Solving Atomic Multicast when Groups Crash

Nicolas Schiper[†] Fernando Pedone[†]

[†]Faculty of Informatics
University of Lugano
6900 Lugano, Switzerland

University of Lugano
Faculty of Informatics
Technical Report No. 2008/002
July 2008

Abstract

In this paper, we study the atomic multicast problem, a fundamental abstraction for building fault-tolerant systems. In the atomic multicast problem, the system is divided into non-empty and disjoint *groups of processes*. Multicast messages may be addressed to any subset of groups, each message possibly being multicast to a different subset. Several papers previously studied this problem either in local area networks [3, 9, 20] or wide area networks [13, 21]. However, none of them considered atomic multicast when groups may crash. We present two atomic multicast algorithms that tolerate the crash of groups. The first algorithm tolerates an arbitrary number of failures, is genuine (i.e., to deliver a message m, only addressees of m are involved in the protocol), and uses the perfect failures detector $\mathcal{P}$. We show that among realistic failure detectors, i.e., those that do not predict the future, $\mathcal{P}$ is necessary to solve genuine atomic multicast if we do not bound the number of processes that may fail. Thus, $\mathcal{P}$ is the *weakest* realistic failure detector for solving genuine atomic multicast when an arbitrary number of processes may crash. Our second algorithm is non-genuine and less resilient to process failures than the first algorithm but has several advantages: (i) it requires perfect failure detection within groups only, and not across the system, (ii) as we show in the paper it can be modified to rely on unreliable failure detection at the cost of a weaker liveness guarantee, and (iii) it is fast, messages addressed to multiple groups may be delivered within two inter-group message delays only.

1 Introduction

Mission-critical distributed applications typically replicate data in different data centers. These data centers are spread over a large geographical area to provide maximum availability despite natural disasters. Each data center, or *group*, may host a large number of processes connected through a fast local network; a few groups exist, interconnected through high-latency communication links. Application data is replicated locally, for high availability despite the crash of processes in a group, and globally, for locality of access and high availability despite the crash of an entire group.

Atomic multicast is a communication primitive that offers adequate properties, namely agreement on the set of messages delivered and on their delivery order, to implement partial data replication [16, 22]. As opposed to atomic broadcast [15], atomic multicast allows messages to be addressed to any subset of the groups in the system. For efficiency purposes, multicast protocols should be *genuine* [14], i.e., only the addressees of some message m should participate in the protocol to deliver m. This property rules out the trivial reduction of atomic multicast to atomic broadcast where every message m is broadcast to all groups in the system and only delivered by the addressees of m.

Previous work on atomic multicast [3, 20, 9, 13, 21] all assume that, inside each group, there exists at least one non-faulty process. We here do not make this assumption and allow groups to entirely crash. To the best of our knowledge, this is the first paper to investigate atomic multicast in such a scenario.

The atomic multicast algorithms we present in this paper use oracles that provide possibly inaccurate information about process failures, i.e., failure detectors [6]. Failure detectors are defined by the properties they guarantee on the set of trusted (or suspected) processes they output. Ideally, we would like to find the *weakest* failure detector $\mathcal{D}_{amcast}$ for genuine atomic multicast. Intuitively, $\mathcal{D}_{amcast}$ provides just enough information about process failures to solve genuine atomic multicast but not more. More formally, a failure detector $\mathcal{D}_1$ is at least as strong as a failure detector $\mathcal{D}_2$, denoted as $\mathcal{D}_1 \succeq \mathcal{D}_2$, if and only if there exists an algorithm that implements $\mathcal{D}_2$ using $\mathcal{D}_1$, i.e., the algorithm emulates the output of $\mathcal{D}_2$ using $\mathcal{D}_1$. $\mathcal{D}_{amcast}$ is the weakest failure detector for genuine atomic multicast if two conditions are met: we can use $\mathcal{D}_{amcast}$ to solve genuine atomic multicast (sufficiency) and any failure detector $\mathcal{D}$ that can be used to solve genuine atomic multicast is at least as strong as $\mathcal{D}_{amcast}$, i.e., $\mathcal{D} \succeq \mathcal{D}_{amcast}$ (necessity) [5].

We here consider *realistic* failure detectors only, i.e., those that cannot predict the future [10]. Moreover, we do not assume any bound on the number of processes that can crash. In this context, Delporte *et al.* showed in [10] that the weakest failure detector $\mathcal{D}_{cons}$ for consensus may not make any mistakes about the alive status of processes, i.e., it may not stop trusting a process before it crashes.[1] Additionally, $\mathcal{D}_{cons}$ must eventually stop trusting all crashed processes. In the literature, $\mathcal{D}_{cons}$ is denoted as the perfect failure detector $\mathcal{P}$. Obviously, atomic multicast allows to solve consensus: every process atomically multicasts its proposal; the decision of consensus is the first delivered message. Hence, the weakest realistic failure detector to solve genuine atomic multicast $\mathcal{D}_{amcast}$ when the number of faulty processes is not bounded is at least as strong as $\mathcal{P}$, i.e., $\mathcal{D}_{amcast} \succeq \mathcal{P}$. We show that $\mathcal{P}$ is in fact the weakest realistic failure detector for genuine atomic multicast when an arbitrary number of processes may fail by presenting an algorithm that solves the problem using perfect failure detection.

As implementing $\mathcal{P}$ seems hard, if not impossible, in certain settings (e.g., wide area networks), we revisit the problem from a different angle: we consider non-genuine atomic multicast algorithms. For this purpose, as noted above, atomic broadcast could be used. This solution, however, is of little practical interest as delivering messages requires all processes to communicate, even for messages multicast to a single group. The second algorithm we present does not suffer from this problem: messages multicast to a single group

[1]Intuitively, consensus allows each process to propose a value and guarantees that processes eventually decide on one common value.

g may be delivered without communication between processes outside g. Moreover, our second algorithm offers some advantages when compared to our first algorithm, based on $\mathcal{P}$: Wide-area communication links are used sparingly, messages addressed to multiple groups can be delivered within two inter-group message delays, and perfect failure detection is only required within groups and not across the system. Although this assumption is more reasonable than implementing $\mathcal{P}$ in a wide area network, it may still be too strong for some systems. Thus, we discuss a modification to the algorithm that tolerates unreliable failure detection, at the cost of a weaker liveness guarantee. The price to pay for the valuable features of this second algorithm is a lower process failure resiliency: group crashes are still tolerated provided that *enough* processes in the whole system are correct.

Contribution In this paper, we make the following contributions. We present two atomic multicast algorithms that tolerate group crashes. The first algorithm is genuine, tolerates an arbitrary number of failures, and requires perfect failure detection. The second algorithm is non-genuine but only requires perfect failure detection inside each group and may deliver messages addressed to multiple groups in two inter-group message delays. We present a modification to the algorithm to cope with unreliable failure detection.

Road map The rest of the paper is structured as follows. Section 2 reviews the related work. In Section 3 our system model and definitions are introduced. Sections 4 and 5 present the two atomic multicast algorithms. Finally, Section 6 concludes the paper. The proof of correctness of the algorithms can be found in the Appendix.

2 Related Work

The literature on atomic broadcast and multicast algorithms is abundant [8]. We briefly review some of the relevant papers on atomic multicast.

In [14], 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. Moreover, they all assume that groups do not crash, i.e., there exists at least one correct process inside each group.

These algorithms can be viewed as variations of Skeen's algorithm [3], a multicast algorithm designed for failure-free systems, where messages are associated with timestamps and the message delivery follows the timestamp order. In [20], 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 algorithm is not well-suited for wide area networks. In [9], 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 timestamp of m and hands 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. In [13], inside each group g, processes implement a logical *clock* that is used to generate timestamps, this is g's clock (consensus is used among processes in g to maintain g's clock). Every multicast message m goes through four stages. In the first stage, 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. Groups then exchange their proposals and set m's final timestamp to the maximum among all proposals. In the last two stages, 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. In [21], the authors present an optimization of [13] that allows messages to skip the second and third stages in certain conditions, therefore sparing the execution of consensus instances. The algorithms of [13, 21] can deliver messages in two inter-group message delays; [21] shows that this is optimal.

To the best of our knowledge, this is the first paper that investigates the solvability of atomic multicast when groups may entirely crash. Two algorithms are presented: the first one is genuine but requires system-wide perfect failure detection. The second algorithms is not genuine but only requires perfect failure detection inside groups.

3 Problem Definition

3.1 System Model

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. Processes may however access failure detectors [6]. 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 maximum number of processes that may crash is denoted by f. 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. 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. A group g that contains at least one correct process is correct; otherwise g is faulty.

3.2 Atomic Multicast

Atomic multicast allows messages to be A-MCast to any subset of groups in Γ. For every message m, $m.dst$ denotes the groups to which m is multicast. Let p be a process. By abuse of notation, we write $p \in m.dst$ instead of $\exists g \in \Gamma : g \in m.dst \wedge p \in g$. Atomic multicast is defined by the primitives A-MCast and A-Deliver and satisfies the following properties: (i) *uniform integrity:* For any process p and any message m, p A-Delivers m at most once, and only if $p \in m.dst$ and m was previously A-MCast, (ii) *validity:* if a correct process p A-MCasts a message m, then eventually all correct processes $q \in m.dst$ A-Deliver m, (iii) *uniform agreement:* if a process p A-Delivers a message m, then all correct processes $q \in m.dst$ eventually A-Deliver m, and (iv) *uniform prefix order:* for any two messages m and m' and any two processes p and q such that $\{p, q\} \in m.dst \cap m'.dst$, 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'.

Let $\mathcal{A}$ be an algorithm solving atomic multicast. We define $\mathcal{R}(\mathcal{A})$ as the set of all admissible runs of $\mathcal{A}$. We require atomic multicast algorithms to be *genuine* [14]:

- *Genuineness*: 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 $p \in m.dst$.

4 Solving Atomic Multicast with a Perfect Failure Detector

In this section, we present the first genuine atomic multicast algorithm that tolerates an arbitrary number of process failures, i.e., $f \leq n$. We first define additional abstractions used in the algorithm, then explain the mechanisms to ensure agreement on the delivery order, and finally, we present the algorithm itself.

4.1 Additional Definitions and Assumptions

Failure Detector $\mathcal{P}$ We assume that processes have access to the perfect failure detector $\mathcal{P}$ [6]. This failure detector outputs a list of trusted processes and satisfies the following properties[2]: (i) *strong completeness:* eventually no faulty process is ever trusted by any correct process and (ii) *strong accuracy:* no process stops being trusted before it crashes.

Causal Multicast The algorithm we present below uses a *causal multicast* abstraction. Causal multicast is defined by primitives ***C-MCast(m)*** and ***C-Deliver(m)***, and satisfies the uniform integrity, validity, and uniform agreement properties of atomic multicast as well as the following *uniform causal order* property: for any messages m and m', if C-MCast(m) $\rightarrow$ C-MCast(m'), then no process $p \in m.dst \cap m'.dst$ C-Delivers m' unless it has previously C-Delivered m.[3] To the best of our knowledge, no algorithm implementing this specification of causal multicast exists. We thus present a genuine causal multicast algorithm that tolerates an arbitrary number of failures in the Appendix.[4]

Global Data Computation We also assume the existence of a *global data computation* abstraction [12]. The global data computation problem consists in providing each process with the same vector V, with one entry per process, such that each entry is filled with a value provided by the corresponding process. Global data computation is defined by the primitives propose(v) and decide(V) and satisfies the following properties: (i) *uniform validity:* if a process p decides V, then $\forall q : V[q] \in \{v_q, \perp\}$, where v_q is q's proposal, (ii) *termination:* if every correct process proposes a value, then every correct process eventually decides one vector, (iii) *uniform agreement:* if a process p decides V, then all correct processes q eventually decide V, and (iv) *uniform obligation:* if a process p decides V, then $V[p] = v_p$. An algorithm that solves global data computation using the perfect failure detector $\mathcal{P}$ appears in [12]. This algorithm tolerates an arbitrary number of failures.

4.2 Agreeing on the Delivery Order

The algorithm associates every multicast message with a timestamp. To guarantee 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. These properties are implemented as described next.

For simplicity, we initially assume a multicast primitive that guarantees agreement on the set of messages processes deliver, but not causal order; we then show how this algorithm may incur into problems, which

[2]Historically, $\mathcal{P}$ was defined to output a set of suspected processes. We here define its output as a set of trusted processes, i.e., in our definition the output corresponds to the complement of the output in the original definition.

[3]The relation $\rightarrow$ is Lamport's transitive happened before relation on events [17]. Here, events can be of two types, C-MCast or C-Deliver. The relation 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 $e_1 =$ C-MCast(m) and $e_2 =$ C-Deliver(m) for some message m.

[4]The genuineness of causal multicast is defined in a similar way as for atomic multicast.

can be solved using causal multicast. To A-MCast a message m_1, m_1 is thus first multicast to the addressees of m_1. Upon delivery of m_1, every process p uses a local variable, denoted as TS_p, to define its proposal for m_1's timestamp, $m_1.ts_p$. Process p then proposes $m_1.ts_p$ in m_1's global data computation (gdc) instance. The definitive timestamp of m_1, $m_1.ts^{def}$, is the maximum value of the decided vector V. Finally, p sets TS_p to a bigger value than $m_1.ts^{def}$ and A-Delivers m_1 when all *pending* messages have a bigger timestamp than $m_1.ts^{def}$—a message m is pending if p delivered m but did not A-Deliver m yet.

Although this reasoning ensures that processes agree on the message delivery order, the delivery sequence of faulty processes may contain *holes*. For instance, p may A-Deliver m_1 followed by m_2, while some faulty process q only A-Delivers m_2. To see why, consider the following scenario. Process p delivers m_1 and m_2, and proposes some timestamp ts_p for these two messages. As q is faulty, it may only deliver m_2 and propose some timestamp ts_q bigger than ts_p as m_2's timestamp—this is possible because q may have A-Delivered several messages before m_2 that were not addressed to p and q thus updated its TS variable. Right after deciding in m_2's gdc instance, q A-Delivers m_2 and crashes. Later, p decides in m_1 and m_2's gdc instances, and A-Delivers m_1 followed by m_2, as m_1's definitive timestamp is smaller than m_2's.

To solve this problem, before A-Delivering a message m, every process p addressed by m computes m's *potential predecessor set*, denoted as $m.pps$. This set contains all messages addressed to p that may potentially have a smaller definitive timestamp than m's (in the example above, m_1 belongs to $m_2.pps$).[5] Message m is then A-Delivered when for all messages m' in $m.pps$ either (a) $m'.ts^{def}$ is known and it is bigger than $m.ts^{def}$ or (b) m' has been A-Delivered already.

The potential predecessor set of m is computed using causal multicast: To A-MCast m, m is first causally multicast. Second, after p decides in m's instance and updates its TS variable, p causally multicasts an *ack* message to the destination processes of m. As soon as p receives an *ack* message from all processes addressed by m that are trusted by its perfect failure detector module, the potential predecessor set of m is simply the set of pending messages.

Intuitively, m's potential predecessor set is correctly constructed for the two following facts: (1) Any message m', addressed to p and some process q, that q causally delivers *before* multicasting m's *ack* message will be in $m.pps$ (the definitive timestamp of m' might be smaller than m's). (2) Any message causally delivered by some addressee q of m *after* multicasting m's *ack* message will have a bigger definitive timestamp than m's. Fact (1) holds from causal order, i.e., if q C-Delivers m' before multicasting m's *ack* message, then p C-Delivers m' before C-Delivering m's *ack*. Fact (2) is a consequence of the following. As p's failure detector module is perfect, p stops waiting for *ack* messages as soon as p received an *ack* from all *alive* addressees of m. Hence, since processes update their TS variable after deciding in m's global data computation instance but before multicasting the *ack* message of m, no addressee of m proposes a timestamp smaller than $m.ts^{def}$ *after* multicasting m's *ack* message.

4.3 The Algorithm

Algorithm $\mathcal{A}1$ is composed of four tasks. Each line of the algorithm, task 2, and the procedure ADeliveryTest are executed atomically. Messages are composed of application data plus four fields: dst, id, ts, and $stage$. For every message m, $m.dst$ indicates to which groups m is A-MCast, $m.id$ is m's unique identifier, $m.ts$ denotes m's current timestamp, and $m.stage$ defines in which stage m is. We explain Algorithm $\mathcal{A}1$ by describing the actions a process p takes when a message m is in one of the three possible stages: s_0, s_1, or s_2.

[5]Note that the idea of computing a message's potential predecessor set appears in the atomic multicast algorithm of [20]. However, this algorithm assumes a majority of correct processes in every group and thus computes this set differently.

To A-MCast m, m is first C-MCast to its addressees (line 8). In stage s_0, p C-Delivers m, sets m's timestamp proposal, and adds m to the set of pending messages $Pending$ (lines 10-12). In stage s_1, p computes $m.ts^{def}$ (lines 17-19) and ensures that all messages in $m.pps$ are in p's pending set (lines 20-23), as explained above. Finally, in stage s_2, m is A-Delivered when for all messages m' in $m.pps$ that are still in p's pending set (if m' is not in p's pending set anymore, m' was A-Delivered before), m' is in stage s_2 (and thus $m'.ts$ is the definitive timestamp of m') and $m'.ts$ is bigger than $m.ts$ (lines 4-6). Notice that if m and m' have the same timestamp, we break ties using their message identifiers. More precisely, $(m.ts, m.id) < (m'.ts, m'.id)$ holds if either $m.ts < m'.ts$ or $m.ts = m'.ts$ and $m.id < m'.id$. Figure 1 illustrates a failure-free run of the algorithm.

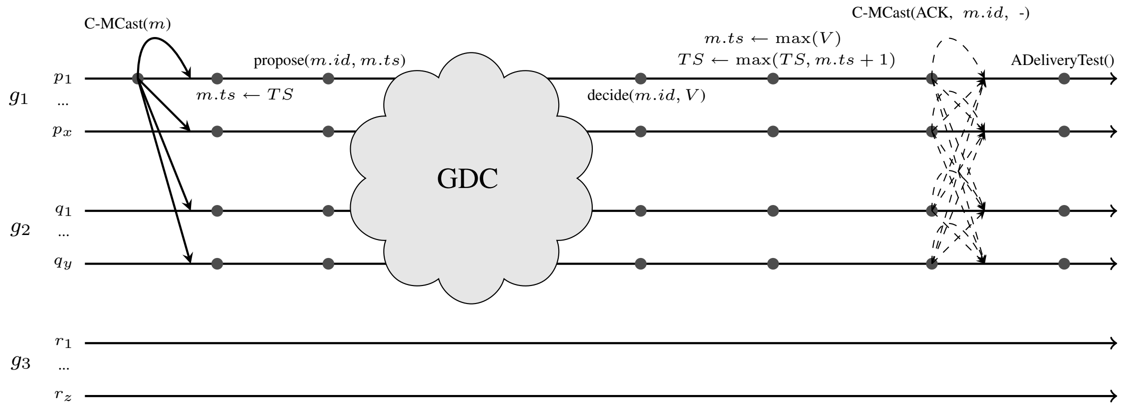

Figure 1: Algorithm $\mathcal{A}1$ in the failure-free case when a message m is A-MCast to groups g_1 and g_2.

5 Solving Atomic Multicast with Weaker Failure Detectors

In this section, we solve atomic multicast with a non-genuine algorithm. The Algorithm $\mathcal{A}2$ we present next does not require system-wide perfect failure detection and delivers messages in fewer communication steps. We first define additional abstractions used by the algorithm and summarize its assumptions. We then present the algorithm itself and conclude with a discussion on how to further reduce its delivery latency and weaken its failure detection requirements.

5.1 Additional Definitions and Assumptions

Failure Detector $\diamond\mathcal{P}$ We assume that processes have access to an eventually perfect failure detector $\diamond\mathcal{P}$ [6]. This failure detector ensures the strong completeness property of $\mathcal{P}$ and the following *eventual strong accuracy* property: there is a time after which no process stops being trusted before it crashes.

Reliable Multicast Reliable multicast is defined by the primitives R-MCast and R-Deliver and ensures all properties of causal multicast except uniform causal order.

Consensus In the *consensus* problem, processes propose values and must reach agreement on the value decided. Consensus is defined by the primitives propose(v) and decide(v) and satisfies the following properties [15]: (i) *uniform validity:* if a process decides v, then v was previously proposed by some process, (ii) *termination:* if every correct process proposes a value, then every correct process eventually decides exactly one value, and (iii) *uniform agreement:* if a process decides v, then all correct processes eventually decide v.

Generic Broadcast Generic broadcast ensures the same properties as atomic multicast except that all messages are addressed to all groups and only *conflicting* messages are totally ordered. More precisely, generic broadcast ensures uniform integrity, validity, uniform agreement, and the following *uniform generalized order* property: for any two conflicting messages m and m' and any two processes p and q, if p G-Delivers m and q G-Delivers m', then either p G-Delivers m' before m or q G-Delivers m before m'.

Assumptions To solve generic broadcast, either a simple majority of correct processes must be correct, i.e., $f < n/2$, and non-conflicting messages may be delivered in three message delays [2] or a two-third majority of processes must be correct, i.e., $f < n/3$, and non-conflicting message may be delivered in two message delays [18]. Both algorithms require a system-wide leader failure detector Ω [5], and thus the eventual perfect failure detector $\diamond\mathcal{P}$ we assume is sufficient. Moreover, inside each group, we need consensus and reliable multicast abstractions that tolerate an arbitrary number of failures. For this purpose, among realistic failure

Algorithm $\mathcal{A}1$ Genuine Atomic Multicast using $\mathcal{P}$ - Code of process p

```
1: Initialization
2:     $TS \leftarrow 1$, $Pending \leftarrow \emptyset$
3: procedure ADeliveryTest()
4:     while $\exists m \in Pending : m.stage = s_2$
              $\forall id \in m.pps : \exists m' \in Pending : m'.id = id \Rightarrow$
                        $m'.stage = s_2 \wedge (m.ts, m.id) < (m'.ts, m'.id)$ do
5:         A-Deliver($m$)
6:         $Pending \leftarrow Pending \setminus \{m\}$
7: To A-MCast message $m$                                                    {Task 1}
8:     C-MCast $m$ to $m.dst$
9: When C-Deliver($m$) atomically do                                         {Task 2}
10:    $m.ts \leftarrow TS$
11:    $m.stage \leftarrow s_0$
12:    $Pending \leftarrow Pending \cup \{m\}$
13: When $\exists m \in Pending : m.stage = s_0$                              {Task 3}
14:    $m.stage \leftarrow s_1$
15:    fork task ConsensusTask($m$)
16: ConsensusTask($m$)                                                       {Task x}
17:    Propose($m.id$, $m.ts$)      ▷ global data computation among processes in $m.dst$
18:    wait until Decide($m.id$, $V$)
19:    $m.ts \leftarrow \max(V)$
20:    $TS \leftarrow \max(TS, m.ts + 1)$
21:    C-MCast(ACK, $m.id, p$) to $m.dst$
22:    wait until $\forall q \in \mathcal{P} \cap m.dst$ : C-Deliver(ACK, $m.id, q$)
23:    $m.pps \leftarrow \{m'.id \mid m' \in Pending \wedge m' \neq m\}$
24:    $m.stage \leftarrow s_2$
25:    atomic block
26:        ADeliveryTest()
```

detectors, $\mathcal{P}$ is necessary and sufficient for consensus [10] and sufficient for reliable multicast [1].[6] Note that in practice, implementing $\mathcal{P}$ within each group is more reasonable than across the system, especially if groups are inside local area networks. We discuss below how to remove this assumption.

5.2 Algorithm Overview

The algorithm is inspired by the atomic broadcast algorithm of [21]. We first recall its main ideas and then explain how we cope with group failures—[21] assumes that there is at least one correct process in every group. We then show how *local* messages to some group g, i.e., messages multicast from processes inside g and addressed to g only, may be delivered with no inter-group communication at all.

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. 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 of group crashes, this solution does not ensure liveness however. Indeed, if a group g crashes there will be some round r after which no process receives the message bundles of g. To circumvent this problem we proceed in two steps: (a) we allow processes to stop waiting for g's message bundle, and (b) we let processes agree on the set of message bundles to consider for each round.

To implement (a), 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 (b), 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. 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 faster message bundle delivery.

Processes may also A-Deliver local messages to some group g without communicating with processes outside of g. As these messages are addressed to g only, members of g may A-Deliver them directly after consensus, and thus before receiving the groups' message bundles.

We note that maintaining a common view of the alive groups in the system resembles what is called in the literature group membership [7]. Intuitively, a group membership service provides processes with a consistent view of alive processes in the system, i.e., processes "see" the same sequence of views. Moreover, processes agree on the set of messages delivered between each view change, a property that is required for message bundles.[7] In fact, our algorithm could have been built on top of such an abstraction. However, doing so would have given us less freedom to optimize the delivery latency of message bundles.

[6] In [1], the authors present the weakest failure detector to solve reliable broadcast. Extending the algorithm of [1] to the multicast case using the same failure detector is straightforward.

[7] Some group membership specifications also guarantee total ordering of the messages delivered between view changes.

5.3 The Algorithm

Algorithm $\mathcal{A}2$ is composed of five tasks. Each line of the algorithm is executed atomically. On every process p, six global variables are used:Rnd denotes the current round number, $\mathit{Rdelivered}$ and $\mathit{Adelivered}$ are the set of R-Delivered and A-Delivered messages respectively, $\mathit{Gdelivered}$ is the sequence of G-Delivered messages, $\mathit{MsgBundle}$ stores the message bundles, and View is the set of groups currently deemed to be alive.

In the algorithm, every G-BCast message m has the following format: $(\mathit{rnd}, g, \mathit{type}, \mathit{msgs})$, where rnd denotes the round in which m was G-BCast, g is the group m refers to, type denotes m's type and is either $\mathit{msgBundle}$, add, or remove, and msgs is a set of messages; this field is only used if m is a message bundle.

To A-MCast a message m, a process p R-MCasts m to p's group (line 5). In every round r, the set of messages that have been R-Delivered but not A-Delivered yet are proposed to the next consensus instance (line 9), p A-Delivers the set of local messages decided in this instance (line 12), and global messages, i.e., non local messages, are G-BCast at line 14 if $\mathit{group}(p)$ belongs to the view. Otherwise, p G-BCasts a message to add $\mathit{group}(p)$ to the view.

Process p then gathers message bundles of the current round k using variable $\mathit{MsgBundle}$: Process p executes the while loop of lines 17-24 until, for every group g, $\mathit{MsgBundle}[g]$ is neither $\perp$, i.e. p is not waiting to receive a message bundle from g, nor $\top$, a value whose signification is explained below. The first message m_g^k of round k related to g of type $\mathit{msgBundle}$ or remove that p G-Delivers "locks" $\mathit{MsgBundle}[g]$, i.e., any subsequent G-Delivered message of round k concerning g is discarded (line 21). If m_g^k is of type $\mathit{msgBundle}$, p stores g's message bundle in $\mathit{MsgBundle}[g]$ (line 24). Otherwise, m_g^k was G-BCast by some process q that suspected g to have entirely crashed, i.e., failure detector $\diamond\mathcal{P}$ at q did not trust any member of g (lines 31-33), and thus p sets $\mathit{MsgBundle}[g]$ to $\emptyset$ (line 23). Note that q sets $\mathit{MsgBundle}[g]$ to $\top$ after G-BCasting a message of the form $(k, g, \mathit{remove}$, -) to prevent q from G-BCasting multiple "remove g" messages in the same round.

While p is gathering message bundles for round k, it may also handle some message of type add concerning g, in which case p adds g to a local variable $\mathit{groupsToAdd}$ (line 22). Note that this type of message is not tagged with a round number to ensure that messages A-MCast from correct groups are eventually A-Delivered by their correct addressees. In fact, tagging add messages with the round number could prevent a group from being added to the view as we now explain. Consider a correct group g that is removed from the view in the first round. In every round, members of g G-BCast a message to add g back to the view. In every round however, processes G-Deliver message bundles of groups in the view before G-Delivering these "add g" messages, and they are thus discarded.

After exiting from the while loop, p A-Delivers global messages (line 26), the view is recomputed as the groups g such that $\mathit{MsgBundle}[g] \neq \emptyset$ or $g \in \mathit{groupsToAdd}$ (line 28), and p sets $\mathit{MsgBundle}[g]$ to either $\perp$, if g belongs to the new view, or $\emptyset$ otherwise (p will not wait for a message bundle from g in the next round). Figure 2 illustrates a failure-free run of the algorithm.

5.4 Further Improvements

Delivery Latency In Algorithm $\mathcal{A}2$, local messages are delivered directly after consensus. Hence, these messages do not bear the cost of a single inter-group message delay unless: (a) they are multicast from a group different than their destination group or (b) they are multicast while the groups' bundle of messages are being exchanged, in which case the next consensus instance can only be started when message bundles of the current round have been received. Obviously, nothing can be done to avoid case (a). However, we can prevent case (b) from happening by allowing rounds to overlap. That is, we start the next round before

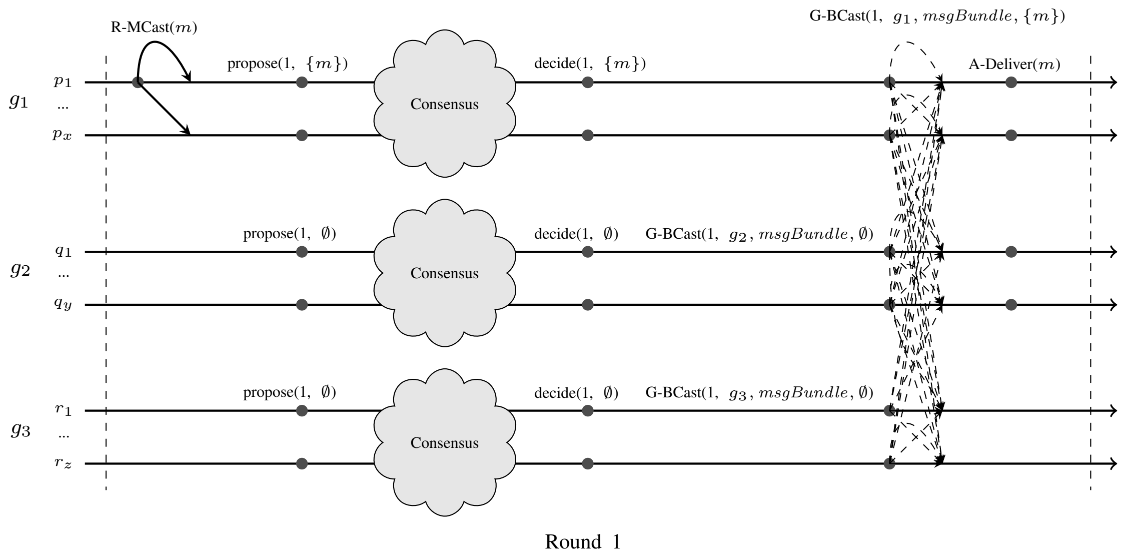

Figure 2: Algorithm $\mathcal{A}2$ in the failure-free case when a message m is A-MCast to groups g_1 and g_2.

receiving the groups' bundle of messages for the current round. Note that to ensure agreement on the relative delivery order of local and global messages, processes inside the same group must agree on when global messages of a given round are delivered, i.e., after which consensus instance. For this purpose, a mapping between rounds and consensus instances can be defined. To control the inter-group traffic, we may also specify that message bundles are sent, say every κ consensus instance. Choosing κ presents a trade-off between inter-group traffic and delivery latency of global messages.

Failure Detection To weaken the failure detector required inside each group, i.e., $\mathcal{P}$ in Algorithm $\mathcal{A}2$, we may remove a group g from the view as soon as a majority of processes in g are suspected. This allows to use consensus and reliable multicast algorithms that are safe under an arbitrary number of failures and live only when a majority of processes are correct. Hence, the leader failure detector Ω becomes sufficient. Care should be taken as when to add g to the view again: this should only be done when a majority of processes in g are trusted to be alive. This solution ensures a weaker liveness guarantee however: correct processes in some group g will *successfully* multicast and deliver messages only if g is *maj-correct*, i.e., g contains a majority of correct processes. More precisely, the liveness guaranteed by this modified algorithm is as follows (uniform integrity and uniform prefix order remain unchanged):

- *weak uniform agreement:* if a process p A-Delivers a message m, then all correct processes $q \in m.dst$ in a maj-correct group eventually A-Deliver m
- *weak validity:* if a correct process p in a maj-correct group A-MCasts a message m, then all correct processes $q \in m.dst$ in a maj-correct group eventually A-Deliver m.

6 Final Remarks

In this paper, we addressed the problem of solving atomic multicast in the case where groups may entirely crash. We presented two algorithms. The first algorithm is genuine, tolerates an arbitrary number of process

Algorithm $\mathcal{A}2$ Non-Genuine Atomic Multicast - Code of process p

```
1: **Initialization**
2:    $Rnd \leftarrow 1$, $Rdelivered \leftarrow \emptyset$, $Adelivered \leftarrow \emptyset$, $Gdelivered \leftarrow \epsilon$
3:    $View \leftarrow \Gamma$, $MsgBundle[g] \leftarrow \bot$ for each group $g \in \Gamma$

4: **To A-MCast** message $m$                                                    $\{Task\ 1\}$
5:    R-MCast $m$ to $group(p)$

6: **When** R-Deliver($m$)                                                       $\{Task\ 2\}$
7:    $Rdelivered \leftarrow Rdelivered \cup \{m\}$

8: **Loop**                                                                      $\{Task\ 3\}$
9:    Propose($Rnd$, $Rdelivered \setminus Adelivered$)                          ▷ consensus inside group
10:   **wait until** Decide($Rnd$, $msgs$)

11:   $localMsgs \leftarrow \{m \mid m \in msgs \wedge m.dst = \{group(p)\}\}$
12:   **A-Deliver** messages in $localMsgs$ in some deterministic order           ▷ A-Deliver local messages
13:   $Adelivered \leftarrow Adelivered \cup localMsgs$

14:   **if** $group(p) \in View$ **then** G-BCast($Rnd, group(p), msgBundle, msgs \setminus localMsgs$)
15:   **else** G-BCast(-, $group(p)$, $add$, -)
16:   $groupsToAdd \leftarrow \emptyset$

17:   **while** $\exists g \in \Gamma : MsgBundle[g] \in \{\bot, \top\}$
18:      **if** $\nexists (rnd, g, type, msgs) \in Gdelivered : (rnd = Rnd \vee type = add)$ **then**
19:         **wait until** G-Deliver($rnd, g, type, msgs$) $\wedge$ $(rnd = Rnd \vee type = add)$
20:      $(rnd, g', type, msgs) \leftarrow$ remove first message in $Gdelivered$ s.t. $(rnd = Rnd \vee type = add)$
21:      **if** $MsgBundle[g'] \in \{\bot, \top\}$ **then**
22:         **if** $type = add$ **then** $groupsToAdd \leftarrow groupsToAdd \cup \{g'\}$
23:         **else if** $type = remove$ **then** $MsgBundle[g'] \leftarrow \emptyset$
24:         **else** $MsgBundle[g'] \leftarrow msgs$
25:   $globalMsgs \leftarrow \{m \mid \exists g \in \Gamma : MsgBundle[g] = msgs \wedge m \in msgs\}$
26:   **A-Deliver** messages in $globalMsgs$ addressed to $p$ in some deterministic order   ▷ A-Deliver global messages
27:   $Adelivered \leftarrow Adelivered \cup globalMsgs$

28:   $View \leftarrow \{g \mid MsgBundle[g] \neq \emptyset\} \cup groupsToAdd$
29:   **foreach** $g \in \Gamma : MsgBundle[g] \leftarrow \bot$ (if $g \in View$) or $\emptyset$ (otherwise)
30:   $Rnd \leftarrow Rnd + 1$

31: **When** $\exists g \in View : MsgBundle[g] = \bot \wedge \forall q \in g : q \notin \Diamond\mathcal{P}$   $\{Task\ 4\}$
32:   G-BCast($Rnd, g, remove$, -)
33:   $MsgBundle[g] \leftarrow \top$

34: **When** G-Deliver($type, m$)                                                $\{Task\ 5\}$
35:   $Gdelivered \leftarrow Gdelivered \oplus (rnd, g, type, msgs)$
```

failures, and requires perfect failure detection. We showed, in Section 1, that if we consider realistic failure detectors only and we do not bound the number of failures, $\mathcal{P}$ is necessary to solve this problem. The second algorithm we presented is not genuine but requires perfect failure detection inside each group only and may deliver messages addressed to multiple groups within two inter-group message delays. We showed how this latter algorithm can be modified to cope with unreliable failure detection, at the cost of a weaker liveness guarantee.

Figure 3 provides a comparison of the presented algorithms. The best-case message delivery latency is computed by considering a message A-MCast to multiple groups in a failure-free scenario when the inter-group message delay is δ and the intra-group message delay is negligible. Note that we took 4δ and 2δ as the best-case latency for causal multicast (cf. Appendix) and global data computation [12] respectively.

Algorithm	genuine?	resiliency	required failure detector(s)	best-case latency
$\mathcal{A}1$	yes	$f \leq n$	system-wide $\mathcal{P}$	10δ
$\mathcal{A}2$	no	$f < n/3$	group-wide $\mathcal{P}$ and system-wide $\Diamond\mathcal{P}$	2δ
		$f < n/2$	(modification of algorithm with weaker liveness tolerates unreliable failure detection)	3δ

Figure 3: Comparison of the presented algorithms.

References

[1] M. K. Aguilera, S. Toueg, and B. Deianov. Revising the weakest failure detector for uniform reliable broadcast. In *Proceedings of DISC'99*, pages 19–33. Springer-Verlag, 1999.

[2] M.K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Thrifty generic broadcast. In *Proceedings of DISC'00*, pages 268–283. Springer-Verlag, 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] Kenneth P. Birman, Andre Schiper, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

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

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

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

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

[9] C. Delporte-Gallet and H. Fauconnier. Fault-tolerant genuine atomic multicast to multiple groups. In *Proceedings of OPODIS'00*, pages 107–122. Suger, Saint-Denis, rue Catulienne, France, 2000.

[10] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. A realistic look at failure detectors. In *Proceedings of DSN'02*, pages 345–353. IEEE Computer Society, 2002.

[11] C. Delporte-Gallet, H. Fauconnier, and R. Guerraoui. Shared Memory vs Message Passing. Technical report, EPFL, 2003.

[12] 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.

[13] U. Fritzke, Ph. Ingels, A. Mostéfaoui, and M. Raynal. Fault-tolerant total order multicast to asynchronous groups. In *Proceedings of SRDS'98*, pages 578–585. IEEE Computer Society, 1998.

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

[15] 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.

[16] Udo Fritzke Jr. and Philippe Ingels. Transactions on partially replicated data based on reliable and atomic multicasts. In *Proceedings of ICDCS'01*, pages 284–291. IEEE Computer Society, 2001.

[17] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

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

[19] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters*, 39:343–350, 1991.

[20] L. Rodrigues, R. Guerraoui, and A. Schiper. Scalable atomic multicast. In *Proceedings of IC3N'98*. IEEE, 1998.

[21] N. Schiper and F. Pedone. On the inherent cost of atomic broadcast and multicast in wide area networks. In *Proceedings of ICDCN'08*, pages 147–157. Springer, 2008.

[22] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.

A Appendix

A.1 Solving Causal Multicast

Several papers investigated the problem of ensuring causal order. However, either the broadcast case is considered [15], messages are not allowed to be multicast to remote groups [4], or messages may only be sent to a single process [19]. We here present the first causal multicast algorithm that allows messages to be addressed to any subset of groups. We solve causal multicast in two steps. Algorithm $\mathcal{A}3$ transforms reliable multicast into a primitive that is called fifo multicast . Fifo multicast ensures the same properties as causal multicast except that uniform causal order is replaced by *uniform fifo order*: if a process p F-MCasts a message m before F-MCasting a message m', then no process in $m.dst \cap m'.dst$ F-Delivers m' unless it has previously F-Delivered m. Algorithm $\mathcal{A}4$ then transforms fifo multicast into causal multicast.

A.1.1 Transforming Reliable Multicast into Fifo Multicast

Algorithm $\mathcal{A}3$ is similar to the fifo broadcast algorithm of [15]. It nevertheless differs from the broadcast algorithm in several aspects as messages are not necessarily addressed to all processes.

As in [15], every message is tagged with information on m's sequence number, denoted as $m.nbCast$. Messages from some process q are then F-Delivered in the sequence number order. To do so, every process p keeps track of the last message from q that p F-Delivered. This information is stored in a variable denoted as $nbDel[q]_p$. Since messages may be addressed to a subset of the groups, messages do not carry a single sequence number, as in [15], but an array of sequence numbers, one for each group (lines 5-9).

Now consider the following problematic scenario specific to fifo multicast. Some process p F-MCasts a message m_1 to some group g_1. Later, p F-MCasts a message m_2 to groups g_1 and g_2 and crashes. Message m_2 is received by processes in g_2, and since m_2 is the first message F-MCast from p to g_2, m_2 is F-Delivered by processes in g_2. On the contrary, m_1 is never received by any process. Note that this can happen because p crashes and links are quasi-reliable. From the uniform agreement property of fifo multicast, processes in g_1 eventually F-Deliver m_2. However, they cannot F-Deliver m_1 before m_2 as m_1 was lost, violating the uniform fifo order property.

To solve this problem, a process $p \in m.dst$ R-MCasts an *acknowledgment* for m when p F-Delivered all messages $m.sender$ F-MCast before m (line 13 or 19). Processes F-Deliver m when they R-Deliver an *acknowledgment* from at least one correct process of every destination group of m (lines 14-15). To do so, processes use a failure detector denoted as Θ [1]. This failure detector satisfies the strong completeness property defined in Section 4.1 as well as the following *accuracy* property: if there exists a correct process then, at every time, every process trusts at least one correct process. Let g by any group. We denote by Θ_g the failure detector Θ restricted to the scope of processes in g. Note that if we restrict the universe of failure detectors to realistic ones and we do not bound the number of process failures, then Θ is equivalent to the perfect failure detector $\mathcal{P}$ [11].

A.1.2 Transforming Fifo Multicast into Causal Multicast

Algorithm $\mathcal{A}4$ is inspired by the blocking transformation of fifo broadcast into causal broadcast of [15]. As we explain below however, solving causal multicast introduces problems nonexistent in causal broadcast.

As in [15], every process p maintains information about how many messages, C-MCast from some process q and addressed to $group(p)$, p C-Delivered since the beginning, and this for every process q. As opposed to [15], this accounting is done on a group basis. This information is thus stored in a vector of vectors, i.e., one vector per group, denoted as $nbDel_p$; we explain below how p maintains $nbDel[g]_p$ for

Algorithm $\mathcal{A}3$ FIFO Multicast - Code of process p

```
1: Initialization
2:    $nbCast[g] \leftarrow 0$, for each group $g$                    ▷ nb. of msgs. F-MCast to $g$
3:    $nbDel[q] \leftarrow 0$, for each process $q$                   ▷ nb. of msgs. F-Delivered originating from $q$
4:    $msgSet \leftarrow \emptyset$                                   ▷ set of messages A-Delivered but not yet F-Delivered
5: To F-MCast message $m$                                            {Task 1}
6:    $m.nbCast \leftarrow nbCast$
7:    R-MCast ($m$) to $m.dst$
8:    foreach $g \in m.dst$ do
9:       $nbCast[g] \leftarrow nbCast[g] + 1$

10: When R-Deliver($m$)
11:    $msgSet \leftarrow msgSet \cup \{m\}$
12:    if $m.nbCast[group(p)] = nbDel[m.sender]$ then
13:       R-MCast(ACK, $m.id, p$) to $m.dst$

14: When $\exists m \in msgSet : \forall q \in \bigcup_{g \in m.dst} \Theta_g$ : R-Deliver(ACK, $m.id, q$) $\wedge$
                       $m.nbCast[group(p)] = nbDel[m.sender]$
15:    F-Deliver($m$)
16:    $nbDel[m.sender] \leftarrow nbDel[m.sender] + 1$
17:    $msgSet \leftarrow msgSet \setminus \{m\}$
18:    if $\exists m' \in msgSet \; : \; m'.nbCast[group(p)] = nbDel[m'.sender]$ then
19:       R-MCast(ACK, $m'.id, p$) to $\{q \mid q \in m'.dst\}$
```

groups g different than $group(p)$. To C-MCast a message m, p F-MCasts m along with $nbDel_p$. Upon F-Delivering m, p inserts m in a list of messages $msgLst_p$ and C-Delivers m as soon as it is the first message in $msgLst_p$ such that $nbDel[group(p)]_p \geq m.nbDel[group(p)]$.[8]

Now consider the following causal relation between two messages m and m' addressed to some group g that we denote as *blind* for g: C-MCast(m) $\rightarrow$ C-MCast(m'), $m.sender \neq m'.sender$, and there exists no message m'' such that $g \in m''.dst$ and C-MCast(m) $\rightarrow$ C-MCast(m'') $\rightarrow$ C-MCast(m'). Note that *blind* causal relations can be of two types:

- Type a: m is addressed to at least two groups g and g' such that there exists a process in g' that C-Delivers m before C-MCasting a message m'' and C-MCast(m'') $\rightarrow$ C-MCast(m').
- Type b: there exists a message m'' such that $m.sender$ C-MCasts m'' after m and C-MCast(m'') $\rightarrow$ C-MCast(m').

These blind causal relations are problematic with the above sketched algorithm because processes in g may C-Deliver m and m' in different orders as for all processes q, $m.nbDel[group(p)][q] = m'.nbDel[group(p)][q]$. We handle blind causal relations of type (a) and (b) by storing extra information on messages and processes to be able to differentiate messages m and m', as we now explain.

Type a: Every process p keeps track of, for every process q and group g (and not only for $group(p)$ as before), the number of messages addressed to g C-MCast from q that were C-Delivered in the causal history.[9] To do so, $nbDel_p$ is piggybacked on C-MCast messages as before and after every message m p C-Delivers, for every group g of the system, p does two things:

1. Process p stores, for every process q, the maximum value between $nbDel[g][q]$ and $m.nbDel[g][q]$ (line 20).

[8]Given any two vectors v_1 and v_2, we write $v_1 \geq v_2$ instead of $\forall q \in \Pi : v_1[q] \geq v_2[q]$ for simplicity.

2. If $g \in m.dst$, p updates $nbDel[g][m.sender]$ to the number of messages C-MCast to g that were C-Delivered in the causal history ($m.nbCast$), including m (line 22). Note that $m.nbCast$ is introduced below.

Type b: Every process p stores, for every process q and group g, the number of messages q C-MCast to a given group g in the causal history.[9] This information is kept in a matrix denoted as $nbCast_p$. Every time p C-MCasts a message m, p piggybacks $nbCast_p$ on m and increments $nbCast[g][p]$ for every group g addressed by m (lines 7-10). When p C-Delivers m, for every process q and group g, p stores the maximum value between $nbCast[g][q]$ and $m.nbCast[g][q]$ (line 19). To C-Deliver m, we add the condition that $nbDel[group(p)]_p \geq m.nbCast[group(p)]$ (line 12).

Algorithm $\mathcal{A}4$ Causal Multicast - Code of process p

```
 1: **Initialization**
 2:     $nbCast[g][q] \leftarrow 0$, for each group $g$ and process $q$        ▷ nb. msgs. $q$ C-MCast to $g$ in causal history
 3:     $nbDel[g][q] \leftarrow 0$, for each group $g$ and process $q$     ▷ nb. msgs. $q$ C-MCast to $g$ C-Delivered in causal history
 4:     $msgLst \leftarrow \epsilon$                                  ▷ list of messages F-Delivered but not yet C-Delivered
 5: **To C-MCast** message $m$                                                                    {*Task 1*}
 6:     $m.nbDel \leftarrow nbDel$
 7:     $m.nbCast \leftarrow nbCast$
 8:     F-MCast ($m$) to $\{q \mid q \in m.dst\}$
 9:     **foreach** $g \in m.dst$ **do**
10:         $nbCast[g][p] \leftarrow nbCast[g][p] + 1$

11: **Function** IsDeliverable($m$)
12:     return $m.nbDel[group(p)] \leq nbDel[group(p)] \wedge$
                $m.nbCast[group(p)] \leq nbDel[group(p)]$

13: **When** F-Deliver($m$)
14:     $msgLst \leftarrow msgLst \oplus m$
15:     **When** $\exists m' \in msgLst$ : IsDeliverable($m'$)
16:         Let $m'$ be the first message in $msgLst$ s.t. isDeliverable($m'$)
17:         C-Deliver($m'$)
18:         **foreach** $g \in \Gamma$ **do**
19:             $nbCast[g] \leftarrow \max(m'.nbCast[g], nbCast[g])$
20:             $nbDel[g] \leftarrow \max(m'.nbDel[g], nbDel[g])$
21:             **if** $g \in m'.dst$ **then**
22:                 $nbDel[g][m'.sender] \leftarrow \max(nbDel[g][m'.sender], m'.nbCast[g][m'.sender] + 1)$
23:         $msgLst \leftarrow msgLst \ominus m'$
```

A.2 The Algorithms' Proofs

In the proofs below, we denote the value of a variable V on a process p at time t as V_p^t. Furthermore, for events of the type C-MCast and C-Deliver, we sometimes add a subscript to denote on which process this event occurred. Note that we use the following definition of *is a prefix of*: S_1 is a prefix of $S_2 \Leftrightarrow \exists \alpha : S_1 \oplus \alpha = S_2$.

[9]Let m be a message. An event C-MCast(m)/C-Deliver(m) is in the causal history of p either if p C-MCasts/C-Delivers m or if there exists a message m' such that p C-Delivers m' and C-MCast(m) $\rightarrow$ C-MCast(m').

A.2.1 The Proof of Algorithm $\mathcal{A}1$

Definition A.1 *For any message m, we define $m.ts_p^{def}$ as the definitive timestamp of m on a process p, i.e., it is m's timestamp after p executed line 19 in Algorithm $\mathcal{A}1$. If p never executes line 19 for m, then $m.ts_p^{def}$ is undefined.*

Proposition A.1 (Uniform Integrity) *For any process p and any message m, (a) p A-Delivers m at most once, and (b) only if $p \in m.dst$ and (c) m was previously A-MCast.*

Proof:

- (a) Follows directly from the uniform integrity property of causal multicast and from the fact that a message is removed from $Pending_p$ after it has been A-Delivered.
- (b) Follows directly from the algorithm.
- (c) Process p A-Delivers m only if p C-Delivered m. From the uniform integrity property of causal multicast, m was C-MCast. Consequently, m was A-MCast. □

Lemma A.1 *For any correct process p and any message m, if p C-Delivers m, then m eventually reaches stage s_2 on p.*

Proof: By the uniform agreement property of causal multicast, all correct processes $q \in m.dst$ eventually C-Deliver m and fork the consensus task for m. By the termination property of global data computation, q eventually decides and C-MCasts (ACK, $m.id, q$). By the strong completeness property of $\mathcal{P}$, eventually no faulty process is ever trusted by any correct process. Therefore, by the validity property of causal multicast, q eventually C-Delivers(ACK, $m.id$,-) for all processes in $\mathcal{P} \cap m.dst$. Therefore, m eventually reaches stage s_2 on q, and in particular on p. □

Lemma A.2 *For any correct process p and any message m, if p C-Delivers m, then p eventually A-Delivers m.*

Proof: By Lemma A.1, m eventually reaches stage s_2 on p. Consider the *transitive* relation on messages *in-pps* defined as follows: m_1 *in-pps* m_2 if and only if $m_1 \in m_2.pps$. Let $PPS(m)$ be the set of messages m' such that m' *in-pps* m. By Lemma A.1, all $m' \in PPS(m)$ eventually reach stage s_2 on p. Because the identifiers of messages are unique, the relation $<$ on messages' timestamps and identifiers defines a total order. Hence, messages in $PPS(m)$ are delivered according to the order defined by $<$ and thus, since $|PPS(m)|$ is finite, p eventually A-Delivers m. □

Proposition A.2 (Uniform Agreement) *If a process p A-Delivers a message m, then all correct processes $q \in m.dst$ eventually A-Deliver m.*

Proof: If p A-Delivers m, p C-Delivered m. By the uniform agreement property of causal multicast, all correct processes $q \in m.dst$ eventually C-Deliver m. By Lemma A.2, q eventually A-Delivers m. □

Proposition A.3 (Validity) *If a correct process p A-MCasts a message m, then eventually all correct processes $q \in m.dst$ A-Deliver m.*

Proof: If p A-MCasts m, then p C-MCasts m. By the validity property of causal multicast, all correct processes $q \in m.dst$ eventually C-Deliver m. By Lemma A.2, q eventually A-Delivers m. □

Lemma A.3 *For any two messages m_1, m_2, and any two processes p and q such that $\{p, q\} \in m_1.dst \cap m_2.dst$, if p A-Delivers m_1, q A-Delivers m_2, and $(m_1.ts_p^{def}, m_1.id) < (m_2.ts_q^{def}, m_2.id)$, then q A-Delivers m_1 before m_2.*

Proof: Let t_1 and t_2 be the times at which p gathered ACK messages for m_1 and q gathered ACK messages for m_2 respectively. Either (a) $t_1 \leq t_2$ or (b) $t_1 > t_2$.

- In case (a), by the strong accuracy property of $\mathcal{P}$, p C-Delivers (ACK, $m_1.id, q$). Hence, q C-Delivers m_1 before t_1 (and t_2). Consequently, q adds m_1 to $Pending_q$ before A-Delivering m_2. At the time q computes m_2's potential predecessor set (line 23), either (a-i) $m_1 \in Pending_q$ or (a-ii) not.
  - In case (a-ii), because a message is removed from $Pending$ only after being A-Delivered (line 6), q A-Delivers m_1 before m_2.
  - In case (a-ii), from line 23, $m_1 \in m_2.pps$. From line 4, q does not A-Deliver m_2 before m_1 reaches stage s_2. Since q A-Delivers m_2, m_1 reaches stage s_2 on q. By the uniform agreement property of global data computation, when m_1 reaches stage s_2 on q, $m_1.ts_q^{def} = m_1.ts_p^{def}$. Since $(m_1.ts_p^{def}, m_1.id) < (m_2.ts_q^{def}, m_2.id)$, $(m_1.ts_q^{def}, m_1.id) < (m_2.ts_q^{def}, m_2.id)$. Consequently, from the condition of line 4, q A-delivers m_1 before m_2.
- In case (b), by the strong accuracy property of $\mathcal{P}$, q C-Delivers (ACK, $m_2.id, p$).

  We now show that p C-Delivers m_1 before C-MCasting (ACK, $m_2.id, p$). Suppose, by way of contradiction, that (*) p does not C-Deliver m_1 before C-MCasting (ACK, $m_2.id, p$). Since p A-Delivers m_1, p C-Delivers m_1. From (*), p does so after p executes line 20 in the consensus task of m_2. Since p C-MCasts (ACK, $m_2.id, p$), p decides in m_2's global data computation instance. By the uniform agreement property of global data computation, (**) $m_2.ts_p^{def} = m_2.ts_q^{def}$ (line 19). Since p A-Delivers m_1, p decides in m_1's global data computation instance. By the uniform obligation property of global data computation, p decides on vector V such that $V[p] = v_p$. Because p sets TS_p to $\max(m_2.ts_p^{def} + 1, TS_p)$ at line 20, from (*) and (**), $m_2.ts_q^{def} < v_p \leq m_1.ts_p^{def}$, a contradiction to the fact that $(m_1.ts_p^{def}, m_1.id) < (m_2.ts_q^{def}, m_2.id)$.

  Consequently, p C-Delivers m_1 before C-MCasting (ACK, $m_2.id, p$). Hence, C-Mcast(m_1) $\rightarrow$ C-Deliver$(m_1)_p$ $\rightarrow$ C-MCast(ACK, $m_2.id, p)_p$ $\rightarrow$ C-Deliver(ACK, $m_2.id, p)_q$. Therefore, from the causal order property of causal multicast, q C-delivers m_1 and adds m_1 to $Pending_q$, before A-Delivering m_2. A similar argument as in (a) is then used to conclude the proof. $\square$

Proposition A.4 (Uniform Prefix Order) *For any two messages m and m' and any two processes p and q such that $\{p, q\} \in m.dst \cap m'.dst$, 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'.*

Proof: Since p A-Delivers m and q A-Delivers m', $m.ts_p^{def}$ and $m'.ts_q^{def}$ are both defined. Either $(m.ts_p^{def}, m.id) < (m'.ts_q^{def}, m'.id)$ or $(m.ts_p^{def}, m.id) > (m'.ts_q^{def}, m'.id)$. By Lemma A.3, either q A-Delivers m before m' or p A-Delivers m' before m. $\square$

A.2.2 The Proof of Algorithm $\mathcal{A}2$

In the proof below, a message of round k concerning a group g is any G-BCast message of the form (k, g, -, -).

Definition A.2 • *We define $MessageBundle_p^k$ as the value of variable $MessageBundle$ on p before p executes line 25 in round k. If p does not execute line 25 in round k, then $globalMsgs_p^k$ is undefined.*

- *We define $View_p^k$ as the value of variable $View$ on p after p executes line 28 in round k. If p does not execute line 28 in round k, then $View_p^k$ is undefined.*
- *We define LM_p^k as the last message process p removes from the sequence $Gdelivered$ at line 20 before p computes the set of global messages of round k at line 25. If p never executes line 25 in round k, then LM_p^k is undefined.*

Lemma A.4 *For any k, any two processes p and q such that group(p) = group(q) = g and any two messages $(k, g, msgBundle,\ msgs_p)$ and $(k, g, msgBundle,\ msgs_q)$ respectively G-BCast by p and q at line 14, $msgs_p = msgs_q$.*

Proof: From the uniform agreement property of consensus, p and q decide on the same set of messages in round k and compute the same set $localMsgs$ at line 11. Consequently, $msgs_p = msgs_q$. □

Lemma A.5 *For any two processes p and q and any k:*

1. *if $MessageBundle_p^k$ and $MessageBundle_q^k$ are both defined, then $MessageBundle_p^k = MessageBundle_q^k$.*
2. *if $View_p^k$ and $View_q^k$ are both defined, then $View_p^k = View_q^k$.*

Proof: In the proof below, we denote as $groupsToAdd_p^k$ the value of variable $groupsToAdd$ on process p before p executes line 25 in round k. We proceed by simultaneous induction on 1 and 2.

- Base step ($k = 1$):
  1. We show that for any group g, $MsgBundle_p^1[g] = MsgBundle_q^1[g]$. Suppose, by way of contradiction, that (*) $MsgBundle_p^1[g] \neq MsgBundle_q^1[g]$. From the condition of line 17, either (a) $MsgBundle_p^1[g] = \emptyset$ or (b) $MsgBundle_p^1[g] \neq \emptyset$.
     - In case (a), since $MsgBundle_p[g]$ is initialized to $\perp$, the first message concerning g that p removes from the $Gdelivered$ sequence is a message of the form (1, g, $remove$, -). Let m_p be this message. Since $MsgBundle_q[g]$ is initialized to $\perp$, the first message concerning g that q removes from the $Gdelivered$ sequence is a message of the form (1, g, $msgBundle$, -). Let m_q be this message. From the uniform generalized order property of generic broadcast, either (a-i) p G-Delivers m_q before m_p or (a-ii) q G-Delivers m_p before m_q. We show that both (a-i) and (a-ii) lead to a contradiction.
       * In case (a-i), from the algorithm, $MsgBundle_p^1 \neq \emptyset$, a contradiction to hypothesis (a).
       * In case (a-ii), from the algorithm, $MsgBundle_q^1 = \emptyset$, a contradiction to (*).
     - In case (b), a similar argument as in (a) is used where every occurrence of $remove$, $msgBundle$, $\neq$, and $=$ are respectively replaced by $msgBundle$, $remove$, $=$, and $\neq$.
  2. From 1, $MsgBundle_p^1 = MsgBundle_q^1$. Therefore, it is sufficient to show that $groupsToAdd_p^1 = groupsToAdd_q^1$. We prove that, for any group g, $g \in groupsToAdd_p^1 \Leftrightarrow groupsToAdd_q^1$.

- ($\Rightarrow$) Process p G-Delivers a message of the form (-, g, add, -). Let m_p^{add-g} be this message. Suppose, by way of contradiction, that (*) there exists a group g in $groupsToAdd_p^1$ that is not in $groupsToAdd_q^1$. From the algorithm, LM_p^1 cannot be of the form (-, -,add, -). Therefore, (**) p G-Delivers m_p^{add-g} before LM_p^1 and LM_p^1 is either (a) of the form (1, g', $remove$, -) or (b) of the form (1, g', $msgBundle$, -) for some group g'.
  - In case (a), from 1, $MsgBundle_p^1 = MsgBundle_q^1$, therefore the first message q G-Delivers concerning g' is a message of the form (1, g', $remove$, -). Let $m_q^{remove-g'}$ be this message. From the uniform generalized order of generic broadcast, either (a-i) p G-Delivers $m_q^{remove-g'}$ before m_p^{add-g} or (a-ii) q G-Delivers m_p^{add-g} before $m_q^{remove-g'}$.
    - In case (a-i), p G-Delivers a message concerning g' before LM_p^1. Therefore, from (**), LM_p^1 cannot concern g' ($MsgBundle_p^1[g']$ is locked before p removes LM_p^1 from the $Gdelivered$ sequence), a contradiction.
    - In case (a-ii), $g \in groupsToAdd_q^1$, a contradiction to (*).
  - In case (b), a similar argument as in (a) is used where every occurrence of $remove$ is replaced by $msgBundle$.
- ($\Leftarrow$) A similar argument as in ($\Rightarrow$) is used where every occurrence of p and q are respectively replaced by q and p.

- Induction step: Suppose that Lemma A.5 holds for $k-1$, we show that Lemma A.5 also holds for k.

  1. We show that for any group g, $MsgBundle_p^k[g] = MsgBundle_q^k[g]$. From the induction hypothesis, $View_p^{k-1} = View_q^{k-1}$, therefore when p and q execute line 29 in round $k-1$, p and q respectively set $MsgBundle_p[g]$ and $MsgBundle_q[g]$ either (a) to $\perp$ or (b) to $\emptyset$.
     - In case (a), a similar argument as in the base step of 1 is used where every occurrence of 1 is replaced by k.
     - In case (b), from the algorithm, $MsgBundle_p^k[g] = MsgBundle_q^k[g] = \emptyset$
  2. A similar argument as in the base step of 2 is used where every occurrence of 1 is replaced by k. □

Proposition A.5 (Uniform Integrity) *For any process p and any message m, (a) p A-Delivers m at most once, and (b) only if $p \in m.dst$ and (c) m was previously A-MCast.*

Proof:

- (a) Process p A-Delivers m either (a-i) at line 12 or (a-ii) at line 26.

  - In case (a-i), m was A-MCast by a process in $group(p)$ and $m.dst = \{group(p)\}$. Let k be the round in which p A-Delivers m for the first time. By the uniform agreement property of consensus, in round k, all processes q in $group(p)$ that decide on consensus, decide on the same set of messages $msgs$. From the algorithm, $m \in msgs$. Consequently, all q A-Deliver m in round k for the first time. Moreover, all q add m to $Adelivered$ at line 13 in round k. Therefore, no process in $group(p)$ proposes m to consensus in a round bigger than k, and p does not A-Deliver m a second time.

- In case (a-ii), let g be the group from which m was A-MCast. Moreover, let k be the first round in which p A-Delivers m. From the algorithm, (*) $\mathit{MsgBundle}_p^k[g] = \mathit{msgs}$ for some set of messages msgs such that $m \in \mathit{msgs}$. By Lemma A.5, for all processes q such that $\mathit{MsgBundle}_q^k$ is defined $\mathit{MsgBundle}_q^k[g] = \mathit{msgs}$. Consequently, all q that start round $k+1$ add m to $\mathit{Adelivered}$ at line 27 in round k and no process in g proposes m to consensus in a round $k' > k$. Therefore, there exists no $k' > k$ such that $m \in MsgBundle_p^{k'}[g]$ and p does not A-Deliver m a second time.

- (b) follows directly from the algorithm

- (c) Process p A-Delivers m only if p R-Delivers m. From the uniform integrity of reliable multicast m was R-MCast. Consequently, m was A-MCast. □

Proposition A.6 (Uniform Prefix Order) *For any two messages m and m' and any two processes p and q such that $\{p, q\} \in m.dst \cap m'.dst$, 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'.*

Proof: Let k and k' be the rounds in which p A-Delivers m and q A-Delivers m' respectively. Either (a) $k < k'$, (b) $k = k'$, or (c) $k > k'$.

- In case (a), either p A-Delivers m (a-i) at line 12 or (a-ii) at line 26.
  - In case (a-i), $m.dst = \{\mathit{group}(p)\}$ and $\mathit{group}(p) = \mathit{group}(q)$. Since $k < k'$ and q A-Delivers m' in round k', q decides in instance k of consensus. Because p A-Delivers m at line 12 in round k, in consensus instance k, p decides on a set of messages msgs such that $m \in \mathit{msgs}$. From the uniform agreement property of consensus, q decides on msgs in consensus instance k. Therefore, q A-Delivers m before m'.
  - In case (a-ii), there exists a group g and a set of messages msgs such that $m \in \mathit{msgs}$ and $\mathit{MsgBundle}_p^k[g] = \mathit{msgs}$. Since $k < k'$, $\mathit{MsgBundle}_q^k$ is defined. By Lemma A.5, $\mathit{MsgBundle}_p^k[g] = \mathit{MsgBundle}_q^k[g]$. Therefore, q A-Delivers m before m'.

- In case (b), either (b-i) both m and m' are A-Delivered at line 12, (b-ii) both m and m' are A-Delivered at line 26, or (b-iii) m and m' are not A-Delivered at the same line.
  - In case (b-i), $m.dst = m'.dst = \{\mathit{group}(p)\}$. Moreover, in consensus instance k, p and q decide on sets msgs and msgs' respectively such that $m \in \mathit{msgs}$ and $m' \in \mathit{msgs}'$. By the uniform agreement property of consensus, $\mathit{msgs} = \mathit{msgs}'$. Therefore, since messages in msgs are A-Delivered at line 12 in a deterministic order, either p A-Delivers m' before m or q A-Delivers m before m'.
  - In case (b-ii), there exist groups g and g' as well as sets of messages msgs and msgs' such that $m \in \mathit{msgs}$, $m' \in \mathit{msgs}'$, $\mathit{MsgBundle}_p^k[g] = \mathit{msgs}$, and $\mathit{MsgBundle}_q^k[g'] = \mathit{msgs}'$. By Lemma A.5, $\mathit{MsgBundle}_p^k = \mathit{MsgBundle}_q^k$. Therefore, since messages are A-Delivered in a deterministic order at line 26, either p A-Delivers m' before m or q A-Delivers m before m'.
  - In case (b-iii), either p A-Delivers m (b-iii-*) at line 12 or (b-iii-**) at line 26.

    - In case (b-iii-*), $m.dst = \{group(p)\}$ and in consensus instance k, p decides on a set of messages $msgs$ such that $m \in msgs$. Moreover, since q A-Delivers m' at line 26, q decides in consensus instance k. From the uniform agreement property of consensus, q decides on $msgs$. Therefore, q A-Delivers m before m'.
    - In case (b-iii-**), the same argument as in (b-iii-*) is used where every occurrence of m, m', p, and q are respectively replaced by m', m, q, and p.

- In case (c), a similar argument as in (a) is used where every occurrence of p, q, m, m', k, and k' are respectively replaced by q, p, m', m, k', and k. □

Lemma A.6 *For any correct process p and any k, p eventually A-Delivers the global messages of round k at line 26.*

Proof: We proceed by induction on k.

- Base step ($k = 1$): Suppose, by way of contradiction, that p never executes line 26 in round 1. Therefore, (*) there exists a group g such that $MsgBundle_p[g] \in \{\bot, \top\}$ forever in round 1. From the termination property of consensus, p eventually decides in consensus instance 1 and executes the while loop of lines 17-24. Hence from (**), p never G-Delivers a message of the form (1, g, *type*, -) where $type$ is equal to $remove$ or $msgBundle$ at line 35. Either (a) g is correct or (b) g is faulty.
  - In case (a), since $View$ is initialized to Γ, there exists a correct process q in g that G-BCasts a message of the form (1, g, $msgBundle$, -) at line 14. From the validity property of generic broadcast p eventually G-Delivers this message, a contradiction to (**).
  - In case (b), from the strong completeness property of $\diamond\mathcal{P}$, p eventually stops trusting processes in g and G-BCasts a message of the form (1, g, *remove*, -) at line 32. From the validity property of generic broadcast, p eventually G-Delivers this message, a contradiction to (**).

- Induction step: Suppose that Lemma A.6 holds for $k - 1$, we show that Lemma A.6 also holds for k. From the induction hypothesis, p eventually starts consensus instance k. By the termination property of consensus, p eventually decides and executes the while loop of lines 17-24 in round k. Suppose, by way of contradiction, that (*) there exists a group g such that $MsgBundle_p[g] \in \{\bot, \top\}$ forever in round k. Hence, (**) p never G-Delivers a message of the form (k, g, *type*, -) where $type$ is equal to $remove$ or $msgBundle$ at line 35. Either (a) $g \in View_p^{k-1}$ or (b) $g \notin View_p^{k-1}$.
  - In case (a), either (a-i) g is correct or (a-ii) g is faulty.
    - In case (a-i), there exists a correct process $q \in g$. From hypothesis (a), $g \in View_p^{k-1}$. By Lemma A.5, $View_p^{k-1} = View_q^{k-1}$ and thus $g \in View_q^{k-1}$. Therefore, q G-BCasts a message of the form ($k, g, msgBundle$, -) at line 14. From the validity property of generic broadcast p eventually G-Delivers this message, a contradiction to (**).
    - In case (a-ii), from the strong completeness property of $\diamond\mathcal{P}$, p eventually stops trusting processes in g and G-BCasts a message of the form ($k, g, remove$, -) at line 32. From the validity property of generic broadcast, p eventually G-Delivers this message, a contradiction to (**).
  - In case (b), p sets $MsgBundle_p[g]$ to $\emptyset$ at line 29 in round $k - 1$. Therefore, there is a time at which $MsgBundle_p[g] \notin \{\bot, \top\}$ in round k, a contradiction to (*).

Proposition A.7 (Uniform Agreement) *For any message m, if a process p A-Delivers m, then all correct processes $q \in m.dst$ eventually A-Deliver m.*

Proof: Let k be the round in which p A-Delivers m and let g be the group from which m is A-MCast. Either (a) $m.dst = \{g\}$ or (b) $m.dst \neq \{g\}$.

- In case (a), in consensus instance k, p decides on a set of messages $msgs$ such that $m \in msgs$. Since q is correct, by Lemma A.6, q eventually A-Delivers the global messages of round $k-1$ at line 26. Consequently, q starts consensus instance k, and by the termination property of consensus, q decides in that instance. By the uniform agreement property of consensus, q decides on $msgs$ in consensus instance k. Therefore, q eventually A-Delivers m.
- In case (b), from the algorithm, $MsgBundle_p^k[g] = msgs$ for some set of messages $msgs$ such that $m \in msgs$. Since q is correct, by Lemma A.6, q eventually A-Delivers the global messages of round k at line 26 and thus $MsgBundle_q^k$ is defined. By Lemma A.5, $MsgBundle_p^k[g] = MsgBundle_q^k[g]$. Therefore, q eventually A-Delivers m. □

Lemma A.7 *For any correct processes p and q, there exists a round k such that for all $k' \geq k$, $group(p) \in View_q^{k'}$.*

Proof: By the eventual strong accuracy of $\diamond\mathcal{P}$, there is a time after which no process stops being trusted before it crashes. Since p is correct, there exists a time after which processes always trust p. Therefore, (*) there exists round k_{no-rmv} such that for all $k' \geq k_{no-rmv}$ no process G-BCasts a message of the form $(k', group(p), remove,$ -). Since process p and processes q are correct, by Lemma A.6, processes p and q execute an infinite number of rounds. From the algorithm, for any round k' such that $group(p) \notin View_p^{k'-1}$, p G-BCasts a message of the form (-, $group(p)$, add, -). Since p is correct, by the validity property of generic broadcast all such messages are eventually G-Delivered by all correct processes. Hence, from (*), there exists a round $k \geq k_{no-rmv}$ such that $group(p)$ is in $View_p^k$ and $group(p)$ is never removed from $View_p$ anymore, i.e., for all $k' \geq k$, $group(p) \in View_p^k$. Thus, by Lemma A.5, for any $k' \geq k$, $group(p) \in View_q^{k'}$. □

Proposition A.8 (Validity) *If a correct process p A-MCasts m, then all correct processes $q \in m.dst$ eventually A-Deliver m.*

Proof: Suppose, by way of contradiction, that there exists a correct process $r \in m.dst$ that never A-Delivers m. By Proposition A.7, no correct process $q \in m.dst$ A-Delivers m (otherwise r would A-Deliver m). If p A-MCasts m, then p R-MCasts m to $group(p)$. Since p is correct, by the validity property of reliable multicast, all correct processes $s \in group(p)$ eventually R-Deliver m and add m to $Rdelivered_s$ at line 7. Since no correct process $q \in m.dst$ A-Delivers m, after t, $m \in Rdelivered_s \setminus Adelivered_s$ forever. Hence, there exists a round k_1 such that for all $k' \geq k_1$, processes s always propose m to consensus instance k' and thus by the uniform integrity and uniform agreement properties of consensus, (*) processes in $group(p)$ decide on a set of messages $msgs$ such that $m \in msgs$ in consensus instance k'. Either (a) $m.dst = \{group(p)\}$ or (b) $m.dst \neq \{group(p)\}$.

- In case (a), r A-Delivers m in round k_1 at line 12, a contradiction.
- In case (b), by Lemma A.7, there exists a round k_2 such that for any $k' \geq k_2$, $group(p) \in View_q^{k'}$. Hence, from (*), there exists a round $k' = \max(k_2, k_1)$ such that: (1) $group(p) \in View_q^{k'}$ and (2) processes in $group(p)$ G-BCast a message at line 14 of the form $(k', group(p), msgBundle, msgs)$ such that $m \in msgs$. Therefore, r A-Delivers m in round k' at line 26, a contradiction. □

A.3 The Proof of Algorithm $\mathcal{A}3$

Proposition A.9 (Uniform Integrity) *For any process p and any message m, (a) p F-Delivers m at most once, and (b) only if $p \in m.dst$ and (c) m was previously F-MCast.*

Proof:

- (a) Follows directly from the uniform integrity property of reliable multicast and from the fact that a message is removed from $msgSet_p$ after it has been F-Delivered.
- (b) Follows directly from the algorithm.
- (c) Process p F-Delivers m only if p R-Delivered m. From the uniform integrity property of reliable multicast, m was R-MCast. Consequently, m was F-MCast. $\square$

Proposition A.10 Uniform Fifo Order *If a process p F-MCasts a message m before F-MCasting a message m', then no process in $m.dst \cap m'.dst$ F-Delivers m' unless it has previously F-Delivered m.*

Proof: Let q be any process in $m.dst \ \cap m'.dst$ that F-Delivers m', we show that q F-Delivers m before. If q F-Delivers m', then there is a time t before q F-Delivers m' at which $nbDel[m.sender]_q^t = m'.nbCast[group(q)]$. By the definition of m, $m.nbCast[group(q)] < m'.nbCast[group(q)]$. From line 14-16, q must have F-Delivered m before t, and thus before q F-Delivers m'. $\square$

Definition A.3 *We define the binary relation $pred$ on messages as follows, m_1 $pred$ m_2 iff:*

1. *$m_1.sender = m_2.sender$,*
2. *$m_1.sender$ F-MCasts m_1 before m_2, and*
3. *There exists at least one correct group in $m_1.dst \cap m_2.dst$*

Moreover, let $\mathcal{G}_{pred(m)} = (V, E)$ be a finite DAG constructed as follows:

1. *add vertex m to V*
2. *while $\exists m_1 \in V : \exists m_2 \notin V : m_2$ $pred$ m_1 do:*
   add m_2 to V and add directed edge $m_2 \rightarrow m_1$ to E

Finally, let m_k be any message in $\mathcal{G}_{pred(m)}$ such that the longest path from m_k to m is of length k.

Lemma A.8 *For any message m, if for all messages m' in $\mathcal{G}_{pred(m)}$ all correct processes in $m'.dst$ R-Deliver m', then all correct processes $p \in m.dst$ eventually F-Deliver m.*

Proof: Assume that for all messages m' in $\mathcal{G}_{pred(m)}$ all correct processes in $m'.dst$ R-Deliver m'. We prove that, for any $k \geq 0$, all messages m_k in $\mathcal{G}_{pred(m)}$ are eventually F-Delivered by all correct processes in $m_k.dst$. Since $m_0 = m$, this shows the claim. Let x be the largest integer such that m_x is in $\mathcal{G}_{pred(m)}$. We proceed by induction on k, starting from $k = x$.

- Base step ($k = x$): From the definition of m_x, (*) there exists no message m_{x+1} such that $m.sender$ F-MCasts m_{x+1} before m_x and there exists at least one correct group in $m_x.dst \cap m_{x+1}.dst$. Since for all messages m' in $\mathcal{G}_{pred(m)}$, all correct processes in $m'.dst$ eventually R-Deliver m', all correct processes in $m_x.dst$ eventually R-Deliver m_x. Let q be any correct process in $m_x.dst$. By (*), m_x is the first message F-MCast by $m.sender$ such that $q \in m_x.dst$, and hence, $m_x.nbCast[group(q)] = 0$. Therefore, all correct processes q in $m_x.dst$ eventually R-MCast (ACK, $m_x.id, q$) and by the validity property of reliable multicast, all q eventually R-Delivers that message. By the strong completeness property of Θ, q eventually stops trusting faulty processes. Consequently, all q eventually F-Deliver m_x.

- Induction step: Suppose the claim holds for k ($0 < k \leq x$), we show it holds for $k-1$. Let q be any correct process in $m_{k-1}.dst$. By the induction hypothesis, q F-Delivers m_k. From the algorithm, (*) there exists a time t at which $nbDel[m.sender]_q^t = m_{k-1}.nbCast[group(q)]$. Since for all messages m' in $\mathcal{G}_{pred(m)}$, all correct processes in $m'.dst$ eventually R-Deliver m', all correct processes in $m_{k-1}.dst$ eventually R-Deliver m_{k-1}. Consequently, from (*), all q R-MCast (ACK, $m_{k-1}.id, q$), either at line 13, if q R-Delivers m_{k-1} after F-Delivering m_k, or at line 19 otherwise. By the validity property of reliable multicast all q eventually R-Deliver that message. By the strong completeness property of Θ, q eventually stops trusting faulty processes. Consequently, all q eventually F-Deliver m_{k-1}. $\square$

Lemma A.9 *For any message m and any process p, if p R-MCasts (ACK, $m.id, p$), then p F-Delivered all messages m' such that $p \in m'.dst$ and $m.sender$ F-MCast m' before m.*

Proof: If m is the first message $m.sender$ F-MCasts to $group(p)$, the claim holds trivially. Otherwise, let m_x be the message such that $p \in m_x.dst$ and $m.sender$ F-MCasts m_x just before m. Since p R-MCasts (ACK, $m.id, p$), there exists a time t at which $nbDel[m.sender]_p^t = m.nbCast[group(p)]$. From lines 14-16, p must have F-Delivered m_x before t. By applying Proposition A.10 multiple times, before t, p also F-Delivered all messages addressed to $group(p)$ that $m.sender$ F-MCast before m_x. $\square$

Proposition A.11 (Uniform Agreement) *If a process p F-Delivers a message m, then all correct processes $q \in m.dst$ eventually F-Deliver m.*

Proof: We first show that, for any $k \geq 0$ such that m_k exists in $\mathcal{G}_{pred(m)}$: (1) m_k is R-Delivered by all correct processes in $m_k.dst$ and (2) for each correct group $g \in m_k.dst$, there is a correct process q in g that R-MCasts (ACK, $m_k.id, q$). We proceed by induction on k.

- Base step ($k = 0$):
  - (1) Since p F-Delivers m, p R-Delivered m. By the uniform agreement property of reliable multicast, all correct processes in $m.dst$ eventually R-Deliver m.
  - (2) Since p F-Delivers m, from the algorithm and the accuracy property of Θ, for every correct group g in $m.dst$, p R-Delivers a message (ACK, $m.id, q$) for some correct process q in g. Hence, by the uniform integrity property of reliable multicast, q R-MCasts (ACK, $m.id, q$).
- Induction step: Suppose that (1) and (2) hold for $k-1$ ($k > 0$), we show that (1) and (2) also hold for k.
  - (1) Because $k > 0$, from the definition of m_k, there exists a message m_{k-1} in $\mathcal{G}_{pred(m)}$ such that there is an edge from m_k to m_{k-1}. From the definition of $\mathcal{G}_{pred(m)}$, m_k $pred$ m_{k-1}, and thus $m_k.sender$ F-MCasts m_k before m_{k-1} and there exists a correct group in $m_k.dst \cap m_{k-1}.dst$. By the induction hypothesis, for each correct group g in m_{k-1} there exists a correct process q in g that R-MCasts (ACK, $m_{k-1}.id, q$). Hence, there exists a correct process q in $m_{k-1}.dst \cap m_k.dst$

such that q R-MCasts (ACK, $m_{k-1}.id, q$). By Lemma A.9, q F-Delivered m_k. From the algorithm, q R-Delivered m_k. By the uniform agreement property of reliable multicast, all correct processes in $m_k.dst$ eventually R-Deliver m_k.

– (2) Using the same argument as in (1), there exists a message m_{k-1} in $\mathcal{G}_{pred(m)}$ such that $m_k.sender$ F-MCasts m_k before m_{k-1} and there exists a correct group in $m_k.dst \cap m_{k-1}.dst$. By the induction hypothesis and the definition of m_{k-1}, there exists a correct process $r \in m_k.dst \cap m_{k-1}.dst$ such that r R-MCasts (ACK, $m_{k-1}.id, r$). By Lemma A.9, r F-Delivered m_k. From the algorithm and the accuracy property of Θ, for each correct group $g \in m_k.dst$ r R-Delivered (ACK, $m_k.id, q$) for some correct process $q \in g$. By the uniform integrity property of reliable multicast, q R-MCasts (ACK, $m_k.id, q$).

Hence, from (1), all messages $m' \in \mathcal{G}_{pred(m)}$ are R-Delivered by all correct processes in $m'.dst$. Therefore, by Lemma A.8, all correct processes in $m.dst$ F-Deliver m. □

Proposition A.12 (Validity) *If a correct process p F-MCasts a message m, then eventually all correct processes $q \in m.dst$ F-Deliver m.*

Proof: Since p is correct, by the validity property of reliable multicast, for all messages $m' \in \mathcal{G}_{pred(m)}$, all correct processes in $m'.dst$ R-Deliver m. By Lemma A.8, all correct processes $q \in m.dst$ F-Deliver m. □

A.4 The Proof of Algorithm $\mathcal{A}4$

Proposition A.13 (Uniform Integrity) *For any process p and any message m, (a) p C-Delivers m at most once, and (b) only if $p \in m.dst$ and (c) m was previously C-MCast.*

Proof:

- (a) Follows directly from the uniform integrity property of fifo multicast and from the fact that a message is removed from $msgLst_p$ after it is C-Delivered.
- (b) Follows directly from the algorithm.
- (c) Process p C-Delivers m only if p F-Delivered m. From the uniform integrity property of fifo multicast, m was F-MCast. Consequently, m was C-MCast. □

Lemma A.10 *For any any message m such that $m.nbDel$ is defined, any group g, and any integer k, if $m.nbCast[g][m.sender] = k$, then m is the $k+1$-th message $m.sender$ C-MCasts to g.*

Proof: From the algorithm, $m.sender$ increments $nbCast[g][m.sender]_{m.sender}$ at line 10 only ($m.sender$ does not increment $nbCast[g][m.sender]_{m.sender}$ at line 19). Moreover, $m.sender$ does so after every message C-MCast to g. Therefore, since $nbCast[g][m.sender]$ is initialized to 0, m is the $k+1$-th message $m.sender$ C-MCasts to g. □

Lemma A.11 *For any two processes p and q, and any group g:*

1. *$nbDel[g][q]_p$ is monotonically increasing with time.*
2. *$nbCast[g][q]_p$ is monotonically increasing with time.*

Proof:

1. Holds trivially from line 20 and line 22.

2. Holds trivially from line 10 and line 19. □

Lemma A.12 *For any two messages m and m' such that C-MCast(m) $\rightarrow$ C-MCast(m'), and any group $g \in m.dst \cap m'.dst$, $m.nbDel[g] \leq m'.nbDel[g]$.*

Proof: If $m.sender = m'.sender$, from Lemma A.11, Lemma A.12 holds trivially. Now suppose that $m.sender \neq m'.sender$. Since C-MCast(m) $\rightarrow$ C-MCast(m'), there exist processes p_1, ..., p_k and messages m_1, ..., $m_k = m'$ ($k \geq 2$) such that:

1. $p_1 = m.sender$

2. p_i C-MCasts m_i for all $1 \leq i \leq k$

3. either (a) $m = m_1$ or (b) p_1 C-MCasts m before m_1, and

4. p_i C-Delivers m_{i-1} before C-MCasting m_i for all $2 \leq i \leq k$

We first show that $m.nbDel[g] \leq m_1.nbDel[g]$. From 3, either (a) $m = m_1$ or (b) p_1 C-MCasts m before m_1.

- In case (a), the claim holds trivially as $m = m_1$.

- In case (b), from Lemma A.11, for any process q, $nbDel[g][q]_{p_1}$ is monotonically increasing with time and thus the claim holds.

To conclude the proof, we show that for all $1 \leq i < k$, $m_i.nbDel[g] \leq m_{i+1}.nbDel[g]$. We proceed by induction on i.

- Base step ($k = 1$): From 4, p_2 C-Delivers m_1 before C-MCasting m_2. From line 12, there exists a time t before p_2 C-MCasts m_2 at which $nbDel[g]_{p_2} \geq m_1.nbDel[g]$. From Lemma A.11, for any process q, $nbDel[g][q]_{p_2}$ is monotonically increasing with time. Therefore, $m_1.nbDel[g] \leq m_2.nbDel[g]$.

- Induction step: Suppose that the claim holds for all i, we show it also holds for $i + 1$ ($1 \leq i < k - 1$). We use the same argument as in the base step to show that $m_{i+1}.nbDel[g] \leq m_{i+2}.nbDel[g]$. □

Lemma A.13 *For any two messages m and m' such that C-MCast(m) $\rightarrow$ C-MCast(m'), and any group $g \in m.dst \cap m'.dst$:*
$m.nbCast[g][m.sender] < m'.nbDel[g][m.sender]$ or $m.nbCast[g][m.sender] < m'.nbCast[g][m.sender]$.

Proof: Since C-MCast(m) $\rightarrow$ C-MCast(m'), either (a) there exists a process p that C-Delivers m and p C-MCasts a message m'' such that C-MCast(m'') $\rightarrow$ C-MCast(m') or (b) $m.sender$ C-MCasts m before m'.

- In case (a), after p C-Delivers m, p sets $nbDel[g][m.sender]_p$ to at least $m.nbCast[g][m.sender] + 1$ at line 22. From Lemma A.11, $nbDel[g][m.sender]_p$ is monotonically increasing with time, and thus $m.nbCast[g][m.sender] < m''.nbDel[g][m.sender]$. Since C-MCast($m''$) $\rightarrow$ C-MCast(m'), from Lemma A.12, $m''.nbDel[g][m.sender] \leq m'.nbDel[g][m.sender]$, and thus $m.nbCast[g][m.sender] < m'.nbDel[g][m.sender]$.

- In case (b), $m.sender$ increments $nbCast[g][m.sender]$ at line 10 after F-MCasting m. From Lemma A.11, $nbCast[g][m.sender]_{m.sender}$ is monotonically increasing with time. Therefore, $m.nbCast[g][m.sender] < m'.nbCast[g][m.sender]$. □

Lemma A.14 *For any process p and q, and any integer k, if there exists a time t at which $nbDel[group(p)][q]_p^t = k$, then, before t, p C-Delivered the first k messages q C-MCasts to $group(p)$.*

Proof: If there exists a time t at which $nbDel[group(p)][q]_p^t = k$, then from line 22, before t, p C-Delivered a message m_k q C-MCast such that $m_k.nbCast[group(p)][q] = k - 1$. From Lemma A.10, m_k is the k-th message q C-MCasts to $group(p)$. Let m_i be the i-th message q C-MCast to $group(p)$. We show that for all $1 \leq i < k$, p C-Delivered m_i before m_k. From the definition of m_i and Lemma A.10, (*) $m_i.nbCast[group(p)][q] < m_k.nbCast[group(p)][q]$. Since C-MCast($m_i$) $\rightarrow$ C-MCast(m_k), from Lemma A.12, (**) $m_i.Del[group(p)][q] \leq m_k.nbDel[group(p)][q]$. From the uniform fifo order property of fifo multicast, p F-Delivers m_i before m_k. Hence, p adds m_i to $msgLst_p$ before F-Delivering m_k. Therefore, from line 16, (*), and (**), p C-Delivered m_i before m_k. □

Proposition A.14 Uniform Causal Order *For any messages m and m', if C-MCast(m) $\rightarrow$ C-MCast(m'), then no process $p \in m.dst \cap m'.dst$ C-Delivers m' unless it has previously C-Delivered m.*

Proof: Let q be any process in $m.dst \cap m'.dst$ that C-Delivers m', we show that q C-Delivered m before. In the proof below, we denote $group(q)$, $m.sender$ and $m'.sender$ as g_q, s_m, and $s_{m'}$ respectively. By Lemma A.13, $m.nbCast[g_q][m.sender] < m'.nbDel[g_q][m.sender]$ or $m.nbCast[g_q][m.sender] < m'.nbCast[g_q][m.sender]$. Since q A-Delivers m', from line 12, there exists a time t before q A-Delivers m' at which $m'.nbDel[g_q][s_m] \leq nbDel[g_q][s_m]_q^t$ and $m'.nbCast[g_q][s_m] \leq nbDel[g_q][s_m]_q^t$. Let k and k' respectively be $m.nbCast[g_q][s_m]$ and $nbDel[g_q][s_m]_q^t$. From the above, (*) $k < k'$. By Lemma A.10, m is the $k+1$-th message s_m C-MCasts to $group(q)$. By Lemma A.14, before t, q C-Delivered the first k' messages s_m C-MCast to $group(q)$. From (*), q C-Delivers m before m'. □

Definition A.4 *Let m be a message, we define the finite DAG $\mathcal{G}_{pred(m)} = (V, E)$ as follows:*

1. *add vertex m to V*
2. *while $\exists m_1, m_2$ s.t. $m_1 \in V \wedge$ C-MCast(m_2) $\rightarrow$ C-MCast(m_1) $\wedge$ $m_1.dst \cap m_2.dst \neq \emptyset$ do:*
   add m_2 to V and add directed edge $m_2 \rightarrow m_1$ to E

Moreover, we define m_k as any message in $\mathcal{G}_{pred(m)}$ such that the longest path from m_k to m is of length k.

Lemma A.15 *For any two processes p and q:*

1. *p increments $nbDel[group(p)][q]_p$ at line 20 or line 22 only when C-Delivering a message m such that $group(p) \in m.dst$.*
2. *p increments $nbCast[group(p)][q]_p$ at line 19 only when C-Delivering a message m such that $group(p) \in m.dst$.*

Proof:

1. Variable $nbDel[group(p)][q]_p$ is updated either (a) at line 20 or (b) at line 22.

- In case (a), suppose, by way of contradiction, that $group(p) \notin m.dst$. From the algorithm, there must exist a message m' such that C-MCast(m') $\rightarrow$ C-MCast(m), $group(p) \in m'.dst$, $m'.sender = q$, and $m'.nbCast[group(p)][q] = m.nbCast[group(p)][q]$. By Proposition A.14, p must have C-Delivered m' before m. Therefore, from line 22, $nbDel[group(p)][q]_p \geq m.nbCast[group(p)][q]+1$ when p C-Delivers m', and thus p does not increment $nbDel[group(p)][q]_p$ when C-Delivering m, a contradiction.
- In case (b), from the condition of line 21, $group(p) \in m.dst$.

2. Suppose, by way of contradiction, that $m \notin group(p)$. From the algorithm, there must exist a message m' such that C-MCast(m') $\rightarrow$ C-MCast(m), $group(p) \in m'.dst$, $m'.sender = q$, and $m'.nbCast[group(p)][q] = m.nbCast[group(p)][q]$. By Proposition A.14, p must have C-Delivered m' before m. Therefore, from line 19, $nbCast[group(p)][q]_p \geq m.nbCast[group(p)][q]$ when p C-Delivers m', and thus p does not increment $nbCast[group(p)][q]_p$ when C-Delivering m, a contradiction. □

Lemma A.16 *For any message m, if for all messages $m' \in \mathcal{G}_{pred(m)}$ all correct processes in $m'.dst$ F-Deliver m', then all correct processes in $m.dst$ eventually C-Deliver m.*

Proof: We prove that, for any $k \geq 0$, all messages m_k in $\mathcal{G}_{pred(m)}$ are eventually C-Delivered by all correct processes in $m_k.dst$. Since $m_0 = m$, this shows the claim. Let x be the largest integer such that m_x is in $\mathcal{G}_{pred(m)}$. We proceed by induction on k, starting from $k = x$.

- Base step ($k = x$): From the definition of m_x, (*) there exists no message m' such that C-MCast(m') $\rightarrow$ C-MCast(m_x) and $m'.dst \cap m_x.dst \neq \emptyset$. Let g be any group in $m_x.dst$ and let p be any process. From (*), $m_x.sender$ never updated $nbDel[g][p]_{m_x.sender}$ at line 20 nor at line 22. Moreover, $m_x.sender$ never updated $nbCast[g][q]_{m_x.sender}$ at line 19 and m_x is the first message $m_x.sender$ C-MCasts to g. Therefore, (**) $m.nbDel[g][p] = 0$ and $m.nbCast[g][p] = 0$. By Lemma A.11, for any process q, $nbDel[g][p]_q$ is monotonically increasing with time. If all correct processes in $m_x.dst$ eventually F-Deliver m_x, from (**) and line 12, all correct processes in $m_x.dst$ eventually C-Deliver m_x.
- Induction step: Suppose that for any l such that $x \geq l > k \geq 0$ the claim holds, we show the claim holds for k. If there exists no correct process in $m_k.dst$, then the claim holds trivially. Otherwise, let q be any correct process in $m_k.dst$ and let p be any process. We show that there exist times t_1, t_2 at which (a) $nbDel[group(q)][p]_q^{t_1} \geq m_k.nbDel[group(q)][p]$ and (b) $nbDel[group(q)][p]_q^{t_2} \geq m_k.nbCast[group(q)][p]$.
  - (a) Let m_p be the message that $m_k.sender$ C-Delivered which set $nbDel[group(q)][p]_{m_k.sender}$ to $m_k.nbDel[group(q)][p]$. We argue that q eventually C-Delivers m_p and thus sets $nbDel[group(q)][p]_q$ to at least $m_k.nbDel[group(q)][p]$ at line 20. By Lemma A.15, m_p is such that $group(q) \in m_p.dst$. From the definition of m_p, C-MCast(m_p) $\rightarrow$ C-MCast(m_k). Hence, since $group(q) \in m_p.dst$, there exists a $k' > k$ such that $m_{k'} \in \mathcal{G}_{pred(m)}$ and $m_p = m_{k'}$. By the induction hypothesis, q eventually C-Delivers m_p.
  - (b) Either (b-i) $p = m_k.sender$ or (b-ii) not.
    - In case (b-i), from Lemma A.10, if $m_k.nbCast[group(q)][m_k.sender] = k$, then m_k is the $k+1$-th message $m_k.sender$ C-MCasts to $group(q)$. If $k = 0$, then the claim holds trivially

since $nbDel[group(q)][p]_q$ is initialized to zero. Otherwise, let m_{k-1} be the k-th message $m_k.sender$ C-MCasts to $group(q)$. From Lemma A.10, $m_{k-1}.nbCast[group(q)][m_k.sender] = k - 1$. By the induction hypothesis, q eventually C-Delivers m_{k-1} and sets $nbDel[group(q)][m_k.sender]_q$ to at least k at line 22.

$*$ In case (b-ii), there must exist a message m_p such that $m_p.sender = p$, $group(q) \in m_p.dst$, C-MCast(m_p) $\rightarrow$ C-MCast(m_k), and $m_p.nbCast[group(q)][p] = m_k.nbCast[group(q)][p]$. From Lemma A.10, if $m_p.nbCast[group(q)][m_k.sender] = k$, then m_p is the $k+1$-th message p C-MCasts to $group(q)$. If $k = 0$, then the claim holds trivially since $nbDel[group(q)][p]_q$ is initialized to zero. Otherwise, let m_{k-1} be the k-th message p C-MCasts to $group(q)$. From Lemma A.10, $m_{k-1}.nbCast[group(q)][m_k.sender] = k - 1$. By the induction hypothesis, q eventually C-Delivers m_{k-1} and sets $nbDel[group(q)][m_k.sender]_q$ to at least k at line 22.

Since (a) and (b) hold, if all correct processes in $m_k.dst$ eventually F-Deliver m_k, by Lemma A.11 and line 12, all correct processes in $m_k.dst$ eventually C-Deliver m_k. □

Proposition A.15 (Uniform Agreement) *If a process p C-Delivers a message m, then all correct processes $q \in m.dst$ eventually C-Deliver m.*

Proof: We show that for any message $m_k \in \mathcal{G}_{pred(m)}$ all correct processes in $m_k.dst$ F-Deliver m'. By Lemma A.16, all correct processes $q \in m.dst$ eventually C-Deliver m. We proceed by induction on k.

- Base step ($k = 0$): From the definition of m_k, $m_0 = m$. Since p C-Delivers m, p F-Delivers m. From the uniform agreement property of fifo multicast, all correct processes in $m.dst$ eventually F-Deliver m.

- Induction step: Suppose the claim holds for k, we show the claims holds for $k + 1$. From the definition of $\mathcal{G}_{pred(m)}$, message m_{k+1} is such that $m_k.dst \cap m_{k+1}.dst \neq \emptyset$ and C-MCast(m_{k+1}) $\rightarrow$ C-MCast(m_k). From the induction hypothesis, all correct processes in $m_k.dst$ eventually C-Deliver m_k. Hence, by Proposition A.14, all processes in $m_{k+1}.dst$ must have C-Delivered m_{k+1} before m_k. □

Proposition A.16 (Validity) *If a correct process p C-MCasts a message m, then eventually all correct processes $q \in m.dst$ C-Deliver m.*

Proof: We show that for any message $m_k \in \mathcal{G}_{pred(m)}$ all correct processes in $m_k.dst$ F-Deliver m'. By Lemma A.16, all correct processes $q \in m.dst$ eventually C-Deliver m. We proceed by induction on k.

- Base step ($k = 0$): From the definition of m_k, $m_0 = m$. From the algorithm, p F-MCasts m. Since p is correct, from the validity property of fifo multicast, all correct processes in $m.dst$ eventually F-Deliver m.

- Induction step: Suppose the claim holds for k, we show the claims holds for $k+1$. The same argument as in the induction step of Proposition A.15 is used. □