Genuine versus Non-Genuine Atomic Multicast Protocols for Wide Area Networks: An Empirical Study*

Nicolas Schiper† nicolas.schiper@usi.ch
Pierre Sutra‡ pierre.sutra@lip6.fr
Fernando Pedone† fernando.pedone@usi.ch

† University of Lugano, Switzerland ‡ Université Paris VI and INRIA Rocquencourt, France

Abstract

We study atomic multicast, a fundamental abstraction for building fault-tolerant systems. We suppose a system composed of data centers, or groups, that host many processes connected through high-end local links; a few groups exist, interconnected through high-latency communication links. A recent paper showed that no multicast protocol can deliver messages addressed to multiple groups in one inter-group delay and be genuine, i.e., to deliver a message m, only the addressees of m are involved in the protocol.

We propose a non-genuine multicast protocol that may deliver messages addressed to multiple groups in one inter-group delay. Experimental comparisons against a latency-optimal genuine protocol show that the non-genuine protocol offers better performance in almost all considered scenarios. We also identify a convoy effect in multicast algorithms that may delay the delivery of local messages, i.e., messages addressed to a single group, by as much as the latency of global messages, i.e., messages addressed to multiple groups, and propose techniques to minimize this effect. To complete our study, we evaluate a latency-optimal protocol that tolerates disasters, i.e., group crashes.

1 Introduction

Atomic broadcast and multicast are powerful group communication abstractions to build fault-tolerant distributed systems by means of data replication. Informally, they allow messages to be propagated to the group members and ensure agreement on the set of messages delivered and on their delivery order. As opposed to atomic broadcast, atomic multicast allows messages to be addressed to a subset of the members of the system. Multicast can thus be seen as the adequate abstraction for applications in which nodes replicate a subset of the data, i.e., partial replication.

In this paper we consider multicast protocols that span multiple geographical locations. We model the system as a set of *groups*, each one containing *processes* (e.g., data centers hosting local nodes). While a few groups exist, each one can host an arbitrary number of processes. Groups are interconnected through high-latency communication links; processes in a group are connected through high-end local links. The latency and bandwidth of intra- and inter-group links are separated by at least two orders of magnitude, and thus, inter-group links should be used sparingly.

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. Such a multicast algorithm is not *genuine* [10] since processes not addressed by the message are also involved in the protocol.

Although most multicast algorithms proposed in the literature are genuine (e.g., [9, 7, 15]), it has been shown that genuineness is an expensive property [16]: no genuine atomic multicast algorithm can deliver global messages in one inter-group message delay,[1] a limitation that is not imposed on non-genuine multicast algorithms. Therefore, when choosing a multicast algorithm, it seems natural to question *the circumstances under which a genuine algorithm is more efficient than a non-genuine algorithm.*

To answer the question, we take a two-step approach: First, we survey, classify, and analytically compare genuine and non-genuine multicast protocols. We identify two existing inter-group-latency optimal protocols: the *disaster-vulnerable* genuine algorithm in [16], which assumes that

*The work presented in this paper has been partially funded by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

[1] This lower bound is tight since the algorithms in [9, 16] can deliver messages in two inter-group delays.

each group contains one correct process, and the *disaster-tolerant* algorithm in [17]. We also introduce a non-genuine algorithm that is an optimization of [17] for the case of *correct groups*, i.e., groups that contain at least one correct process. The resulting algorithm may deliver global messages in a single-group delay (i.e., it is inter-group-latency optimal) and may deliver local messages with no inter-group communication. Second, we experimentally evaluate the three inter-group-latency optimal multicast protocols.

As part of the empirical study, we assess the scalability of the protocols by varying the number of groups, the proportion of global messages, and the load, i.e., the frequency at which messages are multicast. The results suggest that the genuineness of multicast is interesting only in large and highly loaded systems; in all the other considered scenarios the non-genuine protocol introduced in this paper outperforms the optimal genuine algorithm. We measure the overhead of the disaster-tolerant multicast protocol and observe that although it is in general more costly than the two other implemented protocols, it matches the performance of the genuine algorithm when there are few groups.

We also identify a *convoy effect* in multicast algorithms that may delay the delivery of local messages by as much as the latency of global messages. We then propose techniques to reduce this effect in [16, 17] as well as the non-genuine algorithm proposed in this paper. Although simple, these techniques decrease the delivery latency of local messages by as much as two orders of magnitude.

Summing up, this paper makes the following contributions: (a) it reviews and classifies atomic multicast algorithms, (b) it identifies a convoy effect that slows down the delivery of local messages and proposes techniques to remove this undesirable phenomenon, and (c) it empirically compares existing latency-optimal algorithms against a fast non-genuine protocol introduced in this paper. The results of this evaluation show that the non-genuine algorithm performs best except in large and highly loaded systems.

The rest of the paper is structured as follows. In Section 2, we introduce the system model and some definitions. Section 3 surveys and classifies the existing atomic multicast algorithms, and compares them analytically; the non-genuine multicast protocol is introduced in Section 4. Sections 5 and 6 respectively address the convoy effect and some implementation issues. The experimental evaluation of the protocols is presented in Section 7. We conclude the paper in Section 8.

2 System Model and Definitions

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: 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 nor 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. This assumption is not a limitation: [2] proposes an algorithm to build quasi-reliable links on top of fair lossy links, i.e., links that do not lose all sent messages.

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 each group can solve consensus, a problem defined below.

Specifications of Agreement Problems

Consensus. We assume the existence of a *uniform consensus* abstraction. 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 [11]: (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; and (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 groups in Γ. For each message m, $m.dst$ denotes the groups to which the message is reliably multicast. By abuse of notation, we write $p \in m.dst$ instead of $\exists g \in \Gamma : g \in m.dst \wedge p \in g$. 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 $p \in m.dst$ and m was previously R-MCast; (ii) *validity:* if a correct process p R-MCasts a message m, then eventually all correct processes $q \in m.dst$ R-Deliver m; and (iii) *uniform agreement:* if a process p R-Delivers a message m, then eventually all correct processes $q \in m.dst$ R-Deliver m.

Atomic Multicast. Uniform atomic multicast is defined by the primitives A-MCast and A-Deliver and satisfies all the properties of reliable multicast as well as *uniform prefix order:* for any two messages m and m' and any two processes p and q such that $\{p, q\} \subseteq m.dst \cap m'.dst$, if p A-Delivers m and q A-Delivers m', then either p A-Delivers m' before m or q A-Delivers m before m'.

In this context, we say that a message m is *local* iff it is addressed to one group only. On the other hand, if m is multicast to multiple groups, it is *global*.

We define the genuineness of an atomic multicast algorithm as follows [10]: an algorithm $\mathcal{A}$ solving atomic multicast is *genuine* iff for any admissible run R of $\mathcal{A}$ 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 $p \in m.dst$.

3 A Brief Survey of Multicast Algorithms

Although the literature on atomic broadcast algorithms is abundant [6], few atomic multicast protocols exist. In the following, we review, classify, and compare them analytically.

We identify three protocol categories: *timestamp-based*, *round-based*, and *ring-based*. *Timestamp-based* protocols can be viewed as variations of Skeen's algorithm [3], a multicast algorithm designed for failure-free systems. In this type of protocol, messages are assigned timestamps and two properties are ensured: (a) processes agree on the final timestamp of each message and (b) message delivery follows timestamp order. In *round-based* algorithms, processes execute an unbounded sequence of rounds and agree on the set of messages A-Delivered at the end of each round. *Ring-based* algorithms propagate messages along a predefined path, denoted as a ring, and ensure agreement on the message delivery by relying on this topology.

In addition to guaranteeing agreement on the message delivery order, protocols also need to prevent holes in the delivery sequence. Algorithms categorized as *round-based* and as *ring-based* get this property for free; for some *timestamp-based* algorithms ensuring this property requires extra work however.

To compare the algorithms, we use two criteria: best-case message delivery latency and inter-group message complexity. We compute these metrics by considering a failure-free scenario with no failure suspicions where a message is A-MCast by some process p to k groups, $k \geq 2$, including $group(p)$. We let δ be the inter-group message delay and assume that the intra-group delay is negligible.

In [10], 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 consider non-intersecting groups. We first review algorithms that assume that all groups contain at least one correct process, i.e., disaster-vulnerable algorithms, and then algorithms that tolerate group crashes, i.e., disaster-tolerant algorithms.

3.1 Disaster-vulnerable Algorithms

Timestamp-based. In [15], 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. Consensus and gathering timestamp proposals from a majority of processes in each group, respectively, ensure properties (a) and (b) of timestamp-based protocols. Once processes in each destination group of m decide on m's definitive timestamp, they exchange their history of messages, that is, messages that were reliably delivered or decided in consensus previously. This is to prevent holes in the delivery sequence. Because consensus is run among the addressees of a message and can thus span multiple groups, this algorithm is not well-suited for wide area networks. In the best case, global messages are A-Delivered within 4δ.

In [9], inside each group g, processes implement a logical *clock* that is used to generate timestamps; consensus is used among processes in g to maintain g's clock. Every multicast message m goes through four stages: s_0 through s_3. In s_0, in every group g addressed by m, processes define a timestamp for m using g's clock. This is g's proposal for m's final timestamp. In s_1, groups exchange their proposals and set m's final timestamp to the maximum among all proposals to ensure property (a). In the last two stages s_2 and s_3, the clock of g is updated to a value bigger than m's final timestamp and m is delivered when its timestamp is the smallest among all messages that are in one of the four stages. This guarantees property (b). Timestamps can be implemented in different ways. For example, each group g addressed by message m can define g's timestamp by using the consensus instance number that decides on m (stage s_1). Moreover, a consensus instance may decide on multiple messages, possibly in different stages. Since every consensus instance i may decide on messages in stage s_2, after deciding in i, g's clock is set to one plus the biggest message timestamp that i decided on. In the best case, messages are A-Delivered within 2δ, which is optimal for genuine multicast algorithms [16].

In contrast to [9], the algorithm in [16], hereafter denoted as $\mathcal{A}_{ge}$, allows messages to skip stages. Messages that are multicast to one group only 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 skips stage s_2. Since consensus instances are run inside groups, the best-case latency and the number of inter-group messages sent by $\mathcal{A}_{ge}$ are the same as in [9]. Nevertheless, we observe in Section 7.2 that these optimizations allow to reduce the *average* delivery latency under a broad range of loads.

Round-based. In this paper, we introduce a non-genuine algorithm, denoted as $\mathcal{A}_{ng}$, that is faster than [9] and [16]: it can A-Deliver messages within δ, which is obviously optimal. This protocol is a disaster-vulnerable version of the disaster-tolerant non-genuine algorithm in [17], described in Section 3.2. We present $\mathcal{A}_{ng}$ in Section 4.

Ring-based. In [7], consensus is run inside groups exclusively. Consider a message m that is multicast to groups $g_1, ..., g_k$. Message m is first multicast to g_1. Once a con-

sensus instance decides on m, members of g_1 A-Deliver m and hand over this message to group g_2. Every subsequent group proceeds similarly up to g_k. To ensure agreement on the message delivery order, before handling other messages, every group waits for a final acknowledgment from group g_k. Two messages addressed to groups $\{g_1, g_2, g_3\}$ and $\{g_2, g_3, g_4\}$ respectively will thus be ordered by g_2. The latency of this algorithm is $(k + 1)\delta$.

3.2 Disaster-tolerant Algorithms

To the best of our knowledge, [17] is the only paper to address the problem of multicast when groups can crash. This paper presents one genuine and one non-genuine protocol that are respectively timestamp-based and round-based.

Timestamp-based. This algorithm tolerates an arbitrary number of failures but requires perfect failure detection. To ensure properties (a) and (b) of timestamp-based multicast protocols, processes rely on a stronger form of consensus called global data computation [8]. This abstraction allows each process to propose a value, in this case a proposal timestamp for some message m, and ensures that processes agree on a vector V, with one entry per process, such that if a process p_i decides on V, then $V[i]$ must be equal to v_i, the value proposed by p_i. Otherwise, $V[i]$ may be equal to v_i or a special value $\perp$. After deciding on V, processes set m's final timestamp $m.ts$ to the maximum value of V and update their *local* logical clock to a value bigger than $m.ts$. To ensure a hole-free delivery sequence, processes rely on a causal multicast primitive [11]. This protocol has a latency of 6δ and it is an open question whether this is optimal for genuine disaster-tolerant protocols.

Round-based. This protocol, hereafter $\mathcal{A}_{dt}$, is not genuine and requires a two-third majority of correct processes, i.e., $f < n/3$, but only requires perfect failure detection inside each group. Unreliable failure detection can also be tolerated but at the cost of a weaker liveness guarantee.

To A-MCast a message m, a process p R-MCasts m to p's group. In parallel, processes execute an *unbounded* sequence of rounds. At the end of each round, processes A-Deliver a set of messages according to some deterministic order. To ensure agreement on the messages A-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. Local messages addressed to g can already be A-Delivered at this point. In the second step, groups exchange their message bundles. The set of message A-Delivered by some process p at the end of round r is the union of all bundles, restricted to messages addressed to p.

In case some group g crashes, the above protocol does not ensure liveness as there will be some round r after which no process receives the message bundles from g. To circumvent this problem $\mathcal{A}_{dt}$ proceeds in two steps: (i) it allows processes to stop waiting for g's message bundle, and (ii) it lets processes agree on the set of message bundles to consider for each round. This mechanism is similar to *group membership* [5], and is implemented using generic broadcast [13] for efficiency reasons, as we explain next.

To implement (i), processes maintain a common *view* of the groups that are trusted to be alive, i.e., groups that contain at least one alive process. Processes then wait for the message bundles from the groups currently in the view. A group g may be erroneously removed from the view, if it was mistakenly suspected of having crashed. Therefore, to ensure that message m multicast by a correct process will be delivered by all correct addressees of m, we allow members of g to add their group back to the view. To achieve (ii), processes agree on the sequence of views and the set of message bundles between each view change. For this purpose, we use a generic broadcast abstraction to propagate message bundles and view change messages, i.e., messages to add or remove groups.[2] Since message bundles can be delivered in different orders at different processes, provided that they are delivered between the same two view change messages, we define the message conflict relation as follows: view change messages conflict with all messages and message bundles only conflict with view change messages. As view change messages are not expected to be broadcast often, such a conflict relation definition allows for fast message delivery, i.e., within 2δ. As a corollary of the hyperfast learning Lemma of [12], this protocol is optimal.

3.3 Analytical Comparison

Figure 1 provides a comparison of the presented algorithms. To compute the inter-group message complexity, we assume that there are n processes in the system and that every group is composed of d processes. The third and the fifth column respectively indicate whether the protocol tolerates group crashes and whether it is latency-optimal.

It is worth noting that non-genuine atomic multicast could be solved with atomic broadcast protocols. In fact, the broadcast algorithms in [14] and [20] offer similar latencies as $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$ respectively. However, with the former algorithms local and global messages have the same latency; in the contrary, $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$ may deliver local messages without any inter-group communication.

4 Non-Genuine Atomic Multicast

In this section, we present the non-genuine multicast algorithm $\mathcal{A}_{ng}$, and explain the similarities and differences between $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$.

[2] Informally, generic broadcast ensures the same properties as atomic multicast except that messages are always addressed to all groups and only *conflicting* messages are totally ordered. That is, the uniform prefix order property of Section 2 is only ensured for messages that conflict.

Algorithm	genuine?	disaster tolerant?	latency	inter-group msgs.	optimal?
[7]	yes	no	$(k+1)\delta$	$O(kd^2)$	no
[15]	yes	no	4δ	$O(k^2d^2)$	no
[9], [16] ($\mathcal{A}_{ge}$)	yes	no	2δ	$O(k^2d^2)$	yes
this paper ($\mathcal{A}_{ng}$)	no	no	δ	$O(n^2)$	yes
[17]	yes	yes	6δ	$O(k^3d^3)$	?
[17] ($\mathcal{A}_{dt}$)	no	yes	2δ	$O(n^2)$	yes

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

Similarly to $\mathcal{A}_{dt}$, to atomic multicast a message m, a process p reliably multicasts m to p's group (line 4). In parallel, processes execute an unbounded sequence of rounds, and agree on the set of messages A-Delivered in each round. To do so, in each round r, members of each group g define g's message bundle using consensus (lines 8-9), A-Deliver local messages (line 12), and then exchange their message bundle with the other groups. At the end of round r, p A-Delivers the messages contained in the bundles of r that are addressed to p (lines 17-18).

Since $\mathcal{A}_{ng}$ assumes that groups are correct, i.e., groups contain at least one correct process, processes do not need to maintain a common *view* of groups that are deemed to be alive and can thus exchange message bundles using a regular send primitive (lines 13-16): the generic broadcast primitive used in $\mathcal{A}_{dt}$ is unnecessary. This allows $\mathcal{A}_{ng}$ to A-Deliver global messages in δ units of time, as opposed to 2δ for $\mathcal{A}_{dt}$.

Algorithm $\mathcal{A}_{ng}$ is a specialization of $\mathcal{A}_{dt}$ for the case of correct groups. The correctness of $\mathcal{A}_{ng}$ follows from the correctness of $\mathcal{A}_{dt}$, which is provided in [18].

Algorithm $\mathcal{A}_{ng}$

```
1:  Initialization
2:     K ← 1, Rdel ← ∅, Decided ← ∅

3:  To A-MCast message m                                   {Task 1}
4:     R-MCast m to group(p)

5:  When R-Deliver(m)                                      {Task 2}
6:     Rdel ← Rdel ∪ {m}

7:  Loop                                                   {Task 3}
8:     Propose(K, Rdel \ Decided)          ▷ consensus inside group
9:     wait until Decide(K, msgs)
10:    Decided ← Decided ∪ msgs
11:    lMsgs ← {m | m ∈ msgs ∧ m.dst = {group(p)}}
12:    A-Deliver messages in lMsgs in some deterministic order

13:    foreach g ∈ (Γ \ group(p))
14:       toSend ← {m | m ∈ msgs ∧ g ∈ m.dst}
15:       send(K, group(p), toSend) to members of g
16:    wait until ∀g ∈ (Γ \ group(p)) : received(K, g, msgs′)
17:    gMsgs ← {m | p ∈ m.dst ∧
                 ((m ∈ msgs ∧ |m.dst| > 1) ∨
                 (∃g ∈ Γ : received(K, g, msgs′) ∧ m ∈ msgs′))}
18:    A-Deliver messages in gMsgs in some deterministic order

19:    K ← K + 1
```

5 The Convoy Effect

The convoy effect refers to the phenomenon by which the delivery of a *local message* m_l is delayed by a *global message* m_g. This effect may happen for different reasons. In timestamp-based protocols, it may occur if m_g has a smaller timestamp than m_l's and m_g hasn't been A-Delivered yet; in round-based protocols, this undesired phenomenon may happen if m_l cannot be proposed to consensus as the message bundles of the current round are being exchanged.

We observed that all surveyed protocols suffer from this undesired behavior more or less severely but can be modified to remove this effect at least partially. Below, we illustrate how the convoy effect happens in the latency-optimal algorithms $\mathcal{A}_{ge}$, $\mathcal{A}_{ng}$, and $\mathcal{A}_{dt}$, and propose techniques to minimize it.

The timestamp-based algorithm $\mathcal{A}_{ge}$. Consider the following scenario in which a global message m_g delays the delivery of a local message m_l. Messages m_g and m_l are addressed to groups $\{g_1, g_2\}$ and g_1 respectively. Processes in g_1 R-Deliver m_g and define their proposal timestamp for m_g with consensus instance k_1. Shortly after, members of g_1 R-Deliver m_l and decide on m_l in consensus instance k_2, such that $k_2 > k_1$. Message m_l cannot be delivered at this point since m_g has a smaller timestamp than m_l and m_g has not been delivered yet. To deliver the local message m_l, members of g_1 must wait to receive g_2's timestamp proposal for m_g, which may take up to 2δ, if m_g was A-MCast from within g_1.

To deliver local messages faster, global and local messages are handled differently. Local messages are not assigned timestamps anymore and are A-Delivered directly after consensus. More precisely, when some process wishes to A-MCast a local message m_l to a group g, m_l is reliably multicast to g. In each group of the system, we run consensus instances to ensure agreement on the delivery order of local messages, as well as to assign timestamps to global messages as explained in Section 3. As soon as a consensus instance in g decides on m_l, members of g A-Deliver m_l.

To agree on the delivery order of global and local messages, global messages must be A-Delivered after the same consensus instance on members of the same group. To ensure this property, all global messages m_g must go through the four stages defined in Section 3, even in the group that proposed the highest timestamp for m_g. To understand why this is necessary, consider a global message m_g and a local message m_l that are respectively addressed to groups $\{g_1, g_2\}$ and g_1. Group g_1 is the group that assigned the highest timestamp to m_g. If we allow m_g to skip stage s_2 in g_1, two members p and q of g_1 may A-Deliver m_g and m_l in different orders. For example, assume p and q define m_g's proposal timestamp in a consensus instance k_1. Then, p receives g_2's timestamp for m_g, A-Delivers m_g, decides on m_l in a consensus instance k_2, and A-Delivers m_l. How-

ever, q first decides in consensus instance k_2, delivers m_l, receives g_2's timestamp proposal for m_g, and delivers m_g. We refer to this optimized version of $\mathcal{A}_{ge}$ as $\mathcal{A}^*_{ge}$.

The round-based algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$. In $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$, local messages may be delayed by global messages as much as one inter-group delay if they are multicast while the groups' bundle of messages are being exchanged. We can make this scenario unlikely to happen by executing multiple consensus instances per round. The number of consensus instances per round is denoted by parameter κ.

Although this optimization decreases the average delivery latency of local messages, the delivery latency of global messages, can now be increased by as many as κ - 1 consensus instances, this is because each group's bundle of messages is sent every κ consensus instance. Hence, to reduce the delivery latency of global messages, we allow rounds to overlap. That is, we start the next round before receiving the groups' bundle of messages of the current round. In other words, we execute consensus instances while the groups' bundle of messages are being exchanged. In our implementation, message bundles are exchanged after every η consensus instances.

To ensure agreement on the relative delivery order of local and global messages, it is necessary that processes inside the same group agree on when global messages of a given round are delivered, i.e., after which consensus instance. To summarize, processes send the message bundle of some round r after consensus instance $r \cdot \eta$ and A-Deliver messages of round r after instance $r \cdot \eta + \kappa$. We explore the influence of these parameters in Section 7.2.

6 Implementation Issues

We have implemented the three latency-optimal algorithms in Java, using a Paxos library as the consensus protocol [4]; all communication is based on TCP.

Inter-group communication represents a major source of overhead and should be used sparingly. In our implementation, these communications are handled by a dedicated layer. As we explain below, this layer optimizes the communication, reducing the number of inter-group messages.

Message Batching. Inside each group g, a special process is elected as leader [19]. Members of a group use their leader to batch and forward messages to the remote groups' leaders. When a leader receives a message m, it dispatches m to the members of its group.

Message Filtering. In each one of the presented algorithms, inter-group communication originating from processes of the same group g presents some redundancy. In the non-genuine Algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$, at the end of a round r, members of g send the same message bundle. Moreover, in the genuine Algorithm $\mathcal{A}_{ge}$, members of g send the same timestamp proposal for some message m. To avoid this redundancy, only the group leaders propagate these messages. More precisely, message bundles of Algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$, and the timestamp proposals of Algorithm $\mathcal{A}_{ge}$ are only sent by the group leaders. Messages sent by non-leader processes are discarded by the inter-group communication layer.

In the case of a leader failure, these optimizations may lead to the loss of some messages, which will be resent by the new leader.

7 Experimental evaluation

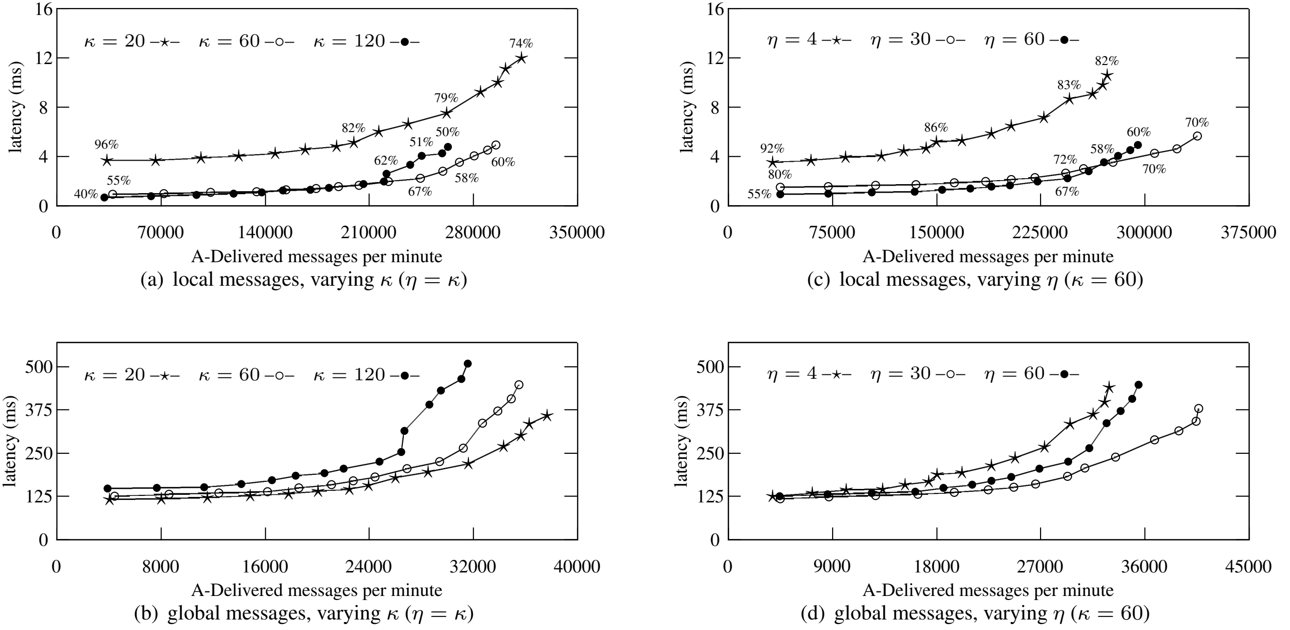
In this section, we evaluate experimentally the performance of the latency-optimal multicast protocols $\mathcal{A}_{ge}$, $\mathcal{A}_{ng}$, and $\mathcal{A}_{dt}$. We start by describing the system parameters and the benchmark used to assess the protocols. We then evaluate the influence of the convoy effect on the algorithms; compare the genuine and non-genuine protocols by varying the load imposed on the system, the number of groups, and the proportion of global messages; and measure the overhead of tolerating disasters.

7.1 Experimental Settings

The system. The experiments were conducted in a cluster of 24 nodes connected with a gigabit switch. Each node is equipped with two dual-core AMD Opteron 2 Ghz, 4GB of RAM, and runs Linux 2.6.20. In all experiments, each group consists of 3 nodes; the number of groups varies from 4 to 8. The bandwidth and message delay of our local network, measured using netperf and ping, are about 940 Mbps and 0.05 ms respectively. To emulate inter-group delays with higher latency and lower bandwidth, we used the Linux traffic shaping tools.

We emulated two network setups. In setup 1, the message delay between any two groups follows a normal distribution with a mean of 100 ms and a standard deviation of 5 ms, and each group is connected to the other groups via a 125 KBps (1 Mbps) full-duplex link. In setup 2, the message delay between any two groups follows a normal distribution with a mean of 20 ms and a standard deviation of 1 ms, and each group is connected to the other groups via a 1.25MBps (10 Mbps) full-duplex link. Due to space constraints, we only report the results using setup 1 and briefly comment on the behavior of the algorithms in setup 2.

The benchmark. The communication pattern of our benchmark was modeled after TPC-C, an industry standard benchmark for on-line transaction processing (OLTP) [1]. TPC-C represents a generic wholesale supplier workload and is composed of five predefined transaction types. Two out of these five types may access multiple warehouses; the other three types access one warehouse only. We assume that each group hosts one warehouse. Hence, the warehouses involved in the execution of a transaction define to which groups the transaction is multicast. Each multicast



(a) local messages, varying κ ($\eta = \kappa$)

(c) local messages, varying η ($\kappa = 60$)

(b) global messages, varying κ ($\eta = \kappa$)

(d) global messages, varying η ($\kappa = 60$)

Figure 2. The influence of κ and η on $\mathcal{A}_{ng}$ with four groups (percentages show convoy effect).

message also contain the transaction's parameters and on average, a message contains 80 bytes of payload.

In TPC-C, about 10% of transactions involve multiple warehouses. Thus, roughly 10% of messages are global. To assess the scalability of our protocols, we parameterize the benchmark to control the proportion p of global messages. In the experiments, we report measurements for $p = 0.1$ (i.e., original TPC-C) and $p = 0.5$. The vast majority of global messages involve two groups. Note that in our benchmark, transactions are not executed; TPC-C is only used to determine the communication pattern.

Each node of the system contains an equal number of clients executing the benchmark in a closed loop: each client multicasts a message and waits for its delivery before multicasting another message. Hence, all messages always address the sender's group; global messages also address other groups. The number of clients per node varies between 1 and 160 and in each experiment, at least one hundred thousand messages are multicast.

For all the experiments, we report either the average message delivery latency (in milliseconds) or the average inter-group bandwidth (in kilo bytes per second) as a function of the throughput, i.e., the number of messages A-Delivered per minute. We computed 95% confidence intervals for the A-delivery latency but we do not report them here as they were always smaller than 2% of the average latency. The throughput was increased by adding an equal number of clients to each node of the system.

7.2 Assessing the Convoy Effect

The round-based Algorithms $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$. We explore the influence of parameters κ, the number of consensus instances per round, and η, the number of consensus instances between two consecutive message bundle exchanges, on the convoy effect. In principle, the higher the value of κ, the less likely the convoy effect is to happen. Indeed, the more consensus instances are run per round, the less probable it is that a local message waits for the message bundles to be exchanged. However, increasing κ also increases the average latency of global messages. The optimal value of κ should be set to allow groups to execute as many local consensus instances as they can while message bundles are being exchanged, to minimize the convoy effect, without affecting the latency of global messages. If we let Δ_{max} and δ_{cons} respectively denote the maximum inter-group delay and the consensus latency, κ should be set to $\lfloor \Delta_{max}/\delta_{cons} \rfloor$. Setting parameter η is less trivial: setting it low potentially decreases the average global message latency but may also saturate the inter-group network.

In failure-free runs, $\mathcal{A}_{ng}$ and $\mathcal{A}_{dt}$ only differ on how the message bundles are exchanged, we thus explore the influence of these parameters on $\mathcal{A}_{ng}$ only. Figures 2(a) and 2(b) illustrate the impact of κ on $\mathcal{A}_{ng}$ in a system with four groups when rounds do not overlap (i.e., $\eta = \kappa$). We report the average percentage of the local message latency that is due to the convoy effect. To make the figures easily readable, we do so only for certain loads.

As explained above, using very low or high values of κ respectively increases the chances of the convoy ef-

fect to happen or the latency of global messages. Hence, we use values of κ close to the theoretical optimum of $\lfloor \Delta_{max}/\delta_{cons} \rfloor$.

We first consider local messages, and note that setting κ too low (i.e., $\kappa = 20$ in our experiments) or too high ($\kappa = 120$) respectively increases the chances of the convoy effect to happen, which impacts the latency, or harms the scalability of the protocol: as the number of global messages sent at the end of each round increased, the inter-group communication became the bottleneck since the traffic was too bursty. In our settings, $\kappa = 60$ gave the best results for local messages (see Figure 2(a)).

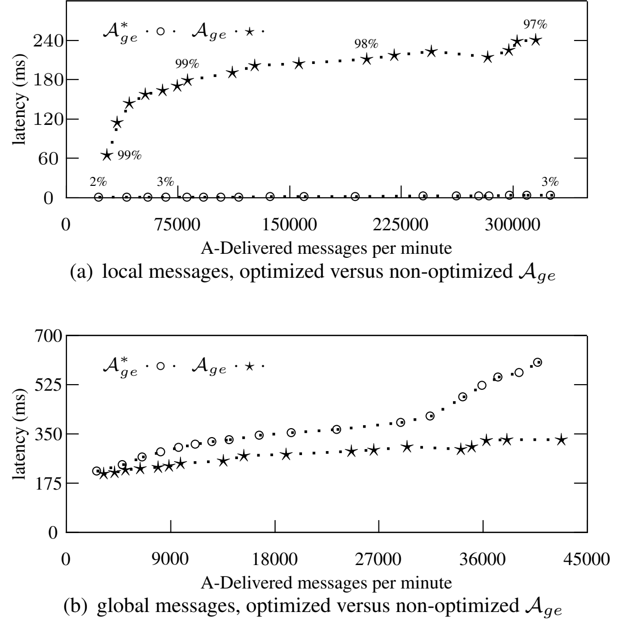
For global messages, setting κ to 60 gives worse performance than setting it to 20 (see Figure 2(b)). We address this problem by tuning parameter η, as illustrated in Figures 2(c) and 2(d). While setting η too low ($\eta = 4$) worsens the latency and the scalability of the protocol, and increases the convoy effect, setting it too high ($\eta = 60$) harms its scalability. When $\eta = 30$, the local message latency is similar to the one when $\kappa = \eta = 60$ (Figure 2(a)), while almost matching the global message latency of $\kappa = \eta = 20$ (Figure 2(b)). Moreover, with respect to Figures 2(a) and 2(b), the scalability of the protocol is improved for both local and global messages.

To further reduce the global message latency, we tried other values for η. However, we did not find a value that gave better performance than $\eta = 30$ nor did we reach the theoretical optimum latency of one δ, i.e., 100 ms. We observed that this was mainly because groups do not start rounds exactly at the same time; consequently, some groups had to wait more than 100 milliseconds to receive all message bundles. Therefore, in all experiments that follow we used $\kappa = 60$ and $\eta = 30$.

The timestamp-based Algorithm $\mathcal{A}_{ge}$. Figures 3(a) and 3(b) evaluate the influence of the convoy effect on $\mathcal{A}_{ge}$ in a system with four groups. Similarly as above, we report the average percentage of the local message latency that is due to the convoy effect. Algorithm $\mathcal{A}^*_{ge}$ delivers local messages much faster than its non-optimized counterpart prone to the convoy effect $\mathcal{A}_{ge}$ (see Figure 3(a)): as the load increases the convoy effect happens more frequently and increases the local message latency until it reaches the latency of global messages, i.e., around 200 ms.

In Figure 3(b), we observe that $\mathcal{A}^*_{ge}$ slows down the delivery of global messages. This phenomenon has two causes. First, all global messages now go through the four stages, thus, an increased number of consensus instances must be run for the same throughput. Second, as an effect of the first cause, global messages have a higher chance to be delayed by other global messages. This is similar to the convoy effect, but for global messages.

These observations underline the importance of allowing global messages to skip stage s_2, an optimization that is present in $\mathcal{A}_{ge}$, but not allowed in $\mathcal{A}^*_{ge}$ (c.f. Section 5), and render $\mathcal{A}^*_{ge}$ interesting only when the decrease in local message latency matters more than the increase in global message latency. As we expect the proportion of local messages to be higher than the proportion of global messages, an assumption that is verified by TPC-C, we only consider $\mathcal{A}^*_{ge}$ hereafter.



(a) local messages, optimized versus non-optimized $\mathcal{A}_{ge}$

(b) global messages, optimized versus non-optimized $\mathcal{A}_{ge}$

Figure 3. The convoy effect in $\mathcal{A}_{ge}$ with four groups (percentages show convoy effect).

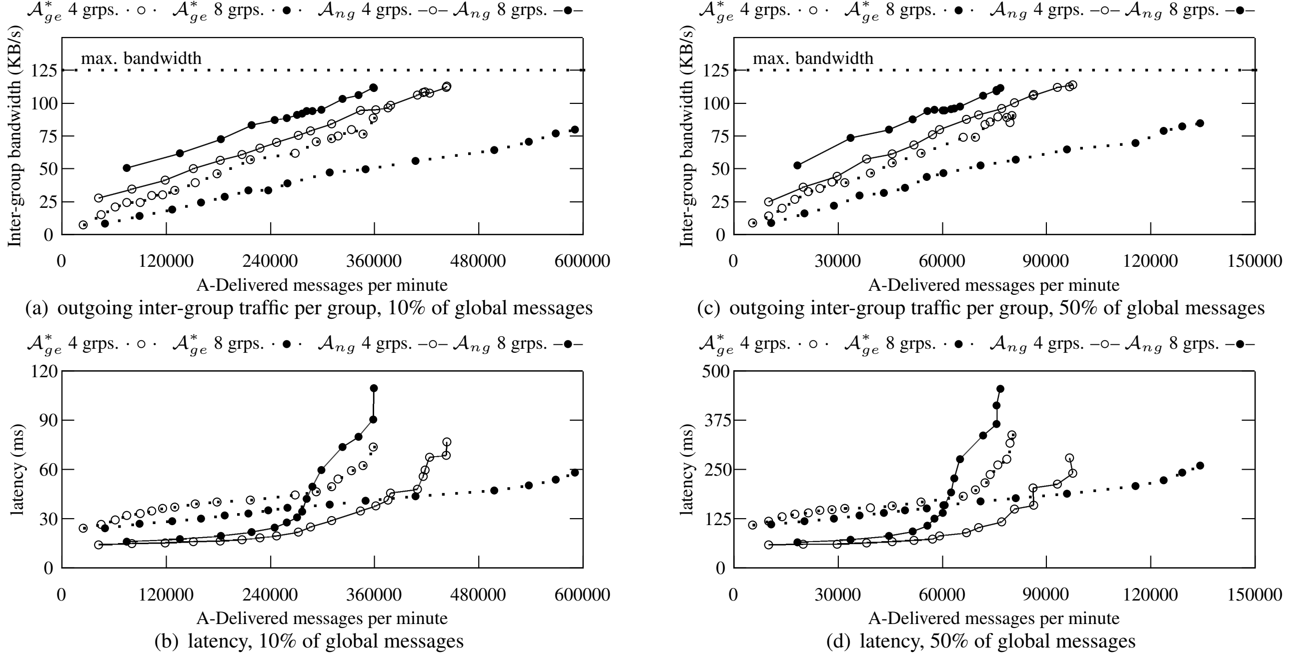
7.3 Genuine vs. Non-Genuine Multicast

We compare $\mathcal{A}_{ng}$ to $\mathcal{A}^*_{ge}$ when the number of groups increases using two mixes of global and local messages. We first set the proportion of global messages to 10% and run the algorithms in a system with four and eight groups. Figures 4(a) and 4(b) respectively report the average outgoing inter-group traffic per group and the average A-Delivery latency, both as a function of the throughput. For brevity we report the overall average delivery latency, without differentiating between local and global messages.

Although $\mathcal{A}_{ng}$ exhibits a better latency than $\mathcal{A}^*_{ge}$ with 4 groups, $\mathcal{A}_{ng}$ does not scale as well as $\mathcal{A}^*_{ge}$ with eight groups (Figure 4(b)). This is a consequence of $\mathcal{A}_{ng}$'s higher demand on throughput: with eight groups the algorithm requires as much as 111 KBps of average inter-group bandwidth, a value close to the maximum available capacity of 125 KBps (Figure 4(a)).

In Figures 4(c) and 4(d), we observe that when half of the messages are global, the two algorithms compare similarly as above but do not scale as well.

As a final remark, we note that in contrast to $\mathcal{A}_{ng}$, $\mathcal{A}^*_{ge}$ delivers messages faster and supports more load with eight



(a) outgoing inter-group traffic per group, 10% of global messages

(b) latency, 10% of global messages

(c) outgoing inter-group traffic per group, 50% of global messages

(d) latency, 50% of global messages

Figure 4. Genuine versus Non-Genuine Multicast.

groups than with four (Figures 4(b) and 4(d)). Indeed, increasing the number of groups decreases the load that each group must handle as, in our benchmark, the vast majority of global messages are addressed to two groups. This effect can be seen in Figures 4(a) and 4(c), where each group needs less inter-group bandwidth with eight groups.

Summary. Figure 5 provides a qualitative comparison between the genuine and non-genuine algorithms. We consider four scenarios generated by all combinations of the two following parameters: the load (high or low) and the number of groups (many or few); the proportion of global messages is not taken into account as it has no influence on the comparison. We note that $\mathcal{A}_{ng}$ is the winner except when the load is high and there are many groups.

We also carried out the same comparison in network setup 2, i.e., a network where the message delay between any two groups follows a normal distribution with a mean of 20 ms and a standard deviation of 1 ms, and each group is connected to the other groups via a 1.25MBps (10 Mbps) full-duplex link. Due to space constraints we only briefly comment on the obtained results. With 4 groups, $\mathcal{A}^*_{ge}$ and $\mathcal{A}_{ng}$ compare similarly as in setup 1. Due to the lower inter-group latency the performance of $\mathcal{A}^*_{ge}$ becomes closer to the one of $\mathcal{A}_{ng}$ however. With eight groups, the non-genuine protocol scales almost as well as the genuine algorithm thanks to the extra available inter-group bandwidth.

7.4 The Cost of Tolerating Disasters

To evaluate the overhead of tolerating disasters, we compare $\mathcal{A}_{dt}$ to the overall best-performing disaster-vulnerable algorithm $\mathcal{A}_{ng}$. With $\mathcal{A}_{dt}$ we set κ and η to 120 and 60 respectively; with $\mathcal{A}_{ng}$, the default values are used, $\kappa = 60$ and $\eta = 30$.

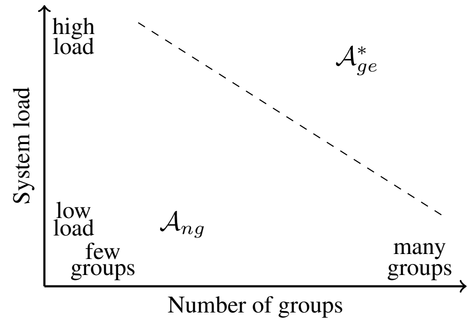
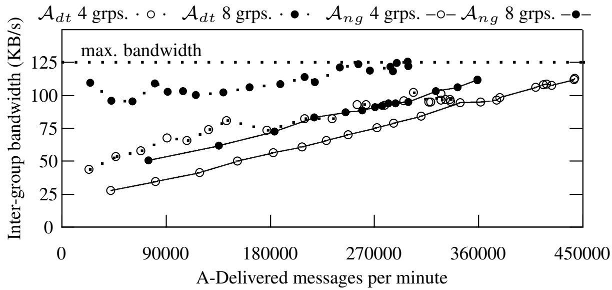


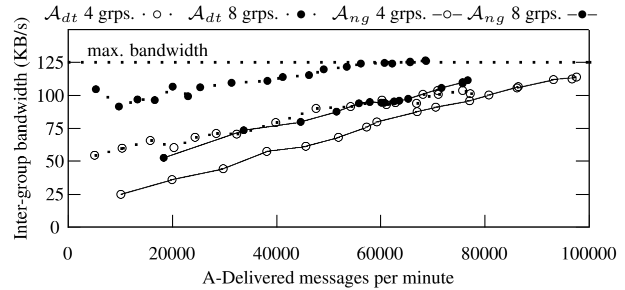
Figure 5. Comparing $\mathcal{A}^*_{ge}$ to $\mathcal{A}_{ng}$.

In Figures 6(b) and 6(d), we observe that with four groups, $\mathcal{A}_{dt}$ roughly needs twice as much time as $\mathcal{A}_{ng}$ to deliver messages. This is expected: local messages take about the same time to be delivered with the two algorithms; global messages roughly need and additional 100 milliseconds to be delivered with $\mathcal{A}_{dt}$. Interestingly, $\mathcal{A}_{dt}$ matches the performance of $\mathcal{A}^*_{ge}$ in a system with four groups (Figures 4(b), 6(b), 4(d), and 6(d)).

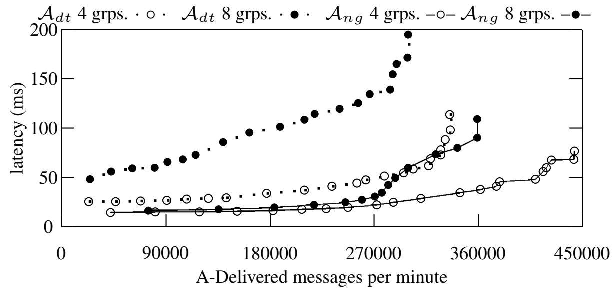
With eight groups, $\mathcal{A}_{dt}$ utilizes the entire inter-group bandwidth under almost every considered load (Figures 6(a) and 6(c)). The latency and scalability of the disaster-tolerant algorithm thus become much worse than $\mathcal{A}_{ng}$'s.



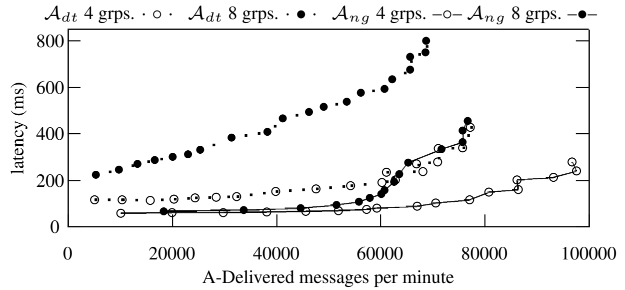
(a) outgoing inter-group traffic per group, 10% of global messages



(c) outgoing inter-group traffic per group, 50% of global messages



(b) latency, 10% of global messages



(d) latency, 50% of global messages

Figure 6. The cost of tolerating disasters.

8 Conclusion

In this paper, we first surveyed and compared multicast protocols. We then proposed a non-genuine multicast algorithm that offers better performance than the latency-optimal genuine algorithm of [16], except in large and highly loaded systems. We also identified a convoy effect that delays the delivery of local messages and proposed techniques to reduce this effect. To complete our study, we showed that the disaster-tolerant protocol [17] does not offer the same level of performance as the non-genuine algorithm introduced in this paper but matches the performance of [16] when there are few groups.

References

[1] Transaction processing performance council (tpc) - benchmark c. http://www.tpc.org/tpcc/.

[2] M. K. Aguilera, W. Chen, and S. Toueg. On quiescent reliable communication. *SIAM J. Comput.*, 29(6):2040–2073, 2000.

[3] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.

[4] L. Camargos. http://sourceforge.net/projects/daisylib/.

[5] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[6] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[7] C. Delporte-Gallet and H. Fauconnier. Fault-tolerant genuine atomic multicast to multiple groups. In *OPODIS*, pages 107–122. Suger, Saint-Denis, rue Catulienne, France, 2000.

[8] C. Delporte-Gallet, H. Fauconnier, J.-M. Helary, and M. Raynal. Early stopping in global data computation. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):909–921, 2003.

[9] U. Fritzke, P. Ingels, A. Mostéfaoui, and M. Raynal. Fault-tolerant total order multicast to asynchronous groups. In *SRDS*, pages 578–585. IEEE Computer Society, 1998.

[10] R. Guerraoui and A. Schiper. Genuine atomic multicast in asynchronous distributed systems. *Theoretical Comput. Sci.*, 254(1-2):297–316, 2001.

[11] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

[12] L. Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, 2006.

[13] F. Pedone and A. Schiper. Handling message semantics with generic broadcast protocols. *Distributed Computing*, 15(2):97–107, 2002.

[14] L. Rodrigues, H. Fonseca, and P. Verissimo. Totally ordered multicast in large-scale systems. In *ICDCS*, page 503, Los Alamitos, CA, USA, 1996. IEEE Computer Society.

[15] L. Rodrigues, R. Guerraoui, and A. Schiper. Scalable atomic multicast. In *IC3N*, pages 840–847. IEEE, 1998.

[16] N. Schiper and F. Pedone. On the inherent cost of atomic broadcast and multicast in wide area networks. In *ICDCN*, pages 147–157. Springer, 2008.

[17] N. Schiper and F. Pedone. Solving atomic multicast when groups crash. In *OPODIS*, pages 481–495, 2008.

[18] N. Schiper and F. Pedone. Solving atomic multicast when groups crash. Technical Report 2008/002, University of Lugano, 2008.

[19] N. Schiper and S. Toueg. A robust and lightweight stable leader election service for dynamic systems. In *DSN*, pages 207–216. IEEE Computer Society, 2008.

[20] P. Zielinski. Low-latency atomic broadcast in the presence of contention. *Distributed Computing*, 20(6):435–450, 2008.