CITI Technical Report 98-2

Lightweight Secure Group Communication

Patrick McDaniel
pdmcdan@eecs.umich.edu
Peter Honeyman
honey@citi.umich.edu
Atul Prakash
aprakash@eecs.umich.edu

ABSTRACT

An advantage of today's high speed networks is the ability to support group communication. Applications that support group communication allow the free exchange of ideas and data in real time, regardless of the physical distance between the participants. Unfortunately, support for additional protocol features such as reliability, secrecy, and total ordering in the multicast context requires more bandwidth and greater complexity than in traditional point-to-point communication. In this

paper we describe a middleware software layer and associated API that attempts to minimize these requirements by providing multiple secure *channels* based on IP multicast within the same logical group. Named LSGC (lightweight secure group communication), the software provides the important features needed by a group application: reliable delivery, best-effort delivery, and security. In

providing both reliable and unreliable channels, an application need pay only for the delivery assurances it needs. We conclude with a description of our implementation and supporting performance data.

April 14, 1998

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

Lightweight Secure Group Communication

Patrick McDaniel
EECS Dept.
University of Michigan
Ann Arbor
pdmcdan@eecs.umich.edu

Peter Honeyman
Center for Information Technology Integration
University of Michigan
Ann Arbor
honey@citi.umich.edu

Atul Prakash
EECS Dept.
University of Michigan
Ann Arbor
aprakash@eecs.umich.edu

1 Introduction

An advantage of today's high speed networks is the ability to support group communication. Applications that support group communication allow the free exchange of ideas and data in real time, regardless of the physical distance between the participants. For group applications such as videoconferencing, high bandwidth requirements dictate a multicast communication model. Unfortunately, support for additional protocol features such as reliability, secrecy, and total ordering in the multicast context requires more bandwidth and greater complexity than in traditional point-to-point communication. Furthermore, these requirements increase as more protocol features are needed.

In this paper we describe a middleware software layer and associated API that attempts to minimize these requirements by providing multiple communication *channels* based on IP multicast [1] within the same logical group. Named LSGC (lightweight secure group communication), the software will be used by application developers to create high-bandwidth group applications with a minimum of protocol and infrastructure overhead.

While there are many architectures and protocols described in the literature aimed at enabling reliable, fault-tolerant, and/or secure group communication [2, 3, 4, 5, 6, 7], they incur significant costs. Integrating with these systems requires additional application infrastructure, bandwidth, or both. We attempt to avoid these costs with a single, lightweight software layer that provides the important features needed by a group application: reliable delivery and security. Moreover, applications choose which features they wish to pay for on a per message basis, avoiding unnecessary overhead.

LSGC and its associated protocols address three central goals:

1. An application using the API must view the group as a single abstraction. Applications are free to transmit messages to the same group with varying delivery semantics without changing the group context. The underlying protocols used to provide varying delivery semantics must be transparent to the application.

2. All communication within the group must be secured. Communication between the members must be confidential, integrity assured, and fresh.

3. The operation of the security mechanisms must introduce minimum additional latency and throughput limitations. Lightweight protocols and cryptographic algorithms are to be used in attempting to achieve this goal. The dynamic nature of group communication coupled with the bandwidth requirements of the target applications require that the group must adapt to changes in membership with minimal overhead. Moreover, the cryptographic algorithms employed in our system must be fast enough to support the high bandwidth requirements of target applications.

In addressing these goals, we introduce the use of multiple logical channels within the same group with varying delivery semantics. Our current prototype provides two logical channels over which application messages can be transmitted: the *reliable* channel and the *unreliable* channel. The reliable channel is used by applications to send messages that contribute to the global shared state. We use the term *critical* message to refer to this type of traffic. Using this mechanism, application developers need not address the many design issues associated with state consistency in the presence of unreliable communication. Reliability in the group context requires either implicit or explicit *consensus* about the existence and ordering of messages. Control messages are required to achieve this consensus, and thus can introduce latencies and consume bandwidth. The unreliable channel is used in those cases where a message does not need reliability, or where the overhead costs are larger than the application is willing to pay. Messages transmitted to the group on the unreliable channel are delivered as "best effort" traffic, the reliability and timeliness of which depends on the network environment.

Both channels operate under a single logical group, obviating the need for reconciliation of potentially divergent group membership. Generally, applications that require multiple delivery semantics need to create and maintain a separate group for each type of delivery. This requires the application to maintain state for each group, and to ensure synchronization the membership between groups, which introduces additional failure modes that must be handled directly by the application. In LSGC, group membership and failure modes are inclusive: a process is in groups for all channels or none, and if any channel fails, then they all fail. This is consistent with our single group abstraction: the application interacts with a single "logical" group.

The central goal in the design of our security mechanisms is performance: all security related operations must be lightweight. We have adapted the Leighton-Micali [8] key distribution algorithm to the group environment. This algorithm has several desirable properties with respect to our stated goals. First, it uses fast (symmetric key) cryptographic algorithms throughout. Second, the algorithm generates unique shared secrets between any two members. These keys are useful in quickly distributing new group session keys. The security services use the reliable channel to distribute authenticity and session key information.

We have completed our initial implementation of LSGC and ported it to several platforms. LSGC provides a single object interface to the group. Using this object, an application has the ability to communicate via either channel, view the current group membership, or retrieve security related information. We use the multicast extensions [1] of the Internet Protocol (IP) [9] as our communication primitive. As such, our implementation should be widely supported by most existing networks.

Applications with real-time requirements, such as audio and video conferencing, are well suited for LSGC. LSGC allows these applications to transmit data at the highest rates available on the unreliable channel, while maintaining state consistency through the reliable channel. In systems where all communication is required to be reliable, this solution may not provide performance improvements but may still offer value as a secure communication mechanism.

In this paper, we outline the architecture of LSGC and give a detailed description of the security mechanisms. In later sections, we discuss our experiences with the implementation and give a summary of the preliminary performance measurements. The remainder of this paper is organized as follows. Section 2 describes the LSGC design and operation. In Section 3 we analyze the performance of our implementation. In Section 4, we review the literature germane to this work. In Section 5 we assess the contributions of this research and identify directions for future work.

2 Architecture

The purpose of LSGC is to provide a useful interface that aids application developers in building high performance group applications. Applications using our system are provided facilities to send messages with differing delivery guarantees within a single logical group. These delivery guarantees are specified for each message. In many existing systems, separate independent communication channels must be created to support different guarantees. Here, variable delivery guarantees are used for enhanced performance. The use of a single group for separate types of communication, as opposed to separate groups, lowers the overhead of group maintenance. With one group, no coordination is needed to reconcile potentially divergent group membership between the channels. Moreover, the use of unreliable communication within the group lowers the costs of application traffic.

Like most group communication systems, our architecture is based on a dynamic group membership model. Through the API, a process applies for group membership. The identity of the group is specified by a multicast address. If membership is granted, the application is free to transmit and receive messages from the group. Based on the type and urgency of each message, an application determines over which channel it will be transmitted. A group member is free to exit the group at any time. All members are immediately notified when the membership changes.

Based on IP multicast, LSGC provides two channels for group communication: a reliable channel and an unreliable channel. The reliable channel guarantees that either a transmitted message is delivered or (nearly) immediate notification of failure is reported. More specifically, the delivery is guaranteed to be atomic (with respect to failures), totally ordered, and reliable: either all or none of the group members receive a message, and all messages are received by all members in the same order. The unreliable channel makes no guarantees about the delivery of the message.

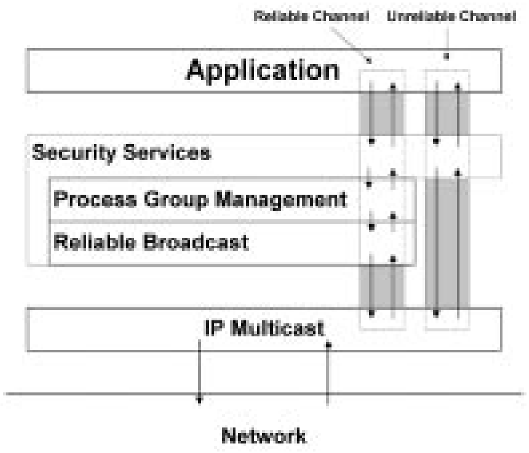

Figure 1: LSGC Protocol Layers - Critical messages are passed between the protocol layers, tagged, and encrypted. After transmitting the message, a confirmation from the group is observed and returned to the sender. Unreliable messages are encrypted and passed directly to the network interface.

Some, none, or all of the group members may receive a message. The underlying IP protocols allow us to assume that, in the absence of a malicious party, the integrity of messages is assured. In the future, we may add channels that provide other delivery semantics.

In most environments it is not desirable to broadcast group communication in the clear over the network, so LSGC secures all communication, guaranteeing the secrecy, integrity, and freshness of each transmission. A secret message has the property that it is computationally infeasible for an untrusted party to recover the plaintext without the encryption key. A receiver is able to determine that a message was sent recently, and is thus fresh. Each message is integrity protected; it is possible for the receiver to verify that a message has not been modified in transit. Although not supported in the current version, we intend to extend to these facilities to include other guarantees, such as sender authenticity.

Described in Figure 1, internally the LSGC consists of three protocol layers. Each layer provides services specific to a feature of LSGC and depends on lower layer services. The security services layer depends on membership in the group, which is established and maintained in the process group management layer. The process group management layer requires the use of the reliable delivery semantics provided by the reliable broadcast layer to correctly maintain group membership.

The security services layer provides facilities for ensuring the secrecy, integrity, and freshness of the group communication. In providing these facilities, we adapt the Leighton-Micali [8] key distribution algorithm to a group environment. Using this algorithm, LSGC periodically distributes a unique cryptographic key that is used to encrypt the group communication. A detailed description of these mechanisms and protocols is given in Section 2.1.

The goal of the process group management layer is to provide all processes with a consistent view of group membership. Knowledge of group membership drives the content and tone of a session, so its management is critical. Failure detection must also be provided, so that participants are informed when members are unexpectedly removed from the session. This layer ensures that each member is aware of the current membership and is informed in a timely manner when group membership changes.

The management of the group membership is based on a server granting scheme. A potential member attempts a join by notifying a distinct member, called the *session leader*, of its intent. After joining, the new group membership is distributed by the session leader, and reliable application traffic continues. The success of these algorithms depends on the underlying reliable broadcast mechanisms. The current membership is shared state held between the group members, and as such reliable delivery makes consistency much easier to obtain.

Security services require a member to obtain credentials to present to the group before joining. These credentials are obtained through a pre-authentication protocol, described in Section 2.1. During the join phase, the credentials are authenticated by each current member. If the credentials are found to be authentic, the new member is added to the group.

The reliable broadcast layer provides reliable, atomic, totally ordered delivery of group traffic. The messages are atomic; either all processes receive a message or none do. The messages are totally ordered; all processes receive the messages in the same order. These delivery semantics are used to ensure that all critical data are delivered to the group members correctly.

We base our design on a sequencer arbitrated commit scheme [5, 6, 10]. The sequencer is responsible for maintaining the order of messages by associating a sequence number with each application level message. This association is broadcast to the group. Ordering and loss detection is achieved by each member observing the sequence number of messages and delivering them to the application accordingly.

Membership can be potentially large, so we have selected a negative acknowledgement (**NAK**) scheme for loss recovery. When an application detects a missed message, it requests a retransmission by sending a NAK message. Positive acknowledgement, wherein all receivers acknowledge each received message, is unsuitable in our system for two reasons. First, with positive acknowledgement, the costs of a single application message grows linearly with group size (in terms of the number of messages). Second, a receiver must know the identity of all the receivers. At this protocol layer, group membership information is unavailable.

An implication of using NAKs is that the sequencer must maintain an indefinite backlog of messages for later retransmission, and this backlog can consume significant resources. We avoid this resource problem by designating a maximum number of messages that may be missed by a client. If the sequencer receives a NAK for a message outside this backlog window, it does not respond, eventually resulting in the client exiting the session.

2.1 Security Services

In this section we outline the facilities that ensure secrecy, freshness, and integrity of group communication. We adapted a two-party key distribution algorithm to the group environment to provide all authenticated members with a context specific session key. All application level communication is encrypted using this key. We conclude this section with a discussion of several implementation issues.

The central goal of the security services is to provide security for all application level traffic. Specifically, we ensure that the application level messages are confidential, integrity checked, and fresh. The confidentiality of application messages is ensured by a session key. Any party with access to this key has full access to the plaintext of all application messages. All traffic is integrity checked: all receivers will be able to determine that any message has or has not been altered in transit. The application messages are guaranteed to be fresh. Replay of messages from previous sessions are detectable by receivers.

The threat model for our system is based on the protection of the application level messages. We state the goal of an adversary is to obtain access to these messages. To this end, we assume the adversary may add, modify, or delete any message. We do not address denial of service attacks.

Each potential member, *A*, of a group (including the session leader) has a shared secret K_A registered with a trusted third party called the key distribution center (*KDC*). This secret key is generated and registered with the *KDC* before the party attempts to join any session. We assume an out of band method for establishing key registration. The strength of the security is based on the protection of these keys. We assume the application has some secure way of obtaining the keying information; pass phrases are normally used for this purpose.

Before describing our solution, we state our trust in the member applications. We assume that all processes that have achieved membership, and thus have been authenticated, adhere to the system specification. We assume no member willingfully discloses its long term or session keys. All members trust the *KDC* not to disclose their long term key, and to generate pair keys according to the specification.

Group communication systems found in the literature provide varying levels of support for secure communication. The simplest form of secure communication can be found in ISIS [5], where group communication is encrypted via a session key. At the other end of the spectrum, RAMPART [4] uses a highly sophisticated, fault-tolerant framework to enable secure group communication in the presence of actively malicious processes. Our goals of secure and high performance multiparty communications leads us to less ambitious security goals. We define a protocol for session key distribution and message encryption whose primary goal is high performance: key distribution and encryption must be as lightweight as possible.

We achieve these performance goals by leveraging on the broadcast and group membership semantics provided by the underlying protocols. Membership is used implicitly to state authenticity, and session keys are distributed on this basis. Before being allowed to join, each prospective member is pre-authenticated. During authentication, a potential member is given a context sensitive *authentication token*, which is presented to the group during the join phase. Each token is specific to both the joining process and the current group membership *view*. [1] The token can be validated by each existing member without obtaining additional information.

All messages are encrypted with a session key specific to each view of group membership. As each new view is installed, a new session key is generated and distributed. We use separate session keys for each view, for two reasons. First, we do not want to grant recently joined members access to previous communication. Second, this reduces the amount of time a session key is in use. If a session key is compromised, it will disclose only the session traffic generated during the duration of the view. Possession of the session key for one view provides no information with which to infer the key for the next view. Because of our lightweight mechanism, the impact of the potentially large number of key distributions is minimal.

We chose the Leighton-Micali key distribution algorithm [8] to authenticate and transmit the authentication token to the joining member. The main advantage of Leighton-Micali is low cost: it uses symmetric key encryption throughout, with none of the modular exponentiation operations associated with public key cryptosystems. Public key cryptography requires significantly more computation than symmetric algorithms. The de-facto standard for public-key cryptography, RSA, can be up to 100 times slower in software and 1000 times slower in hardware than DES, the predominant symmetric algorithm [11].

Before describing the protocols, we define the notation used in this section. We use the term *SL* to refer to the identity of the session leader, A to refer to a potential member of the session, and *KDC* to refer to the key distribution center. $\{X\}_k$ refers to message *X* encrypted under the key *k*. The view identifier, *g*, is used to uniquely tag the changing views of group membership. The term SK_g refers to a session key in view *g*, and SK_{g+1} for the (next) view $g + 1$. The term *I*, possibly with a subscript, refers to a nonce value. Key distribution protocols based on Leighton-Micali define a term $\pi_{A,B}$, called a *pair key*, used to support secret communication between collaborating members *A* and *B*. Derived from the pair key, the session leader and a potential member *A* maintain a shared secret key $\sigma_{SL,A}$. A MD5 hash [12] for the text x is described by H_x.

It is important to note the format of an identity, nonce, key value, and view identifier. Each identity is a unique 16 byte null terminated ASCII string of alphanumeric characters. A potential member is assigned this value when registering a long term key with the *KDC*. Nonce values are unique 64 bit values. To ensure nonces are not reused, some source of monotonic values, such as the system clock, may be used. Key format is algorithm dependent. The DES standard uses an eight byte key (including eight parity bits). In the future, as other ciphers are integrated into LSGC, we will need to support other formats. The view identifier g consists of a two parts. The first part is an eight byte null terminated group name string that identifies the group, used only for displaying and debugging purposes. The second is an eight bit nonce value. Each time a new view is created ($g + 1$), a new nonce in generated and appended to the group name string to create the new view identifier.

A session is started by the initial member, called the session leader. The session leader initializes the session by generating a session key and initiating the process group management protocol. The session leader serves as the arbiter of group membership for that session. We use DES [13] for all encryption in the system; its inherent strength is evident from its 20-year history, yet its 56-bit key length has long been the subject of debate. Related algorithms such as triple-DES [14] or DESX [15] offer the strength of DES with considerably longer keys.

A prospective member initiates a session join by sending a message to the session leader containing her identity and a nonce value. The session leader then obtains the pair key $\pi_{A,B}$ from the *KDC*. Derived from two identities and their associated long term keys, the pair key is used to establish an ephemeral secure channel between the processes. Messages 1-3 in Figure 2 describe the exchange between the prospective member, the session leader, and the certification authority.

To prevent replay attacks, the session leader verifies the encrypted nonce value I_1 included in the *KDC*'s response.

[1] A group *view* is the set of identities associated with the members of the group during a period where no changes in membership occur. If the membership changes (a member joins or leaves the group), then a new view is created.

Join Protocol
1. $A \rightarrow SL : A, I_0$
2. $SL \rightarrow KDC : SL, A, I_1$
3. $KDC \rightarrow SL : [\pi_{SL,A} = \{A\}_{K_{SL}} \oplus \{SL\}_{K_A}], \{I_1\}_{K_{SL}}$
4. $SL \rightarrow A : SL, \{I_0, g, \{g, A\}_{SK_g}\}_{\sigma_{SL,A}}$
5. $A \rightarrow group : A, \{g, A\}_{SK_g}$
6. $SL \rightarrow group : g, (A, \{g+1, SK_{g+1}\}_{\sigma_{SL,A}}), (B, \{g+1, SK_{g+1}\}_{\sigma_{SL,B}}), ...$

Leave Protocol

The session leader generates and transmits message 6.

Application Messages

7. $[groupmember] \rightarrow group : g, \{A, I_2, [message], H_{[message]}\}_{SK_g}$

Figure 2: Protocol description.

The session leader generates the value $\{A\}_{K_{SL}}$. This value is XOR-ed with the pair key $\pi_{SL,A}$ received from the *KDC*. The resulting value is a shared secret key ($\{SL\}_{K_A} = \sigma_{SL,A}$) that is used to create a secure channel between the session leader and the prospective member *A*.

The session leader creates and returns a response message containing a view identifier, an authentication token, and a nonce value encrypted with the shared secret key $\sigma_{SL,A}$. The authentication token contains the view identifier and requesting process identity, encrypted under the session key for that view. Upon receiving this message, the receiver decrypts the contents and verifies the nonce. Mutual authentication is achieved through the verification of the secrets *A* and *SL* share with the *KDC*. The potential member must be in possession of the secret shared with the *KDC* to decrypt the key distribution message. The session leader must be in possession of the secret shared with the *KDC* to determine the secret key shared with *A*. *A* is convinced the key is fresh by validating the nonce value sent in the original request. Message 4 describes the authentication response message.

Note that *A* need not communicate with the *KDC* to obtain the shared secret key $\{SL\}_{K_A} = \sigma_{SL,A}$; she can compute it directly. *A* decrypts the session key with this value and validates her nonce.

When a member tries to join the session, the received authentication token is presented to the group (message 5). All current group members possess the session key, and can validate the contents of the token. After decrypting the token, each group member validates the contents by verifying that the group identifier matches the current group and the joining member identity is consistent with the join request. If the token contents are not valid, the join request is ignored, and the group continues. If the token is valid, the new member is allowed to join the group and a new view and session key is installed. An adversary may intercept the token and attempt to join with it. In this case, the join identity and the one inside the token will not match, and be ignored. If an adversary attempts to impersonate the token owner, he will be able to join the session but unable to obtain the session key for the new view (*see session key distribution below*).

After a joining member presents an authentic token to the group (which includes the session leader), the session leader broadcasts the message containing the new session key and new view identifier. In the session key distribution message the new session key is encrypted under the shared secret key shared with each member. This is similar to the key distribution used during a LEAVE operation in the Iolus system [16]. The session leader caches the shared secret keys, so creating this message is fast: encryption of 24 bytes (8 bytes of new session key plus 16 bytes of new group identifier) per member. Using the cached, shared secret key, the receivers of this message extract the session key out of the message and begin using it immediately. Message 6 describes the session key distribution message. The size of this message grows linearly with group size, and is potentially large. But keying material needs to be sent in any solution, so the size of the message is large by its nature, not as a side effect of our design. Schemes that distribute a key to each member individually will transmit the same amount of data over many more messages. The Iolus system [16] mitigates the need for a large key distribution message by introducing subgroups. The subgroups form a hierarchy of groups to which the keying material is distributed. The keys cascade from supergroup to subgroup, thus reducing the size of any one distribution message. Should it be deemed necessary in the future, we may add support for a similar mechanism.

All communication in the session is encrypted with the session key and tagged with the group view identifier. Included in each application message is the sender's identity and a nonce value. Associated with each group member is a monotonically increasing counter (I_2) that is reset to 1 as a new group membership view is installed. This counter, represented by the nonce value in the message combined with the session key ensures that the message has not be replayed from earlier in the session. To ensure integrity, an MD5 hash of the application data, sender identity, and nonce value is concatenated with the message, and verified upon reception. Message 7, where $[message]$ is the message content, describes its format.

A potential problem occurs during the transition of group views. During the join process, application data such as continuous media may continue to be broadcast over the unreliable channel. Because of delays in the delivery of the session key across the reliable channel, a process may receive an unreliable message encrypted with a session key that it does not yet or will never possess. In the following paragraphs we outline several solutions to this problem.

This session key transition problem is difficult to address with a single general solution. In applications with low throughput characteristics, it may be reasonable to buffer data until the new key arrives. High performance applications, such as videoconferencing, have much higher throughput requirements and thus can not buffer data for long. This is a reasonable solution only if the delay is likely to be very short. Another potential solution is to transmit with both keys for some period after a group change, but this potentially doubles the bandwidth use and increases latency due to the processing of unneeded packets [2].

Another alternative is to establish a waiting period during which all transmitted data is encrypted using the old session key. Here, a newly joined member is unable to process unreliable traffic until the waiting period is over. A potential security risk is that processes that leave the group are able to continue decrypting the unreliable traffic during the wait period.

The target applications generate large amounts of continuous data. If buffering is used, the vast amount of data arriving at the client will likely overflow available buffer space. Because the data is continuous, arbitrary delays in the stream may degrade the service provided by the application. For the similar performance reasons, we are reluctant to introduce double encryption into the system.

After weighing these factors, we have chosen to drop any incoming message for which we do not currently possess the key. In local area networks these delays are short, yielding little data loss during the transition. If in the future we find the costs of the outages outweigh the costs in performance and/or buffer space, and we will reevaluate this decision.

Thought not implemented in our the initial version of LSGC, we note the need to secure the messages of the underlying process group management and reliable broadcast protocols using the session key. In the absence or such protections, an adversary may be able to mount attacks against the group. By intercepting and modifying the messages of the underlying protocols, an adversary may be to block notification of a group membership change and resulting session key distribution. In this way, and adversary is able to get a sender to continue sending application messages using an old session key. Using similar techniques, an adversary will be able to reorder application messages within the group. We are currently integrating cryptographic protection of the underlying protocol messages into LSGC.

3 Performance

We have completed the implementation of LSGC as described in the previous sections, with the exception of the MD5 hash used for secure communication. All code and test applications have been ported to Microsoft Windows95, HP-UX version 10.20, Solaris 2.5.1, and Linux 2.0.18. Using fault insertion, unit, and soak tests, the stability of the system has been established. Our implementation is in prototype form and little attention has been paid to wringing the best performance from it, so the performance reported in this section is preliminary, but provides indicators of the power of our design. Where applicable, we point out where the deficiencies of the current implementation lie and propose potential enhancements.

[2] It is possible to mitigate the costs of encrypting with both keys. A solution would be to generate a random one time key and encrypting it under both the new and old session keys. The message itself would be encrypted using the random key. The ciphertext of the random key encrypted with each session key, plus the encrypted message would be broadcast to the group. A receiver would then reverse the process with an available session key.

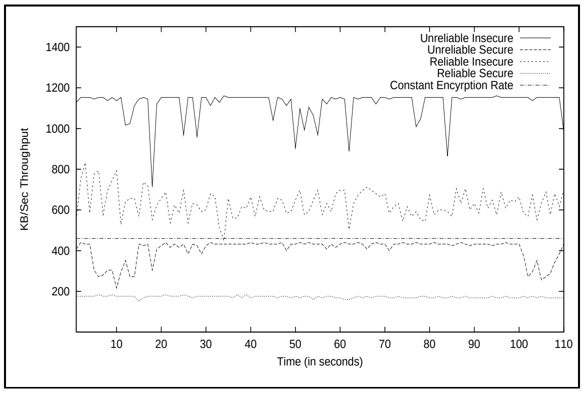

Figure 3: Throughput: per second measurements of throughput taken from single sender, three receiver test. Note that the reliable traffic is limited by a sender window of 1 in out current implementation. Higher throughputs are feasible with larger windows.

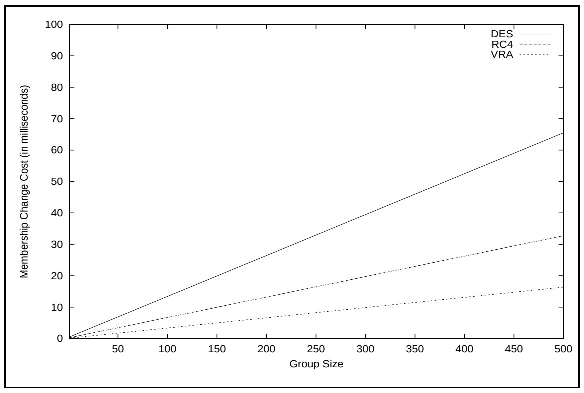

Figure 4: Join/Leave Rate: as calculated from collected throughput statistics. This plot shows the time needed in milliseconds for a single join/leave and session key distribution .

Performance was measured using two test applications that perform client and server functions. The server application creates the sessions, which one or more clients join. The tests were run on Sun Microsystems UltraSparc Model 170E (167MHz) with 128MB memory and 4.0GB hard drives over a 10Mbit Ethernet network. The protocol is self regulating, so the sender transmits reliable traffic as fast as the protocol allows. Unreliable traffic is paced by the sender at the shortest interval that does not incur significant packet loss. Each application message is 8KB. We use a single sender and measurements are taken at all receiver hosts.

Our first series of tests determines whether the unreliable channel performs significantly better than the reliable channel. Intuitively, we expect that the delays caused by the atomic broadcast mechanism reduce system throughput. In our implementation, a sender may have only one uncommitted message outstanding, thus limiting transmission. Performance may be improved by augmenting the protocol to include a sender window. In this way, a sender can have more than one outstanding packet at a time, increasing the total throughput. Unfortunately, this enhancement may require the introduction of a group flow control mechanism. For this work, we assume a small amount of critical data needing reliable delivery semantics, so the enhancement is deemed unnecessary for our initial tests.

In Figure 3, we show measurements of throughput for the several delivery/security communication models. These tests measure a session containing a server and three clients. Four tests lasting 110 seconds each were executed. We show the end to end (application to application) throughput for each test. Each mode of communication was tested (reliable secure, unreliable secure, reliable insecure, and unreliable insecure) to determine the effects of both reliability and security on system throughput.

We point out several features of the throughput described in Figure 3. The most significant, if not surprising, fact is that the unreliable channel has significantly higher throughput. We attribute this to several factors. First, the throughput of the reliable channel is limited by the sender window of 1. Missed packets must be NAKed, requiring additional multicast traffic processing by all hosts, which reduces the total amount of time in which the application data may be processed. The unreliable insecure traffic has very little delay. The test application is able to saturate the 10 Mbps Ethernet using the insecure unreliable delivery.

Another observation is that secure channels have much lower throughput than insecure channels. For both reliable and unreliable traffic, the throughput of secure communication is roughly $\frac{1}{3}$ that of insecure communication. We attribute this largely to the encryption and decryption costs. In our prototype, DES encryption occurs in the critical path of message transmission. This causes the transmission of each packet to be delayed by the amount of time it takes to encrypt the buffer. Modifications may be made to remove this latency by pre-encrypting each message while a previous message is being transmitted. Faster encryption schemes may also be used to decrease this cost. For example, the CITI secure videoconferencing system [17] uses two fast ciphers, RC4 and VRA, which have been found to be four and eight times faster than DES, respectively, when used as stream ciphers. In integrating our work with the CITI

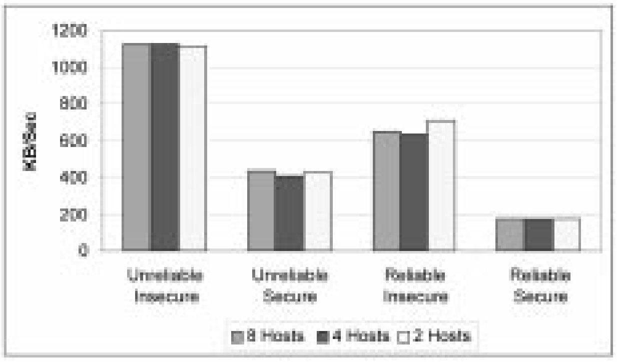

Figure 5: Throughput : as measured with differing group sizes. Throughput is largely independent of group size.

system, we will add support for these ciphers.

A final observation is that while the throughput of the secure traffic remains essentially constant, the insecure traffic shows great variability. Brief periods of change in throughput are due to environmental issues. The unreliable insecure traffic saturates the network, so any traffic from other hosts affects our throughput. These tests were taken on a large network, so usage spikes affect our throughput tests. With reliable insecure traffic we see a much larger variance. This can be attributed to the reliable transport mechanism. Packet loss in our environment is due to receiver overrun. Each lost packet causes one or more hosts to send a NAK. When the server sees any NAK, it responds with the NAK response message, which is slightly larger than the original message. In this way, the buffer overrun problem can be exacerbated by the NAK/response. Hosts may be able to avoid the multiple NAK problem by monitoring the channel for NAKs, sending them only when there is not one outstanding. This is essentially a flow control problem, which we relegate to the application. We do not see this problem with reliable secure communication because of limited throughput: encryption of traffic is slow, so receiver buffers are not overrun.

As described in section 2.1, a group membership change triggers the distribution of a new session key. The cost of key distribution is a function of group size and the speed of the cryptographic algorithm. In Figure 4, we use the throughput characteristics found in the previous tests to estimate the best case costs of join/leave processing supported by LSGC. Note that these numbers are best case: as groups grow larger, the possibility of retransmissions increases. Retransmissions lengthen the membership change processing time. We show this performance with our current implementation (DES), and two other cryptographic algorithms we intend to integrate with LSGC (RC4 and VRA). Note that for group sizes of less than 200, LSGC can process 50 or more membership changes per second, far more than is likely to occur. When LSGC is supporting larger groups (up to 350 or so members), more than 20 membership changes can be processed each second. Very large groups may overwhelm LSGC with membership changes. If support for groups containing 300 or more members is required, the key distribution costs can be reduced by a hierarchy of session leaders.

The next series of tests determines the effects of group size on throughput. We repeated the previous test with group size of two, four, and eight hosts. Figure 5 shows the average throughput for each of the tests in kilobytes per second. For these group sizes the performance is largely constant, which is what we would expect to see. Multiple receivers receive the data in exactly the same manner, regardless of the group size. This is the advantage of multicast. We attribute the small variance reported throughput between the group sizes to environmental issues. Repeated tests of the same type indicate differences comparable to those reported in Figure 5.

One possible result not shown in this figure is that the number of NAKs as a percentage the total traffic increases with group size. At some point, the overhead of NAK processing affects total throughput. This problem is known as *acknowledgement implosion*. Paul *et al.* [18] define a protocol in which a tree of multicast subgroups is connected. In each subgroup, a distinct member, known as the *primary receiver* performs acknowledgements for the group as a whole to a parent subgroup in the tree. Acknowledgements and retransmissions are localized within the subgroup, reducing total protocol overhead. We may elect to augment our work in this way in the future.

4 Related Work

Much of the existing technology on which reliable group communication is based was originally implemented in the ISIS [5] and later HORUS [3] group communication systems. These systems provide a framework in which group applications with reliable delivery requirements can be developed. Using these frameworks, developers can experiment with other features, such as secure communication. One important feature of the HORUS system is the introduction of a comprehensive security architecture. A key element to this architecture is the fault-tolerant key distribution scheme. Process group semantics are used to facilitate secure communication. As group membership changes, new session keys are distributed. The key distribution mechanism can tolerate, up to a point, the loss of the global timing source, which is used to ensure the freshness of the keying material. Public key certificates are used to authenticate message traffic, which are retrieved from a certification authority. The costs of using this architecture for group communication is large. The authors show that a group of 4 members can process $<$ 100kb/second [3], which is far below the requirements for a high bandwidth system such as videoconferencing.

The RAMPART system [4] provides secure communication in the presence of actively malicious processes. The system uses secure channels between two members of the protocol to provide a maximum of security and authenticity. Protocols depend greatly on the *consensus* of processes to reach agreement on the course of action. The costs of the strong guarantees is very high: the authors state the latencies can exceed those of other published protocols by an order of magnitude.

A limitation of many secure group communication systems is that they do not scale to larger networks. The Iolus system [16], addresses this limitation. Scalability in Iolus is achieved by limiting the effects of membership changes to locally maintained subgroups. Each subgroup maintains its own session key, replaced after each membership change. While this approach is effective, little description about the delivery semantics is given. In the future, if scalability becomes a concern, we may choose to integrate these methods into LSGC.

Virtual networks provide developers an abstraction to build applications designed for (logically) local network traffic, but executed across across larger networks. The Enclaves system [2] extends this model to include secure group communication. Group communication in the Enclaves system is controlled at a finer grain, where group membership is controlled not only by authentication, but by policies defined by a group definition. A distinct member authenticates a process wishing to join the group, and validates the new membership using the policy definition. Authentication tokens are passed to an admitted member and presented to other group members as needed.

In the systems mentioned above, we see that a wide range of security threats and fault tolerance requirements can be addressed by applying well known techniques. Yet, fulfilling these requirements comes at the expense of increasing costs. Though appropriate for situations where bandwidth and latency requirements are marginal, these solutions are likely to fail to meet the needs of applications with high performance requirements. This motivates our design, where the requirements for delivery semantics and performance differs between the types of traffic within a single application.

5 Conclusions and Future Work

This paper has presented a design and preliminary performance measurements of middleware for secure, high performance multicast applications. We see two central contributions of this work. The first is the use of multiple delivery semantics within a single group. Messages with reliability requirements are sent through the reliable channel, and those without are sent unreliably. Using these channels, applications pay only for the delivery guarantees commensurate with their needs. The second contribution is the definition of a lightweight pre-authentication mechanism for the group environment. Each potential member is authenticated by a server, and given an *authentication token*. This token is then presented to the group when a join is attempted. Current group members can validate this token without any external information. After entering the group, session key distribution to all members requires only one message. In this design, we have succeeded in creating a lightweight security mechanism by reducing the total number of messages and using symmetric cryptography throughout.

Our initial intuition about the performance characteristics of the system appears valid. We have shown that there is much to be gained through the use of multiple logical channels, and that we can provide a secure solution with minimal overhead. Our tests indicate that throughput is largely unaffected by group size.

The initial implementation is complete. Groups can establish reliable and unreliable communication, the contents of which can be securely transmitted as required. Key distribution, client, and server applications have been developed to further test and optimize the implementation. However, the maturity of the system is not at the point at which we can establish the true throughput and latency. We are encouraged by the initial tests, and have identified several ways in which the implementation can be optimized.

We are integrating LSGC with the CITI secure videoconferencing system. We intend to show the viability of our approach in completing this integration. It seems important to benchmark LSGC against other applications with similar goals (HORUS, RAMPART, *etc.*), and learn from the designs of these systems, so we plan to make these comparisons.

Other work includes the development of additional mechanisms for reliable broadcasting. Inclusion of causal, FIFO, and similar mechanisms can increase the usefulness of LSGC. We will extend the multiple channel model to include these delivery guarantees within the same group. In addition, we seek to identify other target applications to integrate with LSGC.

We also see the need to broaden the range of security services provided. Inclusion of sender authenticity and non-repudiability may prove to be valuable. Possible threat models applicable to group systems are innumerable, and general solutions will prove valuable.

Finally, there may be a need for greater fault tolerance. Reiter points out that distributed systems have much to gain from group based communication [4]. The solutions presented in this paper have a single point of failure, and thus provide limited availability assurances. The application of the techniques presented in this paper may enhance the performance of these services, but without stronger fault tolerance may not be applicable to all environments.

References

[1] S. Deering. Host Extensions for IP Multicasting. Technical Report RFC1112, Internet Engineering Task Force, August 1989.

[2] L. Gong. Enclaves: Enabling Secure Collaboration over the Internet. In *Proceedings of 6th USENIX UNIX Security Symposium*, pages 149–159. USENIX Association, July 1996.

[3] R. Van Renesse, K. Birman, and S. Maffeis. Horus: A Flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, April 1996.

[4] M. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *Proceedings of 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, November 1994.

[5] K. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[6] J. Chang and N.F. Maxemchuk. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):252–273, August 1984.

[7] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, , and C. A. Lingley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, April 1996.

[8] T. Leighton and S. Micali. Secret-key Agreement without Public-Key Cryptography. In *Advances in Cryptology: Proceedings of Crypto 93*, 1994.

[9] Information Sciences Institute. Internet Protocol. Technical Report RFC791, Internet Engineering Task Force, September 1981.

[10] M. Kaashoek and A. Tanenbaum. Group communication in the Amoeba Distributed Operating System. In *Proceedings of the 11th International Conference on Distributed Computing Systems*, pages 222–230. IEEE, May 1991.

[11] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., New York, Chichester, Brisbane, Toronto, Singapore, second edition, 1996.

[12] R. Rivest. The MD5 Message Digest Algorithm. Technical Report RFC1321, Internet Engineering Task Force, April 1992.

[13] Federal Information Processing Standards Publication. Data Encryption Standard. Technical report, National Bureau of Standards, 1977.

[14] American Bankers Association. American National Standard for Financial Institution Key Management. Technical Report ANSI X.917, ANSI, 1985.

[15] J. Kilian and P. Rogaway. How to Protect DES Against Exhaustive Key Search. In *Proceedings of Crypto '96*, August 1996.

[16] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *Proceedings of ACM SIGCOMM '97*. ACM, September 1997.

[17] P. Honeyman, A. Adamson, K. Coffman, J. Janakiraman, R. Jerdonek, and J. Rees. Secure Videoconferencing. In *Proceedings of the Seventh USENIX Security Symposium*, pages 123–130. USENIX Association, January 1998.

[18] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.