A hierarchy of totally ordered multicasts*

Uwe Wilhelm André Schiper
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)

Abstract

The increased interest in protocols that provide a total order on message delivery has led to several different definitions of total order. In this paper we investigate these different definitions and propose a hierarchy that helps to better understand the implications of the different possibilities in terms of guarantees and communication cost. We identify two definitions: *weak total order* and *strong total order*, which are at the extremes of the proposed hierarchy, and incorporate them into a consistent design. Finally, we propose high-level algorithms based on a virtually synchronous communication environment that implement the given definitions.

1 Introduction

The increasing attention for the support of fault-tolerance in distributed computing has led to an increased interest in protocols that provide a total order on message delivery. This is due to the necessity of total order for the implementation of a state machine [Sch90]. The state machine approach is the foremost technique of implementing active replication, which is in turn one of the most important approaches to fault-tolerance in distributed systems. When implementing total order multicasts, two system models can be considered: the synchronous model [KDK+89, CASD85] where the message delays are bounded, or the asynchronous model which has no bounds on message delays. We consider here the asynchronous model which constitutes the most general model.

In the context of asynchronous systems, the usual classification of total order algorithms distinguishes between *symmetric* and *asymmetric* algorithms. In a symmetric algorithm, all processes execute the same code [Lam78, CM84, BJ87, LG90, DKM93, CT93, Anc93]; in an asymmetric algorithm, one process plays a special role, e.g. defines the ordering of messages [KTHB89, BSS91]. The asymmetric algorithms are interesting as they are more efficient (they require less phases and/or fewer messages than the symmetric algorithms). The classification asymmetric/symmetric is however not essential to the application programmer: the semantics ensured by the algorithm is what matters. A closer analysis of the asymmetric algorithms [KTHB89, BSS91] shows that they do not ensure the same properties as symmetric algorithms. This key difference has however never been identified up to now. A symmetric algorithm ensures a property that we define below as *strong order*, whereas an asymmetric algorithm ensures what we define as *weak order*. Informally, the strong order property can be defined as

*Research supported by the "Fonds national suisse" and OFES under contract number 21-32210.91, as part of the ESPRIT Basic Research Project BROADCAST (number 6360), and by PPR-IF under contract number 5003-34344.

follows (for an extensive theoretical discussion of different definitions of total order protocols see [HT93]):

> If processes p and q both deliver messages m_1 and m_2, then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2.

This definition provides a rather powerful guarantee concerning the reliability of the ordered delivery of messages. The definition requires that messages be ordered in any process, independently of the correctness of this process. The provided guarantee is, however, rather costly and not required by all applications. Depending on the application semantics, it might be tolerable that only a well defined subset of the processes deliver the messages in a total order. This observation leads us to the definition of the weak order property:

> If correct processes[1] p and q both deliver messages m_1 and m_2, then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2.

Both of these definitions for total order are useful in the right context. Moreover, as shown in the paper, the weak order property, and the strong order property can be combined to define a hybrid order property. To understand this hybrid order property, consider that messages multicast with an algorithm that ensures the strong order property are labeled *strong*, and messages issued with an algorithm that ensures the weak order property are labeled *weak*. The hybrid order property is identical to the *strong order property* between STO and WTO labeled messages. This property is very similar to the hybrid consistency introduced by Attiya and Friedman [AF92].

The paper is organized as follows. In Section 2 we define our model of computation and the vsc-layer, on which we base further definitions. Next, in Section 3 we formally introduce the different definitions of total order multicasts (strong, weak, and hybrid) as well as the resulting hierarchy. In Section 4 we present the corresponding algorithms for strong, weak and hybrid total order multicasts, and prove their correctness. Finally, Section 5 concludes the paper.

2 System Model

We consider a set of processes $\mathcal{P}$, completely interconnected by a set of communication channels. Communication between processes is by message passing, unordered, and asynchronous, i.e. there is no bound on communication delays. Processes fail by crashing (we do not consider Byzantine failures).

This basic system is augmented by a software layer (the *vsc-layer*) that provides the functionality, on which we can easily build the desired total order multicasts [BSS91]. It offers a very strong communication paradigm, where messages are reliable provided that the sending process is considered correct by a majority of processes. This constitutes a layered approach, which is a widely accepted tool to control the complexity of the design of a sophisticated software system.

In order to formalize our presentation, we introduce some fundamental definitions.[2] The activity of a process is modeled as executing a linear sequence of *events*, called *execution history*, in which e^p_x denotes the x^{th} event executed by p. A system *run* is a tuple of process histories, one for each process. Given two events e^p and e^q, $e^p \rightarrow_\rho e^q$ is the happened-before relation, as defined by Lamport [Lam78], on run ρ. Formula $\textsc{before}_\rho(e^p, e^q)$ holds if $e^p \in \rho$, $e^q \in \rho$, and $e^p \rightarrow_\rho e^q$.

[1] The notion of a correct process is very problematic in an asynchronous system and will be dealt with in Section 2.1.

[2] To distinguish logical formulas from events, formulas will be set in SMALL CAPS.

2.1 VSC-layer

The vsc-layer implements virtually synchronous communication (*vsc*), which consists of a process group *membership service* and *view-synchronous* communication between the processes of a group [BSS91].

The membership service structures processes into groups $g \in G$ and manages them in the form of consecutive views on this group: $v_0(g), v_1(g), ..., v_i(g), ...$ (where $v_i(g) \subseteq \mathcal{P}$); the processes in $v_i(g)$ are called members of view $v_i(g)$. For the sake of simplicity, we will —whenever possible— omit the group g from our formulas. The views are defined in the vsc-layer and furnished as view change events to each member of the view (event $\mathit{vscvwchg}^p(v_i)$ occurs when view v_i is furnished to process p); the occurrence of a view change event in the execution history of a process is also referred to as installing a view. The vsc-layer guarantees that the sequence of views is totally ordered; i.e. if two processes install views v_i and v_j, then they do this in the same order. Moreover, the membership service guarantees liveness[3] of the system: if a process $p \in v_i$ has crashed, the membership service will eventually define a new view v_{i+1}, from which p is excluded and furnish this view to all the processes in v_{i+1}. Finally, the membership service provides a *rank* function that assigns a unique rank to each process in the current view v_i (i.e. an integer value between 0 and $n-1$, with n being the number of processes in v_i).

Based on this membership service we can define whether a process p is correct during a given view v_i in run ρ as well as the set of all correct processes during a given view v_i in run ρ:

$$\text{CORRECT}^p_\rho(v_i) \stackrel{\text{def}}{=} \mathit{vscvwchg}^p(v_i) \in \rho \ \wedge \ \mathit{vscvwchg}^p(v_{i+1}) \in \rho$$

$$\mathcal{C}_\rho(v_i) \stackrel{\text{def}}{=} \{p \in \mathcal{P} \mid \text{CORRECT}^p_\rho(v_i)\}.$$

The intuition behind this definition is that a process p can crash, it can be slow, or it can be partitioned from the majority of the processes. In any case, if a process p is unresponsive for some time interval Δ, the membership service will eventually construct a new view v_{i+1}, from which p is excluded. The incorrect process p will not execute the view change v_{i+1} and is thus incorrect according to the definition given above.

The view-synchronous multicast communication between the processes of a given view is defined in terms of two primitives: $\mathit{vscmcast}^p(m)$ and $\mathit{vscdelv}^p(m)$, where m is a message. When the event $\mathit{vscmcast}^p(m)$ occurs in run ρ, we say that p vsc-multicast message m. Similarly, when the event $\mathit{vscdelv}^p(m)$ occurs in run ρ, we say that p vsc-delivered message m. Messages can contain further control information such as a unique identifier $id(m)$ that can easily be extracted from the message. View-synchronous multicast communication guarantees that (1) membership information (view changes), and (2) messages, are ordered in respect with each other in all processes of the corresponding view; i.e. if two processes in view v_i both install view v_{i+1}, then they have delivered the same set of messages between the corresponding view change events.

We define the vsc-delivery of a message m into view v_i at process p:

$$\begin{aligned}\text{VSC\_DELV}^p_{\rho,i}(m) \ \stackrel{\text{def}}{=} \ & \text{BEFORE}_\rho(\mathit{vscvwchg}^p(v_i), \mathit{vscdelv}^p(m)) \ \wedge \\ & \neg\text{BEFORE}_\rho(\mathit{vscvwchg}^p(v_{i+1}), \mathit{vscdelv}^p(m))\end{aligned}$$

With the above definitions we will now formally define the property that is required for view-synchronicity. For simplicity, we will omit the run from our formulas, unless necessary. The multicast of a message m is view-synchronous, if the following formula holds:

[3]This problem in the membership service requires a consensus, which has been shown to be impossible to solve in an asynchronous system [FLP85]. Our vsc-layer therefore ensures liveness with an algorithm that is based on an *imperfect failure detector* ($\diamond \mathcal{W}$) [CT93], which can exclude processes that are "slow".

$$\left(\exists p \in \mathcal{C}(v_i)\ :\ \text{VSC\_DELV}_i^p(m)\right) \Rightarrow \left(\forall q \in \mathcal{C}(v_i)\ :\ \text{VSC\_DELV}_i^q(m)\right)^4$$

If one process in v_i vsc-delivers m before installing the next view v_{i+1}, then all processes in v_i that install the view v_{i+1}, vsc-deliver m before installing v_{i+1}. Implementation of view-synchrony is discussed in [SS93, MS94].

3 Definitions of total order multicasts

A total order multicast has to satisfy several properties, to which we will refer as *delivery requirements*. In the following we will only discuss two properties: *order* and *agreement*. There are others that have to be satisfied by a total order multicast, namely *validity*, which guarantees that all messages multicast by correct processes will eventually be delivered, and *integrity*, which guarantees that no spurious or duplicate messages are ever delivered. These properties are already satisfied by the reliable multicast in the underlying vsc-layer and are therefore automatically provided.

To implement a total order delivery, messages have to be delayed before they can be delivered in an agreed upon order to the application. This can be done in a software layer between the vsc-layer and the application, called *to-layer* (for total order). We model the actions of this layer by introducing new events: $tomcast^p(m)$ (total order multicast), $todelv^p(m)$ (to-delivery of a message by the to-layer to the application), and $tovwchg^p(v_i)$ (delivery of a view change to the application).

Similar to the definition for vsc-delivery in Section 2.1, we define the to-delivery of a message m into view v_i at process p:

$$\begin{aligned} \text{TO\_DELV}_i^p(m) \ \stackrel{\text{def}}{=}\ & \text{BEFORE}(tovwchg^p(v_i), todelv^p(m)) \ \wedge \\ & \neg\text{BEFORE}(tovwchg^p(v_{i+1}), todelv^p(m)) \end{aligned}$$

There is a very close connection between the view changes generated in the vsc-layer and those generated in the to-layer. As we will see in Section 4, the *tovwchg* is triggered by the *vscvwchg* and only preceded by the to-delivery of delayed messages. Given this behaviour we can conclude that for a correct process p, the formula $\text{TO\_DELV}_i^p(m) \Rightarrow \text{VSC\_DELV}_i^p(m)$ holds. Finally we define a notion to express the order in which messages m_1 and m_2 are to-delivered at process p:

$$\text{TO\_DELV}_i^p(m_1; m_2) \stackrel{\text{def}}{=} \text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge \text{BEFORE}(todelv^p(m_1), todelv^p(m_2))$$

We now formally introduce the different total order multicasts we are interested in.

3.1 Weak total order multicast

This definition for a multicast is the one implemented in [KTHB89, BSS91] (e.g. in the ABCAST of the ISIS system). Here are the delivery requirements for weak total order multicast (wto-delivery requirements). Given two messages m_1,m_2 multicast into view v_i, the following properties have to be satisfied:[5]

[4] Another way to characterize this property is: $\forall p, q \in \mathcal{C}(v_i)\ :\ (\text{VSC\_DELV}_i^p(m) \equiv \text{VSC\_DELV}_i^q(m))$. The two formulas are equivalent, but we prefer the first representation because it is more suggestive.

[5] These properties can be united in one formula that captures both total order and agreement: $(\exists p \in \mathcal{C}(v_i)\ :\ \text{TO\_DELV}_i^p(m_1; m_2)) \Rightarrow (\forall q \in \mathcal{C}(v_i)\ :\ \text{TO\_DELV}_i^q(m_1; m_2))$.

W-Order:

$$\forall p,q \in \mathcal{C}(v_i) : \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge \text{TO\_DELV}_i^q(m_1) \wedge \text{TO\_DELV}_i^q(m_2)\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1;m_2) \equiv \text{TO\_DELV}_i^q(m_1;m_2)\Big)$$

W-Agreement:

$$\Big(\exists p \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^p(m)\Big) \Rightarrow \Big(\forall q \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^q(m)\Big)$$

These properties guarantee that during the view v_i, all correct processes have received the same set of messages in the same order. There is no guarantee whatsoever on processes that are member of view v_i, but not correct (i.e. those which do not install view v_{i+1}). These incorrect processes can deliver an arbitrary set of messages in an arbitrary order. Messages sent with a weak total order multicast (wto-multicast) are called wto-messages.

3.2 Strong total order multicast

We can strengthen the delivery requirements of Section 3.1, by requiring that the total order and agreement properties hold for any process (and not only for "correct" processes). This leads us to the definition of the delivery requirements for strong total order multicast (sto-delivery requirements). Given two messages m_1,m_2 multicast into view v_i, the following properties have to be satisfied:

S-Order:

$$\forall p,q \in v_i \;:\; \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge \text{TO\_DELV}_i^q(m_1) \wedge \text{TO\_DELV}_i^q(m_2)\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1;m_2) \equiv \text{TO\_DELV}_i^q(m_1;m_2)\Big)$$

S-Agreement:

$$\Big(\exists p \in v_i \;:\; \text{TO\_DELV}_i^p(m)\Big) \Rightarrow \Big(\forall q \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^q(m)\Big)$$

These properties guarantee that if any process (whether correct or faulty) has delivered a message m_1 before having delivered message m_2, then all processes that deliver both messages, will deliver m_1 before m_2. Notice, however, that a faulty process could still deliver message m_2 without having delivered m_1 before. Messages sent with a strong total order multicast (sto-multicast) are called sto-messages.

3.3 Hybrid order

The sto-delivery requirements provide more powerful guarantees than the wto-delivery requirements, but are also more costly to implement (greater cost in terms of latency or number of protocol messages). Therefore, it is appealing to choose the implementation of the cheaper weak total order multicast and place the necessary enhancements to satisfy the semantic requirements of an application into the application.

We show that it is possible to offer both multicasts —a wto-multicast and a sto-multicast—, giving the user the option to choose the appropriate multicast for his problem. In order to allow the simultaneous use of both multicasts, we have to define the hybrid order property that controls how sto- and wto-messages are ordered in respect with each other. There are basically three possibilities. Let m_1 and m_2 be two messages issued with different delivery requirements (strong vs. weak):

- *unordered*: if processes p and q (whether correct or not) both deliver messages m_1 and m_2, then they can do this in an arbitrary order;
- *weak incorporated*: if correct processes p and q both deliver messages m_1 and m_2, then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2;
- *strong incorporated*: if processes p and q (whether correct or not) both deliver messages m_1 and m_2, then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2.

The first possibility is obviously not very helpful, since it does not give any guarantees. The second possibility guarantees that the delivery order is consistent within the correct processes, but allows incorrect processes to deliver wto-messages completely arbitrary in respect with sto-messages. We will concentrate on the last possibility, since it provides the strongest guarantees. A formal definition is as follows. Without loss of generality, let m_{s} be a sto-message and m_{w} be a wto-message, multicast into view v_i. The hybrid order is defined by the following property:

Hybrid-Order:

$$\forall p, q \in v_i \;:\; \Big(\text{TO\_DELV}_i^p(m_{\mathrm{s}}) \wedge \text{TO\_DELV}_i^p(m_{\mathrm{w}}) \wedge \text{TO\_DELV}_i^q(m_{\mathrm{s}}) \wedge \text{TO\_DELV}_i^q(m_{\mathrm{w}})\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_{\mathrm{s}}; m_{\mathrm{w}}) \equiv \text{TO\_DELV}_i^p(m_{\mathrm{s}}; m_{\mathrm{w}})\Big)$$

The general idea of this order is that each process delivers the same set of wto-messages (which can be empty) between any two sto-messages. Note that messages that were sent with the same multicast still respect the delivery requirements for the corresponding multicast.

3.4 The entire hierarchy of total order multicasts

Up to this point we have identified two different delivery requirements, which can be interpreted as a very simple hierarchy (see Figure 1a); the wto-delivery requirement defined by W-Order and W-Agreement, and the sto-delivery requirement defined by S-Order and S-Agreement. By combining these two order and agreement properties, we obtain two new specifications that we call uniform weak total order multicast and non-uniform strong total order multicast (see Figure 1b). Given two messages m_1,m_2 multicast into view v_i, the delivery requirements for these two multicasts are defined as follows:

Uniform weak total order multicast

W-Order:

$$\forall p, q \in \mathcal{C}(v_i) : \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge \text{TO\_DELV}_i^q(m_1) \wedge \text{TO\_DELV}_i^q(m_2)\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1; m_2) \equiv \text{TO\_DELV}_i^q(m_1; m_2)\Big)$$

S-Agreement:

$$\Big(\exists p \in v_i \;:\; \text{TO\_DELV}_i^p(m)\Big) \Rightarrow \Big(\forall q \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^q(m)\Big)$$

Non-uniform strong total order multicast

S-Order:

$$\forall p, q \in v_i \;:\; \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge \text{TO\_DELV}_i^q(m_1) \wedge \text{TO\_DELV}_i^q(m_2)\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1; m_2) \equiv \text{TO\_DELV}_i^q(m_1; m_2)\Big)$$

W-Agreement:

$$\Big(\exists p \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^p(m)\Big) \Rightarrow \Big(\forall q \in \mathcal{C}(v_i) \;:\; \text{TO\_DELV}_i^q(m)\Big)$$

These two intermediate multicasts provide different guarantees, and are semantically situated between the weak total order and strong total order multicasts. However, compared with the strong total order multicast, they offer only a marginally lower communication cost. Therefore we will restrict our discussion to the weak and strong total order multicasts.

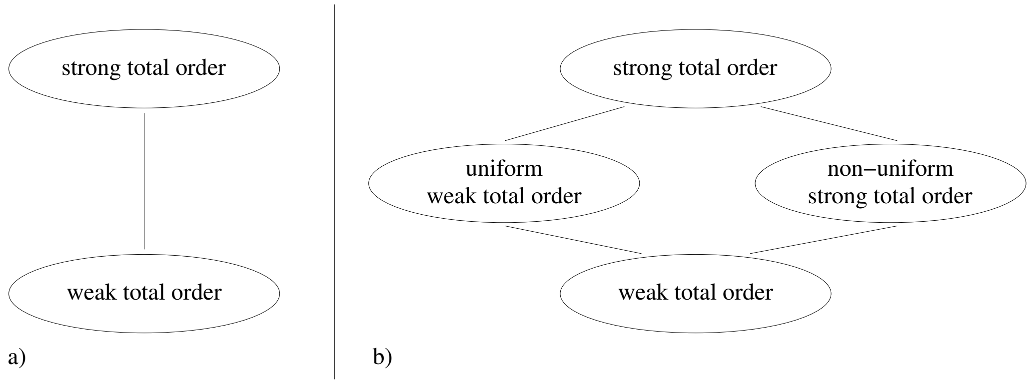

Figure 1: The hierarchy of multicasts

4 Algorithms for the total order multicasts

In this Section, we present algorithms that implement the wto- and sto-delivery requirements, together with the incorporation of the two according to the specification given in Section 3.3.

4.1 Weak total order multicast

The delivery requirements for weak total order multicast are typically ensured using a *sequencer-algorithm.* The idea of this asymmetric algorithm is to designate a special sequencer process that decides upon the sequence in which messages are delivered (usually in the order in which they are received by the sequencer process) and informs the other processes of this sequence through special deliver messages: for any message m, the sequencer defines $sn(m)$. In case failures occur during the execution of this algorithm, there might be several messages that could not be delivered by the processes, due to a missing sequence number for these messages. A termination protocol, which uses a deterministic algorithm to order this set of messages, will then deliver these messages. The algorithm is similar to the ISIS implementation of the ABCAST [BSS91].
Sequencer algorithm: Each process identifies the same special process s in the current view v_i (e.g. using the rank function, such that $\forall p \in v_i$: $\mathrm{rank}(s) < \mathrm{rank}(p) \vee s = p$) as the sequencer process. We assume that the sequencer process receives its proper messages and behaves otherwise like all the other processes. The algorithm requires 1 phase and $2n-2$ messages (n being the number of processes in the group), if no low-level broadcast medium is available.

Multicasting: each process may issue messages at any time (there are no restrictions);

Receiving: upon receiving a message, each process stores it for deferred delivery;

Scheduling: the sequencer assigns each message m it receives a (monotonously increasing) deliver sequence number $sn(m)$ and multicasts the deliver message $<sn(m), id(m)>$ into view v_i (we say that s *issues a schedule* for m);

Delivery: a process p can to-deliver a message m ($todelv^p(m)$), if it has received the message m as well as the deliver message $<sn(m), id(m)>$ for m, and the message m is the next to deliver; i.e. the last message delivered by p had the deliver sequence number $sn(m)-1$ (if this condition is fulfilled, we call m *deliverable*);

Termination: upon receiving a view change for view v_{i+1} from the vsc-layer ($vscvwchg^p(v_{i+1})$), let $\mathcal{ND}_i^p$ be the set of messages received by p during view v_i, but not delivered by p (because the deliver message $<sn(m), id(m)>$ was not received). Process p sorts $\mathcal{ND}_i^p$ according to a deterministic function Φ (e.g. using $id(m)$), to-delivers the messages in the obtained order to the application, and delivers the view change to the application ($tovwchg^p(v_{i+1})$).

There are several ways to further improve this algorithm, e.g. by collecting deliver messages and piggybacking them onto regular messages. We will not discuss these improvements in here, since they do not influence the general behaviour of the algorithm (see also [BSS91]).
Proposition: The sequencer algorithm satisfies the wto-delivery requirements.
Proof (sketch): The W-Agreement property is the same as the one satisfied by the underlying vsc-layer (view-synchronicity) and is therefore guaranteed by the vsc-layer. The termination protocol of the algorithm ensures that the received messages will finally be to-delivered.[6]

If no failures occur, it is easy to see that no process will violate the W-Order property, since each process delivers messages in the order given by the sequencer.

In case failures do occur, the liveness of the underlying vsc-layer guarantees that eventually a view change event will be generated and the view-synchronicity of the underlying vsc-layer guarantees that upon delivery of the view change from the vsc-layer, all correct processes have received the same set of messages. In particular, they have received the same set of deliver messages from the sequencer. Therefore, it is guaranteed that all correct processes will have delivered the same set of messages in the order given by the sequencer due to deliver messages from the sequencer. Thus, they will also have the same set of messages stored for deferred delivery; i.e. $\forall p, q \in \mathcal{C}(v_i) \;:\; \mathcal{ND}_i^p = \mathcal{ND}_i^q$. Every process can now independently sort these messages according to Φ and deliver them in the obtained sequence to the application, which will finally result in the delivery of the same sequence of messages at all correct processes and thus the W-Order property is satisfied. □

Note that the proposed algorithm does not satisfy the sto-delivery requirements. The sequencer process of view v_i might receive messages m_1 and m_2 immediately before being partitioned from the other processes. Since the sequencer does not know that it is partitioned, it will issue a schedule for m_1 and m_2, say m_1 before m_2, and then deliver the messages in this order. The other processes will eventually receive the viewchange v_{i+1} from which the sequencer process is excluded, and may (assume $\Phi(id(m_2)) < \Phi(id(m_1))$) deliver m_2 before m_1. Since the sequencer process if faulty, this case does not contradict the wto-delivery requirements, but is forbidden by the S-Order property of the sto-delivery requirements. A similar example for S-Agreement could be constructed.

4.2 Strong total order multicast

In order to enforce the delivery requirements for strong total order multicast, we propose the *round-order algorithm*. The general idea of this algorithm, is to put additional control information into the messages, which is used to define the total order $\prec$ (see below) on the messages.

[6] A process might still fail before actually to-delivering some messages. This is, however, indistinguishable from the case, where the process fails immediately after to-delivering the messages, and can therefore be neglected. All the required information is, however, guaranteed to be available to the process.

This total order is completely independent from events local to some process (the order is defined solely on the control information in the messages). The round-order algorithm introduces the exchange of additional messages (acknowledgments), which enable the process that obtains these acknowledgments to respect the order $\prec$ on the messages. Processes (whether correct or faulty) deliver messages according to the order $\prec$, which results in a total order delivery of all messages. Each process keeps a counter for the current round that is monotonously increasing and all processes start with the same round value. The control information consists of the identification of the sender of the message $sender(m)$ and the current round of the sender $round(m)$ (we say messages are issued *into* rounds). With these definitions the total order $\prec$ on the messages is defined as follows:

$$
\begin{aligned}
m_1 \prec m_2 \quad \Leftrightarrow \quad & round(m_1) < round(m_2) \;\vee \\
& (round(m_1) = round(m_2) \wedge \mathrm{rank}(sender(m_1)) < \mathrm{rank}(sender(m_2)))
\end{aligned}
$$

The idea of this algorithm is similar to the one described in the ABP-algorithm [Anc93]. Both algorithms use the notion of rounds to establish the total order on the messages and require an information from each participating process to decide upon the delivery of a given message. However, the introduction of the vsc-layer as the underlying communication paradigm allows for a more modular, and thus simpler design. The protocol for sto-multicast is basically symmetric, as is the ABP-algorithm given in [Anc93]. We describe here an asymmetric version of the protocol for sto-multicast[7] that will allow later on to implement the hybrid order, which incorporates both weak and strong total order.

Round-order algorithm: Each process identifies the same special process c in the current view v_i (e.g. using the rank function as before) as the coordinator.[8] We assume that the coordinator receives its proper messages and behaves otherwise like all the other processes. The algorithm requires $1^1/_2$ phases and $3n-4$ messages (n being the number of processes in the group), if no low-level broadcast medium is available.

Multicasting:

- upon multicasting a message, each process puts the control information (sender and round) into the message;
- a process can issue at most one message into any given round;
- a process can never issue a message into a round for which it has already issued an acknowledgment;

Receiving: upon receiving a message m, a process p stores it for deferred delivery and issues an acknowledgment for m to the coordinator, indicating whether p has previously issued a message into round $round(m)$;

Scheduling: the coordinator c awaits acknowledgments from all processes in the current view for a given message m, which allow c to find the position of m in the total order $\prec$ (c knows from each process whether it has sent a message into round $round(m)$ and the order of the messages in a given round is defined by the rank of their sender). The coordinator then issues a schedule (see Scheduling in Section 4.1) for m that respects the order $\prec$ (Notice that the coordinator does not define an order, but only observes a predefined order);

[7] In a symmetric version, the processes would multicast the information required to determine the total order on the messages (acknowledgments) to all the processes in the current view, so that each process could independently determine the correct total order. This approach has the advantage that fewer phases are needed to deliver a message; the disadvantage is that if multicast messages have to be sent as one low level message to each recipient (as is the case in a wide area network), more low level messages are needed to deliver a single high level message.

[8] In this context, we prefer to use the name coordinator instead of sequencer, because this reflects the passive, observing nature of the coordinator versus the active sequencer.

Delivery: a process p can to-deliver a message m ($todelv^p(m)$), if m is deliverable (see Delivery in Section 4.1);

Termination: upon receiving a view change for view v_{i+1} from the vsc-layer ($vscvwchg^p(v_{i+1})$), let $\mathcal{ND}_i^p$ be the set of messages received by p during view v_i, but not delivered by p. Process p sorts $\mathcal{ND}_i^p$ according to $\prec$, to-delivers the messages in the obtained order to the application, and delivers the view change to the application ($tovwchg^p(v_{i+1})$).

Again, this algorithm could be further improved by piggybacking acknowledgments and deliver messages onto regular messages, or by reducing the number of acknowledgments necessary to decide on the delivery of a given message (a majority of acknowledgments might be sufficient); we will not discuss these issues, since they are beyond the scope of this paper.

Lemma: The round-order algorithm guarantees that given two messages m_1,m_2 multicast into view v_i, the following property holds:

$$\forall p \in v_i \ : \ \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge m_1 \prec m_2\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1; m_2)\Big)$$

We say that the processes deliver messages according to $\prec$.

Proof (sketch): The coordinator of the algorithm first orders messages from different rounds according to ascending round numbers (contained in $round(m)$). Since all processes start with the same initial round value, there is a first round from which c can start to schedule messages (i.e. assign a deliver sequence number and multicast the deliver message $<sn(m), id(m)>$). The coordinator c decides to schedule a message m only after it has scheduled all messages from previous rounds and after it has received an acknowledgment for m from every other process in the current view. Hence, c knows about all the messages that were issued into the same round as m and it is assured that no other process will ever send another message into this round. The coordinator will schedule all these messages based on the rank of their sender and thus according to $\prec$.

If no failures occur, every process will deliver messages only due to a deliver message from the coordinator and thus according to $\prec$. Since communication is reliable, c will eventually schedule all the messages and every process will eventually receive all the messages and all the deliver messages.

In case failures do occur, the liveness of the underlying vsc-layer guarantees that eventually a view change will be generated and the view-synchronicity of the underlying vsc-layer guarantees that upon delivery of the view change from the vsc-layer, all correct processes have received the same set of messages. In particular, they have received the same set of deliver messages from the coordinator (this holds also for the coordinator, if c is correct). Therefore, it is guaranteed that all correct processes will have delivered the same set of messages due to deliver messages from the coordinator in the order given by the coordinator. Thus, they will also have the same set of messages stored for deferred delivery (i.e. $\forall p, q \in \mathcal{C}(v_i) \ : \ \mathcal{ND}_i^p = \mathcal{ND}_i^q$), and due to the construction of the delivery rule and the fact that the coordinator schedules messages according to $\prec$, each of these messages is greater (according to $\prec$) than the last message delivered due to a deliver message from the coordinator. Every process can now independently sort these messages according to the total order $\prec$ and deliver them in the obtained sequence to the application. Any faulty process has delivered messages only due to the deliver messages from the coordinator and therefore according to $\prec$. $\square$

Proposition: The round-order algorithm satisfies the sto-delivery requirements.

Proof (sketch): The S-Agreement property for strong total order is also called *uniformity*, as defined by Sandoz and Schiper in [SS93]. The round-order algorithm employs the same technique as the one for uniform message delivery, i.e. to stabilize a message by waiting for an acknowledgment from each process, before actually delivering a message. Thus, the proof is very similar to the one in the cited paper — we omit it here for conciseness.

In order to ensure the S-Order property, it suffices to show that $\prec$ is a total order; i.e. given two messages m_1, m_2 either $m_1 \prec m_2$ or $m_2 \prec m_1$ holds. Messages from different rounds are ordered due to ascending round numbers and messages from the same round are ordered according to the rank of their sender. Since each process is allowed to issue at most one message into any given round, all messages are totally ordered. This together with the Lemma guarantees S-Order. □

4.3 Hybrid order

The goal of this incorporation is to offer the possibility to issue concurrently wto- and sto-multicasts. We define an extension $\prec\cdot$ to the order $\prec$ that comprises the messages sent with a wto-multicast; this is achieved by allocating them to a certain round like the sto-messages. $\prec\cdot$ is a partial order, in which sto-messages are totally ordered with all other messages, while wto-messages are only ordered with sto-messages and with wto-messages from a different round (wto-messages from the same round are not ordered). Messages not ordered by $\prec\cdot$ are ordered by the coordinator of the algorithm for sto-multicast, which functions as sequencer. The algorithm requires the same additional messages as the round-order algorithm for sto-multicast and similar control information within the messages.

We assume again that each process keeps a counter for the current round that is monotonously increasing and that all processes start with the same round value. The control information consists of the identification of the sender of the message $sender(m)$, the current round of the sender $round(m)$, and the protocol with which the message was issued (possible values are *sto* and *wto*). With these definitions the partial order $\prec\cdot$ on the messages is now defined as follows:

$$\begin{aligned} m_1 \prec\cdot\, m_2 \quad \Leftrightarrow \quad & round(m_1) < round(m_2) \quad \vee \\ & (round(m_1) = round(m_2) \ \wedge \ proto(m_1) = \text{wto} \ \wedge \ proto(m_2) = \text{sto}) \quad \vee \\ & (round(m_1) = round(m_2) \ \wedge \ proto(m_1) = \text{sto} = proto(m_2) \ \wedge \\ & \qquad \text{rank}(sender(m_1)) < \text{rank}(sender(m_2))) \end{aligned}$$

Hybrid-order algorithm: Each process identifies the same special process c in the current view v_i (e.g. using the rank function as before) as the coordinator. We assume that the coordinator receives its proper messages and behaves otherwise like all the other processes. The algorithm has the same communication cost as the corresponding previous algorithms.[9]

Multicasting:

- upon multicasting a message, each process puts the control information (sender, round, and protocol) into the message;
- upon multicasting a sto-message m_s, each process further adds a number to the control information that indicates how many wto-messages this process issued during round $round(m_s)$;
- a process can issue an arbitrary number of wto-messages, but at most one sto-message into any given round;
- a process can never issue a message (with neither protocol) into a round for which it has already issued an acknowledgment;

Receiving:

[9] Due to interdependencies between messages, the communication delay may, however, be slightly higher.

- upon receiving a wto-message m_w, a process p stores it for deferred delivery;
- upon receiving a sto-message m_s, a process p stores it for deferred delivery and issues an acknowledgment for m_s to the coordinator, indicating how many wto-messages p issued and whether p has already issued a sto-message into round $round(m_s)$;

Scheduling:

- the coordinator keeps a counter for the deliver round of which it currently delivers messages and upon receiving a message m that does not belong to the deliver round, it stores m for deferred treatment[10] (a message that belongs to the deliver round is called *in-round* message);
- upon receiving an in-round wto-message m_w, the coordinator issues a schedule (see Scheduling in Section 4.1) for m_w;
- upon receiving an in-round sto-message m_s, the coordinator awaits acknowledgments from all processes in the current view, which allow c to find the position of m_s in the partial order $\prec\cdot$ (c knows from each process p how many wto-messages p issued and whether p has already issued a sto-message during round $round(m_s)$). The coordinator then issues a schedule for m_s that respects the order $\prec\cdot$;
- when the last sto-message of a given round has been scheduled, the coordinator switches to the next round and treats the deferred messages (if any);

Delivery: a process p can to-deliver a message m ($todelv^p(m)$), if m is deliverable (see Delivery in Section 4.1);

Termination: upon receiving a view change for view v_{i+1} from the vsc-layer ($vscvwchg^p(v_{i+1})$), let $\mathcal{ND}_i^p$ be the set of messages received by p during view v_i, but not delivered by p. Process p sorts $\mathcal{ND}_i^p$ according to $\prec\cdot$ by choosing an arbitrary (yet deterministic) order for the messages that are not ordered by $\prec\cdot$, to-delivers the messages in the obtained order to the application, and delivers the view change to the application ($tovwchg^p(v_{i+1})$).

Lemma: The hybrid-order algorithm guarantees that given two messages m_1,m_2 multicast into view v_i, the following property holds:

$$\forall p \in v_i \ : \ \Big(\text{TO\_DELV}_i^p(m_1) \wedge \text{TO\_DELV}_i^p(m_2) \wedge m_1 \prec\cdot m_2\Big) \Rightarrow \Big(\text{TO\_DELV}_i^p(m_1; m_2)\Big)$$

We say that the processes deliver messages according to $\prec\cdot$.

Proof (sketch): The coordinator of the algorithm first orders messages from different rounds according to ascending round numbers (contained in $round(m)$). Since all processes start with the same initial round value, there is a first round from which c can start to schedule messages (i.e. assign a deliver sequence number and multicast the deliver message $< sn(m), id(m) >$). c decides to schedule a sto-message m_s only after it has scheduled all messages from previous rounds and after it has received an acknowledgment for m_s from every other process in the current view. Hence, c knows from each process p whether p has sent a sto-message and how many wto-messages p sent into round $round(m_s)$. Furthermore c is assured that no process will send any further message into round $round(m_s)$. The coordinator will now wait until it has received and scheduled all the wto-messages sent into round $round(m_s)$ and then schedule the sto-messages based on the rank of their sender. Thus, c will schedule messages according to $\prec\cdot$, using an arbitrary order for the wto-messages of the same round that are not ordered by $\prec\cdot$.

[10] A message can not belong to an earlier round — the algorithm prevents this from happening.

If no failures occur, every process will deliver messages only due to a deliver message from the coordinator and thus according to $\prec\cdot$. Since communication is reliable, c will eventually schedule all the messages and every process will eventually receive all the messages and all the deliver messages.

In case failures do occur, the liveness of the underlying vsc-layer guarantees that eventually a view change will be generated and the view-synchronicity of the underlying vsc-layer guarantees that upon delivery of the view change from the vsc-layer, all correct processes have received the same set of messages. In particular, they have received the same set of deliver messages from the coordinator (this holds also for the coordinator, if c is correct). Therefore, it is guaranteed that all correct processes will have delivered the same set of messages due to deliver messages from the coordinator in the order given by the coordinator. Thus, they will also have the same set of messages stored for deferred delivery (i.e. $\forall p, q \in \mathcal{C}(v_i) \ : \ \mathcal{ND}_i^p = \mathcal{ND}_i^q$), and due to the construction of the delivery rule and the fact that the coordinator schedules messages according to $\prec\cdot$, each of these messages is either not ordered with, or greater (according to $\prec\cdot$) than the last message delivered due to a deliver message from the coordinator. Every process can now independently sort these messages according to the total order $\prec\cdot$, using an arbitrary order for the messages that are not ordered by $\prec\cdot$, and deliver them in the obtained sequence to the application. Any faulty process has delivered messages only due to the deliver messages from the coordinator and therefore according to $\prec\cdot$. $\square$

Proposition: The hybrid-order algorithm satisfies the wto- and sto-delivery requirements as well as the hybrid-order property.

Proof (sketch): The W-Agreement property for weak total order is still guaranteed by the underlying vsc-layer, and the S-Agreement property for strong total order is ensured with the same technique as in the algorithm for strong total order multicast. It remains to show that the algorithm satisfies the three order properties: W-Order, S-Order, and Hybrid-Order.

The order defined by $\prec\cdot$ constitutes a total order on all sto-messages, and $\prec\cdot$ orders each wto-message in respect with the sto-messages; i.e. given a sto-message m_s and a message m (a sto- or a wto-message), either $m_s \prec\cdot m$ or $m \prec\cdot m_s$ holds. This is due to the fact that each process is allowed to issue at most one message into any given round and the special construction of $\prec\cdot$. This together with the Lemma guarantees that the S-Order and Hybrid-Order properties are satisfied.

If no failures occur, each process will deliver messages in the order given by the coordinator, thus, all correct processes deliver messages in the same sequence. In case failures do occur, we have shown (see Lemma) that each correct process has delivered the same set of messages due to deliver messages from the coordinator in the order given by the coordinator. Furthermore, all correct processes have the same set of messages stored for deferred delivery, which they will sort according to $\prec\cdot$, using a deterministic order for the wto-messages of the same round that are not ordered by $\prec\cdot$. This will result in the delivery of the same sequence of messages at all correct processes, which guarantees the W-Order property. $\square$

5 Conclusion

In this paper we studied communication paradigms that provide total order delivery of multicast messages. We identified a hierarchy of total order multicasts, defined with different delivery requirements. Within this hierarchy, the topmost sto-delivery requirements offer the strongest guarantees while the lowest wto-delivery requirements offer the weakest guarantees for the total order message delivery. However, there is a tradeoff in terms of communication overhead, since an algorithm that satisfies the sto-delivery requirements incurs a considerably higher communication overhead than one that satisfies the wto-delivery requirements. The intermediate multicasts within the hierarchy have no significant advantage over sto-multicast (in terms of communication overhead); therefore we restricted our discussion to the extremes of the hierarchy.

Depending on the application semantics, both multicasts can be useful, so that an incorporation of these multicasts into a consistent design is desirable. This incorporation can be used to optimize the communication cost of an application by selecting the cheapest communication paradigm for the required task. We identified several possibilities for this incorporation and chose the strongest one that we named hybrid order for further investigation. Finally we presented possible implementations of the weak total order multicast, strong total order multicast, and their incorporation based on the hybrid order.

References

[AF92] H. Attiya and R. Friedman. A correctness condition for high-performance multiprocessors. In *24th ACM Annual Symposium on the Theory of Computing*, pages 679–690, 1992.

[Anc93] E. Anceaume. *Algorithmique de Fiabilisation de Systèmes Répartis*. PhD thesis, Université de Paris-Sud, Centre d'Orsay, Jan 1993.

[BJ87] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, February 1987.

[BSS91] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, August 1991.

[CASD85] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *IEEE Symposium on Fault-Tolerant Computing*, pages 200–206, June 1985.

[CM84] J. M. Chang and N. Maxemchuck. Realiable Broadcast Protocols. *ACM Trans. Comput. Syst.*, 2(3):251–273, August 1984.

[CT93] T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Technical Report 93-1374, Department of Computer Science, Cornell University, August 1993. A preliminary version appeared in the *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 325–340. ACM Press, August 1991.

[DKM93] D. Dolev, S. Kramer, and D. Malki. Early Delivery Totally Ordered Broadcast in Asynchronous Environments. In *23rd Annual International Symposium on Fault-Tolerant Computing*, pages 544–553, June 1993.

[FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32:374–382, April 1985.

[HT93] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In Sape J. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–146. Addison-Wesley, 1993.

[KDK+89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1):25–40, feb 1989.

[KTHB89] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[Lam78] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, July 78.

[LG90] S. W. Luan and V. D. Gligor. A Fault-Tolerant Protocol for Atomic Broadcast. *IEEE Trans. Parallel & Distributed Syst.*, 1(3):271–285, July 90.

[MS94] C. Malloth and A. Schiper. View Synchronous Communication in the Internet. Technical Report 94/84, Ecole Polytechnique Fédérale de Lausanne, Lausanne (Switzerland), October 1994.

[Sch90] F.B. Schneider. Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial. *Computing Surveys*, 22(4):299–319, December 1990.

[SS93] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *IEEE 13th Intl. Conf. Distributed Computing Systems*, pages 561–568, May 93.