On the Inherent Cost of Atomic Broadcast and Multicast in Wide Area Networks*

Nicolas Schiper and Fernando Pedone

University of Lugano, Switzerland

Abstract. In this paper, we study the atomic broadcast and multicast problems, two fundamental abstractions for building fault-tolerant systems. As opposed to atomic broadcast, atomic multicast allows messages to be addressed to a subset of the processes in the system, each message possibly being multicast to a different subset. Our study focuses on wide area networks where *groups of processes*, i.e., processes physically close to each other, are inter-connected through high latency communication links. In this context, we capture the cost of algorithms, denoted *latency degree*, as the minimum number of inter-group message delays between the broadcasting (multicasting) of a message and its delivery. We present an atomic multicast algorithm with a latency degree of two and show that it is optimal. We then present the first fault-tolerant atomic broadcast algorithm with a latency degree of one. To achieve such a low latency, the algorithm is proactive, i.e., it may take actions even though no messages are broadcast. Nevertheless, it is quiescent: provided that the number of broadcast messages is finite, the algorithm eventually ceases its operation.

1 Introduction

Distributed applications spanning multiple geographical locations have become common in recent years. Typically, each geographical site, or *group*, hosts an arbitrarily large number of processes connected through high-end local links; a few groups exist, interconnected through high-latency communication links. As a consequence, communication among processes in the same group is cheap and fast; communication among processes in different groups is expensive and orders of magnitude slower than local communication. Data is replicated both locally, for high availability, and globally, usually for locality of access. In this paper we investigate the atomic broadcast and multicast problems, two communication primitives that offer adequate properties, namely agreement on the set of messages delivered and on their delivery order, to implement replication [9].

Ideally, we would like to devise algorithms that use inter-group links as sparingly as possible, saving on both latency and bandwidth (i.e., number of

---

messages). As we explain next, however, atomic broadcast and multicast establish an inherent tradeoff in this context. As opposed to atomic broadcast, atomic multicast allows messages to be sent to a subset of processes in the system. More precisely, messages can be addressed to any subset of the system's groups, each message possibly being multicast to a different subset. From a problem solvability point of view, atomic multicast can be easily reduced to atomic broadcast: every message is broadcast to all the groups in the system and only delivered by those processes the message is originally addressed to. Obviously, this solution is inefficient as it implies communication among processes that are not concerned by the multicast messages. To rule out trivial implementations of no practical interest, we require multicast algorithms to be *genuine* [7], i.e., only processes addressed by the message should be involved in the protocol. A genuine atomic multicast can thus be seen as an adequate communication primitive for distributed applications spanning multiple geographical locations in which processes store a subset of the application's data (i.e., *partial replication*).

We show that for messages multicast to at least two groups, no genuine atomic multicast algorithm can hope to achieve a latency degree lower than two.[1] This result is proven under strong system assumptions, namely processes do not crash and links are reliable. Moreover, this lower bound is tight, i.e., the fault-tolerant algorithm $\mathcal{A}1$ of Section 4 and the algorithm in [5] achieve this latency degree ($\mathcal{A}1$ is an optimized version of [5], see Section 4). A corollary of this result is that Skeen's algorithm, initially described in [2] and designed for failure-free systems, is also optimal—a result that has apparently been left unnoticed by the scientific community for more than 20 years.

We demonstrate that atomic multicast is inherently more expensive than atomic broadcast by presenting the first fault-tolerant broadcast algorithm with a latency degree of one. To achieve such a low latency, the algorithm is proactive, i.e., it may take actions even though no messages are broadcast. Nevertheless, we show how it can be made *quiescent*: provided that a finite number of messages is broadcast, processes eventually cease to communicate. In runs where the algorithm becomes quiescent too early, that is, a message m is broadcast after processes have decided to stop communicating, m will not be delivered in a single inter-group message delay, but in two. We show that this extra cost is unavoidable, i.e., no quiescent atomic broadcast algorithm can hope to always achieve a latency degree of one.[2]

These two lower bound results stem from a common cause, namely the *reactiveness* of the processes at the time when the message is cast. Roughly speak-

---

[1] A precise definition of latency degree is given in Section 2.

[2] This result also holds for quiescent (genuine or non-genuine) atomic multicast algorithms. The genuine case is already covered by the first lower bound result and is therefore irrelevant here.

ing, a process p is said to be *reactive* when the next message m that p sends is in response either to a local multicast event or to the reception of another message. In Section 3, we first show that no atomic broadcast or multicast algorithm can hope to deliver the last cast message m with a latency degree of one if m is cast at a time when processes are reactive. To obtain the lower bounds, we then show that (i) in runs of any genuine atomic multicast algorithm where one message is multicast at time t, processes are reactive at t and (ii) in runs of any quiescent atomic broadcast or atomic multicast algorithm where a finite number of messages are cast, processes are eventually reactive forever.

These results help better understand the difference between atomic broadcast and multicast. In particular, they point out a tradeoff between the latency degree and message complexity of these two problems. Consider a partial replication scenario where each group replicates a set of objects. If latency is the main concern, then every operation should be broadcast to all groups, and only groups concerned by the operation handle it. This solution, however, has a high message complexity: every operation leads to sending at least one message to all processes in the system. Obviously, this is inefficient if the operation only *touches* a subset of the system's groups. To reduce the message complexity, genuine multicast can be used. However, any genuine multicast algorithm will have a latency degree of at least two.

The rest of the paper is structured as follows. In Section 2, we present our system model and definitions. Section 3 shows the genuine atomic multicast latency degree lower bound and investigates the cost of quiescence in a unified way. In Sections 4 and 5, we present the optimal multicast and broadcast algorithms. Finally, Section 6 discusses the related work and concludes the paper. The proofs of correctness of the algorithms can be found in [12].

2 System Model and Definitions

2.1 Processes and Links

We consider a system $\Pi = \{p_1, ..., p_n\}$ of processes which communicate through message passing and do not have access to a shared memory or a global clock. We assume the benign crash-stop failure model, i.e., processes may fail by crashing, but do not behave maliciously. A process that never crashes is *correct*; otherwise it is *faulty*. The system is asynchronous, i.e., messages may experience arbitrarily large (but finite) delays and there is no bound on relative process speeds. Furthermore, the communication links do not corrupt or duplicate messages, and are quasi-reliable: if a correct process p sends a message m to a correct process q, then q eventually receives m. We define $\Gamma = \{g_1, ..., g_m\}$ as the set of process groups in the system. Groups are disjoint, non-empty and

satisfy $\bigcup_{g \in \Gamma} g = \Pi$. For each process $p \in \Pi$, $group(p)$ identifies the group p belongs to. Hereafter, we assume that in each group: (1) there exists at least one correct process and (2) consensus is solvable (consensus is defined below).

2.2 Specifications of Agreement Problems

We define the agreement problems considered in this paper, namely consensus, reliable multicast, and atomic multicast/broadcast. Let $\mathcal{A}$ be an agreement algorithm. We define $\mathcal{R}(\mathcal{A})$ as the set of all admissible runs of $\mathcal{A}$.

Consensus In the *consensus* problem, processes propose values and must reach agreement on the value decided. Uniform consensus is defined by the primitives propose(v) and decide(v) and satisfies the following properties [8]: (i) *uniform integrity:* if a process decides v, then v was previously proposed by some process, (ii) *termination:* every correct process eventually decides exactly one value, (iii) *uniform agreement:* if a process decides v, then all correct processes eventually decide v.

Reliable Multicast With *reliable multicast*, messages may be addressed to any subset of the processes in Π. For each message m, $m.dest$ denotes the processes to which the message is reliably multicast. Non-uniform reliable multicast is defined by primitives *R-MCast(m)* and *R-Deliver(m)*, and satisfies the following properties : (i) *uniform integrity:* for any process p and any message m, p R-Delivers m at most once, and only if $p \in m.dest$ and m was previously R-MCast, (ii) *validity:* if a correct process p R-MCasts a message m, then eventually all correct processes $q \in m.dest$ R-Deliver m, (iii) *agreement:* if a correct process p R-Delivers a message m, then eventually all correct processes $q \in m.dest$ R-Deliver m.

Atomic Multicast Atomic multicast allows messages to be addressed to a subset of groups in Γ. For each message m, $m.dest$ denotes the groups to which m is addressed. Let p be a process. By abuse of notation, we write $p \in m.dest$ instead of $\exists g \in \Gamma : g \in m.dest \wedge p \in g$. Hereafter, we denote the sequence of messages delivered by p at time t as S_p^t, and the sequence of messages delivered by p at time t *projected* on processes p and q as $P_{p,q}(S_p^t)$, i.e., $P_{p,q}(S_p^t)$ is the sequence of messages S_p^t restricted to the messages m such that $p, q \in m.dest$. Atomic multicast is defined by the primitives A-MCast and A-Deliver, and satisfies the uniform integrity and validity properties of reliable multicast as well as the two following properties: (i) *uniform agreement:* if a process p A-Delivers m, then all correct processes $q \in m.dest$ eventually A-Deliver m, (ii) *uniform prefix order:* for any two processes p and q and any time t, either $P_{p,q}(S_p^t)$ is a prefix of $P_{p,q}(S_q^t)$ or $P_{p,q}(S_q^t)$ is a prefix of $P_{p,q}(S_p^t)$.

We also require atomic multicast algorithms to be *genuine* [7]: An algorithm $\mathcal{A}$ solving atomic multicast is said to be *genuine* iff for any run $R \in \mathcal{R}(\mathcal{A})$ and for any process p, in R if p sends or receives a message then some message m is A-MCast and either p is the process that A-MCasts m or $p \in m.dest$.

Atomic Broadcast Atomic broadcast is a special case of atomic multicast. It is defined by the primitives A-BCast and A-Deliver and satisfies the same properties as atomic multicast where all A-BCast messages m are such that $m.dest = \Gamma$, i.e., messages are always A-BCast to all groups in the system.

2.3 Latency Degree

Let $\mathcal{A}$ be a broadcast or multicast algorithm and R be a run of $\mathcal{A}$ ($R \in \mathcal{R}(\mathcal{A})$). Moreover, in run R, let m be a message A-XCast (A-BCast or A-MCast) and $\Pi'(m) \subseteq \Pi$ be the set of processes that A-Deliver m. Intuitively, the latency degree of R is the minimal length of the causal path between the A-XCast of m and the last A-delivery of m among the processes in $\Pi'(m)$, when counting inter-group messages only. To define this latency degree we assign timestamps to process events using a slightly modified version of Lamport's logical clocks [9]. Initially, for all processes $p \in \Pi$, p's logical clock, LC_p, is initialized to 0. On process p, an event e is assigned its timestamp as follows:

1. If e is a local event, $ts(e) = LC_p$
2. If e is the send event of a message m to a process q,

$$ts(e) = \begin{cases} LC_p + 1, \; if \; group(p) \neq group(q) \\ LC_p, \quad otherwise \end{cases}$$

3. If e is the receive event of a message m, $ts(e) = \max(LC_p, ts(send(m)))$

The latency degree of a message m A-XCast in run R is defined as follows:
$$\Delta(m, R) = \max_{q \in \Pi'(m)}(ts(A\text{-}Deliver(m)_q) - ts(A\text{-}XCast(m)_p))$$

where $A\text{-}Deliver(m)_q$ and $A\text{-}XCast(m)_p$ respectively denote the A-Deliver(m) event on process q and the A-XCast(m) event on process p. We refer to the latency degree of an algorithm $\mathcal{A}$ as the minimum value of $\Delta(m, R)$ among all admissible runs R of $\mathcal{A}$ and messages m A-XCast in R.

3 The Inherent Cost of Reactiveness

We establish the inherent cost of the genuine atomic multicast problem for messages that are multicast to multiple groups and we show that quiescence has a cost, i.e., in runs where a message m is cast at a time when the algorithm is quiescent, there exists no algorithm that delivers m with a latency degree of one. As

explained in Section 1, we proceed in two steps. We first show that, if processes are reactive when the last message m is cast, then m cannot be delivered with a latency degree of one. We then prove that (i) in runs of any genuine atomic multicast algorithm where one message is multicast at time t, processes are reactive at t and (ii) in runs of any quiescent atomic broadcast or atomic multicast algorithm where a finite number of messages are cast, processes are eventually reactive forever.

The proofs are done in a model identical to the model of Section 2, except that processes do not crash and links are reliable, i.e., they do not corrupt, duplicate, or lose messages.

Definition 1 *In a run R of an atomic broadcast or multicast algorithm, we say that a process p is reactive at time t iff p sends a message m at time $t' \geq t$ only if p A-XCasts m or if p received a message sent in the interval $[t, t']$.*

Proposition 1 *In a system with at least two groups, for any atomic broadcast or any atomic multicast algorithm $\mathcal{A}$, there does not exist runs R_1, R_2 of $\mathcal{A}$ in which processes are reactive at the time the last messages m_1, m_2 are A-XCast to at least two groups, such that $\Delta(m_1, R_1) = \Delta(m_2, R_2) = 1$.*

Proof: Suppose, by way of contradiction, that there exist an algorithm $\mathcal{A}$ and runs R_i of $\mathcal{A}$, $i \in \{1, 2\}$, such that $\Delta(m_i, R_i) = 1$. Consider two groups, g_1 and g_2. In run R_i, process $p_i \in g_i$ A-XCasts message m_i at time t to g_1 and g_2. We first show that (*) in R_i, at or after time t, processes can only send messages m such that for a sequence of events $e_1 = \text{A-XCast}(m_i), e_2, ..., e_k = \text{send}(m)$, $\text{A-XCast}(m_i) \rightarrow e_2 \rightarrow ... \rightarrow \text{send}(m)$.[3] Suppose, by way of contradiction, that there exists a process p in R_i that sends a message m at a time $t'_i \geq t$ such that the event $\text{send}(m)$ is not causally linked to the event $\text{A-XCast}(m_i)$. We construct a run R'_i identical to run R_i except that message m_i is not A-MCast (note that processes are also reactive at time t in R'_i). Since in R_i, there is no causal chain linking the event $\text{A-XCast}(m_i)$ with the event $\text{send}(m)$, runs R'_i and R_i are indistinguishable to process p up to and including time t'_i. Therefore, p also sends m in R'_i. Hence, since processes are reactive at time t and no message is A-XCast at or after t, p must have received a messag m' sent at or after t by some process q. Applying the same reasoning multiple times, we argue that there must exist a process r that sends a message m'' at time t such that for some events $e_1 = \text{send}(m''), e_2, ..., e_{x-1} = \text{send}(m'), e_x = \text{send}(m)$, we have

---

[3] Events $e_1, ..., e_k$ can be of four kinds, either $\text{send}(m)$, $\text{receive}(m)$, $\text{A-XCast}(m)$, or A-Deliver(m) for some message m. Moreover, the relation $\rightarrow$ is Lamport's transitive happened before relation on events [9]. It is defined as follows: $e_1 \rightarrow e_2 \Leftrightarrow e_1, e_2$ are two events on the same process and e_1 happens before e_2 or $e_1 = \text{send}(m)$ and $e_2 = \text{receive}(m)$ for some message m.

send(m'') $\to$ ... $\to$ send(m') $\to$ send(m). However, r cannot send m'' because no message is A-XCast at or after t, a contradiction.

By the validity property of $\mathcal{A}$ and because there is no failure, all processes eventually A-Deliver m_i. Since $\Delta(m_i, R_i) = 1$, by (*), processes in g_i A-Deliver m_i before receiving any message from processes in g_{3-i} sent at or after time t. Let $t_i^* > t$ be the time at which all processes in g_i have A-Delivered message m_i. We now build run R_3 as follows. As in run R_i, p_i A-XCasts m_i. Runs R_i and R_3 are indistinguishable for processes in group g_i up to time t_i^*, that is, all messages causally linked to the event A-XCast(m_{3-i}) (including A-XCast(m_{3-i}) itself) sent from processes in group g_{3-i} to processes in group g_i are delayed until after t_i^*. Consequently, processes in group g_i have all A-Delivered m_i by time t_i^*. By the uniform agreement of $\mathcal{A}$, processes in g_1 eventually A-Deliver m_2 and processes in g_2 eventually A-Deliver m_1, violating the uniform prefix order property of $\mathcal{A}$. □

Proposition 2 *For any run R of any genuine atomic multicast algorithm $\mathcal{A}$ where one message is A-MCast at time t, processes are reactive at time t.*

Proof: In run R, by the genuineness property of $\mathcal{A}$, for any message m' sent, there exist events $e_1 = $ A-MCast(m), $e_2, ..., e_x = $ send(m') such that A-MCast(m) $\to e_2 \to ... \to$ send(m') (otherwise, using a similar argument as in Proposition 1, we could build a run R' identical to run R, except that no message is A-MCast in R', such that a process sends a message anyway, contradicting the fact that in R' no message is A-MCast and $\mathcal{A}$ is genuine).

Consequently, for any process p, if p sends a message m' at $t' \geq t$, then p A-MCasts m' or p received a message in the interval $[t, t']$. □

Proposition 3 *For any run R of any quiescent atomic broadcast or atomic multicast algorithm $\mathcal{A}$ in which a finite number of messages are A-XCast, there exists a time t such that for all $t' \geq t$, processes are reactive at t'.*

Proof: In R, a finite number of messages are A-XCast. Because $\mathcal{A}$ is quiescent, there exists a time t at or after which no messages are sent. It follows directly that for all $t' \geq t$ processes are reactive at t'. □

Although our result shows that if the last message m is cast when processes are reactive, then m cannot be delivered in one inter-group message delay, in practice, multiple messages may bear this overhead. In fact, this might even be the case in runs where an infinite number of messages are cast. Indeed, to ensure quiescence, processes must somehow *predict* whether any message will be cast in the future. Hence, if no message is expected to be cast, processes must stop communicating, and this may happen prematurely.

4 Atomic Multicast for WANs

In this section, we present a latency degree-optimal atomic multicast algorithm which is inspired by the one from Fritzke *et al.* [5], an adaptation of Skeen's algorithm for failure-prone systems. Due to space constraints, we here only present the basic principles of the algorithm, the pseudo-code as well as a detailed explanation can be found in [12].

4.1 Algorithm Overview

The algorithm associates every multicast message with a timestamp. To ensure agreement on the message delivery order, two properties are ensured: (1) processes agree on the message timestamps and (2) after a process p A-Delivers a message with timestamp ts, p does not A-Deliver a message with a smaller timestamp than ts. To satisfy these two properties, inside each group g, processes implement a logical *clock* that is used to generate timestamps—this is g's clock. To guarantee g's clock consistency, processes use consensus to maintain it. Moreover, every message m goes trough the following four stages:

  – *Stage s_0*: In every group $g \in m.dest$, processes define a timestamp for m using g's clock. This is g's proposal for m's final timestamp.
  – *Stage s_1*: Groups in $m.dest$ exchange their proposals for m's timestamp and set m's final timestamp to the maximum timestamp among all proposals.
  – *Stage s_2*: Every group in $m.dest$ sets its clock to a value greater than the final timestamp of m.
  – *Stage s_3*: Message m is A-Delivered when its timestamp is the smallest among all messages that are in one of the four stages and not yet A-Delivered.

As mentioned above, our algorithm differentiates itself from [5] in several aspects. First, when a message is multicast, instead of using a uniform reliable multicast primitive, we use a non-uniform version of this primitive while still ensuring properties as strong as in [5]. Second, in contrast to [5], not all messages go trough all four stages. Messages that are multicast to only one group can *jump* from stage s_0 to stage s_3. Moreover, even if a message m is multicast to more than one group, on processes belonging to the group that proposed the largest timestamp (i.e., m's final timestamp), m can skip stage s_2.

4.2 Latency Degree Analysis

Consider a message m that is multicast by a process p. In [12], we show that if m is multicast to one group, the latency degree of the algorithm, denoted as $\mathcal{A}1$, is zero if $p \in g$, and one otherwise. Moreover, if m is multicast to multiple groups, the latency degree is two, which matches the lower bound of Section 3.

Theorem 1 *There exists a run R of algorithm $\mathcal{A}1$ in which a message m is A-MCast to two groups such that $\Delta(m, R) = 2$.*

5   Atomic Broadcast for WANs

In this section, we present the first fault-tolerant atomic broadcast algorithm that achieves a latency degree of one. Together with the lower bound of Section 3, this shows that atomic multicast is more costly than atomic broadcast. Due to space constraints, we here only present an overview of the algorithm, the pseudo-code as well as a detailed explanation can be found in [12].

5.1   Algorithm Overview

To atomically broadcast a message m, a process p reliably multicasts m to the processes in p's group. In parallel, processes execute an *unbounded* sequence of rounds. At the end of each round, processes deliver a set of messages according to some deterministic order. To ensure agreement on the messages delivered in round r, processes proceed in two steps. In the first step, inside each group g, processes use consensus to define g's bundle of messages. In the second step, groups exchange their message bundles. The set of message delivered at the end of round r is the union of all bundles. Note that we also wish to ensure *quiescence*, i.e., if there is a time after which no message is broadcast, then processes eventually stop sending messages. To do so, processes try to predict when no further messages will be broadcast. Our prediction strategy is simple, it consists in checking, at the end of each round, whether any message was delivered or not. If no messages were delivered, processes stop executing rounds. Note that our algorithm is indulgent with regards to prediction mistakes, i.e., if processes become quiescent too early, they can restart so that liveness is still ensured.

5.2   Latency Degree Analysis

In [12], we analyze the latency degree of the algorithm, denoted as $\mathcal{A}2$. We first show that its best latency degree (among all its admissible runs) is one, which is optimal. We then consider runs where processes become quiescent too early, i.e., processes stop executing rounds before a message is broadcast. In these runs, the latency degree of the algorithm is two.

Theorem 2 *There exists a run R of algorithm $\mathcal{A}2$ in which a message m is A-BCast such that $\Delta(m, R) = 1$.*

Theorem 3 *There exists a run R of algorithm $\mathcal{A}2$ in which the last message m is A-BCast when processes are reactive such that $\Delta(m, R) = 2$.*

It is worth noting that the presented broadcast algorithm never becomes re-active if the time between two consecutive broadcasts is smaller than the time to execute a round. Moreover, in this case, all rounds are *useful*, i.e., they all deliver at least one message. For example, in a system where the inter-group latency is 100 milliseconds, a broadcast frequency of 10 messages per second is enough to obtain this desired behavior. In case the broadcast frequency is too low or not constant, to prevent processes from stopping prematurely, more elaborate prediction strategies based on application behavior could be used.

6   Related Work and Final Remarks

The literature on atomic broadcast and multicast algorithms is abundant [3]. We here review the most relevant papers to our protocols.

Atomic Multicast In [7], the authors show the impossibility of solving genuine atomic multicast with unreliable failure detectors when groups are allowed to intersect. Hence, the algorithms cited below circumvent this impossibility result by considering non-intersecting groups that contain a sufficient number of cor-rect processes to solve consensus. They can be viewed as variations of Skeen's algorithm [2], a multicast algorithm designed for failure-free systems, where messages are associated with timestamps and the message delivery follows the timestamp order. In [10], the addressees of a message m, i.e., the processes to which m is multicast, exchange the timestamp they assigned to m, and, once they receive this timestamp from a majority of processes of each group, they propose the maximum value received to consensus. Because consensus is run among the addressees of a message and can thus span multiple groups, this al-gorithm is not well-suited for wide area networks. In [4], consensus is run inside groups exclusively. Consider a message m that is multicast to groups $g_1, ..., g_k$. The first destination group of m, g_1, runs consensus to define the final times-tamp of m and hands over this message to group g_2. Every subsequent group proceeds similarly up to g_k. To avoid cycles in the message delivery order, be-fore handling other messages, every group waits for a final acknowledgment from group g_k. The latency degree of this algorithm is therefore proportional to the number of destination groups. In [5], to ensure that processes agree on the timestamps associated to every message and to deliver messages according to the timestamp order, every message goes through four stages. In contrast to [5], the algorithm presented in this paper allows messages to skip stages, therefore reducing the number of intra-group messages sent by sparing the execution of consensus instances.

Atomic Broadcast In [1], the authors consider the atomic broadcast and multi-cast problems in a publish-subscribe system where links are reliable, publishers

do not crash, and cast infinitely many messages. Agreement on the message ordering is ensured by using the same deterministic merge function at every subscriber process. Given the cast rate of publishers, the authors give optimal algorithms with regards to the merge delay, i.e., the time elapsed between the reception of a message by a subscriber and its delivery. Both algorithms achieve a latency degree of one.[4] In [13], a time-based protocol is introduced to increase the probability of *spontaneous* total order in wide area networks by artificially delaying messages. Although the latency degree of the *optimistic* delivery of a message is one, the latency degree of its *final* delivery is two. Moreover, their protocol is non-uniform, i.e., the *agreement* property of Section 2 is only ensured for correct processes. In [14], a uniform protocol based on multiple sequencers is proposed. Every process p is assigned a sequencer that associates sequence numbers to the messages p broadcasts. Processes optimistically deliver a message m when they receive m's sequence number. The final delivery of m occurs when the sequence number of m has been validated by a majority of processes. The latency degree of this algorithm is identical to [13].

In Figure 1, we compare the latency degree and the number of inter-group exchanged messages of the aforementioned algorithms. In this comparison, we consider the best-case scenario, in particular there is no failure nor failure suspicion. We denote n as the total number of processes in the system, d as the number of processes in each group, and k as the number of groups to which a message is cast ($k \geq 2$). To compute the latency degree and number of inter-group messages sent, we consider the oracle-based uniform reliable broadcast and uniform consensus algorithms of [6] and [11] respectively (note that [6] can easily be modified to implement reliable multicast). The latency degrees of [6] and [11] are respectively one and two. Furthermore, considering that a process p multicasts a message to k groups (we consider that p belongs to one of these k groups) or that k groups execute consensus, the algorithms respectively send $d(k-1)$ and $2kd(kd-1)$ inter-group messages.

From Figure 1, we conclude that, among uniform fault-tolerant broadcast protocols, Algorithm $\mathcal{A}2$ achieves the best latency degree and message complexity. In the case of the atomic multicast problem, although Algorithm $\mathcal{A}1$ and [5] achieve the best latency degree among fault-tolerant protocols, [4] has

---

[4] Note that this does not contradict the latency degree lower bound of genuine atomic multicast. Indeed, their assumptions are different than ours, i.e., to ensure liveness of their multicast algorithm, they require that each publisher multicast infinitely many messages to *each* subscriber.

[5] This paper considers a strong model where links are reliable, *multicaster* processes do not crash, and multicast infinitely many messages to every process.

[6] This algorithm is non-uniform, i.e., it guarantees the agreement property of Section 2 only for correct processes.

Algorithm	latency degree	inter-group msgs.
[4]	$k+1$	$O(kd^2)$
[10]	4	$O(k^2d^2)$
[5]	2	$O(k^2d^2)$
Algorithm $\mathcal{A}1$	2	$O(k^2d^2)$
[1][5]	1	$O(kd)$

(a) Atomic Multicast

Algorithm	latency degree	inter-group msgs.
[13][6]	2	$O(n)$
[14]	2	$O(n^2)$
Algorithm $\mathcal{A}2$	1	$O(n^2)$
[1][5]	1	$O(n)$

(b) Atomic Broadcast

Fig. 1. Comparison of the algorithms (d : nb. of processes per group, k : nb. of destination groups)

a lower message complexity. Deciding which algorithm is best is not straightforward as it depends on factors such as the network topology as well as the latencies and bandwidths of links.

References

1. M. K. Aguilera and R. E. Strom. Efficient atomic broadcast using deterministic merge. In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 209–218, New York, NY, USA, 2000. ACM Press.
2. K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.
3. X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.
4. C. Delporte-Gallet and H. Fauconnier. Fault-tolerant genuine atomic multicast to multiple groups. In *Proceedings of the 4th International Conference on Principles of Distributed Computing*, pages 107–122, 2000.
5. U. Fritzke, Ph. Ingels, A. Mostéfaoui, and M. Raynal. Fault-tolerant total order multicast to asynchronous groups. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pages 578–585, October 1998.
6. S. Frolund and F. Pedone. Ruminations on domain-based reliable broadcast. In *DISC '02: Proceedings of the 16th International Conference on Distributed Computing*, pages 148–162, London, UK, 2002. Springer-Verlag.
7. R. Guerraoui and A. Schiper. Genuine atomic multicast in asynchronous distributed systems. *Theor. Comput. Sci.*, 254(1-2):297–316, 2001.
8. V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In Sape J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.
9. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
10. L. Rodrigues, R. Guerraoui, and A. Schiper. Scalable atomic multicast. In *Proceedings of the 7th IEEE International Conference on Computer Communications and Networks (IC3N'98)*, pages 840–847, Lafayette, Louisiana, USA, 1998.
11. A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, 1997.
12. N. Schiper and F. Pedone. Optimal atomic broadcast and multicast algorithms for wide area networks. Technical Report 2007/004 Revision 1, University of Lugano, 2007.
13. A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, pages 190–199. IEEE CS, October 2002.
14. P. Vicente and L. Rodrigues. An indulgent uniform total order algorithm with optimistic delivery. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems*, page 92, Washington, DC, USA, 2002. IEEE Computer Society.