. The *multicast-buffer* contains a set of messages. A process p_i is said to *multicast* a message m, if p_i puts m in its *multicast-buffer*. A message m is uniquely identified, and it contains the identity of the process that multicasts m, noted $\mathsf{Snd}(m)$, as well as the set of processes to which m is multicast, noted $\mathsf{Dst}(m)$. The *delivery-buffer* contains a set of messages. A process p_j is said to *deliver* a message m, if p_j puts m in its *delivery-buffer*.

It is important to note here the distinction between the *"receive"* event and the *"deliver"* event (see Figure 2). As we will see below, atomic multicast is defined on the *"deliver"* event. For any initial state of a process p_i, we assume that the *delivery-buffer* of p_i is empty.

[3] As class $\diamond\mathcal{S}$ contains class $\diamond\mathcal{P}$, the result holds also for any failure detector of class $\diamond\mathcal{P}$.

For simplicity of presentation and without loss of generality, we assume that all messages multicast by p_i in some run R, are in the *multicast-buffer* of p_i in the initial state of R.

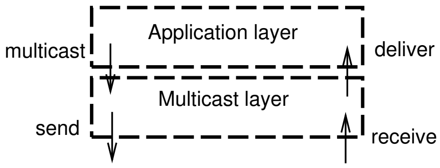

Figure 2: send-receive vs multicast-deliver

An algorithm A is a reliable multicast algorithm if for any run of A, the following properties are satisfied [3]:

- **Validity:** If a correct process multicasts a message m, then some correct process in $\mathsf{Dst}(m)$ eventually delivers m, or no process in $\mathsf{Dst}(m)$ delivers m.
- **Agreement:** If a correct process delivers a message m, then every correct process in $\mathsf{Dst}(m)$ eventually delivers m.

An atomic multicast algorithm A is a reliable multicast algorithm, such that for any run of A, the following additional property is satisfied:

- **Pairwise Total Order:** If correct processes p and q deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'. [4]

A genuine atomic multicast algorithm A is an atomic multicast algorithm, such that for any run R of A, the following additional property is satisfied:

- **Minimality:** For any process $p \in \Omega$, p does not send or receive any (non null) message in R, unless there exists in R a multicast of m, and $p \in \mathsf{Dst}(m) \cup \{\mathsf{Snd}(m)\}$ (i.e. p is either the sender of m or one of the addressees of m).

In a system with one process ($|\Omega| = 1$), genuine atomic multicast is trivially implemented using reliable channels. In a system with two processes ($|\Omega| = 2$), genuine atomic multicast can be implemented with reliable channels and atomic broadcast: whenever a message is multicast to one process, send it on the reliable channel, and whenever a message is multicast to two processes, use atomic broadcast. Therefore, we consider in the next section a system with at least three processes.

[4] In Section 5.5, we will come back to alternative definitions of atomic multicast, as presented in [3].

4 The impossibility result

We show here that in a system with at least three processes, among which one can crash, there can be no genuine atomic multicast algorithm using unreliable failure detectors. We first give an intuitive idea of the proof. Then we state a sequence of lemmas that are needed in the formal proof. Finally, we formally state our result and prove it.

4.1 The intuition

Informally, the idea behind the impossibility result is the following. Assume (by contradiction) that there is some genuine atomic multicast using an unreliable failure detector. Consider a message m multicast to a destination set $\mathsf{Dst}(m)$, a message m' multicast to a destination set $\mathsf{Dst}(m')$, and assume $\mathsf{Dst}(m) \cap \mathsf{Dst}(m') = \{q_1, q_2\}$. Roughly speaking, we build the proof by showing that we can build a partial run R in which no process crashes, but where the processes of $\mathsf{Dst}(m)$ think q_2 has crashed and then they deliver m, whereas the processes of $\mathsf{Dst}(m')$ think q_1 has crashed and then they deliver m'. As a consequence, q_1 delivers m and not m', whereas q_2 delivers m' and not m, violating the properties of atomic multicast. In the following we give an intuitive idea of how such a run R is built. The scenario is illustrated for $\mathsf{Dst}(m) = \{r_1, p_1, q_1, q_2\}$ and $\mathsf{Dst}(m') = \{q_1, q_2, p_2, r_2\}$.

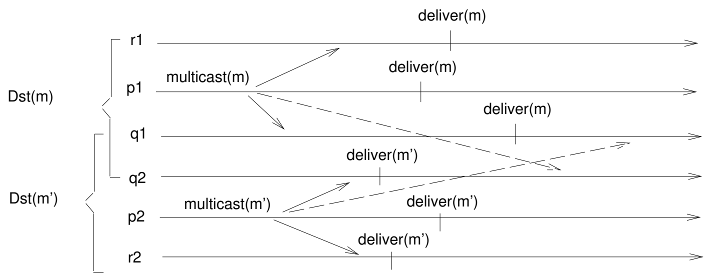

Figure 3: In Run R, q_1 delivers m and not m', whereas q_2 delivers m' and not m.

Building partial runs where q_2 does not deliver m. Consider the case where, except q_2 which initially crashes, all the processes of $\mathsf{Dst}(m)$ are correct. By the *Validity* property of atomic multicast, there is a partial run R_1 where, except q_2, the processes of $\mathsf{Dst}(m)$ (among which q_1) deliver m (see Fig. 4). Consider now a partial run R'_1 similar to R_1, except that q_2 is correct but extremely slow, and all other processes suspect q_2. Since the failure detector is unreliable, the processes cannot distinguish between the runs R_1 and R'_1. Hence, in R'_1, all the processes of $\mathsf{Dst}(m)$, except q_2, deliver m (see Fig. 4).

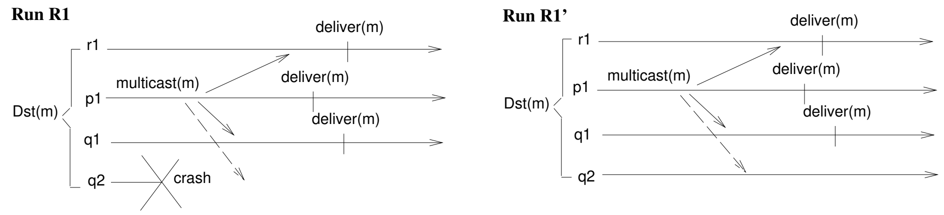

Figure 4: In runs R_1 and R_1', all but q_2 deliver m.

Building partial runs where q_1 does not deliver m'. Consider now the case where, except q_1 which initially crashes, all the processes of $\mathsf{Dst}(m')$ are correct. There is a partial run R_2 where, except q_1, the processes of $\mathsf{Dst}(m')$ (among which q_2) deliver m' (see Fig. 5). Consider a partial run R_2' similar to R_2, except that q_1 is correct but extremely slow, and all other processes suspect q_1. In R_2', all the processes of $\mathsf{Dst}(m')$, except q_1, deliver m' (see Fig. 5).

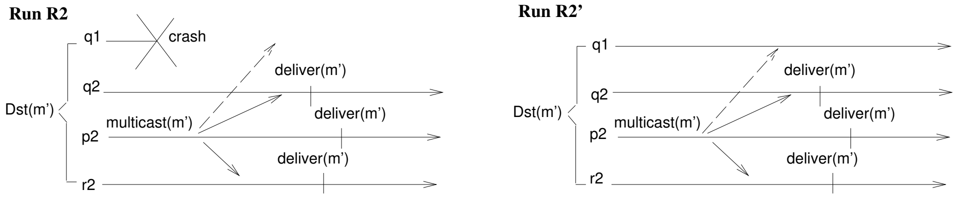

Figure 5: In runs R_2 and R_2', all but q_1 deliver m'.

Composing the partial runs. We show next that we can build a partial run R (where no process crashes, see Fig. 3), by composing R_1' and R_2', in such a way that in R, q_1 delivers m without delivering m', and q_2 delivers m' without delivering m. In R, either the *Pairwise Total Order* property or the *Agreement* property of atomic multicast is violated.

4.2 On Validity and Minimality

We introduce below a lemma which characterizes algorithms that satisfy the *Validity* and *Minimality* properties. Roughly speaking, this lemma states that any genuine atomic multicast algorithm allows partial runs such as R_1 in Figure 4, and R_2 in Figure 5.

Lemma 1. *Let A be any genuine atomic multicast algorithm using a failure detector $\mathcal{D}$. Let C be any initial configuration where exactly one message m is multicast by some process $p \in \mathsf{Dst}(m)$. Let $q \neq p$ be any process in $\mathsf{Dst}(m)$. Let F be any failure pattern where q initially crashes and all other processes are correct. For any failure detector history $H \in F(\mathcal{D})$, there* *is a partial run of A, $R = < F, H, C, S, T >$ such that, every process $p \in \mathsf{Dst}(m) - \{q\}$ delivers m, and no process $s \notin \mathsf{Dst}(m) - \{q\}$ takes any step in R.*

PROOF: In F, all processes are correct except q. By the *Validity* property of atomic multicast, for any failure detector history $H \in F(\mathcal{D})$, there is a partial run $\overline{R} = < F, H, C, \overline{S}, \overline{T} >$ of A, where every process in $\mathsf{Dst}(m) - \{q\}$ delivers m. As q has initially crashed then it has not taken any step. By the *Minimality* property of genuine atomic multicast, in $\overline{S}$ no process $s \notin \mathsf{Dst}(m)$ has sent or received any non null message.

Let S be the restriction of $\overline{S}$ to the events taken by the processes of $\mathsf{Dst}(m)$ (i.e S is obtained by removing from $\overline{S}$ the events taken by the processes that are not in $\mathsf{Dst}(m)$). Let $\overline{T}$ be the sequence of times corresponding to the events taken in $\overline{S}$ by the processes in $\mathsf{Dst}(m)$. As $\overline{S}$ is applicable to C, then S is also applicable to C. The partial run $R = < F, H, C, S, T >$ is thus a partial run of A and, in R, (1) every process $p \in \mathsf{Dst}(m) - \{q\}$ delivers m, and (2) no process $s \notin \mathsf{Dst}(m) - \{q\}$ takes any step in R. □

4.3 On algorithms using unreliable failure detectors

We introduce here a lemma which characterizes algorithms that use unreliable failure detectors. The lemma states, in terms of partial runs, the fact that an algorithm using an unreliable failure detector can never know whether a process has crashed or not. Roughly speaking, the lemma states that if an algorithm A allows a run such as R_1 in Figure 4 (resp. R_2 in Fig. 5), A also allows a run such as R'_1 in Figure 4 (resp. R'_2 in Fig. 5).

First we define a failure detector, noted $\mathcal{D}_\mathcal{D}$, which belongs to the classes of unreliable failure detectors $\mathcal{S}$, $\Diamond\mathcal{S}$ and $\Diamond\mathcal{P}$. The important characteristic of $\mathcal{D}_\mathcal{D}$ is that no algorithm using $\mathcal{D}_\mathcal{D}$ can ever know whether a process that is suspected has really crashed or not [5].

Failure detector $\mathcal{D}_\mathcal{D}$. We define $\mathcal{D}_\mathcal{D}$ as follows: for any failure pattern F, $\mathcal{D}_\mathcal{D}(F)$ is the set of all histories H such that, (1) eventually every process that crashes in F is permanently suspected by every correct process *(Strong Completeness)*, (2) at least some correct process in F is never suspected by any process *(Weak Accuracy)*, and (3) eventually no correct process in F is ever suspected *(Eventual Strong Accuracy)*. By construction, the failure detector $\mathcal{D}_\mathcal{D}$ belongs to $\mathcal{S}$, $\Diamond\mathcal{S}$ and $\Diamond\mathcal{P}$. As stated by the following lemma, an algorithm using $\mathcal{D}_\mathcal{D}$ can never know whether a process that is suspected has really crashed or not.

[5] $\mathcal{D}_\mathcal{D}$ stands for *Devil's Failure Detector*.

Lemma 2. *Let A be any algorithm using $\mathcal{D}_{\mathcal{D}}$. Let q be any process, and F_1 any failure pattern where q crashes. If $R_1 =< F_1, H_1, C, S, T >$ is a partial run of A, then there is a failure pattern F_1', which similar to F_1 except that q is correct in F_1', and a history $H_1' \in \mathcal{D}_{\mathcal{D}}(F_1')$, such that $R_1' =< F_1', H_1', C, S, T >$ is also a partial run of A.*

PROOF: The failure pattern F_1' is similar to F_1, except that q is correct in F_1'. Consider the failure detector history H_1' such that, (1) until time $T[|T|]$, H_1' is similar to H_1, (2) after time $T[|T|]$, no correct process is ever suspected, and (3) every process that crashes is permanently suspected by every correct process. The history H_1' satisfies both *Strong Completeness* and *Eventual Strong Accuracy* properties. Furthermore, as in H_1 at least some correct process p is never suspected, then in H_1' the same process p is also never suspected. Therefore, H_1' also satisfies *Weak Accuracy*. Altogether, we have $H_1' \in \mathcal{D}_{\mathcal{D}}$. As R_1 is a partial run of A, then R_1' is also a partial run of A. □

4.4 Concatenation of runs

The following lemma states that we can build a partial run out of two different partial runs, as long as both partial runs involve disjoint subsets of processes, and have the same failure pattern and failure detector history. Roughly speaking, this lemma states that we can build a partial run, such as R in Figure 3, out of two partial runs such as R_1' and R_2' (Fig. 4 and Fig. 5).

Notation. We consider below partial runs of the form $R_\alpha =< F, H, (I_\alpha[0], M_\alpha[0]), S_\alpha, T_\alpha >$; (I_α, M_α) denotes the sequence of configurations of R_α; $(I_\alpha[0], \emptyset)$ denotes an initial configuration (i.e. a set of initial states and an empty set of messages); $S_\alpha[1](I_\alpha[0], \emptyset) = (I_\alpha[1], M_\alpha[1])$, $S_\alpha[2](I_\alpha[1], M_\alpha[1]) = (I_\alpha[2], M_\alpha[2])$, etc; $P(S_\alpha)$ denotes the set of processes that have at least one step in S_α, and $I_\alpha|_{P(S_\alpha)}$ denotes the set of initial states of the processes in $P(S_\alpha)$. Given S_α (resp. T_α) and S_β (resp. T_β) two finite event sequences (resp. time sequences), we note $S_\alpha.S_\beta$ (resp. $T_\alpha.T_\beta$) the concatenation of S_α with S_β (resp. of T_α with T_β).

Lemma 3. *Let A be any algorithm. Let $R_\alpha =< F, H, (I_\alpha[0], \emptyset), S_\alpha, T_\alpha >$ and $R_\beta =< F, H, (I_\beta[0], \emptyset), S_\beta, T_\beta >$ be any two partial runs of A. If $P(S_\alpha) \cap P(S_\beta) = \emptyset$, and $T_\alpha[|T_\alpha|] < T_\beta[1]$, then $R =< F, H, (I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset), S_\alpha.S_\beta, T_\alpha.T_\beta >$, is also a partial run of A.*

PROOF: Let (I_α, M_α) be the sequence of configurations of R_α, and (I_β, M_β) the sequence of configurations of R_β. As R_α is a partial run of A, then S_α is applicable to $(I_\alpha[0], \emptyset)$. As no process in $P(S_\beta)$ takes any step in S_α, and S_α is applicable to $(I_\alpha[0], \emptyset)$, then S_α is also applicable to $(I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset)$. As no process of $P(S_\beta)$ takes any step in S_α, then $S_\alpha.S_\beta[|S_\alpha|]|_{P(S_\beta)} = I_\beta|_{P(S_\beta)}$, i.e. the processes of $P(S_\beta)$ keep the same state after S_α. As S_β is applicable to $(I_\beta, \emptyset)$, then S_β is also applicable to $(I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset)$. Therefore, R is a partial run of A. □

4.5 On Agreement and Pairwise Total Order

We introduce below a lemma which characterizes algorithms that satisfy the *Agreement* and *Pairwise Total Order* properties. Roughly speaking, this lemma states that a partial run such as R in Figure 3, is not acceptable for any atomic multicast algorithm.

Lemma 4. *Let A be any atomic multicast algorithm. Let R be any partial run of A where two messages m and m' are multicast. For any two correct processes $q_1, q_2 \in \mathsf{Dst}(m) \cap \mathsf{Dst}(m')$, if there is a time t_1 at which q_1 has delivered m but not m', then there is no time $t_2 > t_1$ at which q_2 has delivered m' but not m.*

PROOF: (BY CONTRADICTION) Assume that there is a time t_1 at which q_1 has delivered m but not m', and a time $t_2 > t_1$ at which q_2 has delivered m' but not m. Let R_∞ be any run which is an extension of R. By the *Agreement* property of atomic multicast, as q_1 and q_2 are correct, then, in R_∞, eventually q_1 delivers m', and eventually q_2 delivers m_1. Hence, in R_∞, there is a time at which q_1 and q_2 have delivered both m and m', but in a different order: in contradiction with the *Pairwise Total Order* property. □

4.6 The impossibility result

In this section, we state our impossibility result, and we prove it using the lemmas defined above. Roughly speaking, the proof shows that by combining Lemma 1, Lemma 2, and Lemma 3, a genuine atomic multicast algorithm can lead to a partial run such as R in Figure 3, in contradiction with Lemma 4. The proof is in the appendix.

Proposition 1. *In a system with at least three processes among which one can crash, there exists no genuine multicast algorithm using $\mathcal{D}_\mathcal{D}$.*

5 Circumventing the impossibility result

In this section we discuss five ways of circumventing the impossibility result: (1) by considering failure-free systems, (2) by considering reliable failure detectors, (3) by considering failure detectors that can only make a bounded number of mistakes, (4) by considering multicasts to subsets of process groups (rather than to subsets of individual processes), and (5) by considering a weaker definition af atomic multicast.

5.1 Failure free systems

Skeen has described in 1985 a genuine atomic multicast algorithm [6], assuming a failure free system (no process can crash). The principle of the algorithm is the following. When a process p multicasts a message m to $\mathsf{Dst}(m)$, p sends the message to every member of $\mathsf{Dst}(m)$. Every process $q \in \mathsf{Dst}(m)$ that receives m, stores m in a *pending buffer*, and sends back to p, a timestamp $ts_q(receive(m))$ corresponding to q's current logical clock [5]. Process p then collects the timestamps from all the processes in $\mathsf{Dst}(m)$, defines a sequence number $sn(m)$ as the maximun of the timestamps, and sends $sn(m)$ to every member of $\mathsf{Dst}(m)$. Every process $q \in \mathsf{Dst}(m)$ delivers m when there is no message in q's pending buffer for which $ts_q(receive(m')) < sn(m)$.

This algorithm is an atomic multicast algorithm, and it is a "genuine multicast" as only the sender p and the members of $\mathsf{Dst}(m)$ take part in the protocol needed to deliver m. This algorithm does obviously not tolerate even a single crash failure. If, for instance, one process in $\mathsf{Dst}(m)$ crashes while p is waiting for the timestamps, the algorithm blocks.

5.2 Reliable failure detectors

Skeen's algorithm can actually be transformed into a fault-tolerant algorithm by considering a decentralized way of determining the sequence number $sn(m)$, e.g. using a *consensus* protocol. The resulting algorithm works as follows. Instead of sending back the timestamp to p, every process $q \in \mathsf{Dst}(m)$ sends its timestamp to all the processes. Every process $q' \in \mathsf{Dst}(m)$ collects the timestamps from all the processes that q' does not suspect. Then q defines a temporary sequence number $sn_q(m)$ as the maximum of these timestamps, and q calls a consensus protocol with $sn_q(m)$ as initial value. The consensus protocol ensures that all the processes agree on the same sequence number $sn(m)$. This sequence number is used to deliver m, as in Skeen's original algorithm (see Sect. 5.1). If we assume a reliable failure detector of class $\mathcal{P}$, and a consensus protocol using such a failure detector (e.g. in [1]), the resulting algorithm is a genuine atomic multicast algorithm, and for every message m, it tolerates up to $|\mathsf{Dst}(m)| - 1$ crash failures.

Garcia-Molina and Spauster have introduced in [2] a genuine atomic multicast assuming a failure detector of class $\mathcal{P}$. The algorithm is based on a *propagation graph* built out of the destinations of every message. An optimisation of this algoritm is given in [4], also assuming a failure detector of class $\mathcal{P}$.

5.3 Failure detectors with a bounded number of mistakes

Up to now, we have considered two categories of failure detectors: the *unreliable* ones that can make an infinite number of mistakes, and the *reliable* ones that make no mistakes. We consider here a third category of failure detectors that only make a bounded number of mistakes. We address the question: can we solve genuine atomic multicast with such failure detectors ?

The answer is *"no"* with failure detectors that can make at least *two* mistakes. The intuitive idea of the proof is the following. Consider a message m multicast by a correct process p_1, a message m' multicast by a correct process p_2, and $\mathsf{Dst}(m) = \mathsf{Dst}(m') = \{p_1, p_2\}$. Assume furthermore a failure detector such that p_1 suspects p_2, whereas p_2 suspects p_1 (two mistakes). To avoid infinite blocking (i.e. to satisfy the *Validity* property), p_1 might deliver m without knowing the existence of m', and p_2 might deliver m' without knowing the existence of m: in contradiction with Lemma 1.

It is however possible to solve genuine atomic multicast with failure detectors that can make at most *one* mistake. The intuitive reason why the impossibility result does not apply here is that, in order to violate the properties of atomic multicast, two processes must deliver messages in a different order. These two processes must falsely suspect each others. Hence, two mistakes are necessary. [6]

5.4 Atomic multicast to multiple groups

According to our model, a process can multicast a message m to any subset of processes. In [7], we restrict this model by considering multicasts to subsets of process groups (rather

[6] The algorithm based on Skeen's principle (Sec. 5.2) can easily be transformed to suit a failure detector that makes at most one mistake, as long as this failure detector satisfies the *Strong Completeness* property, and the *Repentance* property, i.e. if a correct process p knows that q is correct, p removes q from its list of suspected processes. The class of such failure detectors is noted $\mathcal{SF}(1)$ in [1].

than to subsets of individual processes), and we present an algorithm based on (1) unreliable failure detectors of class $\diamond\mathcal{S}$, (2) the assumptions that there is a majority of correct processes within each group, and (3) causally ordered messages. This model corresponds to the practical situations where processes are used to manage replicated objects, and each group of processes represents a logical replica.

In fact, having a majority of correct processes within each group, leads to a model that is somehow similar to a failure free model. The logical destinations of messages are groups, of which a majority of processes are always correct. The algorithm we propose in [7] can be seen as an extension of Skeen's original algorithm (Sect. 5.1).

5.5 Atomic multicast with Local Total Order

Our impossibility result obviously applies to stronger definitions of atomic multicast presented in [3]. Among these definitions are those which consider *Uniform* versions of the *Agreement* and *Pairwise Total Order* properties, or which consider the *Global Total Order* property.

Hadzilacos and Toueg have also defined a property, called *Local Total Order*, that is weaker than the *Pairwise Total Order* property we have considered. The *Local Total Order* property ensures that messages sent to the same destination set are totally ordered, but no guarantee is however provided for messages in intersecting sets. An interesting question is whether a definition of atomic multicast based on *Local Total Order* impacts on our impossibility result. The answer is *"yes"*, and we show below that with this definition, atomic multicast and atomic broadcast are equivalent problems, and thus all results on solving atomic broadcast with unreliable failure detectors also apply to atomic multicast.

Indeed, remember that atomic broadcast is used to send messages to all the processes in a given system, ensuring the properties *Validity*, *Agreement*, and *Total Order* [3]. Let *broadcast* and *b-deliver* be the two primitives that define atomic broadcast. Consider a message m to be multicast to $\mathsf{Dst}(m)$: (1) broadcast m to the system $\{p\} \cup \mathsf{Dst}(m)$, (2) when process $q \in \mathsf{Dst}(m)$ b-delivers m, q immediatly delivers m. This means that we can transform atomic broadcast into atomic multicast. As the opposite is trivial, the two problems are thus equivalent.

To summarize, our definition of atomic multicast is, among those defined in [3], the weakest for which the impossibility result holds. This is exactly why we have considered that definition.

6 Summary

This paper has two main contributions. First, it gives a precise definition of a genuine atomic multicast. This leads to a better understanding of the difference between atomic multicast and atomic broadcast, and we can distinguish between genuine atomic multicast algorithms (e.g [2, 4]), and not genuine atomic multicast algorithms (e.g [5]).

Second, the paper shows that genuine atomic multicast cannot be solved with unreliable failure detectors, in a system with at least three processes among which one can crash. As atomic broadcast can be solved with unreliable failure detectors, the impossibility result means that genuine atomic multicast is harder than atomic broadcast [7].

References

[1] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. To appear in Journal of the ACM (1996). A preliminary version appeared in the *Proc of the 10th ACM Symp on Principles of Distributed Computing*, pages 325-340, ACM Press, 1991.

[2] G. Molina and A. Spauster. Ordered and Reliable Multicast Communication. ACM Transactions on Computer Systems, 9(3), pages 242-271, August 1991.

[3] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In Distributed Systems, Sape Mullender Editor, ACM Press Publisher, pages 97-145, 1993.

[4] X. Jia. A Total Ordering Multicast Protocol Using Propagation Trees. IEEE Transactions on Parallel and Distributed Systems, 6(6), pages 617-627, June 1995.

[5] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), pages 558-565, July 1978.

[6] D. Skeen. Unpublished communication, February 1985.

[7] A. Schiper and R. Guerraoui. Total Order Multicast to Multiple Groups. Technical Report, Computer Science Departement, EPFL, 1996.

[7] As consensus and atomic broadcast are equivalent problems [1], a corollary of our result is that atomic multicast is harder than consensus.

Appendix: impossibility result

Proposition 1. *In a system with at least three processes among which one can crash, there exists no genuine multicast algorithm using $\mathcal{D}_{\mathcal{D}}$.*

PROOF: (BY CONTRADICTION) Consider p, q_1 and q_2, three processes in the system. Assume a genuine atomic multicast algorithm A using the failure detector $\mathcal{D}_{\mathcal{D}}$. Let m be a message multicast by some process in $\mathsf{Dst}(m)$, m' a message multicast by some process in $\mathsf{Dst}(m')$, and $\mathsf{Dst}(m) \cap \mathsf{Dst}(m') = \{q_1, q_2\}$.

Let $(I_1[0], \emptyset)$ be some initial configuration where only the message m is multicast. Let F_1 be a failure pattern where q_2 initially crashes and all other processes are correct. Let $H_1 \in \mathcal{D}_{\mathcal{D}}(F_1)$ be a failure detector history where p is never suspected. By Lemma 1, there is a partial run of A, $R_1 =< F_1, H_1, (I_1[0], \emptyset), S_1, T_1 >$ such that the processes in $\mathsf{Dst}(m) - \{q_2\}$ deliver m, and the processes not in $\mathsf{Dst}(m) - \{q_2\}$ take no step. Hence in R_1, q_1 delivers m whereas q_2 does not.

Let $(I_2[0], \emptyset)$ be some initial configuration where only the message m' is multicast. Let F_2 be a failure pattern where q_1 initially crashes and all other processes are correct. Let $H_2 \in \mathcal{D}_{\mathcal{D}}(F_2)$ be a failure detector history where p is never suspected. By Lemma 1, there is a partial run of A, $R_2 =< F_2, H_2, (I_2[0], \emptyset), S_2, T_2 >$ such that, the processes in $\mathsf{Dst}(m) - \{q_1\}$ deliver m', and the processes not in $\mathsf{Dst}(m) - \{q_1\}$ take no step. Hence in R_2, q_2 delivers m' whereas q_2 does not.

Let F be the failure pattern where all processes are correct. By Lemma 2, there is a failure detector history $H'_1 \in \mathcal{D}_{\mathcal{D}}(F)$ and a failure detector history $H'_2 \in \mathcal{D}_{\mathcal{D}}(F)$, such that $R'_1 =< F, H'_1, (I_1[0], \emptyset), S_1, T_1 >$ and $R'_2 =< F, H'_2, (I_2[0], \emptyset), S_2, T_2 >$ are partial runs of A.

Let T'_2 be the time sequence defined as follows: $|T'_2| = |T_2|$, and $\forall i$ s.t. $1 \leq i \leq |T'_2|$, $T'_2[i] = T_2[i] + T_1[|T_1|]$, i.e. T'_2 is the time sequence with the same length as T_2, but which starts immediatly after T_1.

Let H be the failure detector history defined as follows: $\forall p \in \Omega$, $\forall t$ s.t. $1 \leq t \leq T_1[|T_1|]$, $H(p,t) = H_1(p,t)$, $\forall t$ s.t. $T_1[|T_1|] < t \leq T'_2[|T'_2|]$, $H(p,t) = H_2(p,t)$, $\forall t$ s.t. $T'_2[|T'_2|] < t$, $H(p,t) = \emptyset$, i.e. H is similar to H_1 until the end of $T_1[|T_1|]$, H is similar to H_2 from $T_1[|T_1|]$ to $T'_2[|T'_2|]$, then from $T'_2[|T'_2|]$ on, no process is ever suspected in H. In H, p is never suspected, and there is a time after which no process is suspected. Therefore $H \in \mathcal{D}_{\mathcal{D}}(F)$.

Consider $R_\alpha =< F, H, (I_1[0], \emptyset), S_1, T_1 >$ derived from R'_1, and $R_\beta =< F, H, (I_2[0], \emptyset), S_2, T'_2 >$

derived from R'_2. As R'_1 and R'_2 are partial runs of A, then R_α and R_β are also partial runs of A. Furthermore, as $\mathsf{Dst}(m) \cap \mathsf{Dst}(m') = \{q_1, q_2\}$, $P(S_1) = \mathsf{Dst}(m) - \{q_2\}$, and $P(S_2) = \mathsf{Dst}(m') - \{q_1\}$, we have $P(S_1) \cap P(S_2) = \emptyset$. As we also have $T_1[|T_1|] < T'_2[1]$, by Lemma 3, $R =< F, H, (I_1[0]|_{P(S_1)} \cup I_2[0]|_{P(S_2)}, \emptyset), S_1.S_2, T_1.T'_2 >$, is also a partial run of A.

In R, q_1 delivers m but not m' whereas q_2 delivers m' but not m, and $q_1, q_2 \in \mathsf{Dst}(m) \cap \mathsf{Dst}(m')$: in contradiction with Lemma 4. $\square$