New Rules for Early Delivery in an Atomic Multicast System

Theodore Johnson and Lionnel Maugis
Dept. of CIS, University of Florida
Gainesville, Fl 32611-2024
ted@cis.ufl.edu

April 19, 1994

Abstract

An atomic multicast facility guarantees that a multicast message is delivered to the multicast group, and that every process receives messages in the same order. The availability of an atomic multicast facility greatly simplifies the implementation of many distributed system functions. Atomic multicast can be built on top of a reliable causal multicast by waiting until each processor in the group has multicast a message, then delivering the roots of the causal DAG. However, this approach makes the system run at the speed of the slowest processor. Performance can be improved by permitting an *early delivery* of messages. Dolev, Malki, and Kramer gave a protocol, ToTo, for early delivery atomic multicast built on a reliable causal multicast. In this paper, we show that the ToTo protocol is more restrictive than it needs to be, and present a simplification that improves performance. Next, we show how to extend the simplified ToTo protocol to permit early delivery in a wider range of operating conditions. Finally, we discuss a simple method to deliver some of the multicast messages even earlier.

Keywords: Atomic Broadcast, Causal Broadcast, Asynchronous System, Early Delivery, Distributed Protocol.

1 Introduction

Group communication protocols have been proposed to simplify the development of distributed applications. They guarantee reliable delivery and some defined order of messages for all recipients, even if the underlying communication subsystem is unreliable. In this paper, we do not consider synchronous systems, as they make strong assumptions about communication and processing delays [Cri90]. Such systems are designed to cope with the worst possible situation, and may not take advantage of the average case.

As processes replicate to provide fault-tolerance and availability, they need to cooperate when executing a global event – updating a replica for example. They generally agree on the construction of a global order aiming at mastering the uncertainties induced by asynchronism and failures. With this unique perception of the system's evolution, all nodes can take consistent decisions, preventing a replica from diverging. Atomic multicast primitives provide this total ordering consensus, as the following properties are guaranteed :

1. **Atomicity**. If a message is received by one member of the group, then it is received by all non-failed members.

2. **Order**. The sequence of delivered messages is identical for all group members.

3. **Termination**. Once a message has been successfully issued by a sender, eventually all group members will receive the message.

Early distributed agreement protocols are based on variations of two or three phase commit [LG90], central control [BSS91] or token ring [BG93, AMMS+93]. In addition to reliable and atomic multicast primitives, a membership facility provides virtually synchronous group of processes : all working processes receive the same set of messages between configuration changes. The communication topology is assumed to be fully connected and point to point. The channels are also reliable (eventual, exactly once delivery of uncorrupted messages) and first-in first-out.

More recent protocols take advantage of the communication medium. Indeed, simulating reliable point to point links over an unreliable broadcast domain significantly wastes network bandwidth[Bra92].

From a theoretical standpoint, the requirements for distributed agreement in a broadcast domain are given by Melliar-Smith, Moser and Agrawal [MMSA93]. They proved that a necessary and sufficient condition for the existence of a deterministic consensus protocol is delivery of each broadcast message to at least $\lceil (n+k+1)/2 \rceil$ process in an n-process system subject to k crash failures. In their broadcast model, if a message is delivered, it is delivered immediately and in order but not necessarily to all processes.

In the Trans system [MSMA90], they designed a reliable atomic broadcast primitive for unreliable datagrams. Messages contain piggyback information to perform recovery and decide which process has received a particular message. A message m_p sent by p contains two lists of message identifiers, $ack(m_p)$ and $nack(m_p)$. All i in $ack(m_p)$ have been received by p (positive acknowledgments), and all j in $nack(m_p)$ have been lost by p and need to be retransmitted (negative acknowledgments). Using the transitive closure of these acknowledgments, they derived a partial order which can in turn be converted into a total order. Even if reliable sends and receives are achieved efficiently, the processing cost of the total ordering protocol is high : the delivery predicate is not built incrementally, the ordering has to take into account the recovery of lost

messages.

Dolev, Kramer and Malki refined this approach in the Transis system [DKM92]. If i is in $ack(m_p)$ then p has received *i and all j* in $ack^*(i)$, where ack^* is the transitive closure of ack. Therefore, acknowledgments directly create a causal order, simplifying the total ordering (Toto) and group membership protocol. Both use this partial order and have a reasonable processing cost. Their membership algorithm has also the unusual capability of supporting more than one network partition [DMS93]. As we studied the Toto protocol, we found that the delivery rules are somewhat redundant and could be improved to make the delivery even more aggressive. This is the goal of the next sections.

2 Early Delivery

In this paper, we assume that we are building an atomic multicast facility on top of a reliable causal multicast facility. The reliable multicast facility provides a reliable group membership facility and the causality DAG.

Reliable Group Membership : Reliable broadcast requires either that every working processor receive a message, or that none of the processors receive the message. A method for keeping track of the currently working processors is required, or otherwise the first failure prevents all future messages from being delivered. The current set of messages that receive the multicasts is the *process group*. The process group is modified by a *configuration change*. Every process sees a consistent set of configuration changes.

When a processor fails, it is still desirable to deliver as many of the undelivered messages as possible. Birman [BSS91] introduced the concept of *virtual synchrony*, which requires that if one processor delivers a message m before a configuration change, then all non-failed processors deliver m before the configuration change. The use of virtual synchrony gives a solid meaning to the delivery list. In addition, it permits the delivery of the pending messages, since the requirement is only that non-failed processors receive the message.

Implementing reliable group membership and virtual synchrony requires a solution to the asynchronous agreement problem. Many solutions exist, and we assume that one of the solutions is available. We

also assume that the group membership facility guarantees virtual synchrony.

Causality DAG : The *causality DAG* is formed by the acknowledgments, ack, carried by the messages. The vertices are the messages, and $(m1, m2)$ is an edge if and only if message $m1$ acknowledges message $m2$. Note that the DAG could be constructed using any causal multicast primitive. We assume that such DAG is available for use by the total ordering algorithm.

Because of virtual synchrony, we only need to order the messages that are delivered between configuration changes. Because we have causal message delivery, messages are delivered to the processors in a mutually consistent order. In particular, the causality DAG is the same at all processors (although it is not necessarily revealed in the same order at all processors). That is, if message m is added to the causality DAG G_p at processor p, then all messages that causally precede m have already been added to G_p. Therefore, if processor q has also added m to its causality DAG G_q, q has also added all messages that precede m to G_q.

A simple algorithm for totally ordered delivery is to wait until there is a message in the causality DAG from every processor, then deliver the root messages in lexicographic order. While this algorithm provides total ordering at no additional cost in messages or acknowledgements, it delays delivery until every processor has sent a message. The delivery delay can be large, and it forces the system to operate at the speed of the slowest processor. Messages in the causality DAG can be delivered early (before the receipt of a message from every processor) if the messages satisfy a condition that will be detected by all processors, no matter in what order the causality DAG is revealed.

At processor p, the total ordering protocol maintains the most recent portion of the causality DAG, G. A message is added to G when the causal multicast facility releases it to the total ordering protocol. When a message is delivered to the processor, it is removed from G. The messages that can be delivered are the roots of G. We call the roots the *candidate* messages. Early delivery is triggered when *early delivery rule* is satisfied. At this point, some number of candidates are eligible for early delivery, we call them *source* messages. The source messages can be delivered when the protocol is guaranteed that every other processor will deliver the same set of messages. Thus the set of source messages must be stable:

Internal Stability : No non-source candidate message in G can become a source message.

External Stability : No unseen message (not in G) can become a source message.

If the current set of source messages is both internally and externally stable, they can be delivered. The trick is to devise a stability rule that works with the early delivery rule to guarantee correctness without requiring that a message be delivered from every candidate.

3 A Refinement of the ToTo Delivery Rule

In this section, we present a simplification and improvement of the ToTo early delivery algorithm of Dolev, Kramer, and Malki. We generally follow their terminology, but modify it where convenient and to simplify the presentation.

The ToTo protocol is built on top of a reliable causal broadcast, as described above. When a message can be causally delivered, the causality DAG G is augmented and the ToTo protocol is invoked. ToTo monitors G until it detects that some messages can be delivered. The delivery of messages creates a new *activation* of the ToTo protocol. The functions we define are to be evaluated on the current causality DAG when ToTo is invoked.

In the following discussion, we define some functions that the early delivery protocol uses. The functions are evaluated on the current causality DAG G at processor p. If there can be confusion about which causality DAG on which processor the functions are evaluated, we annotate the functions. Otherwise, we suppress the extra notation for simplicity.

Let P be the set of processors in the group. The first message in the causality DAG from a particular processor casts votes for the candidate messages that it causally follows. In addition, a candidate message votes for itself. Each candidate message, m, tracks the processors that voted for it in its *vote vector*, $VV(m)$. The i^{th} component of the vote vector, $VV(m)[i]$ is vote that processor i casts for m, and is defined by:

$$VV(m)[i] = \begin{array}{ll} 1 & \text{If processor } i \text{ votes for } m \\ 0 & \text{If processor } i \text{ votes, but not for } m \\ * & \text{if processor } i \text{ has not cast a vote.} \end{array}$$

We define the *tail* of a candidate message, m, to be the set of processors that sent a message that causally follow m. Then, $Ntail(m) = |tail(m)|$. The tail of the causality DAG G is the set of all processors that have a message in G, and $Ntail(G) = |tail(G)|$. Let u be the number of processors that have not voted yet (unseen votes) :

$$u = |P \ - ntail(G)| = |\{j \ : \ VV(m1)[j] = *\}| = n - Ntail(G)$$

We denote the number of votes that a candidate message m receives to be

$$nvt(m) = |\{i \ : \ VV(m)[i] = 1\}|$$

Candidate messages are compared on the basis of the votes they receive. Let $\Phi \geq n/2$ be a threshold parameter. Let $m1$ and $m2$ be two candidate messages. We define a per-vote function, $beats$, and $votes$ by

$$beats(m1, m2)[j] = \begin{array}{ll} 1 & \text{If } VV(m1)[j] = 1 \text{ and } VV(m2)[j] = 0 \\ 0 & \text{Otherwise} \end{array}$$

$$votes(m1, m2) = |\{j \ : \ beats(m1, m2)[j] = 1\}|$$

Candidate message $m1$ wins over candidate message $m2$ if it beats $m2$ in more than Φ votes. We define a function, Win as

$$Win(m1, m2) = \begin{array}{ll} 1 & \text{If } votes(m1, m2) > \Phi \\ 0 & \text{If } votes(m2, m1) > \Phi \\ X & \text{Otherwise} \end{array}$$

The value of Win is defined on the current causality DAG, G. We need a function that tells us about all

possible future extensions of G, G'. Thus, we use $Future$, which is all possible future values of Win:

$$Future_G(m1, m2) = \{Win_{G'}(m1, m2) \text{ in } G' \text{ extending } G\}$$

The only interesting ways that G' can extend G is by specifying the votes of the processors that have no message in G. Thus, $1 \in Future(m1, m2)$ if and only if the number of processors that vote for $m1$ but not $m2$, plus the number of unseen votes ("*" votes) is greater than Φ. That is,

$$\begin{aligned} 1 \in Future(m1, m2) \quad &\text{If} \quad votes(m1, m2) + u > \Phi \\ X \in Future(m1, m2) \quad &\text{If} \quad X \in Win(m1, m2) \\ 0 \in Future(m1, m2) \quad &\text{If} \quad 1 \in Future(m2, m1) \end{aligned}$$

Note that

$$\begin{aligned} 1 \notin Future(m1, m2) \quad &\Leftrightarrow \quad votes(m1, m2) + u \leq \Phi \Leftrightarrow votes(m1, m2) \leq \Phi + Ntail(G) - n \\ Future(m1, m2) = \{1\} \quad &\Leftrightarrow \quad Win(m1, m2) = 1 \end{aligned}$$

Let M_G be the set of candidate messages in the causality DAG G, The set of *source* messages S_G is defined as

$$i \in S_G \Leftrightarrow i \in M_G \wedge \forall j \in M_G,\ 1 \notin Future(j, i)$$

A message i is a source if it will never lose against any other candidate message j already in G (j cannot *beat* i in more than Φ votes in any possible extension of G).

The ToTo protocol is specified by its delivery rules:

ToTo Delivery Rules

1) (early delivery rule)

if a) (internal stability)

$$\forall m \in M_G \backslash S_G,\ \exists m' \in S_G,\ Future(m', m) = \{1\},$$

and b) (external stability)

$$\exists s \in S_G,\ nvt(s) > \Phi \wedge \forall s' \in S_G,\ Ntail(s') \geq n - \Phi.$$

Then deliver S_G.

2) (default delivery rule)

Else, if $Ntail(G) = n$,

Then deliver M_G.

The ToTo protocol can be improved and simplified by observing that part of rule 1b) is redundant. This observation leads to the Simplified ToTo (S-ToTo) protocol:

Simplified ToTo Delivery Rules

1) (early delivery rule)

if a) (internal stability)

$$\forall m \in M_G \backslash S_G,\ \exists m' \in S_G,\ Future(m', m) = \{1\}.$$

and b) (external stability)

$$\exists s \in S_G,\ nvt(s) > \Phi,$$

Then deliver S_G.

2) (default delivery rule)

Else, if $Ntail(G) = n$,

Then deliver M_G.

S_G is stable when a) all non-source candidate messages will not become sources in the current activation (if they lose once) and b) all unseen messages will not become sources (if a source message gets more than Φ votes, they will lose against it). The observation is that the first part of rule 1b) already guarantees external stability.

As messages are inserted in a causal order, a processor p that already has a message in the graph will not vote for an unseen message. The first message from p, m_p, votes for the candidate messages it causally follows which are already in G (or itself if it is a candidate message).

Suppose that the early delivery rule is satisfied. Then there exists a source s with at least $\Phi + 1$ votes. s can share these votes with other candidate messages in G, but all $\Phi + 1$ processors that have voted for s will not vote for an unseen message s'. When G is extended by inserting s', if s' is a candidate message, s' will not become a source message. Indeed, s' will lose against s, as a majority of processors will still vote for s and not for s' : $votes(s, s') > \Phi$. If s' gather all unseen votes, $nvt(s) + nvt(s') = n \Rightarrow nvt(s') < n - \Phi$. As $\Phi \geq n/2$, $nvt(s') < \Phi$. Therefore, $\forall i \in S_G, votes(s', i) < \Phi$ (s' will not take any source message i out of S_G).

The internal stability rule then guarantees that every process delivers the same set of messages on every activation. A non-source candidate message i will not become a source message if i loses against one candidate message j (if $win(j, i) = 1$ in G, $win(j, i) = 1$ in any extension of G).

We have presented an informal proof of correctness based on internal and external stability arguments while ignoring the interaction of the protocol with failures and group changes. These details are discussed in [DKM92]. We note that ToTo delivery rule 1b) is only used to guarantee that when the delivery of messages occurs, the processor has received every source message.

4 Generalized ToTo

The S-ToTo protocol can permit the delivery of a message after messages have been received from as few as $n/2 + 1$ processors. Thus, the S-ToTo protocol will work well if all processors are connected by a bus. If the network consists of several LANs connected by gateways, it might become unlikely that any message will receive half of the votes. Thus, it would be useful to trigger early delivery by messages that receive only a large minority of the votes. The cost to the protocol is a requirement that more total votes be gathered, to ensure external stability.

Again, we define our threshold to be Φ, and we restrict $1 < \Phi < n$. We need a definition of a source message that is simultaneously weak enough to allow the delivery of many source messages, but strong enough to quickly determine which candidates are non-source messages. We retain the meanings of Win

and $Future$ from the ToTo protocol, but we generalize the meaning of a source message as follows:

$$i \in S_G \Leftrightarrow i \in M_G \wedge [nvt(i) > \Phi \vee \forall j \in M_G, 1 \notin Future(j,i)]$$

It is necessary to allow a message with at least $\Phi + 1$ votes to become source message, because otherwise two messages with non-overlapping sets of $\Phi + 1$ votes may block each other from becoming sources. Note that the generalized definition of a source is the same as that for the ToTo protocol when $\Phi \geq n/2$.

With the generalized definition of a source the Generalized ToTo (G-ToTo) protocol is:

Generalized ToTo Delivery Rules

1) (early delivery rule)

if a) (internal stability)

$$\forall i \in M_G \backslash S_G,\ nvt(i) + u \leq \Phi \wedge \exists j \in S_G,\ Future(j,i) = \{1\}$$

and b) (external stability)

$$Ntail(G) \geq n - \Phi \wedge \exists s \in S_G,\ nvt(s) > \Phi,$$

Then deliver S_G.

2) (default delivery rule)

Else, if $Ntail(G) = n$,

Then deliver M_G.

Note that when $\Phi \geq n/2$, the G-ToTo protocol reduces to the S-ToTo protocol. The correctness of the G-ToTo protocol is a matter of showing that delivery occurs only when internal and external stability are achieved. Internal stability is explicitly tested. External stability is ensured because any possible source s must be in the causality DAG G when the delivery occurs. Suppose s is a source not in G when delivery occurs. But, s cannot receive more than Φ votes, because $Ntail(G) \geq n - \Phi$. Since the early delivery rule was triggered, there is a source s' with at least $\Phi + 1$ votes, so $1 \in Future(s', s)$. Therefore s cannot be a source.

5 Early Delivery in an Activation

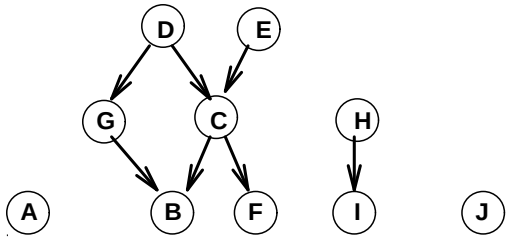
We make one final observation to permit the delivery of some messages while the activation is being calculated (GL-Toto). The sources messages are delivered in a deterministic order, by process id for example. If a message m_A from processor A becomes a source and will not be taken out of the sources when further votes are gathered, then m_A can be delivered even if the protocol has not reached a stable state.

More specifically, if a prefix w of M_G(ordered set) is stable, we can deliver all source messages in w. w is constructed iteratively. Each time the leftmost $m_i \in M_G \backslash w$ is considered. If $i > min(P \backslash ntail(G))$, $min(P \backslash ntail(G))$ has not voted yet and we have to stop the construction of w. If $m_i \in S_G$, m_i is added to w if no unseen message can win over m_i : $nvt(m_i) > \Phi \vee Ntail(G) \geq n - \Phi$, otherwise we stop. If $m_i \in M_G \backslash S_G$, m_i can be added to w if m_i will not become a source message : $nvt(m_i) + u \leq \Phi \wedge \exists m_j \in M_G,\ Future(m_j, m_i) = \{1\}$. Note that delivering early sources messages in w is valid even if the default delivery rule is applied ($Ntail(G) = n$). If the delivery cannot occur before all votes are gathered, this means that either the set of sources needs to be extended, or that the unseen message is not a candidate message. In both cases, when $Ntail = n,\ S_G = M_G$.

Here is a pseudo-code for such rules that is to be tested in the current activation, every time a new message is inserted in G :

```
for i = min(P) to min(P\ntail(G)) − 1
    if m_i is not marked
        if m_i ∈ S_G
            if (nvt(m_i) > Φ ∨ Ntail(G) ≥ n − Φ)
                mark m_i ; deliver m_i ;
            else stop;
        else if (m_i ∈ M_G)
                if (nvt(m_i) + u ≤ Φ ∧ ∃j ∈ M_G Future(j, i) = {1}) /* m_i is stable, continue */
                else stop;
        else /*m_i ∉ M_G, continue */;
    else /* m_i has already been delivered in the same activation, continue */;
```

6 Example



The following example[1] presents an activation of the protocol. Only the first message from a particular processor is shown. The causality DAG is depicted after the insertion of message H.

$n = 12,\ \Phi = 4,\ Ntail(G) = 10,\ u = Ntail(G) - n = 2,\ M_G = \{A, B, F, I, J\}$

m_i	A	B	F	I	J
$nvt(m_i)$	$\lvert\{A\}\rvert = 1$	$\lvert\{B,C,D,E,G\}\rvert = 5$	$\lvert\{C,D,E,F\}\rvert = 4$	$\lvert\{H,I\}\rvert = 1$	$\lvert\{J\}\rvert = 1$

$votes(row, col)$	A	B	F	I	J
A	$-$	$\lvert\{A\}\rvert = 1$	1	1	1
B	$\lvert\{B,C,D,E,G\}\rvert = 5$	$-$	$\lvert\{B,G\}\rvert = 2$	5	5
F	$\lvert\{C,D,E,F\}\rvert = 4$	$\lvert\{F\}\rvert = 1$	$-$	4	4
I	$\lvert\{H,I\}\rvert = 2$	2	2	$-$	2
J	$\lvert\{J\}\rvert = 1$	1	1	1	$-$

$S_G = \{B, F\}$ is computed as follows :

- $A \notin S_G$ as $nvt(A) = 1 \leq \Phi \wedge \exists B \in M_G, votes(B, A) + u = 5 + 2 > \Phi$
- $B \in S_G$ as $nvt(B) > \Phi$
- $F \in S_G$ as $votes(A, F) + u = 1 + 2 \leq \Phi \wedge votes(B, F) + u = 4 \leq \Phi \wedge$ $votes(I, F) + u = 4 \leq \Phi \wedge votes(J, F) + u = 3 \leq \Phi$
- $I \notin S_G$ as $nvt(I) = 2 \leq \Phi \wedge \exists B \in M_G, votes(B, I) + u = 5 + 2 > \Phi$
- $J \notin S_G$ as $nvt(J) = 1 \leq \Phi \wedge \exists B \in M_G, votes(B, J) + u = 5 + 2 > \Phi$

Sp is stable and can be delivered :

[1] Neither Total [MSMA90] nor ToTo [DKM92] are able to elect messages with such DAG, as they require a candudate set to receive more than $n/2$ votes to be elected.

1. $M_G \backslash S_G = \{A, I, J\}$

   $nvt(A) + u = 3 \leq \Phi \wedge \exists B \in S_G,\ votes(B, A) = 5 > \Phi \wedge$

   $nvt(I) + u = 4 \leq \Phi \wedge \exists B \in S_G,\ votes(B, I) = 5 \wedge$

   $nvt(J) + u = 3 \leq \Phi \wedge \exists B \in S_G,\ votes(B, J) = 5$

2. $Ntail(G) = 10 \geq n - \Phi \wedge \exists B \in S_G,\ nvt(B) > \Phi$

Now suppose that H has not been inserted yet. The functions are modified as follows :

$Ntail(G) = 9,\ u = 3,\ nvt(I) = 1,\ votes(I, A) = votes(I, B) = votes(I, F) = votes(I, J) = 1$

$S_G = \{B\}$ as

- $\forall i \in \{A, I, J\},\ i \notin S_G$ as $nvt(i) \leq \Phi \wedge \exists B \in M_G, votes(B, i) + u = 5 + 3 > \Phi$
- $B \in S_G$ as $nvt(B) > \Phi$
- $F \notin S_G$ as $nvt(F) \leq \Phi \wedge \exists B \in M_G, votes(B, F) + u = 2 + 3 > \Phi$

Furthermore, S_G is not internally stable, thus can cannot be delivered :

$$\exists F \in M_G \backslash S_G = \{A, F, I, J\},\ nvt(F) + u = 4 + 3 > \Phi$$

Even if the protocol has not reached a stable state, we can safely deliver B. The prefix $\{A, B\}$ of M_G is stable.

1. $A \in M_G \backslash S_G \wedge nvt(A) + u = 4 \leq \Phi \wedge \exists B \in M_G, votes(B, A) = 5 > \Phi$: A will not become a source message.
2. $B \in S_G \wedge nvt(B) > \Phi$: B will stay a source message.

7 Improvements

The family of total ordering protocols presented in this paper is geared towards reducing the average latency of agreed message delivery. We measure this latency as being the number of processors one need to hear from

before delivering a message ($Ntail(G)$). Each time a message m_i is agreed and delivered, we record also its number of votes $nvt(m_i)$. Our initial study on performance shows that the improvements are quite noticeable. We point out that the modifications needed for S-Toto, G-Toto and GL-Toto are straightforward and do not generate additional processing cost. Experimental results with our implementation of the protocols and simple simulations of a wider range of causal DAG showed that :

1. As much as 10% of the messages delivered early by S-Toto are not delivered by Toto.
2. As much as 5% of all messages can be delivered early with the lexicographic order.
3. GL-Toto performs better than Toto with wide causal graphs. The latter uses the default delivery rule ($Ntail(G) = n$) most of the time, while GL-Toto can delivery messages when it receives more than $n - \Phi$ votes.
4. We can order up to 400 messages per second on a 1MBit/sec ethernet, on a $n = 8$ processor system. With $\Phi = n/2$, the average $Ntail(G)$ is 5.12. In our testbed, performance depends heavily on the communication patterns, if there is a lot of traffic on the net for example. Furthermore, the above results do not take into account process failure or message recovery.

8 Conclusion

In a broadcast domain, a reliable causal multicast facility can be efficiently implemented without sending additional messages. Using this causal order and a special voting scheme, we have presented a family of protocols for delivering group messages in a unique order (again, without additional messages). The protocol reduces delivery latency by inserting messages in the total order as soon as they get elected, even if votes from all members have not been gathered (yet).

The delivery rules are parameterized to cope with a wide class of communication patterns. The set of elected messages is made as large as possible, is easy to compute, but depends on a majority threshold parameter Φ. Small values of Φ improve performance for wide causal DAG, while $\Phi = n/2$ seems to be a good choice for narrow graphs.

References

[AMMS+93] Y. Amir, L. E. Moser, P.M. Melliar-Smith, V. Agrawal, and P. Ciarfella. Fast message ordering and membership using a logical taken passing ring. *Intl. Conference on Distributed Computing Systems*, May 1993.

[BG93] T. Becker and K. Grieger. An efficient atomic muticast protocol for client server models. *IEEE International parallel processing symposium*, 1993.

[Bra92] R. Braden. Extending {TCP} for transactions. *Internet Network Group – Request for Comments 1379*, 1992.

[BSS91] K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Transactions on computer systems*, 9(3), 1991.

[Cri90] F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. *The Journal of Real-Time systems*, 2, 1990.

[DKM92] D. Dolev, S. Kramer, and D. Malki. Total ordering of messages in broadcast domains. Technical report, The Hebrew University of Jerusalem, November 1992.

[DMS93] D. Dolev, D. Malki, and Ray Strong. An asynchronous membership protocol that tolerates partitions. Technical report, The Hebrew University of Jerusalem, November 1993.

[LG90] S. Luan and V. Gligor. A fault tolerant protocol for atomic broadcast. *IEEE Transactions on Parallel and Distributed Systems*, 1(3), 1990.

[MMSA93] L. E. Moser, P. M. Melliar-Smith, and V. Agrawala. Necessary and sufficient conditions for broadcast consensus protocols. *Journal of Distributed Computing*, 7(2), December 1993.

[MSMA90] P.M. Melliar-Smith, L. Moser, and V. Agrawal. Broadcast protocols for distributed systems. *IEEE transactions on parallel and distributde systems*, 1(1), 1990.