Timestamp Acknowledgments for Determining Message Stability

K. Berket, R. Koch, L. E. Moser, P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{karlo, ruppert, moser, pmms}@alpha.ece.ucsb.edu

Abstract

Determination of message stability is important for multicast group communication systems, both for assuring applications that messages have been delivered and also for message buffer management. We present a protocol for determining message stability that uses a single scalar timestamp to acknowledge messages from many sources. The protocol avoids the linear growth in the acknowledgment size that occurs with vector acknowledgment protocols. Compared to other protocols, the protocol consumes significantly less network bandwidth, and its latency to the determination of message stability is only slightly larger. Synchronized physical clocks are shown to yield lower latency than logical clocks. We also present a simple connection establishment and group membership protocol for use in conjunction with the message stability protocol.

Key words: Message buffer mangagement, group communication, message stability, timestamp acknowledgment

1 Introduction

Message stability is an important concept in multicast group communication systems. A message is stable if every intended receiver of the message has received it. Knowledge that a message is stable allows the sender to remove the message from its message buffers, because it will never need to retransmit the message subsequently. Existing approaches to determining message stability require explicit acknowledgments [1, 2] or transitive acknowledgments [3, 4]. An explicit acknowledgment is sent by a receiver in direct response to a message it receives; explicit acknowledgments may be sent separately for each sender or may be grouped to form a vector acknowledgment [5] for all of the senders. A transitive acknowledgment, included in a newly generated message, provides information about the receipt of messages that causally precede [6] the message.

The major performance considerations for determining the stability of a message are: (1) the bandwidth consumed by the acknowledgments, (2) the computing power required to generate and process the acknowledgments, (3) the memory (buffers) required, and (4) the latency to determine that the message is stable.

The bandwidth consumed by the acknowledgments causes the explicit acknowledgment scheme to scale poorly. As the number N of group members increases, the bandwidth required increases as N^2. If the number of receivers is large, a sender may be overwhelmed by the acknowledgments (acks) it receives from them. Even if this ack implosion problem is remedied, the bandwidth consumed grows linearly with the size of the sender group. Transitive acknowledgments use less bandwidth, because only the identifiers of the direct causal predecessors of a message need to be sent as acknowledgments. The number of direct causal predecessors is typically smaller than the size of the sender group, but the bandwidth required still increases with the size of the receiver group due to ack implosion.

For the transitive acknowledgment scheme, the computing power required at the receivers is large because the receivers must build and maintain causal order graphs. The explicit acknowledgment scheme may require substantial computing power at the senders to handle the effects of ack implosion; however, a hierarchical structure can be used to mitigate the ack implosion problem.

The memory required for message buffering for both the transitive acknowledgment and the explicit acknowledgment protocols is proportional to the mean latency to the determination of message stability. Messages that are not determined to be stable must be retained in the buffers of the sender because it may need to retransmit them to achieve reliable delivery.

For explicit acknowledgments, the mean latency to the determination of message stability is on the order of the round-trip time between the sender and the receiver farthest from it. The transitive acknowledgment scheme is of the same order, but may incur additional delays due to the processing required at each receiver.

The timestamp acknowledgment protocol presented here determines the stability of messages, using a single scalar timestamp to acknowledge messages from all of the senders. A similar approach using a single logical (Lamport) timestamp for acknowledgments was presented in [7]. With the

This research has been supported in part by DARPA, Contract N00174-95-K-0083, and by Lawrence Berkeley National Laboratory for the U.S. Department of Energy, Contract W-7405-ENG.48.

timestamp acknowledgment approach, the bandwidth required for acknowledgments is independent of the number of senders, and the computational cost for acknowledgments is low.

2 Assumptions and Requirements

A distributed system consists of a finite set $\mathcal{N}$ of nodes that are connected by a network that offers unreliable point-to-multipoint communication channels. The number of nodes is assumed to be large, and the structure of the network can be arbitrary.

Each sender s has a receiver group $\mathcal{R}_s$ of receiver nodes to which it is sending messages, and each receiver r has a sender group $\mathcal{S}_r$ of sender nodes to which it is sending acknowledgments. A node $n \in \mathcal{N}$ can be both a sender and a receiver and, typically, all of the nodes are both senders and receivers, *i.e.*, for all $r, s \in \mathcal{N}$, $\mathcal{S}_r = \mathcal{R}_s = \mathcal{N}$. Each sender s must know the members of its receiver group $\mathcal{R}_s$, and each receiver r must know the members of its sender group $\mathcal{S}_r$. Without this requirement, the acknowledgment mechanisms cannot function properly; a membership algorithm must be employed to provide this information (see Section 6).

For the messages sent by the senders, the message channels are assumed to be reliable FIFO channels, *i.e.*, all of the intended receivers receive all of the messages sent by a given sender in the order in which they were sent. A receiver sends acknowledgments to all of the senders from which it receives messages. For the acknowledgments sent by the receivers, the message channels are assumed to be unreliable non-FIFO channels that deliver infinitely many acknowledgments if infinitely many acknowledgments are sent.

Each sender has access to a local clock, which can be either a physical clock or a logical clock.[*] The clocks of the senders are assumed to be synchronized.[†] The clock synchronization can be done either in hardware (*e.g.*, GPS-based clocks or radio-controlled clocks) or in software [8, 9]. The protocol does not rely on tightly synchronized clocks. However, the greater the degree of synchrony in the system, the smaller the mean latency to the determination of message stability, as discussed in Section 4. To achieve efficiency, the skew between the clocks at any two nodes should be less than the mean latency between the most distant sender-receiver pair.

A sender sends messages on a regular basis and, thus, the higher protocol layers must generate messages regularly. The mean latency to the determination of message stability is directly related to the minimum rate at which the senders send messages.

A sender determines that a message, which it originated, is stable if it has received an acknowledgment for that message from every member of its receiver group. The timestamp acknowledgment protocol presented in Section 3 determines the stability of messages.

```
sending a message:
  place current timestamp into timestamp field of message
  place message in messageBuffer

receiving a message:
  timestamp = msg.getTimestamp()
  sender = msg.getSender()
  if timestamp ≤ timestampVector[sender]
    ignore message
  else
    timestampVector[sender] = timestamp
    deliver message

sending an acknowledgment:
  place smallest timestamp from timestampVector[] into
    acknowledgment

receiving an acknowledgment:
  ack = msg.getAcknowledgment()
  receiver = msg.getSender()
  if ack > ackVector[receiver]
    ackVector[receiver] = ack
  minAck = min(ackVector)
  for all messages in messageBuffer with
      timestamp ≤ minAck
    mark message stable
```

Figure 1. Use of timestamp acknowledgments for determining message stability.

3 The Protocol

We now describe the timestamp acknowledgment protocol; the pseudocode is shown in Figure 1. Each sender uses its local clock to timestamp each message it sends. When sending an acknowledgment, a receiver acknowledges previous messages that the senders in its sender group originated, by means of an acknowledgment timestamp. The acknowledgment timestamp that a receiver sends is distinct from the sender's timestamp. When receiving an acknowledgment

[*] Both kinds of clocks represent time as a monotonically increasing function. A physical clock maintains its time value as a function of real time, while a logical clock determines its current time according to two rules postulated by Lamport [6]: (1) If two events e_1 and e_2 happen on the same processor and e_1 precedes e_2, then the time of e_1 is less than the time of e_2. (2) If e_1 represents the sending of message m and e_2 represents the reception of message m, then the time of e_1 is less than the time of e_2.

[†] For correctness, synchronized clocks are not necessary; however, for reasonable latency to the determination of message stability, they are necessary.

from a receiver, a sender determines whether the receiver has received previous messages that it sent.

If the underlying network does not provide the receiver with information about the sender of a message, then the message must contain the sender identifier. In some networks, such as ATM, each sender uses a different virtual channel and, thus, sender identifiers are not necessary.

Each sender s has a *send buffer* in which it places messages that it has sent but for which it has not yet received acknowledgments from all nodes in its receiver group $\mathcal{R}_s$. The sender s also maintains a *local acknowledgment vector* with one entry for each node in $\mathcal{R}_s$. Similarly, each receiver r maintains a *local timestamp vector* with one entry for each node in its sender group $\mathcal{S}_r$. At startup, all entries of these vectors are initialized to 0, and the send buffer is cleared.

The protocol requires a sender to send a message at least every t_{send} time units. Likewise, it requires a receiver to send an acknowledgment at least every t_{ack} time units.

Upon receiving a message, a receiver checks whether the timestamp of the message is less than or equal to the value in the entry of its local timestamp vector corresponding to the sender of the message. If so, it ignores the message (because of the membership rules stated in Section 6.) Otherwise, the receiver copies the timestamp of the message into the entry of its local timestamp vector corresponding to the sender of the message and then delivers the message. The local timestamp vector of the receiver holds the timestamp of the last message it has received from each of the senders in its sender group. Because the message channels are reliable FIFO channels, the timestamps for a particular sender are monotonically increasing. Before sending an acknowledgment, a receiver chooses the smallest entry in its local timestamp vector and places this timestamp in the acknowledgment. This timestamp serves as an acknowledgment for all messages that this node has received with smaller or equal timestamps. The values of the acknowledgments that a receiver sends increase monotonically.

Thus, if a receiver r sends an acknowledgment a_1, the next acknowledgment a_2 that r sends contains at least the information necessary to acknowledge all of the messages that a_1 acknowledges. If a sender s does not receive a_1 but does receive a_2, then it has received all of the acknowledgment information contained in a_1 and so does not need to receive a_1. This allows the protocol to use unreliable non-FIFO channels for acknowledgments. Moreover, if r sends the acknowledgments a_1 and a_2 in that order and the sender s receives those acknowledgments in the reverse order, then s can ignore a_1.

Upon receiving an acknowledgment timestamp, a sender knows that the receiver that sent the acknowledgment has received all messages with timestamps less than or equal to the acknowledgment timestamp. The sender inserts the new timestamp value into the entry of its local acknowledgment vector corresponding to the receiver that sent the acknowledgment timestamp, if the new timestamp value is larger than the timestamp value already in that vector entry.

The smallest entry of the sender's local acknowledgment vector determines the message timestamp up to which every receiver in its receiver group has received messages sent by any of the senders in its sender group. The sender can discard from its send buffer all messages that carry a timestamp less than or equal to the smallest entry in its local acknowledgment vector.

We now prove the following theorems, which establish the correctness of the protocol.

Theorem 1. (a) If a sender s determines that a message m that it sent is stable, then every member of s's receiver group $\mathcal{R}_s$ has received m.
(b) If a sender s sends a message m, then eventually s will determine that m is stable.

Proof. (a) A sender s determines that a message m that it sent with timestamp t_m is stable only if it receives an acknowledgment timestamp $at_r \geq t_m$ from every member r of its receiver group $\mathcal{R}_s$. A receiver r sends an acknowledgment timestamp $at_r \geq t_m$ only if it has received a message m' from each member s' of its sender group $\mathcal{S}_r$ with timestamp $t_{m'} \geq at_r$. Because, for all $r \in \mathcal{R}_s$, it is the case that $s \in \mathcal{S}_r$, it follows that all $r \in \mathcal{R}_s$ have received messages from s with timestamps greater than or equal to t_m and, therefore, that message m is stable.

(b) If a sender s sends a message m with timestamp t_m, then eventually every member of s's receiver group $\mathcal{R}_s$ will receive m. Eventually, each sender s' sends a message m' with a timestamp $t_{m'} \geq t_m$. Thus, by the reliable FIFO channel assumption, eventually every receiver r of s's receiver group $\mathcal{R}_s$ also receives a message from every member of its sender group $\mathcal{S}_r$ with timestamp $t_{m'} \geq t_m$ and, thus, sends an acknowledgment $at_r \geq t_{m'}$. Eventually, s will receive an acknowledgment $at'_r \geq at_r$ from every receiver r in its receiver group $\mathcal{R}_s$, and will determine that message m is stable. □

If all of the nodes in the system are both senders and receivers, the timestamp acknowledgment mechanism not only allows a sender to determine stability of its own messages but also allows it to determine the stability of messages from other senders. When a node receives a timestamp acknowledgment from another node, it knows that the node has received all messages with timestamps less than or equal to the acknowledgment timestamps from all other nodes. Thus, the timestamp acknowledgments carry information about the stability of messages from all nodes in the system (all-stable abcast in Isis [10] and safe delivery in Totem [11]). The determination of message stability is important for achieving virtual synchrony [10] and extended virtual synchrony [11], which are important for ensuring the consistency of replicated data and for coordinating processing within a distributed system.

Theorem 2. Suppose that every node in $\mathcal{N}$ is both a sender and a receiver. Then

(a) If a node $n \in \mathcal{N}$ determines that a message m sent by a node n' in $\mathcal{N}$ is stable, then every node in $\mathcal{N}$ has received m.
(b) If a node $n \in \mathcal{N}$ sends a message m, then eventually all nodes in $\mathcal{N}$ will determine that m is stable.

Proof. The proof is a simple extension of the proof of Theorem 1. □

4 Timestamp vs. Vector Acknowledgments

We now compare our timestamp acknowledgment scheme with the traditional vector acknowledgment scheme [5, 10] based on four different criteria: bandwidth used, cycles used, memory used, and latency to the determination of message stability. Because there is a minimum rate at which each sender sends messages and because each receiver sends acknowledgments at a constant rate, the number of messages increases with the number of senders and the number of acknowledgments increases with the number of receivers.

Bandwidth Requirements. In a system consisting of N_s senders and N_r receivers, the bandwidth requirement for the timestamp acknowledgment protocol is $O(N_r)$. One timestamp value is sent per receiver, independent of the number of senders in the system.

In contrast, a vector acknowledgment protocol sends one vector per receiver. Because the length of the vector is linear in the number of senders, the consumed bandwidth is $O(N_r N_s)$. Compared to the timestamp acknowledgment protocol, the bandwidth required by a vector acknowledgment protocol is larger by a factor of N_s.

Computation. For both timestamp acknowledgment and vector acknowledgment schemes, a sender must compute the smallest acknowledgment it has received every time it receives a new acknowledgment. It does so using a local acknowledgment vector with N_r entries. Entering a new entry in a presorted vector takes $O(\log N_r)$ steps every time an entry of the vector is updated. Because the number of acknowledgments is $O(N_r)$, the computational cost for each sender is $O(N_r \log N_r)$ for both acknowledgment schemes.

For the receivers the situation is different. Although a vector acknowledgment protocol requires each receiver to maintain a vector of length N_s to store the identifiers of the last messages it has received from each of the senders, sorting is not necessary. Therefore, the computational cost for each receiver is $O(N_s)$.

The timestamp acknowledgment protocol requires each of the receivers to compute the smallest timestamp of the last message it received from each of the senders. The computational cost at each of the receivers therefore is $O(N_s \log N_s)$.

Buffer Requirements. Because a sender must store messages (to be able to resend them in case of message loss) until it determines that they have become stable, the buffer size grows linearly with the latency to the determination of message stability, which is analyzed below. The buffer size of any sender must be at least as large as the worst-case acknowledgment time divided by the maximum message send rate of the sender.

Latency to the Determination of Message Stability. The timestamp acknowledgment protocol saves bandwidth by transmitting only one timestamp value, instead of a vector of length $O(N_r)$. This saving comes at the cost of additional latency to the determination of message stability.

Because the receivers acknowledge messages sent by all senders with a single scalar timestamp, the slowest sender determines the timestamp up to which messages are acknowledged. If one of the senders lags behind in sending messages (due to message loss or poor clock synchronization), the receivers can acknowledge messages only up to the slowest sender's last message. In this case, messages from other senders are not acknowledged when they could have been.

To analyze the latency to the determination of message stability in more detail, we consider a non-empty group $\mathcal{S}$ of senders and a non-empty group $\mathcal{R}$ of receivers. These sets may overlap and may be identical. For simplicity, we assume perfectly synchronized physical clocks. Clock skew and message loss are incorporated into the channel latency.

We let l_{sr} denote the message latency for a message m sent by a sender s and received by a receiver r. In the case of message loss, this latency includes the time for issuing a retransmission request and resending the message. Because the acknowledgment channels are neither reliable nor FIFO, it might happen that the acknowledgment is lost and a later acknowledgment from the same receiver acknowledges m. Therefore, we let l_{rs} be the acknowledgment latency from the time r sent its first acknowledgment of m to the time s received an acknowledgment of m from r. This definition of l_{rs} can be used if acknowledgments are delivered out of order. The latencies l_{sr} and l_{rs} are non-negative and need not be equal.

In the vector acknowledgment protocol, a receiver sends a vector acknowledgment at least every t_{ack} time units, which reaches the sender l_{rs} time units later. A message m sent by a sender s' is determined to have become stable

$$t_{1,s'} = \max_{r \in \mathcal{R}} \{l_{s'r} + l_{rs'}\} + t_{ack}$$

time units after s' sent it, which is the maximum round trip delay from sender s' to any receiver r plus t_{ack}. Note that $t_{1,s'}$ is specific to the sender s' and can be different for different senders in $\mathcal{S}$.

In our timestamp acknowledgment protocol, a sender s' sends a message m at least every t_{send} time units. A receiver r' can acknowledge m at the latest $t_{send} + \max_{s \in \mathcal{S}} \{l_{sr'}\}$ time units after s' sent m. The sender s' receives an acknowledgment of m from a receiver $r \in \mathcal{R}$ at most $t_{ack} + l_{r's'}$ time units later. Calculating the maximum over all receivers $r \in \mathcal{R}$, we obtain a latency to the determination of message stability of

$$
\begin{aligned}
t_{2,s'} &= \max_{r\in\mathcal{R}}\{t_{send} + \max_{s\in\mathcal{S}}\{l_{sr}\} + t_{ack} + l_{rs'}\} \\
&= t_{send} + t_{ack} + \max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs'}\}
\end{aligned}
$$

time units. Thus, $t_{2,s'}$ time units after s' sent m, s' has received acknowledgments from all receivers in $\mathcal{R}$ and determines m to be stable. Like $t_{1,s'}$, $t_{2,s'}$ is sender-specific.

Comparing t_1 and t_2, we find that they relate to each other as follows:

$$
\begin{aligned}
t_{1,s'} &\leq t_{2,s'} && (1) \\
t_{2,s'} &\leq \max_{s\in\mathcal{S}}\{t_{1,s}\} + t_{1,s'} + t_{send} - t_{ack} && (2)
\end{aligned}
$$

Inequality (1) results from the additional term t_{send} and the fact that the argument in the max function of t_1 is a subset of the argument in the max function of t_2. Inequality (2) holds because

$$
\begin{aligned}
\max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs'}\} &\leq \max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs} + l_{s'r} + l_{rs'}\} && (3) \\
&\leq \max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs}\} && (4) \\
&\quad + \max_{r\in\mathcal{R}}\{l_{s'r} + l_{rs'}\} \\
&= \max_{s\in\mathcal{S}}\{\max_{r\in\mathcal{R}}\{l_{sr} + l_{rs}\}\} && (5) \\
&\quad + \max_{r\in\mathcal{R}}\{l_{s'r} + l_{rs'}\} \\
&= \max_{s\in\mathcal{S}}\{t_{1,s}\} + t_{1,s'} - 2t_{ack} && (6)
\end{aligned}
$$

In inequality (3) we expanded the argument of the max function by including $l_{rs} + l_{s'r}$, and in inequality (4) we use the triangle inequality $\max_{a,b}\{a+b\} \leq \max_a\{a\} + \max_b\{b\}$. In equation (5) we separated the max functions for s and r, and in equation (6) we substituted $t_{1,s}$ and $t_{1,s'}$.

Note that $\max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs}\}$ is the maximum of the round-trip delays for any sender-receiver pair, whereas $\max_{s\in\mathcal{S}, r\in\mathcal{R}}\{l_{sr} + l_{rs'}\}$ describes the maximum latency from any sender s to any receiver r to a specific sender s'. These paths are not loops. Therefore, $t_{1,s'}$ and $t_{2,s'}$ cannot be compared directly. The expansion performed in inequality (3) allows the formation of an upper bound for $t_{2,s'}$ as a function of $t_{1,s'}$.

If the message and acknowledgment channels have the same latency, *i.e.*, $l_{sr} = l_{rs}$, the maximum latency to the determination of message stability is given by

$$
t_{2,s'} \leq \max_{s\in\mathcal{S}}\{t_{1,s}\} + t_{send}
$$

for a sender s'. This analysis shows that the timestamp acknowledgment protocol exhibits a larger latency to the determination of message stability than the vector acknowledgment protocol. However, it also shows that the latency to the determination of message stability for the timestamp acknowledgment protocol is at most equal to the latency to the determination of message stability for the most distant sender-receiver pair plus the latency to the determination of message stability of the sender s' for the vector acknowledgment protocol (assuming $t_{send} \leq t_{ack}$).

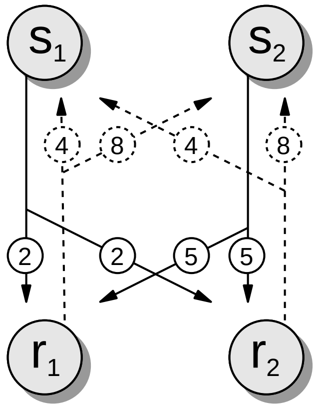

Figure 2. Acknowledgment latencies for the timestamp acknowledgment and the vector acknowledgment protocols. The senders are s_1 and s_2, and the receivers are r_1 and r_2. Solid edges represent message channels, and dashed edges represent acknowledgment channels. The labels on the edges represent latencies.

Figure 2 illustrates the latencies to the determination of message stability for the timestamp acknowledgment and vector acknowledgment protocols. The senders s_1 and s_2 each send at least one message every time unit, and the receivers r_1 and r_2 each send at least one acknowledgment every time unit. In the example, s_1 sends a message m_1 and r_1 receives it two time units later.

Using the vector acknowledgment protocol, r_1 acknowledges m_1 no later than three time units after s_1 sent it. The acknowledgment for m_1 from r_1 reaches s_1 no later than seven time units after s_1 sent m_1. The same applies for s_1 and r_2, resulting in the same maximum round trip delay of seven time units. Having received acknowledgments of m_1 from both receivers, s_1 determines that message m_1 is stable at most seven time units after sending m_1. At s_2, the maximum latency to the determination of message stability for a vector acknowledgment scheme is $\max\{5+1+8, 5+1+8\} = 14$ time units.

Using the timestamp acknowledgment protocol, the system behaves as follows. As above, the sender s_1 sends a message m_1 and the receiver r_1 receives it two time units later. Before r_1 can acknowledge m_1, it must wait for a message m_2 from s_2 with a timestamp at least equal to the timestamp of m_1. Assuming synchronized clocks, s_2 sends m_2 no later than one time unit after s_1 sent m_1. Message m_2 reaches r_1 six time units after s_1 sent m_1. The acknowledgment for m_1 from r_1 might be delayed another time unit. Thus, s_1 receives an acknowledgment for m_1 from r_1 no later than time 11. Similarly, s_1 receives an acknowledgment for m_1 from r_2 no later than time 11. Therefore, s_1 can determine that m_1 is stable at 11 time units.

5 Physical Clocks vs. Logical Clocks

The designers of distributed systems can choose between physical clocks and logical (Lamport) clocks. The previous analysis was based on physical clocks. The protocol also works for logical clocks, but does not perform as well, because logical clocks are more loosely synchronized than physical clocks.

Now we present an example, shown in Figure 3, that points out the drawbacks of logical clocks. For simplicity, we disregard the loss of acknowledgments. Initially, the clocks of the two senders have identical time values (we choose a value of zero for the example). In Figure 3a, sender s_1 sends five messages. The last of them carries the timestamp 5. Sender s_2 then sends a message with timestamp 1.

If physical clocks are used, the receivers r_1 and r_2 can acknowledge all six messages because s_2's message carries a timestamp greater than the timestamps of all of s_1's messages.

With logical clocks, the receivers r_1 and r_2 can acknowledge messages only up to the smaller timestamp, which is 1 (Figure 3b). Upon receiving the acknowledgment, both senders s_1 and s_2 remove the message with timestamp 1 from their buffers and update their clocks to 6. In Figure 3c, s_1 and s_2 send messages with timestamp 7. Now r_1 and r_2 can acknowledge all messages that have been sent. Upon reception of these acknowledgments, each sender can determine that all of its messages are stable and can remove them from its buffers (Figure 3d).

Physical clocks would have allowed the sender s_1 to determine that the messages are stable and to clear them from its buffers immediately after it had received the first acknowledgments from r_1 and r_2. With logical clocks, the additional round of message exchange is necessary to update the clocks so that the acknowledgments can be sent in the next round.

When using a single timestamp value for acknowledgments, clock synchronization is necessary to reduce the latency. The latency is determined by the largest skew between the clocks at any two senders. Before acknowledging a message from a sender with a faster clock, a receiver must wait for a message from a sender with a slower clock. The timestamp of the message from the sender with the slower clock must be at least equal to the timestamp of the message from the sender with the faster clock. Poorly synchronized physical clocks with a large clock skew exhibit an increase in latency similar to that for logical clocks. The larger the maximum clock skew (largest clock skew between any sender and any receiver), the longer it takes to determine that a message is stable.

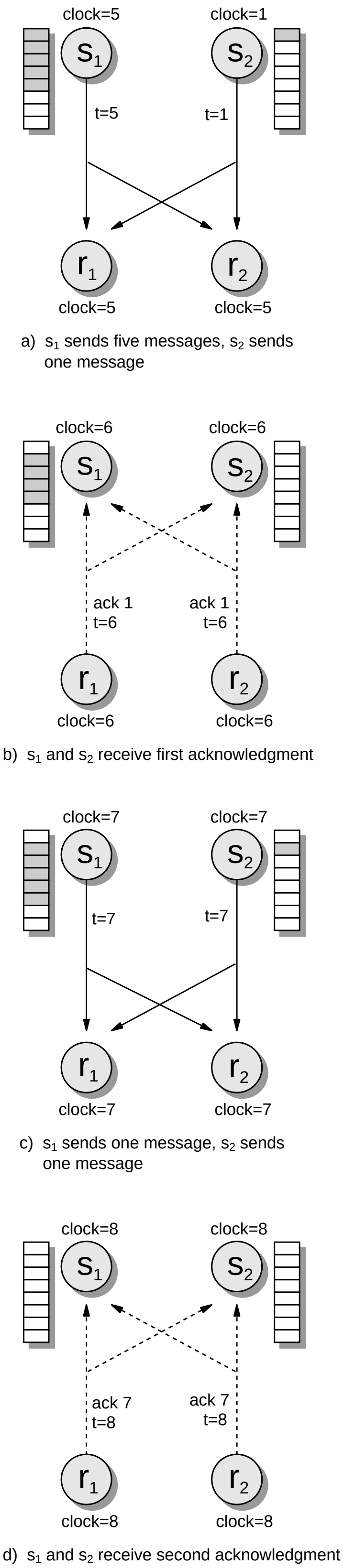

Figure 3. Timestamp acknowledgment protocol being used in a system based on logical clocks. T denotes the current reading of the logical clock, and t denotes the message timestamp.

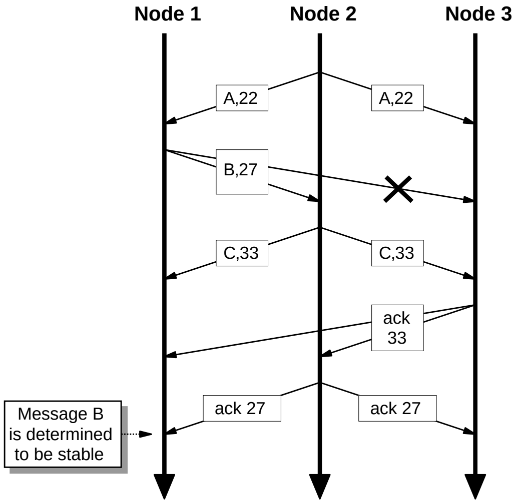

Figure 4. Lack of correct membership information can cause incorrect determination of message stability.

6 Group Membership

With timestamp acknowledgments, a sender knows which receiver created the acknowledgment, but it does not know which messages from other senders gave rise to the timestamp acknowledgment. This might cause a problem if the nodes do not know the correct sender and receiver memberships.

Figure 4 shows a group membership consisting of three nodes, each of which acts as both a sender and a receiver. A reliable multicast protocol that resides between the unreliable network and the timestamp acknowledgment protocol provides reliable FIFO delivery. In the example, node 3 lacks correct membership information. In its view, only node 2 and itself belong to the group.

The scenario begins with node 2 sending a message A with timestamp 22, followed by node 1 sending a message B with timestamp 27. Due to a network failure, node 3 does not receive message B. The underlying reliable multicast protocol does not detect the failure until it receives the next message from node 1.

Incomplete membership information causes node 3 to acknowledge all messages with timestamps less than or equal to 33, after it has received message C. Node 1 now assumes that node 3 has received message B and, therefore, incorrectly regards message B as stable after it has received node 2's acknowledgment with timestamp 27. Subsequently, the reliable multicast protocol requests the missing message B, which node 1 has already removed from its buffers.

This example shows the importance of correct membership information. Each of the senders must know the receivers in its sender group and vice versa. A lack of correct membership information can cause a sender to regard a message as stable, even though some nodes in its receiver group have not received the message.

The determination of accurate membership information is a difficult problem in general [12]. For connection-oriented networks (*e.g.*, ATM), the problem is more tractable. Correct membership information needed for determining message stability can be obtained by applying the following rules for a sender s and a receiver r:

- If a receiver r is a member of a sender s's receiver group $\mathcal{R}_s$, then a message channel has been established from s to r, an acknowledgment channel has been established from r to s, and s has received an acknowledgment from r, confirming that s is a member of r's sender group $\mathcal{S}_r$.
- If a sender s is a member of a receiver r's sender group $\mathcal{S}_r$, then an acknowledgment channel has been established from r to s, a message channel will be established from s to r, and r will receive a message from s.

For connection, the steps for adding a sender or receiver to a group membership are:

1. A receiver r establishes an acknowledgment channel from r to a sender s.
2. A receiver r adds a sender s to its sender group $\mathcal{S}_r$ when r sends its first acknowledgment to s, and records that acknowledgment timestamp in s's entry of r's local timestamp vector.
3. A sender s establishes a message channel from s to a receiver r.
4. A sender s adds a receiver r to its receiver group $\mathcal{R}_s$ when s sends the first message to r with a timestamp greater than the acknowledgment timestamp received from r.

For disconnection, the steps for removing a sender or receiver from a group membership are:

1. A receiver r or a sender s closes both the acknowledgment channel and the message channel.
2. A receiver r removes a sender s from its sender group $\mathcal{S}_r$, and similarly a sender s removes a receiver r from its receiver group $\mathcal{R}_s$, when it determines that either the acknowledgment channel or the message channel has been closed.

This abrupt termination suffices for determining message stability. Other properties of importance to distributed systems might require more precise coordination of both connection and disconnection.

By comparison with other membership algorithms, this membership algorithm is simple because each sender maintains its own receiver group membership and each receiver maintains its own sender group membership, and because each membership change is established pairwise between a sender and a receiver.

7 Conclusion

We have presented an elegantly simple protocol for determining message stability in multicast group communication systems, and have established its correctness. The protocol uses a single scalar timestamp to acknowledge messages from many sources and, thus, scales well to systems with large numbers of senders and receivers --- a scaling not achieved by message stability protocols using explicit or transitive acknowledgments. The network bandwidth required (the expensive resource) is substantially less than that for other protocols, while the latency to the determination of message stability and the required size of message buffers (the less expensive resources) are only slightly greater. As large multicast groups develop over ATM and over the Internet, improved message stability protocols, such as that presented here, will become increasingly important.

References

[1] V. G. Cerf and R. E. Kahn, ''A protocol for packet network intercommunication,'' *IEEE Transactions on Communications*, vol. 22, no. 5 (May 1974), pp. 647-648.

[2] P. B. Danzig, ''Finite buffers and fast multicast,'' *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, Berkeley, CA (May 1989), pp. 108-117.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki, ''Transis: A communication subsystem for high availability,'' *Proceedings of the 22nd IEEE International Symposium on Fault-Tolerant Computing*, New York, NY (July 1992), pp. 76-84.

[4] P. M. Melliar-Smith, L. E. Moser, and V. Agrawala, ''Broadcast protocols for distributed systems,'' *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1 (January 1990), pp. 17-25.

[5] G. T. J. Wuu and A. J. Bernstein, ''Efficient solutions to the replicated log and dictionary problems,'' *Operating Systems Review*, vol. 20, no. 1 (January 1986), pp. 57-66.

[6] L. Lamport, ''Time, clocks, and the ordering of events in a distributed system,'' *Communications of the ACM*, vol. 21, no. 7 (July 1978), pp. 558-568.

[7] P. Ezhilchelvan, R. Macedo, and S. Shrivastava, ''Newtop: A fault-tolerant group communication protocol,'' *Proceedings of the IEEE International Conference on Distributed Computer Systems*, Vancouver, Canada (May 1995), pp. 296-306.

[8] F. Cristian, ''A probabilistic approach to distributed clock synchronization,'' *Proceedings of the 9th IEEE International Conference on Distributed Computing Systems*, Newport Beach, CA (June 1989), pp. 288-296.

[9] T. K. Srikanth and S. Toueg, ''Optimal clock synchronization,'' *Journal of the ACM*, vol. 34, no. 3 (July 1987), pp. 626-645.

[10] K. P. Birman and R. Van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press (1994).

[11] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, ''Extended virtual synchrony,'' *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, Poznan, Poland (June 1994), pp. 56-65.

[12] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, ''On the impossibility of group membership,'' Technical Report 95-1548, Department of Computer Science, Cornell University, Ithaca, NY.