Genuine Atomic Multicast

Rachid Guerraoui André Schiper

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland

Abstract. This paper addresses the problem of atomic *multicasting* messages in asynchronous systems with unreliable failure detection. First, we give a characterisation of the notion of algorithms *tolerating unreliable failure detection*. Second, we give a characterisation of the notion of *genuine atomic multicast*. This characterisation leads to a better understanding of the difference between atomic multicast and atomic broadcast, and to a clear distinction between genuine atomic multicast algorithms and non-genuine atomic multicast algorithms. Third, we show that in a system with at least three processes, the genuine atomic multicast problem cannot be solved with unreliable failure detection, if at least one process may crash. We mention however two ways of circumventing this impossibility result.

1 Introduction

The motivation of this work was to better understand the characteristics of the atomic multicast problem, and in particular to find out whether, as for consensus and atomic broadcast problems [1], the atomic multicast problem can be solved with unreliable failure detectors.

1.1 Atomic multicast versus 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 provides 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 addresses of every message agree either to deliver or not the message, and (2) no two correct processes deliver any two messages in different order.

1.2 Genuine multicast vs feigned multicast

At first glance, it might seem that atomic broadcast and atomic multicast are similar 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, because for $\mathsf{Dst}(m) \subset \Omega$, all the processes in the system are involved in the algorithm, *even those that are not concerned with the message* m, i.e., a multicast to a small subset turns out to be as costly as a broadcast (to all).

We require from a *genuine* multicast algorithm that it satisfies a specific *minimality* property. Roughly speaking, this property reflects the scalability of the algorithm, by requiring that only the sender and the addresses of a message be involved in the protocol needed to deliver the message. 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.

1.3 Impossibility result

We consider in this paper a distributed system with process crash failures, reliable channels but unreliable failure detection (one can never know whether a process has crashed or not). We show that, in a system with at least three processes, among which one can crash, there exists no genuine atomic multicast algorithm tolerating unreliable failure detection. This impossibility result explains why, among the atomic multicast algorithms proposed in the literature, those that turn out to be genuine are either not fault-tolerant or are based on reliable failure detection ([2, 5]). A corollary of this result is that genuine atomic multicast is strictly harder than atomic broadcast, as the latter can be solved with unreliable failure detection [1].

Our impossibility result is somehow frustrating because the minimality property that makes an atomic multicast algorithm scalable (and hence well suited for large scale distributed systems), is precisely the property that makes the problem impossible to solve with unreliable failure detection (which is the typical case in large scale systems). We point out two ways of circumventing our impossibility result: (1) by weakening the definition of atomic multicast, and (2) by considering a stronger system model where messages are always multicast to sets of process groups (rather than to sets of individual processes).

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 characterizes algorithms that tolerate unreliable failure detection. Section 4 defines the notion of *genuine atomic multicast*. Section 5 gives an intuitive idea of the impossibility result and Section 6 presents its proof. Section 7 discusses two ways of circumventing this impossibility result and Section 8 summarises the contributions of the paper.

2 Model

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.

2.1 Processes and 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 actions: (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 action 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 action 1, and (d) the value d seen by p in action 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 S 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. $T[1] \leq i \leq T[|T|]$, $S'[i] = S[i]$, $T'[i] = T[i]$.

3 On the use of unreliable failure detectors

3.1 Failure detector properties

Failure detectors are abstractly characterized by completeness and accuracy properties. The following *strong completeness* property was introduced in [1]: eventually every process that crashes is permanently suspected by every correct process. The following accuracy properties were 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. A failure detector class is a set of failure detectors defined by some accuracy and some completeness property. An algorithm A is said to solve a problem P with a failure detector class $\mathcal{C}$, if A solves P using any failure detector of class $\mathcal{C}$. Figure 1 shows some of the notations of the failure detector classes introduced in [1].

Among failure detectors, some never make any false failure detection. These are failure detectors of class $\mathcal{P}$ (defined by *strong completeness* and *strong accuracy* properties). 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*. Intuitively,

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

Fig. 1. Failure detector classes

an unreliable failure detector is a failure detector which does not distinguish a crashed process from a correct one. Except class $\mathcal{P}$, all failure detector classes in Figure 1 contain unreliable failure detectors. In the following, we define what it means for an algorithm to tolerate unreliable failure detection.

3.2 Algorithms tolerating unreliable failure detection

Intuitively, an algorithm that uses an unreliable failure detector can never know whether a process has crashed or not. Hence, such algorithm *tolerates* unreliable failure detection. We characterise such algorithms with the following property. We say that an algorithm A using a failure detector D, *tolerates* unreliable failure detection if A satisfies the following property.

- **Unreliable failure detection tolerance.** *For every failure pattern F_1 where q crashes, for every a partial run $R_1 =< F_1, H_1, C, S, T >$ of A, there is a failure pattern F_1', which is similar to F_1 except that q is correct in F_1', and a failure detector history $H_1' \in \mathcal{D}(F_1')$, such that $R_1' =< F_1', H_1', C, S, T >$ is also a partial run of A.*

If a process q is suspected of having crashed, the A algorithm cannot distinguish the situation where q has indeed crashed (failure pattern F_1), from the situation where q has not crashed (failure pattern F_1'). Any algorithm that solves a problem with failure detector classes $\mathcal{S}$, $\diamond\mathcal{S}$ or $\diamond\mathcal{P}$ (Figure 1) tolerates unreliable failure detection (i.e., satisfies the unreliable failure detection property above).

4 Definition of genuine atomic multicast

4.1 On *multicast* and *delivery*

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*. Each of these buffers contain a (possibly empty) set of messages. A process p_i is said to *multicast* a message m, if p_i puts m in its *multicast-buffer*. A process p_j is said to *deliver* a message m, if p_j puts m in its *delivery-buffer*. Every 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 $\mathsf{Dst}(m)$. It is important to notice here the distinction between the *receive* event and the *deliver* event. As we will see below, atomic multicast is defined on the *deliver* event (see Figure 2).

For any initial state of a process p_i, we assume that the *delivery-buffer* of p_i is empty. 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.

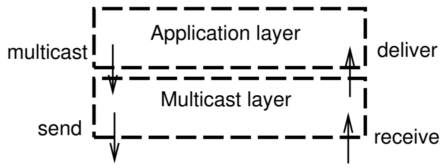

Fig. 2. send-receive vs multicast-deliver

4.2 Reliable multicast

An algorithm A is a *reliable multicast* algorithm, if for any run of A, the following properties are satisfied [4]:

- **Validity:** If a correct process multicasts a message m, then some correct process in $\mathsf{Dst}(m)$ eventually delivers m.
- **Agreement:** If a correct process delivers a message m, then every correct process in $\mathsf{Dst}(m)$ eventually delivers m.

4.3 Atomic multicast

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'. [1]

4.4 Genuine atomic multicast

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:

[1] In Section 6, we will come back to alternative definitions of atomic multicast, as presented in [4].

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

The minimality property reflects the scalability of a genuine multicast algorithm. It is important to notice that this property applies to the atomic multicast algorithm layer and not to the underlying network. Typically, nothing would prevent the underlying network from making a message m transit through nodes (routers) whose processes are not the addresses of the message m (this would not violate our minimality property). The routing issue is related to the network topology, and not to the minimality property of a genuine multicast algorithm.

4.5 Genuine atomic multicast and failure detection

The following algorithm is a simple example of a genuine atomic multicast algorithm. The principle of this algorithm, generally attributed to Skeen, 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 [6]. Process p then collects the timestamps from all the processes in $\mathsf{Dst}(m)$, defines a sequence number $sn(m)$ as the maximum 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 $m' \neq m$ 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. The algorithm does not however tolerate even a single crash failure: if one process in $\mathsf{Dst}(m)$ crashes while p is waiting for the timestamps, the algorithm blocks. The algorithm can easily be transformed to tolerate failures, but would require reliable failure detection. In the following, we address the following question: can we find a genuine atomic multicast algorithm that tolerates unreliable failure detection ?

In a system with one process ($|\Omega| = 1$), genuine atomic multicast is trivially implemented using reliable channels (whatever the failure detection is). In a system with two processes ($|\Omega| = 2$), genuine atomic multicast can be implemented with reliable channels and atomic broadcast (hence with unreliable failure detection, see [1]) as follows: 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. In the next sections, we assume a system with at least three processes ($|\Omega| > 2$) among which one can crash, and we show that there exists no genuine multicast algorithm that tolerates unreliable failure detection.

5 The impossibility result: sketch

We give here an overview of the result and an intuitive idea of its proof. The complete proof is given in Section 6.

5.1 Atomic multicast versus atomic broadcast

In [1], Chandra and Toueg have shown that in a system Ω, (1) atomic broadcast can be solved with any failure detector of class $\diamond\mathcal{P}$ iff a majority of processes in Ω are correct, and (2) atomic broadcast can be solved with any failure detector of class $\mathcal{S}$ if at least some process in Ω is correct.

One can intuitively see that these results do not apply to genuine atomic multicast. Consider result (1) and a subset Π of Ω within which the assumption of a majority of correct processes does not hold. In executions where messages are only multicast by processes of Π to Π, solving the genuine atomic multicast problem turns out to solve the atomic broadcast problem inside Π, which is impossible (as the assumption of a majority of correct processes does not hold in Π). Consider now result (2). If the failure detector in Ω is of class $\mathcal{S}$, it guarantees *weak accuracy*, i.e., some correct process p is not suspected by any process. In fact, this process p is actually the process that enables to ensure the total order property of atomic broadcast [1]. Consider a subset Π of Ω within which the assumption of *weak accuracy* does not hold, i.e., the process p is outside Π. In executions where messages are only multicast by processes of Π to Π, p will not be allowed to participate in the protocol, and hence no total order will be reached.

In the following, not only we formally confirm that results (1) and (2) do not apply to the genuine atomic multicast problem, but furthermore, we show that no algorithm tolerating unreliable failure detection solves the genuine atomic multicast problem (in a system with three processes, among which one can crash).

5.2 Overview of the proof

Assume (by contradiction) that there is some genuine atomic multicast tolerating unreliable failure detection. 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 $\mathsf{Dst}(m) \cap \mathsf{Dst}(m') = \{q_1, q_2\}$. 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 that q_2 has crashed and then they deliver m (1), whereas the processes of $\mathsf{Dst}(m')$ think that q_1 has crashed and then they deliver m' (2). As a consequence, q_1 delivers m and not m', whereas q_2 delivers m' and not m, violating the properties of atomic multicast (3). 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 $\mathsf{Dst}(m') = \{q_1, q_2, p_2, r_2\}$.

(1) Building a partial run where q_2 does not deliver m. Consider the case where all the processes of $\mathsf{Dst}(m)$ are correct, except q_2 which initially crashes. By

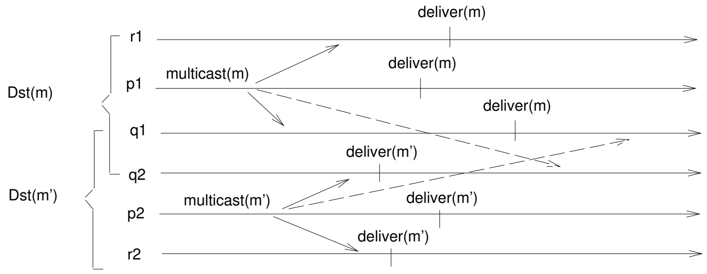

Fig. 3. In Run R, q_1 delivers m and not m', whereas q_2 delivers m' and not m.

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 (e.g., q_2 is just extremely slow). Since the failure detection is unreliable, the processes cannot distinguish run R_1 from run R_1' (Sect. 2.5.). Hence, in R_1', all the processes of $\mathsf{Dst}(m)$, except q_2, deliver m (see Fig. 4).

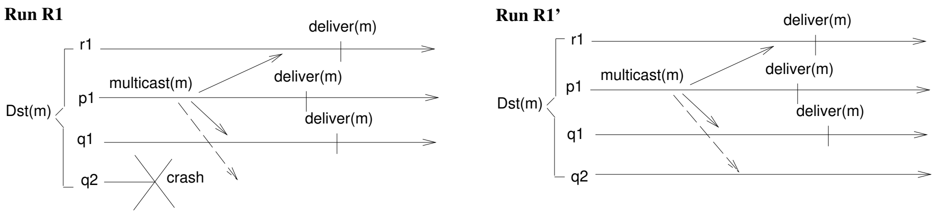

Fig. 4. In runs R_1 and R_1', all but q_2 deliver m.

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

(3) Composing the partial runs. We build a partial run R (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: a contradiction with atomic multicast.

6 The impossibility result: proof

In the following, we state the lemmas that are used in the proof, and we formally state the impossibility result.

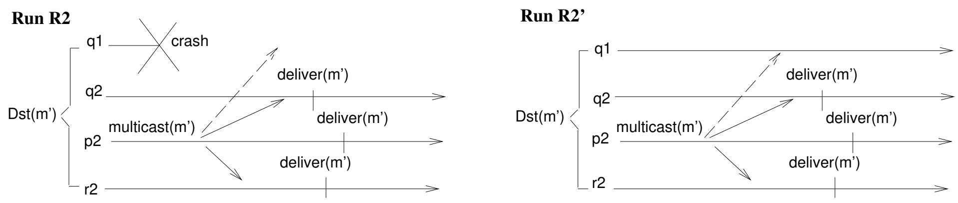

Fig. 5. In runs R_2 and R_2', all but q_1 deliver m'.

6.1 On validity and minimality

Lemma 1 below characterises algorithms that satisfy the validity and minimality properties of atomic multicast. 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 any 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 $r \in \mathsf{Dst}(m) - \{q\}$ 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 $r \in \mathsf{Dst}(m) - \{q\}$ delivers m, and (2) no process $s \notin \mathsf{Dst}(m) - \{q\}$ takes any step in R. $\square$

6.2 On concatenation of runs

Lemma 2 below 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 2. *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 any algorithm 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 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 $(I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset)$. Hence, R is a partial run of A. □

6.3 On agreement and pairwise total order

Lemma 3 below characterises 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 3. *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 $t_2 > t_1$ such that at t_2, 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. □

6.4 The result

Proposition 1 below states our impossibility result. To prove this result, we show that by using the characterisation of an algorithm tolerating unreliable failure detection (Sect. 3.2), and by combining Lemma 1 and Lemma 2 above, we build a partial run such as R in Figure 3, in contradiction with Lemma 3.

Proposition 1. *In a system with at least three processes among which one can crash, there exists no genuine multicast algorithm tolerating unreliable failure detection.*

PROOF: (BY CONTRADICTION) Consider p, q_1 and q_2, three processes in the system. Assume a genuine atomic multicast algorithm A tolerating unreliable failure detection. 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, and $H_1 \in \mathcal{D}(F_1)$. 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 no process outside $\mathsf{Dst}(m) - \{q_2\}$ takes any 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, and $H_2 \in \mathcal{D}(F_2)$. 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 no process outside $\mathsf{Dst}(m) - \{q_1\}$ takes any 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. As we assume an algorithm that tolerate unreliable failure detection, there is a failure detector history $H_1' \in \mathcal{D}(F)$ and a failure detector history $H_2' \in \mathcal{D}(F)$, 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 $H \in \mathcal{D}(F)$.

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 2,
$R =< F, H, (I_1[0]|_{P(S_1)} \cup I_2[0]|_{P(S_2)}, \emptyset), S_1.S_2, T_1.T_2' >$, 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 3. □

A simple corollary of our result is that no algorithm using failure detector classes $\mathcal{S}$ or $\diamond\mathcal{P}$ can solve the genuine atomic multicast problem. As atomic broadcast was shown to be solvable with those classes [1], a consequence of our result is that genuine atomic multicast is strictly harder than atomic broadcast (and hence strictly harder than consensus, as consensus and atomic broadcast are equivalent problems [1]).

7 Circumventing the impossibility result

7.1 Weakening the definition of atomic multicast

Our impossibility result also applies to stronger definitions of atomic multicast presented in [4]. 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. There is however 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. With this definition, atomic multicast and atomic broadcast are equivalent problems, and thus all results on solving atomic broadcast with unreliable failure detection also apply to atomic multicast. In fact, our definition of atomic multicast is, among those defined in [4], the weakest for which the impossibility result holds. This is exactly why we have considered that definition to state our result.

7.2 Atomic multicast to multiple groups

According to our model, a process can multicast a message m to any subset of processes. In [3], we restrict this model by considering multicasts to subsets of process groups (rather 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 actually corresponds to the practical situations where processes are used to manage replicated objects, and each group of processes represents a logical replica.

8 Summary

This paper has three contributions. First, we define what it means for an algorithm to tolerate unreliable failure detection. Second, we characterise the notion of 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, and feigned genuine atomic multicast algorithms.

Third, we show that genuine atomic multicast cannot be solved with an algorithm that tolerates unreliable failure detection, in a system with at least three processes among which one can crash. As atomic broadcast can be solved with unreliable failure detection, the impossibility result means that genuine atomic multicast is strictly harder than atomic broadcast. We mention two ways of circumventing our impossibility result: (1) by weakening the definition of atomic multicast, and (2) by strengthening the underlying system model.

Acknowledgement

We are very grateful to Vassos Hadzilacos and Bernadette Charron-Bost for their helpful comments on earlier drafts of this paper.

References

1. T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2), pages 225-267, 1996.
2. H. Garcia Molina and A. Spauster. Ordered and Reliable Multicast Communication. ACM Transactions on Computer Systems, 9(3), pages 242-271, August 1991.
3. R. Guerraoui and A. Schiper. Total Order Multicast to Multiple Groups. Proceedings of the 17th IEEE Int. Conf. on Distributed Computing Systems, Baltimore, pages: 578-585, May 1997.
4. V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. In Distributed Systems, Sape Mullender Editor, ACM Press Publisher, pages 97-145, 1993.
5. X. Jia. A Total Ordering Multicast Protocol Using Propagation Trees. IEEE Transactions on Parallel and Distributed Systems, 6(6), pages 617-627, June 1995.
6. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), pages 558-565, July 1978.