Reliable Group Communication in Distributed Systems *

S. Navaratnam
S. Chanson
G. Neufeld

Department of Computer Science
University of British Columbia
Vancouver, British Columbia
Canada V6T 1W5

Abstract

The design and implementation of a reliable group communication mechanism is presented. The mechanism guarantees a form of atomicity in that messages are received by all operational members of the group or by none of them. In addition, the order of messages is the same at each of the recipients. The message ordering property can be used to simplify distributed database and distributed processing algorithms. The proposed mechanism can survive despite process, host and communication failures. Survivability is essential in fault-tolerant applications.

1 Introduction

This paper describes the design and implementation of reliable one-to-many inter-process communication (IPC) mechanism. One-to-many IPC, also known as multicast or group communication, refers to an activity by which a single message may be transferred from one process to many other processes which may be in the same or in different hosts in the distributed system. The mechanism guarantees that the message will be received by all the operational receivers or by none of them. It also ensures that the messages sent by the senders will be delivered in the same order to all the receivers.

In a distributed environment a set of cooperating processes, possibly residing on different hosts, can be viewed as a single logical entity called a process group. The clients can communicate with the group as a single logical entity using the group’s logical name. Such a mechanism permits critical resources to be maintained on several hosts and be conveniently shared by client processes with enhanced modularity, performance and reliability[5].

2 System Model and Assumptions

The prototype model of the group communication mechanism is built using the V Kernel[7] operating on SUN workstations. Interconnection of the workstations is via an Ethernet. Within each workstation there exists two types of processes: processes responsible for implementing the group communication mechanism, and application processes which make use of the group communication mechanism. We assume that the application processes may fail but the processes responsible for implementing the group communication mechanism never fail unless the machine itself fails. We also assume that when processes or machines fail they simply cease execution without taking any malicious action (i.e., fail stop)[18]. If the machine at which a failed process was executing remains operational, we assume that this failure...
is detected by the underlying operating system. On the other hand, if the machine itself fails, all the processes executing in it fail and processes at other machines can detect this only by timeouts. Furthermore, we assume that the underlying system provides a reliable one-to-one message transport protocol. In other words, error detection and correction mechanisms (such as checksum, timeout and retransmission) exist which guarantee a unicast message to be delivered free of errors to its destination.

In the environment where the group communication mechanism is built, no information survives machine failures. Therefore, the case of a process receiving a message before machine failure and one where the machine fails before the message is delivered to it are indistinguishable. Thus, our group communication mechanism only guarantees that all operational members of a group will receive all the messages in the same order.

### 3 The Group Communication Mechanism

The design of the group communication mechanism should be general and not dependent on specific characteristics of the underlying network. For example, the underlying hardware may or may not support broadcast or multicast facilities. Consider the case where the underlying hardware supports only a single-destination message transport mechanism (unicast). In this case, delivery of a message to the group can be achieved by maintaining the list of members in the group, and sending the message to individual members using one-to-one IPCs. However if the underlying hardware supports broadcast or multicast, then the members of a group can subscribe to a particular multicast address. A message intended for the group can be sent to this address and only those machines where one or more members of this group reside will read the message.

Messages transmitted in broadcast networks, such as Ethernet, are available to all the receivers. However it is possible that a message may be lost or damaged. Unlike the reliable transport of a unicast message where the sender can retransmit the message until the receiver acknowledges, it is hard to support reliable transport of multicast packets unless the number and identity of the group members are known. If the membership list is maintained, then the message can be multicast to the members in a datagram fashion first and members whose acknowledgments are not received within a fixed time interval can be sent the message again on a one-to-one basis.

Thus, for reliable delivery of messages to all members of a group, some coordination mechanism is needed to maintain the group membership list. For a static group where the group membership never changes, the coordination mechanism can be built into the underlying system. However for a dynamic group where members may join or exit at any time, a group manager is necessary to maintain the membership list. This scheme is ineffective, however, if the machine where the group manager is executing fails. We therefore choose to replicate the group manager at all member sites, with one group manager acting as the primary and the remaining group managers as secondaries. Secondary group managers act as backup in case the primary group manager fails.

In addition to reliable delivery of messages to all members of the group, the group communication mechanism must also ensure that messages are delivered in the same order to all the members. In a system with a single sender and many receivers, sequencing messages to all of the receivers is trivial. If the sender initiates the next multicast transmission only after confirming that the previous multicast message has been received by all the members, then the messages will be delivered in the same order. On the other hand, in a system with many senders and a single receiver, the messages will be delivered to the receiver in the order in which they arrive at the receiver’s machine. Ordering in this case is simply handled by the receiver.

In general, group communication mechanism must operate between many senders and many receivers. In such a system, a message sent from a sender may arrive at a destination before the arrival of a message from another sender; however this order may be reversed at another destination.

A solution for such a system is to make it appear as a combination of the two simple systems. That is, the senders send messages to a single receiver which then transmits the messages to the rest of the receivers in an orderly fashion. Thus, the single receiver acts as a funnel process. This idea can be incorporated into our design without any additional cost because the group manager which we use to guarantee reliable delivery can also be used as a funnel process. Thus in our scheme, the senders will send the messages intended for a particular group to the group’s primary manager which will reliably and orderly transmit them to the members of the group.

Although the order property is essential in many applications, some applications do not require an order to be enforced between messages as the outcome of one may not affect the other. Since the overhead in enforcing the order property is non-trivial, our group communication mechanism provides two types of message transmission. One guarantees delivery of the messages in the same order to all members of a group and the other guarantees only atomicity, but where messages may be delivered in some arbitrary order. The former type of message transmission is known as OGSEND (Ordered Group Send) and the latter UGSEND (Unordered Group Send).

The group communication mechanism supports dynamic group creation as well as dynamic group membership. Thus,
in addition to communication, the mechanism should provide facilities to create, join and leave a group. A new group is dynamically created when a process invokes the creategroup() operation which returns the group_id of the group to the invoker. New members may join the group by invoking joingroup( group_id ) and members may leave a group by invoking leavengroup( group_id ). Processes send UGSEND messages by invoking ugsend( msg, group_id, msgtype ) and OGSEND messages by invoking ogsend( msg, group_id, msgtype ) respectively. If the IMMEDIATE_REPLY bit is set in msgtype, then the sender may be unblocked by the group communication mechanism before the message is delivered to the members of the group. Otherwise the sender will be unblocked only after all the members have received and acknowledged the message.

4 UGSEND and OGSEND

Each group has a primary manager and zero or more secondary managers. When a new group is created, a primary manager for this group is also created in the same machine. When a member from a different machine joins the group, and a secondary manager for this group does not already exist on the joining member’s machine, a secondary manager for this group will be created. The primary manager as well as the secondary managers maintain the process identifiers (pids) of those members of the group which are co-resident with the manager in the local group member list. Also the primary manager and secondary managers maintain the pids of all the group managers for the group in their group manager member lists. When a new secondary manager is created, the primary manager’s group manager member list is copied into the new secondary manager’s group manager member list. The primary group manager then updates its group manager member list with the pid of the new secondary manager and informs all the secondary managers of the group to update their lists as well.

When a member joins the group from a machine where the primary manager or a secondary manager for this group already exists, the pid of the new member is simply added to the local group member list. Although group membership information is distributed across all the machines, the primary manager and secondary managers maintain information only about their local members. Thus, when a message is sent to a group, the group communication mechanism must make sure that the message is delivered to all the group managers each of which will then deliver the message to its local members. This requires less space, less network traffic and reduced code complexity compared to the case of replicating the entire membership information in the primary manager and in all the secondary managers[8].

An OGSEND message for a group is first sent to the group’s primary manager. The primary manager is responsible for delivering the message to its local members and to the secondary managers of the group. The secondary managers receive the message, and deliver it to their local members. Only after the primary manager has successfully delivered the message will it operate on the next OGSEND message.

Unlike ordered delivery, unordered delivery does not require that the messages be funneled through a single primary manager. Thus, we have multiple senders and multiple receivers. If each sender maintains a list of all the receivers’ pids then every sender can participate in the message transmission activity. Even though the messages from senders can be guaranteed to be delivered to all the members, they may not be delivered in the same order. In our group communication mechanism, each secondary manager has information about the primary manager as well as all the secondary managers. Thus, every secondary manager can initiate UGSEND messages. When UGSEND messages are transmitted, the group communication mechanism first checks to see whether there is a group manager for this group available in the sender’s machine. If there is, the message will first be sent to it which will then transmit it to its local members and to the rest of the group managers, each of which in turn delivers the message to their local members. However if a local group manager does not exist, then the message is sent to the primary manager for the group which then transmits it to its local members and to the secondary managers for the group.

The method of transmitting a message from the primary manager to secondary managers depends on the functionality of the underlying network architecture. If the network supports only unicast, then a manager can send the message to the other managers using one-to-one IPC. However, if the network also supports broadcast facility then the message can be first broadcast to the group’s managers. The sending manager then waits for a specific time period for acknowledgements from each of the receiving group managers. If acknowledgements are not received at the expiration of the time interval, the sending manager redelivers the message to these managers using one-to-one IPC. This resend may result in duplicate message delivery to some of the receivers. In our proposed mechanism, the primary manager and secondary managers use transaction identifiers to detect and discard duplicate messages. Transaction identifiers are simply integer values. For example, consider the OGSEND message transmission. The primary manager maintains a variable called ogsend-send-seq-no (ossno) which keeps track of the next OGSEND message’s transaction identifier. When an OGSEND message transmission is initiated, the primary manager assigns the ossno to the message and transmits it to the secondary managers. The ossno is then incremented by one, to be used with the next OGSEND message. On the receiving end, the secondary managers maintain a variable called ogsend-receive-seq-no (orsno) to keep track of the
transaction identifier of the next incoming UGSEND message. When an UGSEND message is received by a secondary manager, the \texttt{oserno} associated with the message and the local \texttt{oserno} are compared and depending on whether they match, the message is either delivered to the local members or discarded.

5 Failure Detection and Recovery

To ensure that the group communication mechanism provides reliable service, the survivability property must be guaranteed. Survivability is a measure of how well a system can tolerate and recover from failures. Our discussion in this section will focus on two aspects of failures: process failures and machine failures. We have assumed that application processes such as group members may fail, but operating system processes such as primary or secondary managers which are used to implement the group communication mechanism are well debugged and do not fail unless the host machine itself fails. When a machine fails all the processes in it fail; thus machine failures are more serious than process failures.

Suppose the machine on which a group manager resides fails while the manager is delivering a message. In this case, it is possible that some of the members will not receive the message resulting in a partial delivery. Therefore, the group communication mechanism must be able to detect these failures and finish off any incomplete message transmission. The following two subsections describe how this problem can be handled for the cases of secondary manager host failures and primary manager host failures.

5.1 Secondary Manager Host Failure

If a secondary manager fails, the primary manager has to detect this and finish any incomplete UGSEND message transmission initiated by the failed secondary manager. To detect the failure of secondary managers, the primary manager has many options\cite{15}. All these schemes exploit the positive acknowledgement property of one-to-one IPC to determine process failure. We have implemented the following mechanism. The primary manager creates a single process called a \texttt{prober} to probe the liveness of the secondary managers. The \texttt{prober} periodically sends an \texttt{ARE.YOU.LIVE} probe message to each secondary manager. The probe message uses one-to-one IPC to which the secondary managers reply with an \texttt{LAM.LIVE} message. If a secondary manager fails, then the underlying system will inform the \texttt{prober} that it is trying to send a message to a nonexistent process and the \texttt{prober} will notify the failure to the primary manager.

Once the failure of a secondary manager is detected, the primary manager should delete the failed secondary manager’s pid from its manager member list and inform the rest of the operational secondary managers to do the same in order to maintain a consistent group view. However, before doing this, the primary manager must finish any incomplete UGSEND message transmission initiated by the failed secondary manager. To do this, the primary manager requests all the secondary managers to send to it their last UGSEND message received. If the returned messages as well as the last message received by the primary manager have the same transaction identifier value, then the failed secondary manager has either successfully completed its last UGSEND message transmission activity or no member has received its last UGSEND message. Either of these outcomes assures atomicity. However if there is a discrepancy among the transaction identifier values, then the primary manager takes the message with the highest transaction identifier and transmits it to the secondary managers. Those secondary managers that have already received the message simply discard the duplicates, but others receive the message and deliver it to their local members. Once this message retransmission activity is completed, the primary manager deletes the failed secondary manager’s pid from its group manager member list and informs the rest of the operational secondary managers about the failure.

5.2 Primary Manager Host Failure

The primary group manager fails when the machine on which it is executing fails. Primary manager failure is more serious than secondary manager failure. If the primary manager fails, UGSEND activities cannot be carried out and new members cannot join the group from a machine where a secondary manager for this group does not reside. Also, failure of secondary managers cannot be detected and incomplete message transmission activities initiated by the failed secondary managers cannot be completed. Even though the operational secondary managers may be able to participate in UGSEND message transmission, one cannot guarantee atomic delivery. Thus, a group cannot exist without a primary manager and function correctly for any extended length of time. In order to provide a continuous group communication mechanism, secondary managers must employ a scheme to detect the failure of the primary manager and select a new primary manager from among themselves. Similar to secondary manager failure detection, primary manager failure can be detected using different options. In the proposed group communication mechanism each secondary manager creates a “vulture” process to look for the failure of the primary manager. Since this mechanism is built on top of the V Kernel, the vulture process takes advantage of the \texttt{ReceiveSpecific} IPC primitive\cite{2} provided by the underlying system to detect primary manager failure. The vulture is receive blocked on the primary manager indefinitely. How-

\footnote{The transaction identifier values will differ by at most one.}

\footnote{The \texttt{ReceiveSpecific} IPC primitive periodically probes the remote machine to detect if the specified process is still alive.}
ever, if the primary manager fails, the underlying kernel in the vulture's machine will unblock the vulture and notify it that it is trying to receive a message from a nonexistent process. The vulture then informs its secondary manager about the failure of the primary manager.

5.3 New Primary Manager Selection Algorithm

Every secondary manager is a potential candidate to become the next primary manager due to the fact that each of them has the same global view of the group and each of them has the capability of detecting primary manager's failure. The algorithm to select a new primary manager must deal with several problems which may arise. For example, there may be inconsistencies due to two or more secondary managers attempting to become the new primary manager. Failures may even occur during the selection of the new primary manager itself. Therefore the algorithm must guarantee that when the selection is over, the group must be left with only one primary manager and all the secondary managers must know the identity of the new primary manager.

In a distributed environment such as ours, many algorithms are available to select a new leader[3,11,13]. We propose a succession list algorithm which is state driven. In this algorithm each potential candidate has a list of the identities (pids) of the other potential candidates. Normally this information is in the form of an ordered list. In our group communication mechanism the entry of each secondary manager in the manager member list is ordered by the time it joined the group. Since this list is updated only by the primary manager, this order will be the same in all the manager member lists. In case of primary manager failure, the first operational secondary manager in the list will become the next primary manager.

In explaining this algorithm we focus only on the state of one secondary manager, say smi, and not on the state of the entire distributed program (see Figure 1). When smi learns about the failure of the primary manager it checks if there are secondary managers in front of it in the manager member list. If so, it sends a probe message INFORM-STATUS to smj, the first in the list. If the underlying system informs smi that smj is not operational, smi probes the next secondary manager down the list.

If smi finds out that it is the first operational secondary manager in the manager member list, it broadcasts a CANDIDATE message to all the secondary managers through an aide process. When a secondary manager receives CANDIDATE, it informs its vulture process to look for the failure of the primary manager candidate. This is necessary because if the primary manager candidate fails then other secondary managers will be notified and they can restart the selection algorithm.

Each secondary manager receiving the CANDIDATE message from the aide process will acknowledge with the CANDIDATE,ACCEPT message. After the aide process has confirmed that all the secondary managers have received and accepted smi's candidacy it will notify sm. The primary manager candidate will then finish any incomplete OGSEND or group administration (joining or leaving a group) transmissions initiated by the failed primary manager, and any incomplete UGSEND message transmissions by a failed secondary manager. The new primary manager will then broadcast the message PMGR,ACTIVE to all the secondary managers through its aide process. After the aide process has re-

Figure 1: State transition diagram for primary manager selection algorithm.

---

443
ceived the CANDIDATE.ACCEPT message from all the operational secondary managers, it notifies the primary manager candidate which will then destroy the aide process, create the prober process and resume normal operation.

5.4 Network partition

Since we have assumed that all state information are lost in the event of process and machine failures, no state inconsistency can result due to these failures. However, this is not the case with respect to network partitions. In some types of systems, a process cannot distinguish between network partition and host failure. In this case, each partition without a primary manager will invoke the algorithm described above to elect a new primary manager and continue to function. The problem arises when partitions remerge - multiple primary managers with the same multicast address may be created and the state information among the group members may not be consistent. We believe the resolution of this situation should be left to the individual applications. A simple scheme to detect the remergence of network partitions is outlined below.

In our scheme, the prober process used to detect secondary manager failure also periodically broadcasts a RESOLVE message to the group's multicast address. This message contains the primary manager's identifier (pid) and is discarded by the secondary managers. However, a primary manager receiving this message from a different primary manager will know that the network had partitioned and remerged. Depending on the application, a suitable resolution scheme is carried out. In order to avoid confusion before resolution of the problem, each message transmitted by a manager always contains the identifier of the primary manager of the group in its message header. A manager receiving an OSEND or USEND message compares this identifier with the pid of its primary manager and discards the message if they are different (see [15] for details).

We have also proposed a scheme to merge the subgroups using an approach similar to that described in the election of a new primary manager. The scheme allows one of the primary managers to be elected and state information made consistent among all members of the resultant group. However, whether this action is desirable depends on the application. Details of the scheme can be found in [15].

6 Performance

We have done some preliminary measurements on the elapsed time for the group send primitives ogsend and ugsend. Elapsed time is the length of time during which the sender remains blocked after invoking an ogsend or ugsend routine. Elapsed time for these primitives depends on a number of factors, including the underlying system's workload, number of secondary managers of the group and whether the IMMEDIATE.REPLY bit is set in msgtype. The elapsed time is also dependent on the speed of the processor and the type of network interface. For our measurements we used four 16 MHz 68020 based SUN workstations, each connected to a 10Mbps Ethernet interface.

The measurements were made by performing OSEND and USEND message transmission $N$ times and dividing the total elapsed time by $N$ to obtain a reasonably accurate estimate for a single operation. Table 1 gives the elapsed time for the USEND and OSEND message transmissions as a function of the size of the remote group members. In this case the process which invokes the ugsend or ogsend primitives resides in the host where the primary manager of the group executes. The corresponding elapsed time for the unreliable and unordered group send is the V kernel running on the same hardware is also given for comparison.

<table>
<thead>
<tr>
<th>No. of managers</th>
<th>V group send</th>
<th>ugsend</th>
<th>ogsend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>7.7</td>
<td>8.2</td>
</tr>
<tr>
<td>2</td>
<td>18.5</td>
<td>19.1</td>
<td>19.6</td>
</tr>
<tr>
<td>3</td>
<td>19.3</td>
<td>20.2</td>
<td>21.0</td>
</tr>
<tr>
<td>4</td>
<td>20.0</td>
<td>22.2</td>
<td>23.5</td>
</tr>
</tbody>
</table>

Elapsed time (milli seconds) for ugsend and ogsend. The sending process is in the same host as the primary manager.

Table 1: Local primary manager

Table 2 is similar to Table 1, except that the process which invokes the primitives resides in a host where a secondary manager for that group executes. In both cases the receiving managers acknowledge a message only after the message is delivered to and acknowledged by their local members (i.e., IMMEDIATE.REPLY bit is off). $N$ is chosen to be 30,000 for both measurements.

<table>
<thead>
<tr>
<th>No. of managers</th>
<th>V group send</th>
<th>ugsend</th>
<th>ogsend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18.5</td>
<td>19.2</td>
<td>20.6</td>
</tr>
<tr>
<td>3</td>
<td>19.3</td>
<td>20.6</td>
<td>22.0</td>
</tr>
<tr>
<td>4</td>
<td>20.0</td>
<td>22.4</td>
<td>24.81</td>
</tr>
</tbody>
</table>

Elapsed times (milli seconds) for ugsend and ogsend. The sending process is in the same host as the secondary manager.

Table 2: Local secondary manager
The first observation from these figures is that the elapsed time for both primitives doubles when the first remote member is added to the group. However, the increase in the elapsed time for additional remote members is not very significant. This behavior is understandable, since the underlying network is a broadcast network and the time to transmit a group message to one remote site or multiple remote sites is the same, assuming that the probability of a packet loss is negligible.

The second observation is that the elapsed time for the UGSEND message transmission is less than that for the OGSEND transmission. However, this difference is not significant when the process which invokes these primitives resides in the same host as the primary manager. The reason is that the UGSEND message transmission is carried out by either the primary or secondary manager for the group executing locally, but the OGSEND message is sent to the primary manager which may be executing in a host different from that of the sending process. Finally, the additional cost to provide atomicity and ordering is found to be reasonably low.

7 Related Work

Although group communication has received considerable attention[1,2,4,8,9,10], only a few distributed systems have actually implemented such facilities. We mention four such projects which we consider relevant to our work.

In the V system[8], a group send operation terminates successfully if at least one member of the group receives the message and replies to it. Each machine has information only about local members of the groups. This information includes the identities of the local members and their group addresses. So when messages are sent to a group address, machines where members of this group are executing will receive it and deliver it to the members. The underlying kernel will retransmit the packet until at least one of the members of the group acknowledges the message. Therefore the V Kernel supports a very basic group communication mechanism to transport a message to multiple processes; additional properties such as reliability and order have to be built on top of it.

Cristian et al.[9] proposed a protocol for the reliable and ordered delivery of a message to all machines in a distributed system (i.e., broadcast) whereas our focus is on the delivery of a message to a set of processes, several (or all) of which could reside on a single machine. Their protocol is based on a simple information diffusion technique. A sender sends a message on all its outgoing links and when a new message is received on some incoming link by a machine, it is forwarded on all other outgoing links. After the reception of the message at a machine, it's delivery is delayed for a period of time determined by the inter-site message delivery latency. The messages are time stamped to enable order delivery and to detect duplicates. The performance of this protocol is dependent on the accuracy with which the clocks are synchronized and the operating system's task scheduling mechanism which is responsible for scheduling the relay task which relays an incoming message to the adjacent machines[2].

Chang et al.[8] proposed a protocol which, like Cristian's work, is responsible for the delivery of a message to all the machines in the distributed system. However their philosophy is similar to ours where the messages are funneled through a coordinator called token machine. A sender sends a message to the token machine which then transmits the message to the rest of the machines. The protocol places the responsibility on the receiver machines for reliable delivery. The token machine sequences the messages and transmits them to the rest of the machines in a datagram fashion. If a machine misses a sequence number then it sends the token machine a negative acknowledgement for the missing message. The token machine is rotated among the operational machines to provide reliability and resilience.

Birman's ISIS system[2] supports reliable group communication similar to ours. However, to ensure the order property, the messages are not funneled through a coordinator; instead a two-phase protocol is used. The protocol maintains a set of priority queues for each member, one for each stream of messages, in which it buffers messages before placing them on the delivery queue. When a message is received by a member, it temporarily assigns this message an integer priority value larger than the priority value of any message that was placed in the priority queue corresponding to the message's stream. Each member sends back this priority value to the sender. The sender collects all the replies and computes the maximum value of all the priorities received. It sends this value back to the recipients which assign this priority to the new message and place it on the priority queue. The messages are then transferred from the priority queue to the delivery queue in order of increasing priority. This guarantees order. However, the sender has to reliably communicate with the members twice before the message is delivered.

8 Conclusions

This paper describes the design and implementation details of a reliable group communication mechanism. A group communication mechanism is reliable if it has the two aspects of reliability: full delivery and correctness. In order to ensure full delivery the sender must know the identities of the members of the group. Issues related to correctness are atomicity, order and survivability. Atomicity ensures that every message sent to a group will be delivered to all operational members or to none of them. Order ensures that messages arrive in the same order they were sent at all the members of the group. Survivability guarantees that messages will be delivered correctly despite partial failures of processes, machines or communication links.

In order to provide the reliable properties transparently
to the application processes, a group manager process exists at each machine. One of the group managers is selected to be the primary manager; the rest are secondary managers. The messages from multiple senders are funneled through the primary manager, ensuring full delivery as well as order. If the primary manager fails, a new primary manager is selected from among the secondary managers. The new primary manager will finish any incomplete message transmission initiated by the failed primary manager. This guarantees atomicity and survivability.

For reasons of efficiency, two operations are provided, UGSEND and OGSEND. Both operations are reliable in that they support atomicity and survivability. However, OGSEND also provides ordered communication, while UGSEND is unordered.

In conclusion, we have demonstrated that reliable group communication can be implemented with only a modest increase in cost over unreliable communication.

References


