Genuine Atomic Multicast in Asynchronous Distributed Systems *

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 distributed systems. Firstly, we give a characterization of the notion of *genuine atomic multicast*. This characterization leads to a better understanding of the difference between atomic multicast and atomic broadcast, and to make a clear distinction between genuine atomic multicast algorithms and non-genuine atomic multicast algorithms. Secondly, we consider a system with at least two processes among which one can crash, and we show that, in contrast to atomic broadcast, genuine atomic multicast is impossible to solve with failure detectors that are unreliable, i.e., that cannot distinguish crashed processes from correct ones. Finally, we discuss a way to circumvent the impossibility result, by restricting the destinations of multicasts to sets of disjoint process groups, each group behaving like a logically correct entity.

1 Introduction

The motivation of this paper is to better understand the characteristics of the atomic multicast problem, and in particular to find out whether the possibility and impossibility results stated for atomic broadcast [5], also apply to atomic multicast.

An *atomic broadcast* primitive enables messages to be sent 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 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 not the message, and (2) no two correct processes deliver any two messages in a different order.

It is easy to see that atomic multicast can be used to implement atomic broadcast, simply by atomic multicasting every message to all the processes in the system. A consequence of this transformation, together with the *FLP* result [9] (which states that consensus is impossible to solve in asynchronous systems if one process can crash), and the equivalence of atomic broadcast and consensus [5], is that atomic multicast is impossible to solve in asynchronous systems if one process can crash. Another consequence is that any lower bound result on the knowledge about failure detection needed to solve atomic broadcast, directly applies to atomic multicast, e.g., [4].

A natural question is then whether we can implement atomic multicast with atomic broadcast. This means that we could solve atomic multicast in asynchronous systems augmented with failure detectors, even if such failure detectors are unreliable, i.e., cannot distinguish the situation where some process has crashed from the situation where the process is correct, but just slow for instance [5].

At first glance, it might seem that the answer is yes, as a simple atomic multicast algorithm can be obtained from any atomic broadcast algorithm as follows: consider a message m to be multicast to a subset $Dst(m)$ of the processes in the system Ω: (1) m is broadcast, together with the information

*Submitted for a Journal Publication. This paper is an extended and revised version of a paper by the same authors entitled *Genuine Atomic Multicast*, in Distributed Algorithms (WDAG'97), Springer Verlag (LNCS 1320): pp 141-154.

$Dst(m)$, to all the processes in Ω; (2) a process $p_i \in \Omega$ only delivers m if $p_i \in Dst(m)$. This transformation of atomic broadcast into atomic multicast, leads however to a *feigned* multicast algorithm, because for $Dst(m) \subset \Omega$, all the processes in the system are involved in the algorithm, *even those that are not concerned with the message m*. Hence, a multicast to a small subset turns out to be as costly as a broadcast (to all) and the benefit of a multicast (namely scalability) is in this case lost.

To distinguish such naive implementations of *feigned* multicast from *genuine* multicast, we introduce a property called *minimality* which reflects the scalability of a multicast, and we require from any *genuine* multicast that it satisfies this property. Roughly speaking, the *minimality* property states that only the sender and the addressees of a message should be involved in the protocol needed to deliver the message. It is obvious that the naive atomic multicast implementation above, using atomic broadcast, does 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 show that in a system with at least two processes, among which one can crash, there exists no genuine atomic multicast algorithm using a failure detector that can be wrong about at least two processes. This impossibility result holds even if channels are reliable and all processes but one are correct. A corollary of this result is that genuine atomic multicast is strictly harder than atomic broadcast.

Our impossibility result actually explains why atomic multicast algorithms proposed in the literature either (1) are not fault-tolerant, e.g., [20], (2) require reliable failure detection, e.g., [2, 10, 14], (3) ensure only local total order, e.g., [3], (local total order does not prevent processes in intersecting destination sets from delivering messages in different orders [13]), or (4) are not genuine multicast algorithms [7, 1, 8].

Our impossibility result is somehow frustrating because the minimality property that makes an atomic multicast algorithm scalable, and hence well suited for large scale 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 present a way to circumvent the impossibility result, by restricting the possible destination sets of multicasts. We assume that messages are not multicast to sets of individual processes, but are rather multicast to sets of non-intersecting process groups, each group behaving like a logically correct entity. This is for instance the case when implementing static transactions on replicated objects: each replicated object is represented by one group [19] and a majority of processes are assumed to be correct within each group. We discuss two algorithms that implement a genuine atomic multicast primitive to sets of non-intersecting process groups.

The rest of the paper is structured as follows. Section 2 presents the system model. Section 3 defines *genuine atomic multicast*. Section 4 characterizes the notion of unreliable failure detector. Section 5 states and proves our impossibility result. Section 6 discusses how to circumvent this result. Section 7 summarizes the main contributions of the paper.

2 Model

Our model of asynchronous computation with failure detection is similar to the one described in [5]. In the following, we recall some important definitions and we introduce new ones that are needed to prove our result. The reader interested in specific details about the original model should consult [5].

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 n processes (i.e., $|\Omega| = n$). Processes fail by *crashing* and a process p is said to *crash at time t* if p does not perform any *action* after time t (the notion of *action* is recalled in Section 2.3). Failures are permanent, i.e., no process *recovers* after a crash. A *correct* process is a process that does not crash. Otherwise the process is said to be *faulty*. 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 [5], that in any failure pattern, there is at least one correct process. The set of correct processes in a failure pattern F is noted $correct(F)$, while the set of faulty processes in F is noted $faulty(F)$.

2.2 Failure detectors

A *failure detector history* is a function H from $\Omega \times \Phi$ to 2^Ω, where for every process $p \in \Omega$, for every time $t \in \Phi$, $H(p, t)$ denotes the set of processes *suspected* (to have crashed) 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, each history representing a possible behavior of $\mathcal{D}$ for the failure pattern F. Given $\mathcal{D}$ any failure detector, F any failure pattern, and H any history in $\mathcal{D}(F)$, we call $H(p,t)$ the *value* of $\mathcal{D}$ at time t, for process p, and history H.

In [5], several classes of failure detectors are introduced. In particular:

- $\mathcal{P}$ *(Perfect)* denotes the set of failure detectors that satisfy (1) *strong completeness*, i.e., eventually every process that crashes is permanently suspected by every correct process, and (2) *strong accuracy*, i.e., no process is suspected before it crashes.
- $\diamond\mathcal{P}$ *(Eventually Perfect)* denotes the set of failure detectors that satisfy (1) *strong completeness*, and (2) *eventual strong accuracy*, i.e., eventually no correct process is suspected by any correct process.
- $\mathcal{S}$ *(Strong)* denotes the set of failure detectors that satisfy (1) *strong completeness*, and (2) *weak accuracy*, i.e., some correct process is never suspected by any correct process.
- $\diamond\mathcal{S}$ *(Eventually Strong)* denotes the set of failure detectors that satisfy (1) *strong completeness*, and (2) *eventual weak accuracy*, i.e., eventually some correct process is never suspected by any correct process.

2.3 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 atomically performs the following three actions: (1) p receives a message from some process 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 $M = \emptyset$ (no message is initially in the buffer): in this case, the states to which I maps the processes are called *initial states.* 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 configuration $C = (I, M)$ is noted $e(C)$.

2.4 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 $S[1](C)$ (the configuration obtained from applying $S[1]$ to C), etc. Given any schedule S, we denote by $P(S)$ the set of the processes that take 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 such that $H \in \mathcal{D}(F)$, 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 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, and (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. $T[1] \leq i \leq T[|T|]$: $S'[i] = S[i]$ and $T'[i] = T[i]$.

3 Genuine atomic multicast

In this section, we define the notion of *genuine atomic multicast.* Then we give a simple example of a (non-fault-tolerant) genuine atomic multicast algorithm.

3.1 On *TO-multicast* and *TO-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 contains a (possibly empty) set of messages. A process p is said to *TO-multicast (Total Order multicast)* a message m if p has m in its *multicast-buffer*. A process q is said to *TO-deliver* a message m if q puts m in its *delivery-buffer*. Every message m is uniquely identified and contains the identity of the process that TO-multicasts m, noted $Orig(m)$, as well as the set of processes to which m is multicast, noted $Dst(m)$. It is important to notice here the distinction between the *receive* event and the *TO-deliver* event. As we will see below, atomic multicast is defined on the *TO-deliver* event (see Figure 1).

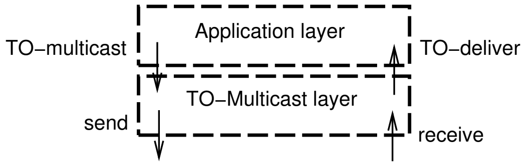

Figure 1: send-receive vs multicast-deliver

For any initial state of a process p, we assume that the *delivery buffer* of p is empty. For simplicity of presentation, and without loss of generality, we assume that all messages TO-multicast by p in some run R are in the *multicast-buffer* of p in its initial state.

3.2 Genuine atomic multicast

An algorithm A is an *atomic multicast* algorithm, if in every run R of A, the following properties are satisfied [13]:

- **Agreement:** If a correct process TO-delivers a message m, then every correct process in $Dst(m)$ eventually TO-delivers m.
- **Validity:** If a correct process TO-multicasts a message m, then every correct process in $Dst(m)$ eventually TO-delivers m.
- **Integrity:** For any message m, every correct process p TO-delivers m at most once, and only if $p \in Dst(m)$ and m was TO-multicast by some process $Orig(m)$.
- **Pairwise Total Order:** If two correct processes p and q TO-deliver messages m and m', then p TO-delivers m before m' if and only if q TO-delivers m before m'.

The agreement, validity and integrity properties define reliable multicast, and together with the pairwise total order property, they define atomic multicast. Note that there are other definitions of atomic multicast (see [13]). We will discuss those definitions in Section 5.

An atomic multicast algorithm A is said to be a *genuine* atomic multicast if, in every run R of A, the following property is satisfied:

- **Minimality:** If a correct process p sends or receives a (non null) message in run R, then some message m is TO-multicast in R, and $p \in \{Orig(m)\} \cup Dst(m)$ (p is either the process that TO-multicasts m, or one of the addressees of m).

The minimality property reflects the scalability of a genuine multicast algorithm. Other aspects such as the number or the size of the messages could also be taken into account when defining the scalability of a multicast algorithm (see for instance [17]).

Notice that the minimality property applies to the notion of multicast algorithm, whether it is atomic or not. One can define for instance a *genuine reliable multicast* algorithm. Notice also that the minimality property applies to the multicast algorithm 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 addressees of the message m (this would not violate the minimality property). The routing issue is related to the network topology, and not to the minimality property of a genuine multicast algorithm.

3.3 On genuine atomic multicast algorithms

In a failure-free environment, the following algorithm (from [20]) is a genuine atomic multicast. When a process p TO-multicasts a message m to $Dst(m)$, p sends the message to every member of $Dst(m)$. Every process $q \in Dst(m)$ 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 [16]. Process p then collects the timestamps from all the processes in $Dst(m)$, defines a sequence number $sn(m)$ as the maximum of the timestamps, and sends $sn(m)$ to every member of $Dst(m)$. Every process $q \in Dst(m)$ that receives $sn(m)$, removes m from its *pending buffer* and stores it in a *delivery buffer*. Process q TO-delivers m when (1) there is no message $m' \neq m$ in its *pending buffer* for which $ts_q(receive(m')) < sn(m)$ and (2) there is no message $m'' \neq m$ in its *delivery buffer* for which $sn(m'') < sn(m)$.

In a failure-free environment, this algorithm trivially ensures the agreement, validity, integrity and pairwise total order properties of atomic multicast. The algorithm also satisfies the minimality property of a genuine atomic multicast as only the sender $Orig(m)$ and the members of $Dst(m)$ take part in the protocol needed to TO-deliver m. The algorithm does not however tolerate a single crash failure: if one process in $Dst(m)$ crashes while $Orig(m)$ is waiting for the timestamps, the algorithm blocks. The algorithm can actually be transformed to tolerate failures, but would require a perfect failure detector, or at least a failure detector that cannot be wrong about the crash of more than one process.

There are indeed fault-tolerant multicast algorithms that do not require reliable failure detectors. Nevertheless, these algorithms are not genuine multicast. For instance, algorithms based on Lamport's ordering technique [16], e.g. [8], are not genuine multicast because they deliver messages according to the order defined by the timestamps initially assigned to messages at multicast time. 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 every process in the system, p might need to interact with the whole system. The algorithm given in [7] also violates the minimality property of genuine atomic multicast: the delivery of a message m may require interacting with processes that are neither $Orig(m)$ nor members of $Dst(m)$. Finally, token-based algorithms, e.g., [1], require the involvement of the token holder, which may be neither $Orig(m)$ nor a process in $Dst(m)$, and hence are not genuine. In the following, we address the question: can we find a genuine atomic multicast algorithm that uses a failure detector that can be wrong about the crash of any subset of the processes in the system?

In [5], Chandra and Toueg have shown that in a system Ω, (1) atomic broadcast can be solved with any failure detector of class $\diamond\mathcal{S}$ if 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. Failure detectors of class $\mathcal{S}$ can be unreliable in the sense that they can be wrong about the crash of any subset of the processes in Ω. The same observation obviously holds for failure detectors of class $\diamond\mathcal{S}$ which is weaker than $\mathcal{S}$. In the following, we formally show that in a system with at least two processes among which one can crash, results (1) and (2) do not apply to the genuine atomic multicast problem. More generally, we show that no algorithm can solve the genuine atomic multicast problem with failure detectors that can be wrong about the crashes of at least two processes.

4 On unreliable failure detection

4.1 Overview

Intuitively, an unreliable failure detector $\mathcal{D}$ is one which can never distinguish the situation where some process q has crashed, from the situation where q is correct, but just very slow for example. In other words, $\mathcal{D}$ provides the same suspicion information for the failure pattern where q has crashed, and the failure pattern where q is correct: $\mathcal{D}$ is said to be *wrong* about process q. Any suspicion by $\mathcal{D}$ of some process q may turn out to be false, i.e., q may actually be correct. In that sense, $\mathcal{D}$ is said to be unreliable as we cannot rely on its suspicions.

We consider different degrees of *failure detection unreliability*, according to the number of processes on which the failure detector can be wrong. Some failure detector $\mathcal{D}_1$ can for instance be wrong about one process whereas a different failure detector $\mathcal{D}_2$ can be wrong about two processes. For instance, the failure detector $\mathcal{D}_2$ cannot distinguish the failure pattern F where processes q_1 and q_2 have both crashed, from the failure pattern F' where only one of them has crashed, or from the failure pattern F'' where neither of them has crashed.

4.2 Definition

Roughly speaking, we say that a failure detector $\mathcal{D}$ is $k-$*unreliable* if $\mathcal{D}$ can be wrong about k processes. Such a failure detector $\mathcal{D}$ may not distinguish any pair of failure patterns F and F', as long as the faulty processes in F and F' are members of a subset W of size k (W denotes the *Wrong subset*) .

- **Definition (k-unreliable failure detection).** *A failure detector $\mathcal{D}$ is $k-$unreliable if for every failure pattern F such that $|faulty(F)| \leq k$, for every history $H \in \mathcal{D}(F)$, there is a subset W of Ω such that [$|W| = k$ and $faulty(F) \subseteq W$], for every failure pattern F' such that [$faulty(F') \subseteq W$], for every time $t_i \in \Phi$, there is a history H' in $\mathcal{D}(F')$ such that [$\forall p \in \Omega$: $\forall t \leq t_i$, $H'(p,t) = H(p,t)$].* [1]

Consider for instance the system $\Omega = \{p_1, p_2, p_3\}$. A failure detector $\mathcal{D}$ is $2-$unreliable if, for instance, for every failure pattern F where p_3 is correct, for every history H in $\mathcal{D}(F)$, for every failure pattern F' where p_3 is correct, at any time t_i, there is a history H' in $\mathcal{D}(F')$ that is equal to H up to time t_i. In this case, $W = \{p_1, p_2\}$ and the only process about which $\mathcal{D}$ is not wrong is p_3. Hence, at any time, $\mathcal{D}$ gives the same information for a failure pattern F where both p_1 and p_2 have crashed, and for a failure pattern F' where both of them are correct, i.e., $\mathcal{D}$ does not distinguish F and F'.

We note $\mathcal{U}_k$ the set (i.e., the class) of failure detectors which are $k-$unreliable[2]. It is easy to see that if $k' < k$, then any failure detector that is $k-$unreliable is also $k'-$unreliable. However, some failure detector might be $k'-$unreliable but not $k-$unreliable. For instance, $\mathcal{U}_3$ (respectively $\mathcal{U}_2$) is the set of failure detectors which can be wrong about 3 processes (respectively 2 processes). Any failure detector of class $\mathcal{U}_3$ is also of class $\mathcal{U}_2$.

In Section 5, we show that no failure detector of class $\mathcal{U}_2$ ($2-$unreliable) can solve the genuine atomic multicast problem, in a system with at least two processes among which one can crash (Proposition 2). This impossibility results consequently applies to any failure detector class $\mathcal{U}_k$, where $k \geq 2$. This impossibility result cannot be stated with failure detectors of classes $\mathcal{U}_0$ or $\mathcal{U}_1$. Class $\mathcal{U}_0$ contains failure detectors which can be wrong about the crash of at least 0 process, and hence contains failure detectors that are perfect (i.e., of class $\mathcal{P}$), which are sufficient to solve genuine atomic multicast. Consider failure detectors that can be wrong about the crash of at most one process, and satisfies the strong completeness property. These failure detectors are of class $\mathcal{U}_1$, and it is easy to transform the algorithm sketched in Section 3.3 [20] so that it can work with such failure detectors. Hence, class $\mathcal{U}_1$ contains failure detectors that can solve the genuine atomic multicast problem.

4.3 On the unreliability of $\mathcal{S}$ and $\Diamond\mathcal{P}$

In this section, we show that, in a system with $n > 1$ processes (i.e., $|\Omega| = n$), failure detector classes $\mathcal{S}$ and $\Diamond\mathcal{P}$ (and hence class $\Diamond\mathcal{S}$, as $\Diamond\mathcal{P} \subset \Diamond\mathcal{S}$) contain at least one failure detector of class $\mathcal{U}_{n-1}$ (Proposition 1). Hence, in a system with at least three processes ($n \geq 3$), both $\mathcal{S}$ and $\Diamond\mathcal{P}$ contain at least one failure detector of class $\mathcal{U}_2$.

Proposition 1. *In a system with at least one correct process ($n \geq 1$), there is at least one failure detector of class $\mathcal{U}_{n-1}$ that also belongs to classes $\mathcal{S}$ and $\Diamond\mathcal{P}$: in other words, $\mathcal{U}_{n-1} \cap \mathcal{S} \cap \Diamond\mathcal{P} \neq \emptyset$.*

PROOF: To show this result, we define a *typical unreliable* failure detector $\mathcal{TU}$ of class $\mathcal{U}_{n-1}$ which satisfies the strong completeness, weak accuracy and eventual strong accuracy properties (i.e., which belongs to classes $\mathcal{S}$ and $\Diamond\mathcal{P}$). We consider a subset W of Ω such that $|W| = 2$ and we define the failure detector $\mathcal{TU}$ as follows:

- *Failure detector $\mathcal{TU}$.* *For every failure pattern F,* $\mathcal{TU}(F) = \{H \mid \exists t_0 \in \Phi, \exists r \in correct(F), \forall p \in \Omega : \forall t \leq t_0, r \notin H(p,t) \text{ and } \forall t > t_0, H(p,t) = F(t)\}$.

Roughly speaking, in every failure pattern F, $\mathcal{TU}$ might suspect all but some correct process r until some time t_0, and after time t_0, $\mathcal{TU}$ suspects exactly the crashed processes ($\mathcal{TU}$ does not suspect correct processes).

[1]It is important to notice that (1) a $k-$unreliable failure detector $\mathcal{D}$ may also be wrong about correct processes, i.e., W may contain correct processes; and (2) for a given failure pattern F, the set W is fixed (in other words, $\mathcal{D}$ cannot be wrong about every subset W in F: this would mean that $\mathcal{D}$ can be wrong about every process).

[2]It is important to notice that the notion of $k-$unreliable failure detector is different from the notion of $k-$mistaken failure detector, defined in [5]. Roughly speaking, a failure detector is said to be $k-$mistaken if it does not make more than k false suspicions (in any history). A $k-$unreliable failure detector can make more than k false suspicions, but cannot make false suspicions on more than k processes (in any history).

(i) We first show that $\mathcal{TU}$ is of class $\mathcal{U}_{n-1}$.

Consider the definition of a $(n-1)-$unreliable failure detector. Consider any failure pattern F such that $|correct(F)| \geq 1$ and any history $H \in \mathcal{TU}(F)$. By the definition of $\mathcal{TU}$, there is a time $t_0 \in \Phi$ and a correct process r in F such that, in H, until time t_0 no process suspects r, and after time t_0 no correct process is suspected. Consider the subset $W = \Omega \setminus \{r\}$, any failure pattern F' where r is correct, and any time t_i. We construct the failure detector history H' such that [$\forall p \in \Omega$: $\forall t \leq t_i$, $H'(p,t) = H(p,t)$ and $\forall t > t_i$, $H'(p,t) = F'(t)$]. Hence, in H', until time t_i no process suspects r, and after time t_i no correct process is suspected. The history H' is thus in $\mathcal{TU}(F')$ which means that $\mathcal{TU}$ is of class $\mathcal{U}_{n-1}$.

(ii) We show that $\mathcal{TU}$ satisfies the strong completeness, eventual strong accuracy, and weak accuracy properties:

1. Strong completeness. By the definition of $\mathcal{TU}$, for every failure pattern F, in every history $H \in \mathcal{TU}(F)$, there is a time t_0 such that, for every correct process p, for every time $t > t_0, H(p,t) = F(t)$ (eventually every crashed process is suspected by every correct process).
2. Eventual strong accuracy. By the definition of $\mathcal{TU}$, in every failure pattern F, in every history $H \in \mathcal{TU}(F)$, there is a time t_0 such that, for every correct process p, for every time $t > t_0$, $p \notin H(p,t)$ (as $H(p,t) = F(t)$). After time t_0, only crashed processes are suspected and hence no correct process is ever suspected by any correct process.
3. Weak accuracy. By the definition of $\mathcal{TU}$, in every failure pattern F, in every history $H \in \mathcal{TU}(F)$, there is some process $r \in correct(F)$, such that $\forall p \in \Omega, \forall t \in \Phi : r \notin H(p,t)$, i.e., there is some correct process r that no process ever suspects.

$\mathcal{TU}$ is thus of classes $\mathcal{U}_{n-1}$, $\mathcal{S}$ and $\diamond\mathcal{P}$. $\square$

5 The impossibility result

In the following, we show that in a system with at least two processes among which one can crash, no algorithm using a $2-$unreliable failure detector can solve the genuine atomic multicast problem. We first give an intuitive idea of the proof (Section 5.1), then we describe it in detail (Section 5.2) and finally we discuss its consequences and generality (Section 5.3 and Section 5.4).

5.1 The impossibility result: proof overview

The basic idea of the proof is the following. It is by contradiction: we assume that there is some genuine atomic multicast algorithm A using a $2-$unreliable failure detector, and we exhibit a run of A where two processes TO-deliver two messages m and m' in different orders. More precisely, we consider a message m which is TO-multicast to a destination set $Dst(m)$ and a message m' which is TO-multicast to a destination set $Dst(m')$, such that $Dst(m) \cap Dst(m') = \{q_1, q_2\}$. Then we exhibit a partial run R of A in which no process crashes, but (1) the processes of $Dst(m)$ think that q_2 has crashed and then TO-deliver m, whereas (2) the processes of $Dst(m')$ think that q_1 has crashed and then TO-deliver m'. As a consequence, process q_1 TO-delivers m but not m', whereas q_2 TO-delivers m' but not m, violating the properties of atomic multicast.

In the following, we give an intuitive idea of how such a run R is built. For presentation generality (see Section 5.3), we consider a scenario where each destination set contains several correct processes (see Section 5.2). We assume that $Orig(m) = p_1$, $Dst(m) = \{r_1, p_1, q_1, q_2\}$, $Orig(m') = p_2$ and $Dst(m') = \{q_1, q_2, p_2, r_2\}$.

(1) Building a partial run where q_2 does not TO-deliver m. Consider the case where only m is TO-multicast and all the processes of $Dst(m)$ are correct, except q_2 which initially crashes. By the validity property of atomic multicast, there is a partial run R_1 where, except q_2, the processes of $Dst(m)$ (including q_1) TO-deliver m (see Figure 2). By the minimality property of a genuine multicast, no process outside $Dst(m)$ sends or receives any message.

(2) Building a partial run where q_1 does not TO-deliver m'. Consider now the case where only m' is TO-multicast and all the processes of $Dst(m')$ are correct, except q_1 which initially crashes. By the validity property of atomic multicast, There is a partial run R_2 where, except q_1, the processes of

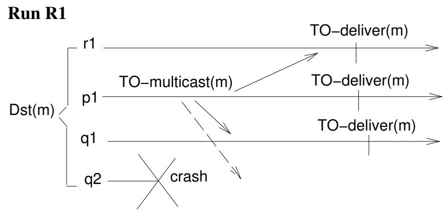

Figure 2: In run R_1, except q_2, all members of $Dst(m)$ TO-deliver m.

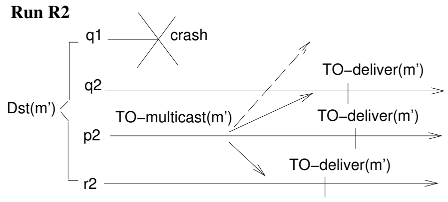

Figure 3: In run R_2, except q_1, all members of $Dst(m')$ TO-deliver m'.

$Dst(m')$ (including q_2) TO-deliver m' (Figure 3). By the minimality property of a genuine multicast, no process outside $Dst(m')$ sends or receives any message.

(3) Building run R by composing R_1 and R_2. The basic idea behind building the partial run R (Figure 4), is to show that both the scenario of processes r_1, p_1 and q_1 in run R_1, and the scenario of processes q_2, p_2 and r_2 in run R_2 can occur in a single failure-free run R. Hence, in R, q_1 TO-delivers m without TO-delivering m' whereas q_2 TO-delivers m' without TO-delivering m: a contradiction with the properties of atomic multicast.

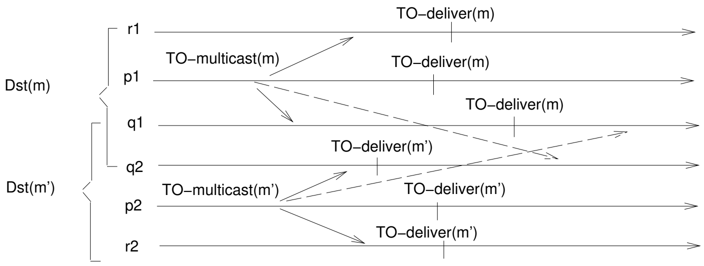

Figure 4: In Run R, q_1 TO-delivers m but not m', whereas q_2 TO-delivers m' but not m.

5.2 Proof of the impossibility result

Lemma 1 below (Section 5.2.1) characterizes 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 2 and R_2 in Figure 3. Lemma 2 (Section 5.2.2) characterizes algorithms that satisfy the agreement and pairwise total order properties of atomic multicast. Roughly speaking, this lemma states that a partial run such as R in Figure 4 is not acceptable for any atomic multicast algorithm. Proposition 2 (Section 5.2.3) states our impossibility result: we prove the result by showing that with the 2-unreliable failure detector definition (Section 4) and Lemma 1, we build a partial run such as R in Figure 4, in contradiction with Lemma 2.

5.2.1 On validity and minimality

Lemma 1. *Let A be any genuine atomic multicast algorithm using any failure detector $\mathcal{D}$, C be any initial configuration where exactly one message m is TO-multicast by some process p in $Dst(m)$, $q \neq p$ be any* *process in $Dst(m)$, t_0 be any time in Φ, F be any failure pattern where q crashes at time t_0 and all other processes are correct, and H be any history in $\mathcal{D}(\mathcal{F})$. There is a partial run of A, $R =< F, H, C, S, T >$ such that, $T[1] = t_0$, every process $r \in Dst(m) - \{q\}$ TO-delivers m, and no process $s \notin Dst(m) - \{q\}$ takes any step in R.*

Proof: In F, all processes are correct except q which crashes at time t_0. By the validity property of atomic multicast, for any failure detector history H in $\mathcal{D}(F)$, there is a partial run $\overline{R} =< F, H, C, \overline{S}, \overline{T} >$ of A, such that (1) $\overline{T}[1] = t_0$ (in asynchronous systems, steps can be arbitrarily delayed), (2) $\overline{S}[1]$ is a step taken by a process in $Dst(m)$ (the process that takes the first step can be arbitrarily chosen), and (3) every process in $Dst(m) - \{q\}$ TO-delivers m. As q has crashed at time t_0, then q has not taken any step in $\overline{T}[1]$. By the minimality property of genuine atomic multicast, in $\overline{S}$ no process in $\Omega - Dst(m)$ sends or receives any non null message.

Let S be the restriction of $\overline{S}$ to the steps taken by the processes in $Dst(m)$ and T the sequence of times corresponding to the events taken in $\overline{S}$ by the processes in $Dst(m)$. As (1) $\overline{S}$ is applicable to C, (2) S is obtained by removing from $\overline{S}$ the steps taken by the processes that are not in $Dst(m)$, and (3) these steps correspond only to null messages, then S is also applicable to C. Hence, the partial run $R =< F, H, C, S, T >$ is also a partial run of A. In R, (1) $T[1] = t_0$, (2) every process $r \in Dst(m) - \{q\}$ TO-delivers m, and (3) no process $s \notin Dst(m) - \{q\}$ takes any step in R. □

5.2.2 On agreement and pairwise total order

Lemma 2. *Let A be any atomic multicast algorithm, R be any partial run of A where two messages m and m' are TO-multicast, and q_1, q_2 be any pair of correct processes in $Dst(m) \cap Dst(m')$. If there is a time t_1 at which q_1 has TO-delivered m but not m', then there is no time t_2 at which q_2 has TO-delivered m' but not m.*

Proof: (by contradiction) Assume that there is a time t_1 at which q_1 has TO-delivered m but not m', and a time t_2 at which q_2 has TO-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 TO-delivers m' and eventually q_2 TO-delivers m_1. Hence, in R_∞, there is a time at which q_1 and q_2 have TO-delivered both m and m', but in different orders: in contradiction with the pairwise total order property of atomic multicast. □

5.2.3 The impossibility result

Proposition 2 (impossibility result). *In a system with at least two processes among which one can crash, there exists no genuine atomic multicast algorithm that uses a 2-unreliable failure detector.*

Notation. In the following, we denote partial runs by $R =< F, H, (I[0], M[0]), S, T >$; (I, M) denotes the sequence of configurations of R; $(I[0], \emptyset)$ denotes an initial configuration (i.e., a set of initial states and an empty set of messages); $S[1](I[0], \emptyset) = (I[1], M[1])$, $S[2](I[1], M[1]) = (I[2], M[2])$, etc; $P(S)$ denotes the set of processes that have taken at least one step in S, and $I|_{P(S)}$ denotes the set of initial states of the processes in $P(S)$. Given S_α (respectively T_α) and S_β (respectively T_β) two finite event sequences (respectively time sequences), $S_\alpha.S_\beta$ (respectively $T_\alpha.T_\beta$) denotes the concatenation of S_α and S_β (respectively of T_α and T_β).

Proof: (by contradiction) Assume a genuine atomic multicast algorithm A using a 2-unreliable failure detector $\mathcal{D}$ in a system Ω with at least two processes among which one can crash.

Consider the failure pattern F_α where q_2 crashes at time 0 and all other processes are correct. Let H_α be any history in $\mathcal{D}(F_\alpha)$. By the definition of a 2−unreliable failure detector (Section 4.2), there is a pair of processes q_1 and q_2 in Ω such that for every failure pattern F' where all processes outside $\{q_1, q_2\}$ are correct, for every time $t_i \in \Phi$, there is a history H' in $\mathcal{D}(F')$ such that [$\forall p \in \Omega$: $\forall t \leq t_i$, $H'(p, t) = H(p, t)$] (the set of processes for which $\mathcal{D}$ is wrong is $W = \{q_1, q_2\}$).

Let m be a message TO-multicast by a process in $Dst(m) - \{q_2\}$ (to the subset $Dst(m)$) and m' be a message TO-multicast by a process in $Dst(m') - \{q_1\}$ (to the subset $Dst(m')$), such that $\{q_1, q_2\} = Dst(m) \cap Dst(m')$.

Let $(I_\alpha[0], \emptyset)$ be some initial configuration where only the message m is TO-multicast. By Lemma 1, there is a partial run of A denoted by $R_1 =< F_\alpha, H_\alpha, (I_\alpha[0], M_\alpha[0]), S_\alpha, T_\alpha >$, such that $T_\alpha[1] = 0$, the processes in $Dst(m) - \{q_2\}$ TO-deliver m, and no process outside $Dst(m) - \{q_2\}$ takes any step. Hence, in R_α process q_1 TO-delivers m whereas q_2 does not.

In F_α all processes but q_2 are correct. Consider the failure pattern F_β where q_1 crashes at time 0 and all other processes are correct. Consider time $T_\alpha[|T_\alpha|]$. By the 2-unreliable failure detection definition, there is a failure detector history $H_\beta \in \mathcal{D}(F_\beta)$, such that for every $t \leq T_\alpha[|T_\alpha|]$, for every process $p \in \Omega$, $H_\alpha(p,t) = H_\beta(p,t)$.

Let $(I_\beta[0], \emptyset)$ be some initial configuration where only the message m' is TO-multicast. By Lemma 1, there is a partial run of A denoted by $R_2 =< F_\beta, H_\beta, (I_\beta[0], M_\beta[0]), S_\beta, T_\beta >$, such that $T_\beta[1] = T_\alpha[|T_\alpha|] + 1$, the processes in $Dst(m) - \{q_1\}$ TO-deliver m', and no process outside $Dst(m) - \{q_1\}$ takes any step. Hence, in R_2 process q_2 TO-delivers m' whereas q_1 does not. (Note that run R_2 starts right after R_1 ends.)

Consider F the failure pattern where all processes (including q_1 and q_2) are correct. Consider time $T_\beta[|T_\beta|]$. By the definition of a 2-unreliable failure detector, there is a history $H \in \mathcal{D}(F)$, such that for every $t \leq T_\beta[|T_\beta|]$ and every process $p \in \Omega$, $H(p,t) = H_\beta(p,t)$.

Let (I_α, M_α) be the sequence of configurations of R_1 and (I_β, M_β) be the sequence of configurations of R_2. As R_1 is a partial run of A, then S_α is applicable to $(I_\alpha[0], \emptyset)$. As $P(S_\alpha) = Dst(m) - \{q_2\}$, and $P(S_\beta) = Dst(m') - \{q_1\}$, we have $P(S_\alpha) \cap P(S_\beta) = \emptyset$ (i.e., no process of $P(S_\alpha)$ takes any step in S_β, and no process of $P(S_\beta)$ takes any step in S_α). As S_α is applicable to $(I_\alpha[0], \emptyset)$, and S_β is applicable to $(I_\beta, \emptyset)$, then S_α is also applicable to $(I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset)$, S_β is also applicable to $(I_\alpha|_{P(S_\alpha)} \cup I_\beta|_{P(S_\beta)}, \emptyset)$, and $S_\alpha.S_\beta[|S_\alpha|]|_{P(S_\beta)} = I_\beta|_{P(S_\beta)}$ (i.e., the processes of $P(S_\beta)$ keep the same state after S_α). As we have $T_\alpha[|T_\alpha|] < T_\beta[1]$ and all processes are correct in F, 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. In R, q_1 TO-delivers m but not m' whereas q_2 TO-delivers m' but not m and $q_1, q_2 \in Dst(m) \cap Dst(m')$: a contradiction with Lemma 2. $\square$

5.3 Genuine atomic multicast harder than atomic broadcast

A corollary of Proposition 1 (Section 4.3) and our impossibility result (Proposition 2 in Section 5.2), is that genuine atomic multicast cannot be solved with failure detector classes $\mathcal{S}$ or $\diamond\mathcal{P}$, in a system with at least three processes. Hence, genuine atomic multicast is strictly harder than atomic broadcast (which was shown to be solvable using failure detector classes $\mathcal{S}$ or $\diamond\mathcal{P}$ [5]).

5.4 On the generality of the impossibility result

We discuss below the generality of the impossibility result with respect to alternative definitions of genuine atomic multicast and stronger assumptions about the number of correct processes in destination sets.

5.4.1 On atomic multicast definition

Our impossibility result also applies to a weaker form of genuine atomic multicast where we would not require the integrity property to be satisfied (e.g., even if a process is allowed to TO-deliver the same message twice). Indeed, our proof relies on agreement, validity, pairwise total order and minimality properties of genuine atomic multicast, but never uses the integrity property. Our impossibility result also applies to the stronger definition of genuine atomic multicast presented in [17], which requires scalability, not only in terms of the number of processes involved in a multicast protocol, but also in terms of the number and size of messages involved in the protocol.

Finally, our impossibility result also applies to stronger definitions of atomic multicast presented in [13]. Among these definitions are those which consider *uniform* versions of the agreement and pairwise total order properties, or which consider *global total order*, and *uniform global total order* properties. 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 provides no guarantee for messages in intersecting sets. Our impossibility result does not apply to this property. Indeed, with a weaker definition based on local total order, genuine atomic multicast can be implemented using the atomic broadcast algorithm of Chandra and Toueg described in [5]. Hence all results on solving atomic broadcast with unreliable failure detection also apply to atomic multicast. To summarize, among the ordering properties of [13], the definition we have considered uses the weakest property for which our impossibility result holds.

5.4.2 On the number of correct processes

Assuming a majority of correct processes in every destination set is not sufficient to circumvent our impossibility result. This would mean, if f denotes the maximum number of processes which can crash, that the properties of genuine atomic multicast are ensured in every run where, for every message m that is TO-multicast, $|Dst(m)| > 2f$. Consider the scenario of Figure 4 (Section 5.1). Assume that up to 1 process can crash (i.e., $f = 1$). Even if we assume that each of $Dst(m_1) = \{r_1, p_1, q_1, q_2\}$ and $Dst(m_2) = \{q_1, q_2, p_2, r_2\}$ contains three correct processes (which is the case in the scenario), process q_1 TO-delivers m_1 without even receiving m_2 and process q_2 TO-delivers m_2 without even receiving m_1. This is actually conveyed by the proof, which does not preclude the existence of a majority of correct processes both in $Dst(m)$ or in $Dst(m')$.

6 Circumventing the impossibility result

In the previous sections, we have considered a general multicast model where any process in the system can TO-multicast messages to any subset of the processes in the system. In this section, we restrict this model by considering TO-multicast to sets of *non-intersecting* process groups, rather than to individual processes. We assume furthermore that for every group g, a majority of g's members are correct, i.e., each group acts as a logically correct entity, despite the fact that individual processes can crash. This model does not preclude the possibility of a group of size *one*, as long as the process inside that group is correct. We discuss below how our restricted model enables us to circumvent the impossibility result.

We first point out in Section 6.1 a practical use of a TO-multicast primitive to sets of non-intersecting process groups. Then we discuss in Section 6.2 two ways of circumventing the impossibility result, i.e., two implementations of genuine atomic multicast. Both implementations are extensions of Skeen's (non-fault-tolerant) algorithm sketched in Section 3.3 [20].

6.1 A practical use of TO-multicast to multiple disjoint groups

6.1.1 Static transactions on replicated objects

Multiple groups $g, g', g'', \ldots$ typically represent multiple replicated objects, one per group. Whereas TO-multicast to one single group allows us only to model *independent* operations, TO-multicast to multiple groups *links* multiple operations together. This corresponds to the well known concept of *transaction*. Consider a simple application where (1) a group g manages a bank account #1; (2) a group g' manages a bank account #2; (3) an operation op represents the withdrawal of \$100 from bank account #1; and (4) an operation op' represents the deposit of \$100 to bank account #2 [19].

Performing the two operations op and op' in the context of one transaction means specifically that no operation can observe a state such that op has been performed and not op' (or vice versa). This can be easily achieved by building the message $m = ((op, g), (op', g'))$, and by TO-multicasting m to $g \cup g'$. After the TO-delivery of m, the members of g perform the operation op, while the members of g' perform the operation op'.

6.1.2 On the assumption of non-intersecting process groups

When groups are used for replication, the members of a group g share a common state, denoted by g_s: every member of g has its own copy of the state g_s. Assuming non-intersecting groups means that the state of any two processes p and p', members of two different groups, must be empty.

If we consider the previous bank account example, the assumption of non-intersecting process groups does not mean that every group can manage only a single replicated bank account. In fact, every group can manage many bank accounts, as long as the sets of accounts managed by different groups do not intersect.

It is important to notice that without the assumption of *non-intersecting* process groups, nothing would prevent the scenario of our impossibility proof (Section 5.2.3), e.g., $Dst(m) = g_1$, $Dst(m') = g_2$, $g_1 \cap g_2 = \{q_1, q_2\}$, and any of q_1 and q_2 can crash. Assuming TO-multicast to sets of *non-intersecting* process groups, each group acting as a logically correct entity (i.e., among every group members a majority is correct), means that any intersection of two destination sets is made of logically correct entities, e.g., $Dst(m) = \{g_1, g_3\}$, $Dst(m') = \{g_2, g_3\}$ and only a minority of g_3's members can crash (in contrast to the scenario of our impossibility proof where any of $\{q_1, q_2\}$ can crash, i.e., there is no majority of correct processes).

6.2 Implementation of genuine TO-multicast to multiple groups

The two genuine TO-multicast implementations presented here are extensions of Skeen's algorithm (see Section 3.3), which we recall below. Given a message m TO-multicast to $Dst(m)$, the principle of Skeen's algorithm is the following:

1. every process p in $Dst(m)$ attaches a timestamp $ts_p(m)$ to m;
2. the timestamps $ts_x(m)$ of all the processes in $Dst(m)$ are used to compute the sequence number $sn(m)$ of message m: $sn(m)$ is set to the maximum of the timestamps $ts_x(m)$;
3. the messages are TO-delivered in the order defined by their sequence numbers.

In the context of TO-multicast to multiple groups, this general principle can be adapted in two ways. Let m be a message TO-multicast to $Dst(m)$, where $Dst(m)$ is a set of groups:

- *Solution 1.* Each group g in $Dst(m)$ independently computes a *group* timestamp $ts_g(m)$, using a consensus protocol. The sequence number $sn(m)$ is then set to the maximum of all group timestamps $ts_g(m)$ (see Section 6.2.1);[3]
- *Solution 2.* As in Skeen's protocol, the timestamp $ts_p(m)$ is defined by each individual process p in $Dst(m)$. The sequence number $sn(m)$ is then computed using a consensus protocol among the set of processes in $Dst(m)$ (see Section 6.2.2).

The key difference between the two solutions is related to the use of a consensus protocol. In solution 1, consensus is part of the first step of the protocol (it is a local computation inside each group g of a timestamp $ts_g(m)$). In solution 2, consensus appears in the second step of the protocol (it is a global computation among all groups in $Dst(m)$ of a sequence number $sn(m)$).

6.2.1 Solution 1: one consensus per group g to compute the group timestamp $ts_g(m)$

Consider m TO-multicast to $Dst(m)$, where $Dst(m)$ is a set of non-intersecting groups:

- Every group g in $Dst(m)$ first computes a group timestamp $ts_g(m)$;
- The sequence number $sn(m)$ is then set to the maximum of all the group timestamps $ts_g(m)$.

The computation of the group timestamp $ts_g(m)$ by group g can be implemented using a *Local-TO-multicast* of m to g, ensuring *local total order* inside g. Let message m be *Local-TO-multicast* to g, and assume that the only event that increments the logical clock of any process p in g is the *Local-TO-delivery* of m. The group timestamp $ts_g(m)$ can thus be defined as the logical clock value of the event *Local-TO-delivery(m)*. All the processes of some group g *Local-TO-deliver* m in the same order and hence all the processes of g agree on the same timestamp $ts_g(m)$. The *Local-TO-multicast* of m to any group g can be implemented using the Chandra-Toueg atomic broadcast protocol [5] inside g, which is itself based on a reduction to a consensus protocol. Thus, solution 1 requires the consensus problem to be solvable within each individual group g of the system. Examples of conditions for solving consensus on any group g are: (1) a majority of g's members are correct, and (2) the failure detector is of class $\diamond\mathcal{P}$. Assumption (2) ensures that any subsystem g has a failure detector of class $\diamond\mathcal{S}$, which was shown to be sufficient to solve consensus in a system with a majority of correct processes [5].

6.2.2 Solution 2: one consensus in $Dst(m)$ to compute the sequence number $sn(m)$

Given n the number of groups in $Dst(m)$, Solution 1 involves n consensus protocols (one consensus protocol inside each group). Solution 2, presented below, involves one consensus protocol, however among a larger number of processes. In both cases, a process participates at most in one consensus protocol.

Consider m TO-multicast to $Dst(m)$, where $Dst(m)$ is a set of non-intersecting groups:

- Each member p of a group g in $Dst(m)$, when receiving m, attaches a timestamp $ts_p(m)$ to m. Contrary to Solution 1, reception is not ordered here, which means that two members p and p' of the same group g, can attach two different timestamps to m.

[3] This solution has been suggested by an anonymous reviewer.

- Once a process p has its timestamp $ts_p(m)$, p then sends $ts_p(m)$ to all the processes in $Dst(m)$. Process p then waits to get the timestamp $ts_x(m)$ from a majority of processes of every group in $Dst(m)$. These timestamps are used by p to define its initial value $prop\text{-}sn_p(m)$ for a consensus protocol to decide on the sequence number $sn(m)$: $prop\text{-}sn_p(m)$ is set to the maximum of all timestamps $ts_x(m)$ received by p.
- The sequence number $sn(m)$ is the decision of the consensus protocol among the processes in $Dst(m)$.

The details of this solution can be found in [11]. The solution is based on causal delivery of messages [18], and in order to ensure that consensus is solvable among the processes in $Dst(m)$, it assumes that among group members, a majority is correct, and the failure detector is of class $\diamond\mathcal{P}$. In [17], an optimized version that does not require causal ordering and ensures scalability in terms of number and size of messages is also presented.

7 Summary

Firstly, we have introduced the notion of genuine atomic multicast. This notion 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 feigned genuine atomic multicast algorithms.

Secondly, we have defined what it means for a failure detector to be $k-$unreliable, and using this definition, we have shown that, in a system with at least two processes among which one can crash, genuine atomic multicast cannot be solved with an algorithm that uses a $2-$unreliable failure detector. Such failure detectors are those which can be wrong about at least two processes in the system. This result explains why atomic multicast algorithms proposed in the literature either (1) are not fault-tolerant, e.g., [20], (2) require reliable failure detection, e.g., [2, 10, 14], (3) ensure only local total order, e.g., [3], or (4) are not genuine multicast algorithms [7, 1, 8]. A simple corollary of this result is that genuine atomic multicast is strictly harder than atomic broadcast.

Finally, we have presented a way to circumvent the impossibility result, by assuming that messages are only multicast to sets of disjoint process groups (rather than to sets of individual processes), each group acting as a logically correct entity. In this context, we have sketched two genuine atomic multicast algorithms that tolerate failures and do not require reliable failure detection.

Acknowledgement

We are very grateful to Vassos Hadzilacos and Luis Rodrigues for their helpful comments on earlier drafts of this paper. We would also like to thank the anonymous reviewers for their useful suggestions to improve the quality of the presentation, and particularly the reviewer that suggested Solution 1 of Section 6.

References

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

[2] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. ACM Transactions on Computer Systems, 5(1), pages 47-76, February 1987.

[3] K. Birman, A. Schiper and P. Stephenson. Lightweight Causal and Atomic Group Multicast. ACM Transactions on Computer Systems, 9(3), pages 272-314, August 1991.

[4] T. Chandra, V. Hadzilacos and S. Toueg. The weakest failure detector for solving consensus. Journal of the ACM, 43(4), pages 685-722, July 1996.

[5] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2), pages 225-267, March 1996.

[6] J. Chang and M. Maxemchuck. Reliable Broadcast Protocols. ACM Transactions on Computer Systems, 2(3), pages 251-273, August 1984.

[7] D. Dolev, S. Kramer and D. Malkhi. Early Delivery Totally Ordered Broadcast in Asynchronous Environments. Proceedings of the 23rd IEEE Int. Symp. on Fault-Tolerant Computing, pages 296-306, June 1993.

[8] P. Ezhilchelvan, R. Macedo and S. Shrivastava. Newtop: A Fault-Tolerant Group Communication Protocol. Proceedings of the 15th IEEE Int. Conf. on Distributed Computing Systems, pages 296-306, May 1997.

[9] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32 (2), pages 374-382, April 1985.

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

[11] R. Guerraoui and A. Schiper. Total Order Multicast to Multiple Groups. Proceedings of the 17th IEEE Int. Conf. on Distributed Computing Systems, pages 578-585, May 1997.

[12] R. Guerraoui and A. Schiper. Genuine Atomic Multicast. Proceedings of the Int. Workshop. on Distributed Algorithms (WDAG'97), Springer Verlag (LNCS 1320): pp 141-154, September 1997.

[13] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Cornell University, Technical Report (TR 94-1425), 1994.

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

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

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

[17] L. Rodrigues, R. Guerraoui and A. Schiper. Scalable Atomic Multicast. Ecole Polytechnique Fédérale de Lausanne, Technical Report, December 1997.

[18] A. Schiper, J. Eggli and A. Sandoz. A New Algorithm to Implement Causal Ordering. Proceedings of the 3rd Int. Workshop. on Distributed Algorithms, Springer Verlag (LNCS 392), pages 219-232, September 1989.

[19] A. Schiper and M. Raynal. From Group Communication to Transactions in Distributed Systems. Communications of the ACM, 39(4), pages 84-87, April 1996.

[20] D. Skeen. Unpublished communication. February 1985. Referenced in [2].