A Practical Approach to the InterGroup Protocols [⋆]

K. Berket, D. A. Agarwal, O. Chevassut

Ernest Orlando Lawrence Berkeley National Laboratory
1 Cyclotron Rd, MS 50B-2239
Berkeley, CA 94720

Abstract

Existing reliable ordered group communication protocols have been developed for local-area networks and do not, in general, scale well to large numbers of nodes and wide-area networks. The InterGroup suite of protocols is a scalable group communication system that introduces an unusual approach to handling group membership, and supports a receiver-oriented selection of service. The protocols are intended for a wide-area network, with a large number of nodes, that has highly variable delays and a high message loss rate, such as the Internet. The levels of the message delivery service range from unreliable unordered to reliable timestamp ordered. We also present a secure group layer that builds on InterGroup to provide SSL-like security for groups.

Key words: distributed systems, group communication, reliable multicast, security
PACS: 89.20.Ff, 89.20.Hh

[⋆] This work was supported by the Director, Office of Science. Office of Advanced Scientific Computing Research. Mathematical, Information, and Computational Sciences Division, U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-49126.

Email addresses: `KBerket@lbl.gov` (K. Berket), `DAAgarwal@lbl.gov` (D. A. Agarwal), `OChevassut@lbl.gov` (O. Chevassut).

URL: `http://www-itg.lbl.gov/CIF/GroupComm/` (K. Berket).

1 Introduction

Distributed applications often need to maintain consistency of replicated information and coordinate the activities of many processes. Collaborative applications and distributed computations are both examples of these types of applications. With the advent of grids [11], distributed computations will be spread across multiple computer centers requiring efficient mechanisms for coordination between the processes. Collaborations are by their very nature distributed and built in an incremental, ad hoc manner. Group communication provides a very natural mechanism for supporting these types of applications and allowing them to use a peer-to-peer architecture rather than a server-based architecture.

The MBone video-conferencing tools (vic, vat, and rat), the session directory tool (sdr) and the whiteboard tool (wb) [1] are excellent examples of the peer-to-peer model. These tools are designed to use multicast protocols to send data, which allows groups to form and communicate without coordinating with a server. This peer-to-peer model inherently makes the tools easier to design and to operate for groups of two and groups of hundreds. Because there are no servers, groups can be formed in an ad hoc manner with minimal setup.

There are many applications that can benefit from the use of a peer-to-peer group communication capability. Instant messaging systems, shared remote visualization, shared virtual reality, file-sharing applications and collaborative remote control of instruments are just a few examples. Most of these applications currently use a central server to collect messages and forward them to the participants. A peer-to-peer group communication service that provides group membership services, reliable ordered message delivery, and progress in the presence of process and network faults can be used to provide an efficient mechanism for participants to talk directly to each other.

Although group communication systems can provide these services, the protocols have historically been limited in their scalability. The InterGroup protocol suite, described in this paper, is a group communication system that tackles scalability by redefining the meaning of group membership, allowing voluntary membership changes, adding a receiver-oriented selection of delivery services (which permits heterogeneity of the receiver set), and providing a scalable reliability service. The InterGroup protocols are designed specifically with the intention of scaling to the Internet and to large numbers of participants.

[1] More information on all of these tools and their binaries can be found at http://www-itg.lbl.gov/mbone.

2 Related Work

Group communication systems provide reliable group ordered message delivery and membership services that allow the system to make progress in the presence of process and network faults. The main concerns in scaling these protocols are flow control, congestion avoidance, the reliable multicast service, and the delivery service. The delivery service needs to support reliable group-ordered delivery and a form of virtual synchrony [6,18]. Virtual synchrony is a property that defines consistency constraints for processes in a dynamic group setting. Because of this, it is essential for the membership protocols to be based on algorithms that reach consistent decisions.

Traditionally, group communication systems, such as the Totem Single Ring Protocol (SRP) [4], Transis [3], Isis [6], and RMP [24], were designed for the local-area network environment, where network latencies and losses are minimal, and there is a relatively small number of processes. Because of this, scalability concerns were not a top priority in the design of these systems.

Recently, group communication systems have made advances beyond the local-area network environment, and research into scaling the membership services has intensified. One example protocol is the Totem Multiple Ring Protocol (MRP) [2]. It uses a hierarchy of rings interconnected by gateways. Each ring is an instantiation of the Totem SRP and the gateways provide coordination and message forwarding between the rings.

To reduce the costs encountered in applications requiring the use of a large number of process groups, dynamic light-weight groups [13] were introduced. The idea is to map a large number of application process groups to a smaller number of protocol process groups. For example, this concept has been implemented by Horus [23], its follow-on Ensemble [22], the process group interface of Totem [19], and the Totem MRP. The process group interface of Totem provides a static mapping of the application groups to one protocol group, while the gateways of the Totem MRP filter the forwarding of messages based on application groups.

Another trend in group communication systems has been the breaking up of the system into building blocks. Horus introduced the building block approach to group communication systems. This approach is used in these systems to allow flexible delivery services by providing an interface to the various building blocks.

Moshe[15] is a membership service (building block) designed to be used by a group communication system in a wide-area environment. It provides an optimistic algorithm, for reaching a group-wide decision regarding the membership, that usually finishes in one round. This result is achieved by separating the membership service from the fault detection mechanisms so that most membership changes can be diagnosed by the fault detection algorithm and treated as voluntary. Moshe also separates the membership service from the virtual synchrony service.

Scalable Reliable Multicast (SRM) [10] is a protocol that was designed specifically to scale to the Internet. However, it is not a full group communication system; it only provides the mechanisms for recovering messages in a scalable manner. It achieves this by separating the reliability mechanisms from the loss detection mechanisms. The loss detection is left up to the application. The SRM protocol exchanges *session messages* between group members to update the control information at each individual process. The original SRM protocol used a group-wide multicast for all of its communication, which limited its scalability. One proposed solution to this problem is the organization of the group members in a self-configuring hierarchy [21].

3 The Architecture

InterGroup is designed to operate in an asynchronous environment where no bound can be placed on the time required for a computation or for communication of a message. Processes may fail by stopping and taking no further actions. The network is allowed to partition and re-merge, and messages may be duplicated or reordered by the network.

The InterGroup protocols are designed using a building block approach. The protocols are divided into four separate modules based on functionality. This modular design allows InterGroup to provide a flexible service model. The rest of this paper describes these modules. The message reliability service in the InterGroup protocols is provided by the reliable multicast module. The reliable multicast module is responsible for retransmission mechanisms. It is described in Section 5. The components of message ordering and delivery are in the message delivery module, described in Section 6. The process group membership protocols track the changes in the group membership. The components of process group membership are described in Section 7. The control hierarchy module provides a scalable mechanism for the exchange of control information between sites in the system. The components of the control hierarchy are described in Section 8. We precede the explanation of the individual components with Section 4 where we introduce the primary InterGroup messages and the data transmission algorithm.

4 Messages

4.1 Data Messages

Data messages are used for exchanging application information. The following fields are contained in a data message:

- *type*: The message type.
- *pg*: The identifier of the process group to which this message belongs.
- *sender*: The identifier of the process that sent this message. This uniquely identifies the process within the process group.
- *seq*: The sequence number of this message. It is incremented by one for each data message sent by this sender. If another message from this sender has the same value of this field, that message must be identical to this one or is a Keep Alive message. This field is used to detect message loss and allow FIFO ordering of messages from individual processes.
- *timestamp*: The timestamp of this message. This field is used to determine a process group wide ordering of messages and to indicate causality.
- *payload*: Byte array containing the application data.

4.2 Keep Alive Message

Keep Alive (KA) messages are used by the InterGroup protocols to help detect lost messages, to ensure progress in the delivery of messages, and to help maintain liveness. The fields of a Keep Alive message are identical to those of a data message except: (1) the *payload* is empty and (2) the *seq* field contains the value of the most recently sent data message.

Processes periodically send Keep Alive messages when no data messages are available. A keep-alive timer is set whenever a message is sent. If the timer expires before a data message is sent, a Keep Alive message is sent to the group. The header of this message contains the same information as in the last data message sent by this process, but with an updated timestamp value.

4.3 Data Transmission

The InterGroup system uses an adaptation of the Real Time Control Protocol (RTCP) flow control algorithms [20]. We use the RTCP algorithm for calculating the time to wait before sending the next message. However, we vary the bandwidth parameter used in the calculation, based on the message loss in the group. The bandwidth parameter is slowly increased during times when no losses are reported. When a process receives notification of a loss, it reduces the bandwidth parameter, thus reducing the amount of traffic that the process attempts to send to the group.

We also add a wrapper around the RTCP algorithm that checks to see if the process has too many outstanding messages. If that is the case, the process sends only Keep Alive messages until the number of outstanding messages falls below a threshold. The outstanding message limit stops the process from sending new data messages until the system stabilizes.

This approach to flow control is conservative and we have not yet tried to optimize the flow control mechanisms. The topic of flow control is still an open research topic.

5 Reliable Multicast

The reliable multicast module provides the mechanisms to request the retransmission of messages, and retransmit messages. The InterGroup protocols use reliable multicast mechanisms that are an adaptation of the SRM protocol [10]. SRM provides a nice framework for multi-sender systems.

5.1 Retransmission Requests

When the delivery module at a process detects a missing message, it notifies the reliable multicast module, which schedules a retransmission request. The requests are scheduled based on random timeouts to suppress redundant requests from processes sharing the loss.

The scheduling timeout is randomly chosen from an interval, which is a function of this process's estimated distance to the sender of the missing message. The timeout is thus chosen from the uniform distribution

$$3^i * [lowReqParam * d_S, (lowReqParam + highReqParam) * d_S]$$

seconds, where d_S is the estimated distance (latency) from this process to the sender of the missing message. The variables *lowReqParam* and *highReqParam* are adjustable via an adaptive algorithm and are discussed in more detail in [10] (where they are referred to as C_1 and C_2). The parameter i is the number of times the timeout calculation has been performed for this missing message. It is set to 0 the first time the timeout is calculated.

When the timer for a retransmission request expires, the process sends the retransmission request. It then increments i, recalculates the timeout, and resets the timer to the new timeout. If the process receives a retransmission request for a message for which it also has a request, it resets and reschedules the timer for its own request. A retransmission request and its associated timer are removed from the scheduler when the message is received by the process. This is true whether the message is in its original or retransmitted form.

5.2 Retransmission of Data

When a process receives a retransmission request for a message it has, it schedules the retransmission of the message. The retransmissions are scheduled based on random timeouts to suppress redundant retransmissions from multiple processes that can satisfy the request. The timeout used for the retransmission scheduling is randomly chosen from the uniform distribution

$$[lowRtxParam * d_R, (lowRtxParam + highRtxParam) * d_R],$$

where d_R is the estimated distance from this process to the sender of the retransmission request. The variables *lowRtxParam* and *highRtxParam* are adjustable via an adaptive algorithm, and are discussed in more detail in [10] (where they are referred to as D_1 and D_2).

If a process has scheduled the retransmission of a message and receives a retransmission of that message from another process, then this process cancels the timer and does not retransmit the message. Otherwise, when the timer expires, the process sends the retransmission of the message.

6 Delivery Services

The InterGroup system provides the following delivery services within a group: unreliable unordered, reliable source ordered, and reliable timestamp ordered. Each process chooses the delivery service it expects from a group when it joins the group. This decision applies to how messages are delivered at that process from the group. Each process makes this designation independently and there is no requirement that all the members of a group choose the same service.

Each InterGroup message is ordered based on three pieces of information in the message: the process identifier of the sender, the sequence number, and the timestamp. Each process identifier is unique within a group. The sequence numbers provide a count of the messages sent by a sender. The timestamp is used to preserve the causality between messages in the group. It is obtained from a Lamport clock [16] which guarantees that messages from the same source have increasing timestamps and that each message has a higher timestamp than any message the sending process has received prior to sending that message. The actual delivery order is dependent on the service chosen at the receiver.

6.1 Unreliable Unordered

The unreliable unordered (UU) delivery service is very similar to the service provided by IP Multicast. Messages received for the group are delivered directly to the application. Some messages might never be received, and multiple copies of the same message might be received. There is no guarantee regarding the order in which messages are received. This service also includes an option to allow the application to request information about the membership of the group.

An example application that might use this type of service is a video conference. Video and audio reliability differs from typical data in that a suitable representation of the data can be constructed without necessarily having all of the data. In fact, smooth latency and jitter averages are often more important than reliability. In addition, several codecs already incorporate data redundancy to protect against loss for this type of data.

Applications receiving messages using the UU delivery service, may suffer more overhead, when compared with IP multicast. This is caused primarily by the headers InterGroup places on the messages.

6.2 Reliable Source Ordered

The reliable source ordered (RSO) delivery service delivers messages from a source (or set of sources) to the application. This delivery service preserves the timestamp order of the messages from each source. The RSO delivery service guarantees that there will be no missing messages in the set of messages delivered from a particular source and that no message will be delivered twice at a receiver. Messages missing from the source order are detected by gaps in the sequence numbers of the messages received from a source. If this service detects a missing message, it requests a retransmission of that missing message from the reliability module. Keep Alive messages are used to aid in loss detection. The sequence number of a Keep Alive message is equal to the sequence number of the most recent data message sent by the source. Thus, Keep Alive messages allow us to recognize the loss of the most recent data message sent by the source.

The RSO service is well-suited to applications that currently use multiple TCP/IP [9] connections to transmit information to a group of sites. A specific usage example might be a large chain store distributing pricing information to all its outlets periodically.

6.3 Reliable Timestamp Ordered

Applications using the reliable timestamp ordered (RTO) delivery service receive messages in timestamp order for the process group. All of the processes in the process group keep a local Lamport clock. This Lamport clock has a slight modification such that the time is updated to match the local processor time whenever the Lamport clock lags behind the local time. Using the local processor time allows the time value to increase in the absence of messages. This provides a tighter synchronization mechanism for the clocks and leads to lower delivery latencies for the RTO delivery service.

The data messages are delivered to the application in timestamp order. A message is not delivered until all the messages with a lower timestamp have been delivered. The delivery algorithm keeps a message buffer for each process in the group. Messages are placed in the buffer only if there are no missing messages with a lower sequence number from that process. Missing messages are detected and requested for retransmission in the same manner as with the RSO delivery service. The messages in each buffer are ordered according to their sequence numbers. A message is delivered to the application when it is at the head of its buffer and there is a message with a higher timestamp at the head of every other message buffer.

If any of the message buffers are empty there can be no progress in the delivery of messages. Many applications do not send messages continuously, so Keep Alive messages are used to keep the timestamps moving forward and to allow continued message ordering. A Keep Alive message is placed in the message buffer corresponding to its sender if it has the same sequence number as the last message delivered from this buffer, and there are no messages in the buffer. In the case that a Keep Alive message is already in the buffer, the new message replaces the one already present if they have the same sequence number and the new message has a higher timestamp. When the delivery algorithm determines that a Keep Alive message is to be delivered, that message is removed from its message buffer and is not delivered to the application. The use of Keep Alive messages keeps the latency to delivery of messages at a relatively constant level.

The RTO delivery service is most often used when the application requires consistency across the group participants. A specific example is for database updates to a replicated database. It is also useful to collaborative applications for coordinating activities like a shared whiteboard. The RTO delivery service can also be used to achieve atomic broadcast in systems where all the participants subscribe to the RTO service and the number of participants is known in advance.

6.4 Preserving Causality of Messages

Each process determines the delivery service it desires for a process group at the receiving end. To achieve this receiver-oriented delivery service, we order messages independently at each receiver without restricting the delivery service choices of the other receivers, and separate the reliability service from the ordering service.

One significant benefit of the receiver-oriented selection of delivery service is that the number of acknowledgments that must be gathered from participants is minimized. Only processes subscribed to the RTO delivery service have to participate in this operation. Processes that choose the other delivery services only wait for the messages to be in the order they have requested, thus cutting down on the latency to deliver messages.

Since each process can choose its delivery service, the sender of a message does not know which delivery service(s) will be requested. To ensure causality of messages in the group, the InterGroup protocols subscribe a process to the RTO delivery service when that process is sending messages regardless of the service it is actually subscribed to. This is handled internal to the InterGroup protocols and does not affect the application's delivery service. This results in unnecessary overhead if none of the processes receiving messages in the process group are subscribed to the RTO delivery service. However, the increased flexibility and data abstraction that this method provides offset this negative.

7 Membership

The discussion thus far assumed that the set of processes in a group is static. The membership module provides the InterGroup protocols with the mechanisms to handle dynamic groups where processes may join, leave, or fail and groups can merge or partition.

This section explains the InterGroup approach to dynamic membership and its effect on the application. We refer to a representation of the group membership as a *view*. A view is delivered to the application whenever the membership changes. A view consists of an *identifier* and the *membership*. Each identifier is unique for this group and the identifiers have an ordering relation that guarantees that the identifiers monotonically increase across views at a process. The membership of a view is a set of process identifiers that represent the processes that share this view.

A cornerstone of the InterGroup approach to membership is the recognition that the message order and reliability constraints can be met by tracking only the processes currently sending messages in the group. In the InterGroup system, not all processes are equal. In each process group, a process is classified by its recent activity. If the process has been sending data to the group recently, it is classified as an *active sender*. Each group thus has two memberships; the *receiver membership* that contains all the members of the group, and the *sender membership* that contains only the active senders. The sender group membership is maintained using consistency-based membership mechanisms that are based on the algorithms used in Transis [3]. In this algorithm the membership is kept explicitly. In contrast, our receiver group membership is not needed for the purposes of message ordering and reliable delivery, so it does not need to be explicitly maintained. Processes are moved into and out of the sender group based on their current sending behavior.

The active senders run a membership repair algorithm (MRA) designed so that participation of processes outside the sender group is minimized. The active senders run the membership algorithms to reach a consistent decision on the new membership and on the place in the message flow that a membership change occurs. An active sender enters the MRA if a process in the membership of its view fails, a process that is not in the membership of its view starts sending in the group, or it detects that another process in its view is running the MRA.

In the MRA, the sender executes an algorithm to determine the next view that it will deliver. This algorithm requires that all of the processes in the membership of the next view agree to install the same view. It also requires that the processes in both the membership of the current view and the membership of the next view agree on the messages that must be delivered prior to installing the next view. Once this agreement has been reached the processes deliver the messages that they agreed to deliver and then each process sends a message signaling its readiness to install the next view. Once a process receives this message from every process in the membership of the next view, it installs the next view and ends the MRA. If any process in the membership of the next view fails before the MRA ends, the processes restart the MRA.

The agreements of the next view and messages to deliver in a view of the MRA algorithm are important as they provide consistency and some synchronization between the processes. They also enable us to provide virtual synchrony [6,18] to applications that request the RTO delivery service.

Processes that are not active senders but that have requested the RTO delivery service also participate in the membership repair by running the receiver membership repair algorithm (RMRA). The RMRA is initiated on receipt of a message that signals the beginning of the MRA from a process that is currently an active sender. The process running the RMRA halts delivery of messages to the application. The timestamp of the last message delivered to the application before delivery was halted provides the earliest logical time at which the process can begin a new membership. The process then waits for a message that describes the membership change. This message is sent by an active sender and signals the completion of the MRA. The information in this message includes a list of members in the new view and the logical time at which the view begins. This process then attempts to recover and order all of the messages that precede the new view. If it succeeds in this recovery, it installs the new view, and successfully completes the RMRA. Otherwise, it does not successfully complete the RMRA.

The combination of the MRA and RMRA allows the active senders to provide virtual synchrony information to the entire group, while keeping the number of processes participating in consensus-based membership to a minimum.

The membership mechanisms described above deal with membership changes that occur involuntarily. If membership changes occur at a process's request more efficient mechanisms can be used. A process, wishing to join the sender group, contacts a member of the sender group and that process serves as a *sponsor*. The sponsor sends a message to the group requesting to add this process to the sender group. The delivery of that message signifies the addition of the process to the sender group. A process wishing to leave the sender group sends a message to the group, requesting that it be removed from the sender group. The delivery of that message signifies the removal of the process from the sender group.

InterGroup also employs voluntary membership mechanisms for the receiver group. Processes may enter and leave the receiver group without impact on the other group members. However, when entering the group, a process needs to determine the current view. The joining process first contacts a member of the sender group that serves as a sponsor and requests this information from it. The sponsor replies with the current view and the current sequence number for each of the members of that view.

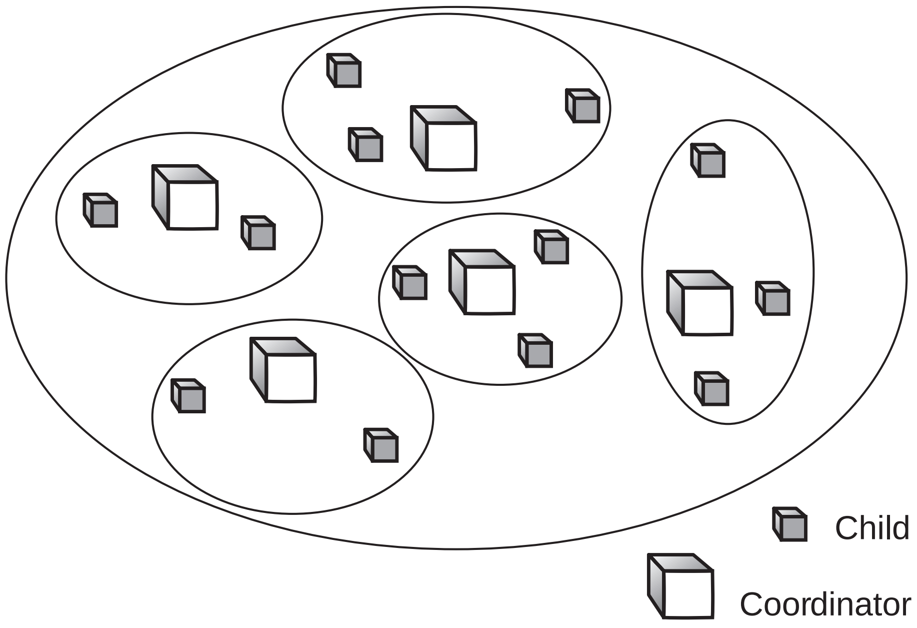

Fig. 1. The control hierarchy.

8 Control Information

There are many types of control information that need to be gathered by the InterGroup protocols. The reliable multicast protocols need information about the latency between processes. The buffer management protocols need information from the reliability service so messages are held in the buffers only until they have been delivered to all of the processes in the group. This information must be obtained from all of the processes participating in InterGroup. To make this operation more efficient and scalable, we use a hierarchically organized structure to gather and disseminate the control information.

The logical structure of this hierarchy attempts to mimic the underlying network topology by considering the latencies between control processes and is based on the work proposed in [21] for SRM. The control processes are organized in multicast trees, with the roots referred to as *coordinators*, and the leaves referred to as *children* (Fig. 1). Each child is associated with at least one coordinator. The *local group* of a coordinator is composed of the children of that coordinator (including the coordinator itself). The *coordinator group* includes all the coordinators. Each control process limits its communication of control information to its local group; the coordinators also communicate with the coordinator group. The frequency of control messages is regulated using a simplified version of the congestion control algorithm used by RTCP [20] which works to keep the bandwidth used by the messages to a constant.

A self-determination protocol is executed periodically to determine whether a control process in the hierarchy should change states (child to coordinator or vice versa) in response to changes in the system. The determination of a state change is a local decision made at each control process based on control information gathered before the self-determination protocol is executed, and based on a predefined set of rules adapted from [21].

A control process, upon startup, checks to see how many coordinators are present in the hierarchy, by checking the messages sent in the coordinator group.[2] If the number of coordinators is less than or equal to the expected average coordinator group size or the control process does not receive a control message from the coordinator group within a given time, the control process starts up as a coordinator. Otherwise, the control process starts up as a child.

A control process, starting up as a coordinator immediately starts sending control messages. A control process, starting up as a child, needs to find a coordinator that will accept it into its local group. This step is accomplished by using an expanding ring search. When it finds a coordinator, the control process starts sending messages to the coordinator's local group and becomes a child. If the expanding ring search doesn't provide a coordinator within a given time, the control process starts up as a coordinator.

The detection of control process faults is accomplished via a fault-detection algorithm that runs periodically. If the information from a control process has not been updated recently, the control process is removed from the data structures and thus removed from this control process's view of the membership of the hierarchy. When a fault regarding the coordinator for this control process is detected, the control process uses the information from the self-determination protocol to decide whether it should find another coordinator and remain a child, or whether it should become a coordinator.

Global control traffic from all control processes is gathered using the hierarchy. The control information is aggregated as it is gathered through the InterGroup control hierarchy, thus limiting the control information message size.

8.1 Determining the Distance Between Processes

The distance or latency between processes is used in the timer calculations of the reliable multicast. It is important to have a good estimate of this value to avoid duplicate retransmission requests and message retransmissions. The structure of the hierarchy allows a good estimate of the latency while maintaining scalability.

The distance between any two processes is calculated using a simplified version

[2] Only one message is necessary to make this determination.

of the NTP [17] algorithm. Each control message includes a local timestamp, that records when the message was sent. Process A sends a control message msg that contains a local timestamp $T1$. When process B receives msg, it records its local timestamp $T2$, stores the timestamp pair $T1, T2$ and associates it with A. Process B then sends another control message msg' that contains a local timestamp $T3$ and the timestamp pair associated with process A $(T1, T2)$. Upon receiving msg', process A records its local timestamp $T4$ and can calculate the one-way distance to process B as:

$$distance_{A,B} = (T4 - T3 + T2 - T1)/2$$

This calculation of the distance assumes symmetric paths (which is not always the case), but it does not assume synchronized clocks.

Within a local group the processes calculate the distance to every other process in that local group. The coordinators do the same within the coordinator group. Each coordinator then distributes its distance calculations for the coordinator group to its local group members. Thus, the distance between members of two different local groups is approximated by the distance between the coordinators of those local groups.

8.2 Message Buffers or Determining Message Stability

A *message buffer* is used to store messages, so that they can be retransmitted if necessary. Since we don't necessarily want to store every message in the system, we need to determine when a message will no longer be requested for retransmission. A message will no longer be requested for retransmission if that message has been received by every process. Such messages are referred to as *stable* messages. A stable message can be removed from the message buffer.

We provide a protocol that determines message stability for buffer management. It is based on the protocol in [5], that uses a timestamp acknowledgment mechanism for stability determination. Each process keeps track of the timestamp of the most recent *locally stable* message delivered to the application. A process determines that a message is locally stable if the InterGroup protocols have finished processing the message at that process. A process will make the determination of local stability based on the delivery guarantee it chooses. A message becomes stable in the process group when every process in the process group has determined that the message is locally stable. The timestamp of the most recent locally stable message is periodically communicated to the control hierarchy which calculates and returns the timestamp of the most currently (system-wide) stable message. All messages with timestamps smaller than this value are marked as stable and removed from the process's message buffer.

9 Securing a Group Communication System

When communicating over an open network like the Internet, security is essential to allow application components to reliably communicate in the face of adversaries. The Secure Group Layer protocol (SGL) [1] adapts the notion of view-based group communication to the context of security. SGL provides distributed applications with a platform they can use to achieve reliable and secure communication among distributed components.

SGL is a security protocol for reliable multicast communications designed to be similar to the well-know SSL protocol [12]. The SSL protocol builds on the lower-layer protocol TCP to establish a secure channel between a client and a server. SSL provides a secure two-party channel with FIFO-ordering of messages and additional security services such as confidentiality, authentication and integrity. Similarly, the SGL protocol builds on the InterGroup group communication system to establish a secure multicast channel between application components distributed over the Internet. The resulting secure multicast channel provides application programmers with the basic properties of reliable multicast communication systems in terms of group membership, and multicast security services.

The dependence of SSL on a lower-layer protocol makes it vulnerable to some attacks against TCP/IP [14] however SSL renders these attacks either harmless or limits them to denial of service. In a similar way, SGL renders attacks against the InterGroup system either harmless or limits them to denial of service.

SGL builds a secure multicast channel by first establishing a session key among the authorized application components through a group Diffie-Hellman key exchange. The group Diffie-Hellman protocols [7,8] are designed to achieve strong security requirements and to provide an authorized set of users in a multicast group with a secret session key. These protocols do not rely on a centralized key distribution center. Once a session key has been established, SGL uses it with symmetric cryptographic algorithms to achieve multicast message confidentiality and multicast data integrity.

The InterGroup group communication system supports groups with a small number of senders and a comparatively large number of receivers. The group Diffie-Hellman scheme has been designed to provide a session key to the smaller group of senders; it does not provide a session key to the receiver group. Dissemination of the key to the receivers will require an extension to the group Diffie-Hellman scheme. The new key exchange scheme will combine both the group Diffie-Hellman key exchange for the sender group and a group key distribution scheme for the receivers. After the senders have established a session key, the control nodes (i.e. sender group) will use the key distribution scheme to disseminate this key to the receiver group.

10 Conclusion and Future Work

The goal in designing the InterGroup protocols has been to provide the application services of group communication systems in a wide-area environment with a large number of participants, prone to large latencies and frequent faults, such as the Internet. To accomplish this, InterGroup takes a slightly different approach to group membership, delivery services and control information exchange.

The InterGroup protocols take a scalable approach to membership protocols, while maintaining consistent message ordering and delivery within groups. A cornerstone of the InterGroup approach is the recognition that the message order and reliability constraints of a group communication system can be met by keeping only the processes currently sending messages in the group membership. The membership protocols require only the sending processes to participate in expensive group-wide decisions.

We also step away from the traditional approach of having the sender of a message choose the delivery service. Instead, we provide more flexibility to the application by allowing each process to choose a delivery service independent of the other processes in the group. This approach allows processes that cannot meet the desired system quality of service to still participate in the group, using a weaker delivery service, and improves scalability of the system.

The InterGroup control hierarchy provides scalable mechanisms for gathering and distributing information within the group. We have shown how two types of information, distances between processes and acknowledgments, are shared in the InterGroup system. This hierarchy can be easily enhanced to exchange additional information if the need arises.

We have designed the architecture to contain four major components: a control information service, a reliability service, a delivery service, and a membership service. We have released an alpha version of the InterGroup protocols that implements the basic InterGroup mechanisms, and continue to augment and enhance the implementation. We are currently undertaking studies of the theoretical aspects of the protocols in order to better understand the consequences of our design. We are also measuring the performance of our implementation in order to investigate the performance and scalability characteristics.

References

[1] D. A. Agarwal, O. Chevassut, M.R. Thompson, and G. Tsudik. An Integrated Solution for Secure Group Communication in Wide-Area Networks. In *Proc. of 6th IEEE Symposium on Computers and Communications*, 2001.

[2] D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia. The Totem multiple-ring ordering and topology maintenance protocol. *ACM Transactions on Computer Systems*, 16(2):93–132, May 1998.

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

[4] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4):311–342, November 1995.

[5] K. Berket, R. Koch, L. E. Moser, and P. M. Melliar-Smith. Timestamp acknowledgments for determining message stability. In *Proceedings of the 2nd International Conference on Parallel and Distributed Computing and Networks*, Brisbane, Australia, December 1998.

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

[7] E. Bresson, O. Chevassut, and D. Pointcheval. Provably Authenticated Group Diffie-Hellman Key Exchange – The Dynamic Case. In *Proc. of Asiacrpt'01*, Dec 2001.

[8] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. Provably Authenticated Group Diffie-Hellman Key Exchange. In *Proc. of the 8th ACM Computer and Communications Security*, Nov 2001.

[9] V. G. Cerf and R. E. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5):647–648, May 1974.

[10] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[11] I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, Inc., 1998.

[12] A. Freier, P. Karlton, and P. Kocher. *The SSL Protocol Version 3.0*, November 1996.

[13] K. Guo and L. Rodrigues. Dynamic light-weight groups. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, pages 33–42, Baltimore, Maryland, May 1997.

[14] B. Harris and R. Hunt. TCP/IP Security Threats and Attack Methods. *Computer Communicastions, Elsevier*, 22(10):885–897, 1999.

[15] I. Keidar, J. Sussman, K. Marzullo, and D. Dolev. A client-server oriented algorithm for virtually synchronous group membership in WANs. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 356–65, Taipei, Taiwan, April 2000.

[16] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[17] D. L. Mills. Network time protocol (version 3) specification, implementation and analysis. IETF Request for Comments: 1305, March 1992.

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

[19] L. E. Moser, P. M. Melliar-Smith, R. K. Budhia D. A. Agarwal, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, April 1996.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF Request for Comments: 1889, January 1996.

[21] P. Sharma, D. Estrin, S. Floyd, and L. Zhang. Scalable session messages in SRM using self-configuration. Technical Report 98-670, USC, February 1998.

[22] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using ensemble. *Software: Practice and Experience*, 28(9):963–979, July 1998.

[23] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[24] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered protocol. In *Proceedings of the International Workshop on Theory and Practice in Distributed Systems*, pages 33–57, Dagstuhl Castle, Germany, September 1994. Springer-Verlag.