Uniform Reliable Multicast in a Virtually Synchronous Environment

André Schiper* Alain Sandoz

Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)

Abstract

This paper presents the definition and solution to the uniform reliable multicast problem in the virtually synchronous *environment defined by the Isis system. A uniform reliable multicast of a message m has the property that if m has been received by any destination process (faulty or not), then m is received by all processes that reach a decision. Uniform reliable multicast provides a solution to the distributed commit problem. The paper defines two multicast primitives in the virtually synchronous model: reliable multicast (called* view-atomic*) and uniform reliable multicast (called* uniform view-atomic*). The view-atomic multicast is used to implement the uniform view-atomic primitive. As view-atomicity is based on the concept of process group membership, the paper establishes a connection between the process group membership and the distributed commit problems.*

1 Introduction

A distributed application is composed of processes communicating through message passing. Point to point is the most simple communication pattern. Because of communication errors (messages can be corrupted or lost), it is however basically unreliable. To circumvent this undesirable behavior, communication protocols can be defined to implement a more adequate channel semantics: the reliable channel abstraction, which can be enriched with FIFO delivery order or the more restrictive causal delivery order [19, 18]. Point to point communication is however perceived as insufficient for many distributed applications. Group communication or multicasts are often desirable [7, 11] and have been considered in many systems [2, 20, 16]. Group communication raises two issues: the reliability issue and the ordering issue. With respect only to the ordering issue, total ordering or causal ordering of multicasts can be considered. In the context of reliability, atomicity (in the sense of all or nothing, without any order semantics) or k-reliability (i.e. tolerance to k failures) are usually considered. Uniform reliable multicast, which defines a stronger semantics than reliable multicast, has also been considered [14, 10, 8, 1, 15] for solving the distributed commit problem [3]. Informally, a uniform reliable multicast of message m has the property that if m has been received by any process (faulty or not), then m is received by all processes that reach a decision. The problem has been considered however mostly in the synchronous model, which assumes bounded communication delays [10, 8]. The case of asynchronous communication has been outlined in [10], and considered from a theoretical point of view in [1, 15]. Considering the synchronous model has its justification in the impossibility result concerning distributed consensus in the asynchronous model [9].

This paper considers the problem of uniform reliable multicast in an asynchronous model with crash-failure process semantics. The precise model is the *virtually synchronous model* introduced by the Isis system [4, 5], a model based on the concept of process group membership [17]. The paper establishes the connections between the process group membership problem (GMP) in the asynchronous model, and uniform reliable multicast, i.e. between GMP and distributed commit in an asynchronous system. In particular we show that it is always possible to implement a distributed commit protocol in an asynchronous environ-

*Current address: Cornell University, Department of Computer Science, Ithaca, NY 14853-7501.

[0]Research supported by the "Fonds national suisse" under contract number 21-32210.91, as part of European ESPRIT Basic Research Project Number 6360 (BROADCAST).

ment that blocks only if the GMP protocol blocks.

The rest of the paper is organized in the following way. Section 2 presents the virtually synchronous model and defines reliable multicast (called *view-atomic*) and uniform reliable multicast (called *uniform view-atomic*) in this model. In section 3 an implementation of view-atomic multicast is given, based on the detection of a global stable property. Section 4 uses the view-atomic multicast primitive to implement uniform view-atomic multicast. Section 5 presents a solution to the distributed commit problem in the virtually synchronous model using the uniform multicast primitive and briefly considers the case of network partitions.

2 Uniform view-atomic multicast

2.1 System model and failure semantics

We consider a set of processes $p_i \in P$, connected through a set of communication channels $c_{ij} \in C$ (c_{ij} connects p_i to p_j). Communication is asynchronous, i.e. there is no bound on communication delays. Processes have crash-failure semantics: a failed process stops sending messages. Byzantine errors [12] are not considered. After a crash, a process might recover (if part or all of its state is saved on permanent storage), or might never recover. Communication channels c_{ij} are assumed to be reliable (a message m sent by process p_i to process p_j will eventually reach p_j if neither p_i nor p_j fail) and to deliver messages in FIFO order.

In the asynchronous model, it is impossible to distinguish a crashed process from a slow process or a process connected through a slow channel. This observation has led to the [9] impossibility result on distributed consensus, and to the definition of the synchronous model with bounded delays. The synchronous model has some attractive features, but has the drawback to lead to poor performances because worst case assumptions must be made on communication delays. The model we consider in the paper has been called *virtually synchronous* [4, 5]. It can be defined as the asynchronous model completed with an *imperfect failure detector* FD satisfying the specification and failure semantics described below. Consider processes structured into groups $g \in G$ used in the context of multicasts: each multicast m is addressed to a group g, i.e. to every process $p_i \in g$. For a group g of processes, the failure detector FD is assumed to construct a sequence of views $v_0(g), v_1(g), \ldots, v_i(g), \ldots$ ($v_i(g) \subseteq P$) corresponding to the successive composition of the group g as perceived by FD, and to deliver each view $v_i(g)$ to the members of the view. The sequence of views satisfies the following conditions:

- if process $p \in g$ has crashed, then FD will eventually detect it and define a new view from which p is excluded (if p recovers after the crash, it comes back with a different process id, and is thus considered as a new process);
- if a process p joins the group g, FD will eventually detect it, and define a new view including p;
- if a view $v_i(g)$ is defined and $p \in v_i(g)$, then either p eventually receives view $v_i(g)$, or $\exists k > 0$, such that $p \notin v_{i+k}(g)$;
- $\forall p, q \in g$, if p and q receive views $v_i(g)$ and $v_j(g)$ ($i \neq j$), then p and q both receive $v_i(g)$ and $v_j(g)$ in the same order.

Note that we don't assume that a process p considered by FD to have crashed, has really crashed: incorrect failure detection is tolerated by the model, and is inherent to any implementation of FD. Building a failure detector is done by a so called GMP (Group Membership Problem) protocol. In [17] it has been shown how such a failure detector can be implemented, considering processes with fail-stop failure semantics and network partitions.

2.2 View-atomic and uniform view-atomic multicast

Definitions D1 and D2 are the traditional (informal) definitions of reliable and uniform reliable multicast.

Definition D1. Reliable multicast. A multicast m to a group g is reliable iff m is received by all non-faulty processes of group g, or by none.

Definition D2. Uniform reliable multicast. A multicast m to a group g is uniform reliable iff the following condition holds: if a process in g (correct or faulty) has received m, then all non-faulty processes of group g eventually receive m.

Definitions D1 and D2 are unsatisfactory, because they don't define an instant from which point the definitions hold: in the asynchronous model, progress in the computation is related to the reception of messages. In the virtually synchronous model however (i.e. with a failure detector), progress in the computation is related either to the reception of messages, or to the reception of new views informing of process crashes. Therefore the virtually synchronous model enables one to consider stronger conditions which can be achieved using specific multicast primitives. These are the *view-atomic* (see D3) and *uniform view-atomic* (see D4) multicasts. These definitions consider for each process $p \in g$ the notion of *current view*. Initially $v_0(g)$ is the current view of every process of group g. A process in current view $v_i(g)$, receiving view $v_{i+1}(g)$ from the failure detector, will only consider $v_{i+1}(g)$ as the new current view once view $v_{i+1}(g)$ *has been installed* [1]. Definition D3 defines thus a condition (on multicast deliveries) for installing a new view $v_{i+1}(g)$. Installation of a new view can be considered as *reaching a decision*.

Definition D3. View-atomic multicast. Consider a group g, a view $v_i(g)$, a message m multicast to g. The multicast m is *view-atomic* in view $v_i(g)$ iff:
if $\exists p \in v_i(g)$ *which has delivered* m *in view* $v_i(g)$[2] *and has installed view* $v_{i+1}(g)$*, then all processes* $q \in v_i(g)$ *which have installed* $v_{i+1}(g)$ *have delivered* m *before installing* $v_{i+1}(g)$.

Remark. Definition D3 is not the multicast semantics of the Isis system [5], as shown by the example sketched below. The protocol in [5] does not exclude the following case. Consider *(i)* $v_i(g) = \{p_1, p_2, p_3, p_4\}$, *(ii)* process p_4 initiating a view-atomic multicast m in $v_i(g)$, and *(iii)* p_4 crashing after the emission of m to p_3:

- view $v_{i+1}(g) = \{p_1, p_2, p_3\}$ is defined and sent to p_1, p_2, p_3;
- p_3 receives and delivers m, receives $v_{i+1}(g)$, installs $v_{i+1}(g)$, and crashes immediately after having installed $v_{i+1}(g)$;
- view $v_{i+2}(g) = \{p_1, p_2\}$ is defined and sent to p_1, p_2;
- processes p_1 and p_2 both receive and install $v_{i+1}(g)$ and $v_{i+2}(g)$ without delivering message m (this can happen if errors occur in the retransmission of m from p_3 to p_1 and p_2).

Definition D3 is clearly not satisfied: process p_3 delivers m and installs view $v_{i+1}(g)$, while p_1 and p_2 both install view $v_{i+1}(g)$ without having delivered m. □

In the virtually synchronous model, we define uniformity by definition D4.

Definition D4. Uniform view-atomic multicast. Consider a group g, a view $v_i(g)$, a message m multicast to g. Multicast m is *uniform view-atomic* in $v_i(g)$ iff:
if $\exists p \in v_i(g)$ *which has delivered* m *in view* $v_i(g)$*, then all processes* $q \in v_i(g)$ *which have installed* $v_{i+1}(g)$ *have delivered* m *before installing* $v_{i+1}(g)$.

Definition D4 is stronger then D3: definition D3 imposes conditions on delivery of a multicast m only on the processes which install $v_{i+1}(g)$, whereas definition D4 considers delivery of message m by any process.

3 View-atomic multicast protocol

This section describes an implementation of definition D3. This is done in three steps. We start by describing the local behavior of each process (which is very close to [5]). Based on this local protocol, we restate definition D3 in terms of a property noted VA (for *View-Atomicity*) which can be evaluated in the system. Property VA is stable, i.e. once true, remains true forever [6]. The final step consists in giving the protocol for detecting the stable property VA. This suggests the following implementation of view-atomic multicasts (definition D3). Consider a process p with current view $v_i(g)$ receiving a new view $v_{i+1}(g)$ from the failure detector. As soon as the stable property VA is true on the view $v_i(g)$, process p can install the new view $v_{i+1}(g)$.

3.1 Process behavior

Consider a group g of processes completely connected through reliable asynchronous channels, and a view v_i (from this point the reference to g will be omitted).

Notation. The set of processes in the intersection of all views $v_i, \ldots, v_j$ $(i \leq j)$ is noted $V_{i,j}$. □

Consider a process $p \in v_i$, where v_i is the current view on p. Behavior of p differs depending on whether p has already received v_{i+1} or not.

[1] Views are received and installed in the same order on all processes. For j, k such that $j < k$, reception by p of view $v_k(g)$ can however occur before installation of view $v_j(g)$.

[2] Message delivery is discussed in §3.1.

Before reception of view v_{i+1}, p multicasts any message m by appending the view number i of v_i to m, and sending m to all $q \in v_i$. The view attribute of m, noted $vn(m)$, is used when a message m is received by a process q: $vn(m)$ is matched against q's current view number $curr(q)$. If $vn(m) < curr(q)$, m is an "old" message and is discarded. If $vn(m) > curr(q)$, m comes from a "future" view (e.g. resulting from a process joining the group) and is buffered until a new view with number equal to $vn(m)$ is installed on q. Finally if $vn(m) = curr(q)$, the message is delivered to q.

As soon as view v_{i+1} is received (but not yet installed), p stops issuing multicasts to g until finally view v_{i+1} is installed (see 3.3). Note that before *installing* view v_{i+1}, p could receive other views $v_{i+2}, \ldots, v_{i+k}$. After receiving v_{i+1} and until installing v_{i+1}, p behaves in the following way (we consider here only messages m with $vn(m) = i$):

1. p starts by multicasting to v_i[3] all messages m received in v_i, that might not have been received by all $q \in v_i$[4] (duplicate messages are discarded by the destination process). In order to distinguish process p from the initial sender of m, p is noted $relay(m)$, and the initial sender is noted $sender(m)$;

2. if p has received view v_{i+k} $(k \geq 1)$, then (i) p rejects all messages m received directly from a sender such that $sender(m) \notin V_{i,i+k}$ or, (ii) if m has been relayed, such that $relay(m) \notin V_{i,i+k}$;

3. whenever p receives for the first time a message m that has not been rejected under condition 2, p delivers it, and then relays m to all $q \in v_i$. Process p is also noted $relay(m)$.

The relay protocol of points 1 and 3 is the classical protocol for *reliable multicast* (definition D1) [13]. According to point 2, a process p discards any message m received or relayed from a process $q \in v_i$ that p considers to have failed. Rejection of these messages is essential in order to ensure that the global predicate VA introduced in paragraph 3.2 is stable.

[3] In fact the multicast needs only be addressed to $V_{i,i+k}$, where v_{i+k} is the most recent view received by p_i.

[4] This property of a message can be *evaluated* locally on p using information piggybacked on messages m' received from the group g (every message sent to the group by $q \in v_i$ piggybacks some encoded description of the list of messages received by q).

3.2 Restatement of definition D3 in terms of detection of a stable property

In order to define a global stable predicate VA equivalent to definition D3, we associate to each process $p \in v_i$ a vector $msg_p(v_i)$ of size $|v_i|$ defined such that:

- $msg_p(v_i)[p]$ is the number of messages multicast by p in v_i (i.e. number of messages m such that $sender(m) = p$);
- for $p \neq q$ $(q \in v_i)$, $msg_p(v_i)[q]$ is the number of messages m multicast in v_i and delivered to p, such that $sender(m) = q$.

The global property *VA* is defined by definition D5.

Definition D5. Property $VA(v_i)$. Consider a group g with currently installed view v_i (i.e. the current view of every $p \in v_i$ is v_i). Property $VA(v_i)$ is true iff there exists a view v_{i+k} $(k \geq 1)$ such that the two following conditions are satisfied:

1. $\forall p \in V_{i,i+k}$, p has received view v_{i+k};
2. $\forall p, q \in V_{i,i+k}$, $msg_p(v_i) = msg_q(v_i)$. □

$VA(v_i)$ stands for *view atomicity is realized in* v_i, and is thus a predicate defined on v_i. As shown by proposition 1 below, condition 1 is necessary to ensure stability of VA, whereas condition 2 is related to definition D3. As the failure detector continues to deliver views, property $VA(v_i)$ can be true for several views v_{i+k}, $k \geq 1$.

Proposition 1 (Stability). If every process $p \in v_i$ follows the protocol of §3.1, the global property *$VA(v_i)$* defined on view v_i is a stable property.

Proof (sketch). Condition 1 of D5 trivially remains true forever. If condition 2 is true, no message is in transit in $V_{i,i+k}$ because of FIFO delivery order. Consider $p \in V_{i,i+k}$ and $msg_p(v_i)[q]$: (1) if $p = q$, the protocol of §3.1 and condition 1 of D5 ensure that $msg_p(v_i)[p]$ remains unchanged; (2) if $p \neq q$, $q \in V_{i,i+k}$, as no message is in transit in $V_{i,i+k}$, then $msg_p(v_i)[q]$ remains unchanged; (3) if $p \neq q$, $q \notin V_{i,i+k}$, because condition 1 of D5 is true, p rejects messages received directly from q, and no message from q is relayed by processes in $V_{i,i+k}$: $msg_p(v_i)[q]$ remains unchanged. Thus condition 2 of definition D5 remains true forever. □

Proposition 2 (Safety). If every process $p \in V_{i,i+k}$ installs view v_{i+1} only when $VA(v_i)$ is true for some v_{i+k} $(k \geq 1)$, then definition D3 is satisfied.

Proof. If $VA(v_i)$ is true (for some v_{i+k}) then D3 is trivially satisfied (compare property $VA(v_i)$ and definition D3). Because property $VA(v_i)$ is stable, D3 can no more be invalidated. □

Proposition 3 (Liveness). Consider v_i the current view in g and occurrence of a view change. If every process $p \in v_i$ follows the protocol of §3.1, then $\exists k \geq 1$ such that property $VA(v_i)$ will eventually become true for view v_{i+k}.

Proof (sketch). If $\exists k$ such that $V_{i,i+k} = \emptyset$ (every process in v_i has failed), then $VA(v_i)$ is trivially true for v_{i+k}. Else, note v_{i+k} $(k \geq 1)$ a view such that $\forall p \in V_{i,i+k}$, p has received view v_{i+k}: condition 1 of D5 is met. Assume then that condition 2 of D5 is not true, which is possible only because $\exists p, q \in V_{i,i+k}$ and $r \in v_i$ such that $msg_p(v_i)[r] > msg_q(v_i)[r]$. Consider any message m sent by r and received by p but not by q. Two cases: (1) if neither p nor q fail, p will relay message m to q (see §3.1). Because channels are reliable, m will eventually be received by q and condition $msg_p(v_i)[r] = msg_q(v_i)[r]$ becomes true (i.e. $VA(v_i)$ true for view v_{i+k}); (2) if p or q fail, a new view $v_{i+k'}$ $(k' > k)$ will be defined, from which p or q are excluded. Condition 2 of D5 is no more violated for view $v_{i+k'}$. Note $v_{i+k''}$ $(k'' \geq k')$ a subsequent view such that $\forall s \in V_{i,i+k''}$, s receives view $v_{i+k''}$. By finite induction, this completes the proof. □

3.3 Protocol for detecting the global property VA

Detection of property $VA(v_i)$ consists of superimposing on the basic computation in group g another computation responsible for detecting if the stable property has become true. We describe in this paragraph the protocol of this superimposed computation, and consider (in order to simplify the presentation) that it is executed for a process p by p itself.

The protocol is based on a *coordinator-participant* scheme. The coordinator for p, with current view v_i and most recently received view v_{i+k}, is a function of $V_{i,i+k}$. For example, if $pid(q)$ is the process identifier of q, the coordinator for p might be the process c such that $c = \min_{q \in V_{i,i+k}}(pid(q))$. As soon as p receives a new view indicating the crash of c, the coordinator changes, ensuring fault-tolerance of the protocol. Moreover a coordinator always exists unless $V_{i,i+k}$ is empty: this occurs only if the failure detector suspects all processes to have crashed. A process that is not the coordinator for the protocol is called a *participant.*

The information sent from participant p to the coordinator is the vector $msg_p(v_i)$ introduced in 3.2, slightly modified to encode reception by p of views indicating failure of some process. For $p \neq q$ $(q \in v_i)$:

- $|msg_p(v_i)[q]|$ is the number of messages m received by p such that $sender(m) = q$;
- $msg_p(v_i)[q] < 0$ iff p has received a view v_{i+k} such that $q \notin V_{i,i+k}$.

Participant protocol

The only task of a participant p is to send $msg_p(v_i)$ to its coordinator when either:

1. p receives a new view v_{i+k} $(k \geq 1)$, or
2. vector $msg_p(v_i)$ changes after reception of v_{i+1} by p.

Coordinator protocol

Consider the coordinator c relative to $V = V_{i,i+k}$, where v_{i+k} is the most recent view received by c. As soon as c has received at least one vector $msg_p(v_i)$ from each $p \in V$, and upon each subsequent reception of such a vector, c evaluates the conditions 1 and 2 below (compare to definition D5):

1. $\forall p \in V$, $\forall q \notin V$, $msg_p(v_i)[q] < 0$;
2. $\forall p, q \in V$, $msg_p(v_i) = msg_q(v_i)$.

When both conditions are true, the global property $VA(v_i)$ has been detected. The coordinator then multicasts a message $\mathsf{install}(v_{i+1})$ to all the processes in $V_{i+1,i+k}$, using a classical reliable multicast [13], and finally installs view v_{i+1}. A process receiving message $\mathsf{install}(v_{i+1})$ installs view v_{i+1}.

Proposition 4 (Safety). Consider a view v_i and a process p. If p installs view v_{i+1} then the property $VA(v_i)$ is true.

Proof (sketch). If p installs view v_{i+1}, then there exists a coordinator c and a view v_{i+k} $(k \geq 1)$ such that conditions 1 and 2 of the coordinator protocol are true. As both conditions are equivalent to conditions 1 and 2 of definition D5, property $VA(v_i)$ is true. □

Proposition 5 (Liveness). Consider a view v_i and a process p. If $\exists k \geq 1$ such that the property $VA(v_i)$ is true for view v_{i+k}, then each process $p \in V_{i+1,i+k}$ eventually installs view v_{i+1} or fails.

Proof (sketch). The proof consists in two parts.
(i) There exists a coordinator c which detects $VA(v_i)$ in some view $v_{i+k'}$ ($k' \geq k$): Because $VA(v_i)$ is true for v_{i+k}, conditions 1 and 2 of D5 are true. By the protocol of §3.1, each process $p \in V_{i,i+k}$ sends its vector $msg_p(v_i)$ to the coordinator c of $V_{i,i+k}$. Three cases: (1) neither p nor c fail, in which case $msg_p(v_i)$ will eventually be received by c, which will be able to detect $VA(v_i)$ in v_{i+k}; (2) p fails, in which case a new view $v_{i+k'}$ ($k' > k$) is defined, and c needs $msg_p(v_i)$ no more in order to detect $VA(v_i)$ in $v_{i+k'}$; [5] (3) c fails, in which case a new view $v_{i+k''}$ ($k'' > k$) is defined and a new coordinator c' is considered. As $VA(v_i)$ is true for v_{i+k} and is a stable property, condition 2 of D5 is also true for each view v_{i+x} ($x > k$). Note v_{i+l} ($l > k''$) the view such that v_{i+l} is received by all $p \in V_{i,i+l}$. Property $VA(v_i)$ is thus also true for v_{i+l}. The same argument applied to v_{i+k} can now be applied to v_{i+l}.
(ii) Each process $p \in V_{i+1,i+k}$ eventually installs view v_{i+1} or fails: This follows directly because the coordinator which detects $VA(v_i)$ in some view $v_{i+k'}$ ($k' > k$), multicasts message $\mathsf{install}(v_{i+1})$ using a reliable multicast. □

4 Uniform view-atomic multicast protocol

The view-atomic multicast protocol of the previous section is now used to implement the uniform view-atomic multicast, which requires a two phase protocol. This leads to distinguish two cases: either (1) no failure (i.e. no view change) occurs before the end of the two phase protocol, or (2) failures occur during the execution of the protocol, requiring a termination protocol. The two phases are described in 4.1, the termination protocol in 4.2.

[5] We assume that the coordinator is defined in such a way that a new coordinator is defined iff the previous is not in the new view.

4.1 The 2 phase uniform view-atomic multicast protocol

Consider a group g, a current view v_i, and a process $p \in v_i$ issuing a uniform view-atomic multicast m to v_i. Message m is delivered to its destination after the following two phase protocol.

Phase I. Process p initiates a view-atomic multicast (see sect. 3) to v_i of message *(m, not deliver)*. The attribute not deliver instructs every destination process q to not *yet* deliver m to the application layer. On receiving *(m, not deliver)*, process q acknowledges reception by sending *(ack, id(m))* to the sender p. When acknowledgments have been received from every $q \in v_i$, the second phase of the protocol is initiated by p.

Phase II. In the second phase, p multicasts message *(id(m), deliver)* to v_i (the multicast need not be view-atomic!). When receiving this message, any $q \in v_i$ delivers m to the application layer. Because channels are FIFO, *(m, not deliver)* is always received before *(id(m), deliver)*. If no failure occurs, each $p \in v_i$ eventually delivers m. Uniform view-atomicity is thus trivially satisfied.

4.2 Termination protocol

If a failure occurs before the end of the protocol above, leading to the reception of view v_{i+1}, a termination protocol is needed to satisfy definition D4. Remarkably, the termination protocol does not however need to issue any message: this is because phase II is necessary only when no error occurs! The termination protocol takes place immediately before the installation of the view v_{i+1} (see 3.3).

Termination protocol. Consider a uniform view-atomic multicast m initiated in view v_i and a process $q \in v_i$ receiving view v_{i+1} before the delivery of m to the application layer. Two cases have to be considered:

Case 1 (normal case). If q is about to install view v_{i+1} and *(id(m), deliver)* has been received by q, then m is delivered to the application layer.

Case 2 (failure case). If q is about to install view v_{i+1} and *(m, not deliver)* has been received, but not *(id(m), deliver)*, then m is also delivered to the application layer.

Proposition 6. The two phase protocol of 4.1 together with the termination protocol satisfies definition D4.

Proof.
Safety. Assume $\exists p \in v_i$ which has delivered m in view v_i. We have to prove that $\forall q \in v_i$, if q has installed v_{i+1}, then q has delivered m (in view v_i). There are two cases to consider: either *(i)* p has delivered m after receiving *(id(m), deliver)*, or *(ii)* p has delivered m without having received *(id(m), deliver)* (m delivered by case 2 of the termination protocol). In case *(i)*, as *(id(m), deliver)* has been received by p, then $\forall q \in v_i$, q has received at least *(m, not deliver)*. By case 2 of the termination protocol, each $q \in v_i$ which installs v_{i+1} delivers m in v_i. Consider case *(ii)*. Message *(m, not deliver)* was sent using a view-atomic multicast. By definition D3, each q which installs v_{i+1} has received *(m, not deliver)*. By case 2 of the termination protocol, each $q \in v_i$ which installs v_{i+1} delivers m in v_i.

Liveness. Follows directly from the liveness of the view-atomic multicast protocol. □

5 General remarks

5.1 Distributed commit

Using the uniform view-atomic multicast primitive, distributed commit [3] is easy to implement. Consider a group g that has to reach a commit decision. This is done in the following way. The coordinator $c \in g$ defined relatively to a view $v_i(g)$ (see 3.3) is the process responsible for taking the decision. Once a decision D is taken, D is sent to $v_i(g)$ using a uniform view-atomic multicast. There are 2 cases to consider: (1) if any process receives D, then each process that installs $v_{i+1}(g)$ also receives D, or (2) no process ever receives D. Suppose that $v_{i+1}(g)$ informs of the crash of c. In case (1) a decision has been reached. In case (2), no decision has been taken by any process, and a new coordinator c' responsible for taking the decision is chosen in $v_{i+1}(g)$.

5.2 Network partitions

Up to this point we haven't considered communication failures leading to network partitions. Although this type of failure requires a careful analysis, one can make the following remarks:

- *definitions D3 and D4* are independent of partitions, provided that the sequence of views $v_i(g)$ is defined consistently in the occurrence of partitions;
- partitions are considered in [17]. The view sequence is defined as an ordered sequence of majority views (this sequence is unique). The implementation described in [17] is resistant to network partitions;
- the view-atomic and the uniform view-atomic multicast protocols of sections 3 and 4, if built on top of a GMP protocol resistant to partitions, are also resistant to partitions.

When partitions are considered, uniformity ensures that a message m received by any process, is received by every process in the majority view sequence. This follows from the way partitions are handled in [17].

6 Conclusion

The paper has defined a uniform reliable multicast in the virtually synchronous model (an asynchronous model with an imperfect failure detector), called *uniform view-atomic multicast.* The uniform reliable multicast problem had been essentially considered up to now in the synchronous model [8], where transmission delays are bounded. As shown by definition D4, uniform view-atomic multicast can be defined quite naturally in the virtually synchronous model, which is based on a view sequence delivered by the failure detector, also called group membership protocol or GMP. In this context, each modification of the group composition consists in reaching a decision among group members. The paper shows that a uniform view-atomic multicast primitive can be implemented ideally using a two phase protocol based on a multicast with a weaker semantics, the *view-atomic multicast.* The view-atomic multicast orders messages respectively to view changes. Using a two phase protocol for the uniform multicast primitive is per se not surprising (two phase protocols, or even three phase protocols, are used for solving a wide range of problems, including the distributed commit problem). The interesting aspect of the implementation described here is however to have clearly identified the correct weaker multicast semantics (i.e. view-atomic multicast) adequate for implementing the uniform view-atomic multicast. This primitive is not equivalent to the reliable multicast defined in the virtually synchronous model by [5]. The paper also shows how the distributed commit problem can be efficiently implemented using one uniform view-atomic multicast.

The contribution of the paper is thus twofold. First it contributes to a better understanding of the virtually synchronous model, which we believe is a very attractive asynchronous model. Second by showing how the GMP protocol can be used to solve the distributed commit problem, the paper establishes a clear link between both problems. In particular, as neither the view-atomic protocol nor the uniform view-atomic protocol introduce blocking conditions, the uniform view-atomic protocol blocks only when the GMP protocol blocks (see [17]).

Acknowledgments

We wish to thank Ken Birman for his useful comments on an earlier version of this paper.

References

[1] R.Bazzi, G.Neiger, *The Possibility and the Complexity of Achieving Fault-Tolerant Coordination*, Proc. 11th ACM Symp. on Principles of Distr. Computing, Vancouver, Aug. 1992, 203-214.

[2] K.Birman et al., *ISIS - A Distributed Programming Environment*, Cornell University, 1990.

[3] P.A.Bernstein, V.Hadzilacos, N.Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley, 1987.

[4] K.Birman, T.Joseph, *Exploiting Virtual Synchrony in Distributed Systems*, Proc 11th Symposium on Operating System Principles, Nov 1987, 123-187.

[5] K.Birman, A.Schiper, P.Stephenson, *Lightweight Causal and Atomic Multicast*, ACM Trans. Comput. Syst. 9, 3 (Aug. 1991), 272-314.

[6] K.M.Chandy, L.Lamport, *Distributed Snapshots: Determining Global States of Distributed Systems*, ACM Transactions on Computer Systems, Vol 3, No 1 (Feb 1985), 63-75.

[7] S.T.Chanson, G.W.Neufeld and L.Liang, *A Bibliography on Multicast and Group Communications*, ACM SIGOSR 23, 4 (Oct 1989), 20-25.

[8] T.D.Chandra, S.Toueg, *Time and Message Efficient Reliable Broadcast*, Proc. 4th Int Workshop on Distributed Algorithms, Bari, Sept 90, LNCS 486, Springer Verlag, 289-303.

[9] M.J.Fisher, N.A.Lynch, M.S.Paterson, *Impossibility of Distributed Consensus with One Faulty Process*, Journal of the ACM, Vol 32, No2 (April 1985), 374-382.

[10] A.Gopal, S.Toueg, *Reliable broadcast in synchronous and asynchronous environments*, Proc 3rd Int Workshop on Distributed Algorithms, Nice, 1989, LNCS 392, Springer Verlag, 110-123.

[11] L.Liang, S.T.Chanson and G.W.Neufeld, *Process Groups and Group Communications: Classifications and Requirements*, IEEE Computer 23, 2 (Feb 1990), 56-66.

[12] L.Lamport, R.Shostak, M.Pease, *The Byzantine Generals Problem*, ACM Trans. on Progr. Languages and Systems, Vol 4, No 3 (July 1982), 382-401.

[13] T.Joseph, K.Birman, *Reliable Broadcast Protocols*, *in* Distributed Systems, Ed. S.Mullender, Addison-Wesley, 1989, 293-317.

[14] G.Neiger, *Techniques for Simplifying the Design of Distributed Systems*, PhD thesis, Cornell University, Department of Computer Science (Aug 1988), TR 88-933.

[15] G.Neiger, R.Bazzi, *Using Knowledge to Optimally Achieve Coordination in Distributed Systems*, Proc of the 4th Conf on Theoretical Aspects of Reasoning about Knowledge, Morgan-Kaufmann, March 1992, 43-59.

[16] L.Peterson, N.Bucholz, R.Schlichting, *Preserving and using context information in interprocess communication*, ACM Trans. on Computer Systems, Vol 7, No 3 (Aug 1989), 217-246.

[17] A.Ricciardi, K.Birman, *Using process groups to implement failure detection in asynchronous environments*, Proc. 10th ACM Symp. on Principles of Distr. Computing, Montreal, Aug. 1991, 341-352.

[18] M.Raynal, A.Schiper and S.Toueg, *The causal ordering abstraction and a simple way to implement it*, Inf. Processing Letters 39 (1991), 343-350.

[19] A.Schiper, J.Eggli, A.Sandoz, *A New Algorithm to Implement Causal Ordering*, Proc 3rd Int Workshop on Distributed Algorithms, Nice, 1989, LNCS 392, Springer Verlag, 219-232.

[20] P.Verissimo, L.Rodrigues, J.Rufino, *The Atomic Multicast Protocol (AMp)*, *in* D.Powell (Ed), *Delta-4: A Generic Architecture for Dependable Distributed Computing*, Esprit Project 818/2252, Springer Verlag, 1991.