Offering Data Confidentiality for Multimedia Overlay Multicast: Design and Analysis

WAI-PUN KEN YIU and SHUENG-HAN GARY CHAN
Hong Kong University of Science and Technology

Application layer multicast (ALM) has been proposed to overcome current limitations in IP multicast for large-group multimedia communication. We address offering data confidentiality tailored for ALM. To achieve confidentiality, a node may need to continuously *re-encrypt* packets before forwarding them downstream. Furthermore, keys have to be changed whenever there is a membership change, leading to *rekey* processing overhead at the nodes. For a large and dynamic group, these reencryption and rekeying operations incur high processing overhead at the nodes. We propose and analyze a scalable scheme called Secure Overlay Multicast (SOM) which clusters ALM peers so as to localize rekeying within a cluster and to limit re-encryption at cluster boundaries, thereby minimizing the total nodal processing overhead. We describe the operations of SOM and compare its nodal processing overhead with two other basic approaches, namely, host-to-host encryption and whole group encryption. We also present a simplified analytic model for SOM and show that there exists an optimal cluster size to minimize the total nodal processing overhead. By comparing with a recently proposed ALM scheme (DT protocol), SOM achieves a substantial reduction in nodal processing overhead with similar network performance in terms of network stress and delay.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Algorithms, Experimentation, Performance, Security

Additional Key Words and Phrases: Key management, multicast security, overlay multicast, performance analysis

ACM Reference Format:

Yiu, W.-P. K. and Chan, S.-H. G. 2008. Offering data confidentiality for multimedia overlay multicast: Design and analysis. ACM Trans. Multimedia Comput. Commun. Appl. 5, 2, Article 13 (November 2008), 23 pages. DOI = 10.1145/1413862.1413866 http://doi.acm.org/10.1145/1413862.1413866

13

1. INTRODUCTION

With the fast penetration of broadband Internet access, networked multimedia applications such as real-time stock quote systems, Internet radio, Internet TV, multi-party video conferencing, etc. are gaining popularity. A scalable way to provide these multimedia services is IP multicast. Unfortunately, global IP multicast still faces many technical and deployment challenges nowadays. To tackle this problem, application layer multicast (ALM) (or overlay multicast) has been proposed. The basic idea of ALM is to form overlay multicast tree among end-hosts. Multicasting is then achieved by transmitting

This work was supported in part by the Hong Kong Innovation and Technology Commission (GHP/045/05) and the Hong Kong Research Grant Council (611107).
Authors' address: Department of Computer Science and Engineering, Hong Kong University of Science and Technology.

data from one peer to another along the tree edges using unicast connections. Therefore, ALM does not require multicast-capable routers and speeds up the deployment of large-scale multicast-based services. There have been many ALM protocols proposed in recent years, such as Narada, NICE, DT, etc. [Chu et al. 2002; Liebeherr et al. 2002; Banerjee et al. 2002]. In many multicast applications (such as stock quotes, news, TV, etc.), the sender would like to encrypt its data so that only authorized and paid subscribers are able to decrypt it. Therefore, supporting a secure communication channel among peers is essential to those applications.

Despite many ALM protocols having been proposed previously, their main concerns being connectivity, failure recovery, scalability, transmission efficiency, etc., not many of them address the multicast security issues in ALM. A secure multicast system generally offers *data confidentiality*, *authentication*, *integrity*, etc [Chan and Chan 2003]. In this paper, we only focus on providing *data confidentiality* in ALM, other security issues are out of the scope of this article. In our system, group members are authenticated and authorized before joining the system. Also, the system trusts all authenticated users that they do not leak any multicast data. For simplicity, we do not distinguish between joining and admission, and leaving and ejection, in our system.

Data confidentiality is one of the most challenging problems in secure multicast. To achieve this, a secure multicast scheme must address key management issues, which include efficient organization and distribution of keys with low communication overheads, key storage cost, and scheme complexity. The general approach to provide data confidentiality in group communication is to encrypt data with a secret cryptographic key K (i.e., the so-called group key). In such a secure multicast system, only the sender and multicast group members (i.e., the receivers) know K, and hence only the group members are able to correctly decrypt the ciphertext. Without knowing K, one would not be able to decrypt the multicast data. Since the group members are authenticated and authorized before joining the system, data confidentiality is addressed if K can be distributed securely and efficiently to all authenticated group members. The process of updating the cryptographic keys and distributing them to the group members is called *rekeying* operation. A secure system should offer the following two properties:

Backward secrecy. A new member should not be able to decrypt any multicast data sent *before* its joining; otherwise, he may be able to store the past data and decrypt it after its joining. In this case, K needs to be changed for every join event.

Forward secrecy. A former member should not be able to decrypt any multicast data sent *after* its leaving the group. In this case, K needs to be changed for every leave event.[1]

Therefore, the data encryption key has to be changed (i.e., keyed) for each membership change, and the corresponding decryption key has to be made known to all the current members. As a multicast group may be very large and highly dynamic, the rekeying mechanism should be scalable in terms of computation requirement, storage requirement and bandwidth consumption.

To offer data confidentiality, one may think of two straightforward basic approaches:

Host-to-host encryption. Each overlay connection on the data delivery tree shares a unique data encryption key (i.e., setting up secure point-to-point connections between neighboring nodes). In forwarding packets from one host to another, each host has to first decrypt the packets received from its parent, and then reencrypt the packets for each of its children using the corresponding encryption key of the connection. Clearly, a node needs to continuously decrypt and reencrypt packets. This leads to continuous *decryption/reencryption processing overhead* which depends on packet arrival rate.

[1]One may argue that data security in ALM can be achieved if an upstream node stops sending data to a leaving node. However, this approach cannot totally prevent access to the data by the leaving node since the node can still perform network sniffing in a public network.

Whole group encryption. All group members share a universal group key, and hence decryption/reencryption processing is not needed between peers. In this case, whenever there is a membership change, a new group key is generated which has to be made known to all the members. Such rekey messages have to be processed by all the peers in the network so as to agree on a common new group key. This leads to *rekey processing overhead* depending on how often group membership changes.

We see from above that host-to-host encryption leads to high decryption/reencryption overhead for high data (packet) rate while whole group encryption leads to high rekey messaging overhead for dynamic group. Therefore, given a certain data (packet) rate and group dynamics, either one of the approaches would not perform satisfactorily in terms of processing overhead. As we will show later in this paper via analysis and simulation, a more efficient way is to use a hybrid scheme where the group members are divided into clusters. Whole group encryption is used within each cluster while host-to-host encryption is used between clusters. This effectively strikes a balance between the two processing overheads, thereby achieving low overhead at each node. We term this system Secure Overlay Multicast (SOM). SOM provides a simple and yet efficient way to offer data confidentiality.

We highlight the contributions of this paper as follows:

—We propose a framework called Secure Overlay Multicast (SOM) to provide data confidentiality for application layer multicast. Our scheme guarantees both forward and backward secrecy during member join and leave. We also provide solution for managing member clusters dynamically so as to provide good performance in the system.

—We introduce two basic approaches, namely, host-to-host encryption and whole group encryption. We also compare their performance with SOM.

—We analyze the storage requirements, message and computation overhead of the system, and compare with other related schemes.

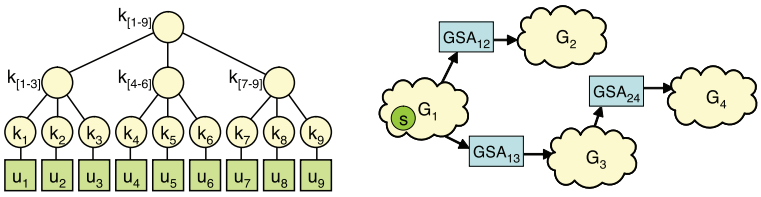
—We present an analytic model for SOM in order to analyze its performance using queueing theory. Our model provides insights on the system performance and shows that there exists an optimal cluster size to minimize nodal processing overhead. With our formula on optimal cluster size, our system could be configured to achieve the minimal processing overhead.

—We perform simulation of the system and show that our simulation results fit well with our analytic model. Our results also show that SOM substantially cuts the processing overhead of the system (by many factors) without compromising network performance (in terms of physical link stress and relative delay penalty).

This article is organized as follows. In Section 2, we briefly review the related work in secure multicast. Next, we describe in detail in Section 3 the two basic approaches and how they are related to our work. Then, we present the framework of SOM in Section 4. In Section 5, we analyze the performance of SOM. In Section 6, we work out the optimal cluster size using queueing theory. We present our simulation results in Section 7, followed by a conclusion in Section 8.

2. RELATED WORK

Traditional multicast protocols such as Distance Vector Multicast Routing Protocol (DVMRP), Core Based Tree (CBT) and Protocol Independent Multicast–Dense Mode (PIM-DM) require the use of multicast-capable routers, making global multicast a challenge [Waitzman et al. 1988; Ballardie et al. 1993; Deering et al. 1994]. To address this problem, application layer multicast (ALM) (or overlay multicast) has been proposed, which moves the multicast functionalities to the application layer of end hosts. Many ALM protocols have been proposed in recent years, [Chu et al. 2002; Liebeherr et al. 2002; Mathy et al. 2004; Zhu et al. 2004; Ganjam and Zhang 2004; Sripanidkulchai et al. 2004; Yiu et al. 2006].



(a) A key tree with degree 3 (k= 3).

(b) Iolus architecture: s denotes the data source and the arrows represent the flow of data in the system.

Fig. 1. Key management approaches used in IP-multicast networks.

However, their main concerns are connectivity, failure recovery, scalability, efficiency, etc. For security issues, for example, Badishi et al. [2006] address the problem of denial of service (DoS) attacks in gossip-based multicast. In this article, we consider another important security issues—data confidentiality.

Secure group communication in IP multicast has been extensively studied [Wong et al. 2000; Mittra 1997; Chan and Chan 2003; 2002; Chang et al. 1999; Shields and Garcia-Luna-Aceves 1999; Canetti et al. 1999; Huang and Mishra 2003; Kim et al. 2004; Lee et al. 2006]. Though those schemes can be directly applied in ALM environment, the multicast-based rekeying operations generate much traffic in the network, which may be inefficient in ALM. There are two main approaches to manage the group key, namely, logical key hierarchy (LKH) and subgrouping, elaborated as follows.

In LKH, keys are logically organized into a tree structure [Wong et al. 2000]. Figure 1(a) shows a key tree with degree 3. The users are divided into three subgroups, $\{u_1, u_2, u_3\}$, $\{u_4, u_5, u_6\}$, and $\{u_7, u_8, u_9\}$. Each user $u_i, i \in [1..9]$ is given three keys, namely, its individual key, a key for its subgroup, and a key for the entire group. In the figure, individual keys are represented by $k_i, i \in [1..9]$, subgroup keys by $k_{[1-3]}$, $k_{[4-6]}$, $k_{[7-9]}$, and whole group key by $k_{[1-9]}$. Suppose u_9 leaves the group, the remaining eight members form a new secure group and require a new group key for backward secrecy. Also, u_7 and u_8 forms a new subgroup and require a new subgroup key. To update the new subgroup key, the key server sends the new subgroup key to u_7 (u_8) encrypted by k_7 (k_8). After that, the key server can securely update the new group key by sending each subgroup the new group key encrypted by the corresponding subgroup keys. This rekeying operation for member departure costs in total five encrypted rekey messages (three of them are multicast messages). As a general case, with a balanced key tree of degree d, the total number of rekey messages is approximately $d \log_d N$ where N is the group size. Joining is less costly in LKH. Suppose u_9 joins the group again later. After the key server compromising the individual key k'_9 with u_9, the key server sends u_9 the new subgroup key $k'_{[7-9]}$ and the new group key $k'_{[1-9]}$ encrypted by k'_9. The key server also securely delivers the new subgroup key to u_7 and u_8 by encrypting it using the old subgroup key. Finally, the key server securely distributes the new group key by encrypting the new group key using the old group key. This rekeying operation costs only four rekey messages (two of them are multicast messages). In general, since only one key is required to update at each level of the key tree and each update costs two rekey messages,[2] the total number of rekey messages is $2 \log_d N$.

LKH achieves sublinear group rekeying overhead for a membership change in the group. However, LKH cannot be directly applied in ALM, mainly due to the absence of multicast-capable network and

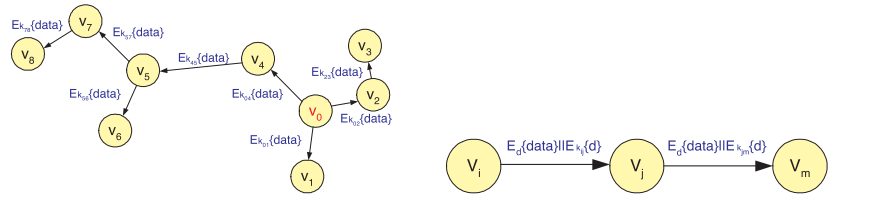
[2]In order to provide forward secrecy, all the keys in the path from the new user to the root of the key tree are required to be updated. There are two re-key messages for updating each key node: one update is for the original subgroup members of the key node, another one is for the new member.

the fundamental difference in overhead accounting. LKH assumes a multicast-capable network, and hence sending one rekey message to all members accounts for only one messaging overhead. Therefore, the rekey message overhead for member joining and leaving is $O(\log_d N)$ and $O(d \log_d N)$, respectively. However, in ALM, the total overhead of simply multicasting one rekey message to all members is made up of $O(N)$ unicast messaging. Moreover, LKH does not take the network locations of end hosts into consideration when building the key tree. Thus, using ALM for rekey message distribution could be very inefficient. Therefore, blindly applying the key tree structure of LKH in ALM is not efficient in terms of network bandwidth usage.

Another approach to offer confidentiality in IP multicast is subgrouping. In Iolus, as shown in Figure 1(b), the scalability of rekeying is achieved by dividing secure multicast group into multiple subgroups G_i [Mittra 1997]. Each subgroup is managed by a trusted group security agent (GSA), which is *statically preconfigured* and located in different parts of the Internet. The GSAs form an overlay tree for a secure multicast group. In addition, GSAs have to decrypt and reencrypt the messages coming from the upstream subgroup. They also multicast the reencrypted messages within the subgroup using IP-multicast and forward them to the downstream GSAs using unicast. Each subgroup has its own subgroup key, and hence rekeying is performed only in the subgroup where a member joins or leaves the subgroup. As a result, the rekey messaging overhead depends only on the size of the subgroup. Suppose a new member u joins a subgroup G, the GSA of G has to update its subgroup key k_G to ensure forward secrecy. This can be done by multicasting the new subgroup key k'_G encrypted by k_G, and unicast k'_G to u using asymmetric encryption. In this case, only two rekey messages are required no matter how large G is. However, in the departure case, each remaining subgroup member should use unicast to communicate the new subgroup key with the GSA. Hence, the rekey operation for departure costs $O(M)$ rekey messages, where M is the subgroup size. When M is very large, a member departure could cost a lot of rekey messages in the network. When M becomes small, decrypting and reencrypting data packets among subgroups costs large processing overhead in the GSAs. Furthermore, how the GSAs are placed in the Internet is also a critical factor affecting the performance of key management. Therefore, it is difficult for Iolus to dynamically and efficiently adapt the change of membership.

SOM dynamically clusters members and manages each cluster in a distributed manner. While subgrouping adopted in Iolus does not consider the impact of subgroup size and nodal processing overhead, we consider and analyze the optimal performance and the effects brought by these parameters in this article. We also propose in our scheme a dynamic subgroup management to control the size of each subgroup and does not need any statically configured agents.

In our scheme, a simple clustering mechanism is proposed to group users into clusters for efficient key management. There have been also other clustering methods proposed in the literature. For example, NICE and ZIGZAG uses hierarchical clustering to organize and manage their clusters into multiple levels [Banerjee et al. 2002; Tran et al. 2003]. The use of multiple levels in these schemes is mainly for efficient data delivery. SOM can also apply their clustering techniques; however, those methods complicate the protocol. Some clustering techniques have also been proposed in the area of wireless ad hoc networks [Lin and Gerla 1997; Basagni 1999; Lin and Chu 2000]. Since those schemes are designed for communication-constricted networks (each node can only directly communicate with its one-hop neighbors), they usually assume that each node only has the knowledge of neighboring nodes one hop away. Therefore, unlike our proposed scheme, those clustering techniques target to form clusters with a limited diameter (which is usually one), rather than control the size of the clusters formed. Our scheme does not put any constraints on the number of hops away from any node, and hence keeps the algorithm simple and easy to implement. As those schemes are designed for mobile networks, they also focus on how to cope with the node mobility which our scheme does not need to consider. Our focus is on how to control the cluster size and allow splitting and merging.



(a) Every pair of peers on the tree shares a data encryption key.

(b) *data* is encrypted by a random key d which is then appended by d encrypted by a neighbor key.

Fig. 2. Host-to-host encryption.

The idea of SOM was previously proposed in Yiu and Chan [2004]. In this paper, we further model and analyze the performance of our system so as to work out the optimal cluster sizes for various applications. With the optimal cluster size for a given application, the system can achieve the minimum processing overhead for providing data confidentiality. We also present results on cluster characteristics and processing load distribution among peers in our system.

3. TWO BASIC APPROACHES

In this section, we describe in detail the two basic schemes, host-to-host encryption and whole group encryption because they are also the building blocks in SOM.

We model the topology of an overlay tree of N nodes (hosts) as $T = (V, E)$, where the source is located at the root v_0. Each node $v_i \in V$ represents a user in the group and each $e_i \in E$ represents a unicast connection between two end-point users.

3.1 Host-to-Host Encryption

In host-to-host encryption, each overlay connection is symmetrically encrypted. The encryption mechanism is shown in Figure 2(a). Two connected members v_i and v_j agree upon a symmetric key called *neighbor key* k_{ij} using key-encryption-key mechanism (i.e., v_i generates k_{ij} and makes the key known to v_j by encrypting it using v_j's public key) or Diffe-Hellman protocol. Clearly, a neighbor key has only local significance, i.e., it is associated between a pair of peers only. In this system, a peer v_i sends an encrypted data packet $E_{k_{ij}}\{data\}$ to its downstream peer v_j. Upon receiving the packet, v_j decrypts it using the shared neighbor key k_{ij}. Then, v_j reencrypts the whole packet for its downstream peer, say v_m. v_j encrypts the data again using another neighbor key k_{jm} which is shared by v_j and v_m, and forwards the encrypted packet $E_{k_{jm}}\{data\}$ to v_m.

As the processing overhead incurred in symmetric encryption is proportional to the size of the data, we can improve the previous scheme by the following (Figure 2(b)):

—Encryption:

(1) Data is encrypted as $E_d\{data\}$ by a symmetric key d randomly generated by the source.

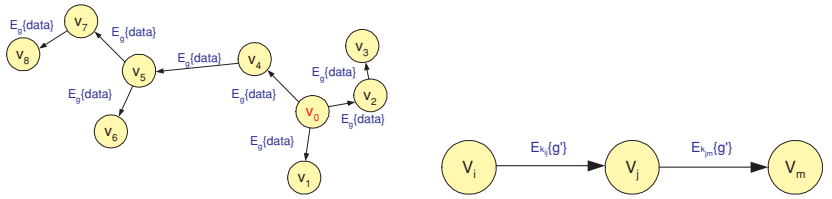
(2) $E_d\{data\}$ is then concatenated with $E_k\{d\}$, where k is the neighbor key between two peers on the data path, i.e., $E_d\{data\}||E_k\{d\}$.

—Decryption:

(1) A peer gets the random key d using its neighbor key k.

(2) It decrypts the "data" using d.

Therefore, reencryption is done by decrypting and reencrypting d rather than the whole packet. Since a symmetric key is usually much smaller in size as compared to a packet (128 or 256 bits versus kilobytes), this scheme reduces the reencryption overhead by orders of magnitude.



(a) All members share one universal data encryption key.

(b) The new group key g' is re-encrypted at every peer.

Fig. 3. Whole group encryption.

Whenever a member joins or leaves the system, its neighbors (i.e., its parent and children) reconfigure to maintain the connectivity of the overlay tree. These peers may form new overlay links and establish new neighbor keys with other peers. This is the rekeying operation in the scheme.

Although we improve the scheme by encrypting a key rather than the whole data packet, this approach still requires per-packet processing on every node for reencryption. Therefore, the nodal processing overhead would be high for high-bandwidth applications such as video delivery. Hence, host-to-host encryption is suitable for low-bandwidth applications.

3.2 Whole Group Encryption

In whole group encryption, the source encrypts data using a universal shared group key g (as shown in Figure 3(a)). In contrast to host-to-host encryption, a member receiving a packet $E_g\{data\}$ simply relays it to its children without any reencryption (it certainly needs to decrypt the packet using g for its own consumption).

Clearly, this scheme incurs no reencryption processing overhead at nodes. However, to ensure backward secrecy, whenever a member joins the group, the group key has to be changed in order to prevent it from decrypting past group data. [3] Similarly, the group key has to be changed when a member leaves the group to ensure forward secrecy. The new group key is encrypted and delivered to all the current group members. This rekeying process incurs a total of $O(N)$ rekey processing overhead to all the existing N members.

For a join event, all existing members are required to decrypt rekey messages $E_g\{g'\}$ for the new group key g' using the old group key g. The rekeying process for a member departure is high. Since the leaving member knows the old group key, we need to apply a host-to-host encryption approach to update the group key as follows:

(1) The new group key is encrypted using neighbor keys established among the peers. As shown in Figure 3(b), each peer first needs to decrypt the new group key g' using the neighbor key shared with its upstream peer.
(2) For each of its downstream peers, the node reencrypts g' with their shared neighbor key.

Clearly, the nodal processing overhead for rekeying is expected to be high for large and dynamic groups. This scheme is, therefore, suitable for small groups or groups with rather static membership.

[3]One may sniff and store the group data before its joining. After joining, the member could use the group key to decrypt the stored data *if* the group key is not changed after its joining. Similarly, one may keep the group key after leaving the group and use that group key to decrypt new group data *if* the group key is not changed after its departure.

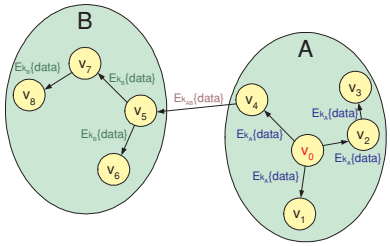


Fig. 4. Only members in the same cluster share the same cluster key. Reencryption is necessary only when transferring data across different clusters.

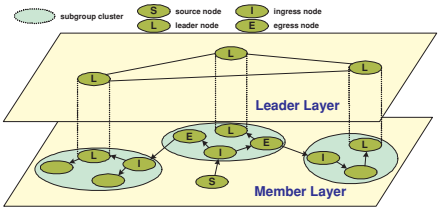


Fig. 5. Peers are organized into two layers in SOM: Leader layer and Member layer. S is the source of data stream. Intra-cluster trees are formed within the clusters while an intercluster tree is formed via the connections between ingress and egress nodes.

4. SECURE OVERLAY MULTICAST

In this section, we first give an overview of SOM framework (Section 4.1). We then describe in detail how the protocol handles member join and departure (Section 4.2 and Section 4.3). Next, we present how the protocol splits and merges clusters to adapt group dynamics (Section 4.4). Finally, we describe the ingress/egress node selection and the data path formation (Section 4.5).

4.1 SOM Framework

As clear from Section 3, given a certain data rate and group dynamics, neither host-to-host encryption nor whole group encryption would lead to minimum processing overhead. A more efficient way is to group members into nonoverlapping clusters of size m as shown in Figure 4. Instead of sharing a group key among all members, members in a cluster share a "cluster key." Whenever a member joins or leaves the group, it also joins or leaves a cluster. Hence, rekeying is done only within a cluster. Consequently, only $O(m)$ rekey messages are processed for each member arrival or departure. SOM loosely maintains the cluster sizes by splitting and merging them. In order not to incur too much splitting and merging overhead while keeping a rather uniform cluster size, SOM bounds the size of each cluster between $c/2$ and $2c$, where c is a system parameter for controlling the cluster size. Packets are reencrypted only when they are transferred across the cluster boundaries, and reencryption only takes place at the ingress and egress nodes of a cluster (i.e., node v_4 and v_5 in the figure). In other words, SOM applies "whole cluster encryption" within clusters and "cluster-to-cluster encryption" between clusters.

In SOM, members are logically organized into two layers as shown in Figure 5: 1) Leader layer and 2) Member layer. The two layers are implemented by two *independent* ALM protocols. While a normal member only joins the ALM protocol running for the member layer, a cluster leader joins both ALM protocols for the member and leader layers. Every cluster has a cluster leader, which forms an overlay network with other cluster leaders for control messaging and for coordinating operations such as joining, merging and splitting. Therefore, the cluster leaders constitute the leader layer. Together with other members, they form the member layer. In our simulation, we apply Delaunay Triangulation

(DT) protocol in both layers [Liebeherr et al. 2002]. Interested readers are referred to Liebeherr et al. [2002] for the details of tree formation and data delivery.

We elaborate the main roles in SOM as follows:

Cluster Leader. It keeps information related to each cluster member (such as the member's ID or address). It also works as an entry point for a new member to join its cluster. The *JoinRequest* from a new member is passed down to the Member layer if the leader finds that its cluster is the closest one to the member. The leader is also responsible for splitting the cluster into two if it is too large. When a cluster is too small, the leader will send *MergeRequest* messages to its neighboring cluster leaders in the Leader layer to find a suitable cluster for merging.

A cluster leader periodically multicasts *HeartBeat* messages within its own cluster to indicate its presence. If any member finds that its cluster leader has left the group, it first randomly backs off and then multicasts a *NewLeader* message within the cluster. It is possible that more than one member declares itself as the new leader. In this case, tie is broken by their network addresses.

Egress Member. An egress member is responsible for *sending* data from its cluster to another cluster. Data sent across clusters, say from A to B, must be reencrypted with another key k_{AB}. It has to establish an intercluster encryption key with each of the downstream ingress nodes.

Ingress Member. As opposed to an egress member, an ingress member *receives* data from another cluster. It also needs to reencrypt the packets received using its cluster key before forwarding them to other members in its cluster.

Normal Member. A normal member does not need to do any reencryption; he only needs to forward the multicast data within the cluster and decrypts it using the cluster key for its own use. Each member periodically sends *Heartbeat* messages to its connected peers so as to indicate its aliveness. Absence of several *Heartbeat*s from a member implies the departure of that member.

Note that the protocol suggested here is a framework; hence, the intercluster network formed by leaders and the intracluster network formed by members can be implemented by any existing ALM protocols. In SOM simulation, we use DT-GNP as the underlying protocol due to its distributed nature. This is described in more detail in the following.

Accurate estimation of network distances can improve network efficiency in terms of latency, bandwidth, and other operations such as joining, leaving, splitting, merging, etc. However, measuring the network distance using `ping` for all pairs of peers is time-consuming and not scalable. Internet coordinate system such as Global Network Positioning (GNP) [Ng and Zhang 2002], Vivaldi [Dabek et al. 2004], Internet Coordinate System (ICS) [Lim et al. 2005] etc. are fast and scalable approaches to determine network distances between peers. The key idea of these systems is to map the Internet locations of peers into some appropriate coordinate system (e.g., an N-dimensional Euclidean space) with only a few network measurements. The computed distances from the coordinates are highly correlated with their network distances.

We have used GNP in our simulation. Using GNP, a cluster leader can summarize its cluster location by calculating the *centroid* of all member coordinates within its cluster. With the centroid, a new member, given its coordinates, can find its the closest cluster to join easily and quickly. Furthermore, the coordinates of members can make the splitting of a cluster more efficient.

4.2 Member Arrival

In SOM, a new member u joins the group in two steps:

(1) It finds the closest cluster to join.

(2) It joins the cluster and attaches itself to the ALM tree within the cluster.

To find the closest cluster, u first contacts one of the cluster leaders in the multicast group. This can be bootstrapped by a server or Rendezvous Point (RP) which keeps the locations of some cluster leaders. When the RP receives a *JoinRequest* message containing u's coordinates, the RP searches for the closest leader from its leader list. Then, it forwards the *JoinRequest* to that closest leader. Upon receiving a *JoinRequest*, the cluster leader compares u's coordinates with the *centroid* of the cluster and determines whether any other neighboring cluster is closer to u according to its centroid. (The centroid of a cluster may be computed from the coordinates of a few members in the cluster.) If so, it forwards the request to the leader of the closer cluster. The process continues until the closest cluster which u belongs to is found. After that, the *JoinRequest* is passed down to the Member layer where u forms an intracluster tree with the cluster members. The details of forming the intracluster tree is the same as the joining procedure of the underlying ALM protocol.

Once the new member u attaches itself on the intracluster tree, rekeying is performed within the cluster. u forms a secure channel with each of its neighbors using asymmetric encryption. It sends its public key p_u in the *JoinRequest* to its neighbor v. v then generates a symmetric key k_{uv} (the so-called *neighbor key*) and sends back a *JoinReply* with the encrypted neighbor key using u's public key (i.e., $E_{p_u}\{k_{uv}\}$). Then, u and v can use the same secret key k_{uv} to update the cluster key in the future. The cluster leader then updates the cluster key to guarantee backward secrecy. It first generates a new cluster key k_c' and then multicasts the rekey message $E_{k_c}\{k_c'\}$ in the cluster, where k_c is the old cluster key. When a peer finds that its downstream peer is a new member, it reencrypts k_c' using the secret key shared with the downstream peer so that the new member can get the new key k_c' without knowing the old key k_c.

4.3 Member Departure

The departure of a member can be known by either 1) a *Goodbye* message from the member to its neighbors (i.e., a graceful leave); or 2) an absence of several *HeartBeat* messages to its neighbors (i.e., an ungraceful leave). Whenever a peer detects the leaving of its neighbor, it informs its cluster leader to trigger the rekeying mechanism. Meanwhile, the intracluster tree may need to be changed. The peers at the two ends of a new link establish a neighbor key using asymmetric cryptography as in the joining process. After the cluster leader generates the new cluster key k_c', it multicasts k_c' within the cluster along the overlay tree using host-to-host encryption, that is, members reencrypt k_c' using neighbor keys before forwarding it.

4.4 Cluster Split and Merge

A cluster leader periodically exchanges with its neighboring leaders in the Leader layer *ClusterInfo*, which contains cluster size and member information. SOM keeps the size of each cluster within the range from $c/2$ to $2c$. When a cluster T becomes too large (i.e., greater than $2c$), the leader x triggers the splitting operation. As x stores all member coordinates of its cluster, it can perform any centralized clustering algorithm [Jain and Dubes 1988] based on these coordinates to split the original cluster into two parts, T_1 and T_2, each of which contains fewer than $2c$ members. Suppose x is in T_1 after splitting. It then randomly picks a member y from T_2 as the leader of that cluster and sends to y a *LeaderTransfer* message containing information of its members for management purpose. After cluster split, both leaders (x and y) need to generate their new cluster keys by multicasting rekey messages as in the joining process.

On the other hand, when a cluster T_1 shrinks and becomes too small (i.e., less than $c/2$), it has to merge with some suitable neighboring clusters. A cluster is considered to be suitable if, after merging, 1) the resultant cluster size is within the size range; or 2) the resultant cluster can then be split into smaller clusters whose sizes are within the range. Once the target cluster T_2 and its leader y are identified,

the leader x of T_1 sends a *MergeRequest* to y. The *MergeRequest* contains some network addresses of the members in T_1. y then selects a pair of members (s, t): s is contained in the *MergeRequest* and t is in T_2. It informs t to connect to s for merging according to the partition recovery mechanism in the underlying ALM protocol. After that, x becomes the leader of the combined cluster. Upon cluster merge is completed, x needs to rekey the new cluster key using the same mechanism in the departure process.

4.5 Data Delivery Path and Ingress/Egress Selection

In SOM, cluster leaders first determine among themselves in the leader layer which cluster is connected to which by means of some ALM protocols (such as DT as presented in Liebeherr et al. [2002] which uses compass routing [Kranakis et al. 1999]). Given the connectivity, cluster leaders then identify their ingress and egress nodes to form overlay connections across clusters. Within a cluster, an intracluster tree (rooted at its ingress node) is also built according to some ALM protocols (which may be different from the intercluster ALM protocol). As shown in Figure 5, an intercluster tree in SOM is formed by connections between egress nodes and ingress nodes rooted at the cluster to which the source connects. In each cluster, an intracluster tree rooted at its ingress node is formed by connecting peers within the same cluster. With all the inter- and intracluster links, an overlay tree is constructed for data delivery.

In order to minimize delay, the closest pair of nodes should be chosen as the egress-ingress pair between two clusters. However, suppose the cluster size is C, finding such pair may require $O(C^2)$ (or $O(C \log C)$ for a more advanced technique [Sahni 1998]) computations, which may not be desirable in reality. To reduce it to $O(C)$, a suboptimal solution is used as follows. A downstream leader x first chooses a number of nodes which are the closest to the centroid of the upstream cluster with leader y. This is the *IngressList* that x sends to y. y then chooses the closest pair of nodes, one from its cluster (the egress node) and the other from the *IngressList* (the ingress node). The egress node then makes a secure connection with the ingress node for packet reencryption.

5. SYSTEM ANALYSIS

In this section, we analyze the storage requirement, number of rekey messages and processing overhead of our protocol, and compare them with logical key hierarchy (LKH) and Iolus [Wong et al. 2000; Mittra 1997].

5.1 Storage Requirement

In a secure multicast system, receivers and the central key server (which may be the source) are required to store a number of keys for data encryption and rekeying purposes.

In LKH, each member in the group needs to store $\log_d N = O(\log N)$ keys (i.e., the number of keys along a branch of the logical key tree, see Figure 1(a)), where N is the size of the secure group and d is the degree of the key tree[4]. The central key server needs to manage all the keys in the key tree, i.e., $(d^{h+1} - 1)/(d - 1) = O(N)$ where h is the height of the key tree.

On the other hand, in Iolus, each member in a subgroup needs to store only two keys, namely, the subgroup key and the secret key shared with its GSA. Each GSA, however, needs to store all the secret keys shared with all the members within its subgroup, and the subgroup keys of the associated subgroups. Therefore, the number of keys stored at each GSA depends on the subgroup size, which may be as large as the group size.

In SOM, each member keeps one cluster key and several neighbor keys. Depending on the ALM protocol for implementation, the number of neighbor keys stored varies. For example, in Delaunay Triangulation protocol [Liebeherr et al. 2002], the average number of neighbors is six. For egress or

[4]In this analysis, we assume the key tree in LKH scheme is well balanced.

ingress peer, one more key is needed for reencryption across clusters. Clearly, as opposed to LKH and Iolus, SOM does not have special entities or members which require disproportionately higher storage than other members. In other words, SOM evenly distributes the key storage load over all its peers.

5.2 Number of Rekey Messages

The bandwidth overhead consumed for rekeying depends on the number and the size of rekey messages. Different from IP-multicast-based schemes, we need to consider the number of unicast transmissions needed for sending the rekey messages in ALM. In the following analysis, we normalize the bandwidth overhead by the size of a key.

In LKH scheme, each join/leave requires a change of all the keys on the path from the member u to the root in the key tree, where the path consists of $\log_d N = O(\log N)$ keys. In the case of member join, for each key on the path, the key server needs to send one unicast rekey message (encrypted using u's individual key) to u and one multicast rekey message to other members in the same subgroup as u. Since each multicast message in ALM is accomplished by $O(N)$ unicast messages. The total number of rekey messages for a member join is $O(N \log N)$. Similarly, in the case of member leave, for each key on the path, the key server needs to send d multicast rekey messages (except the bottommost one, which can be updated with individual user using unicast message.) Hence, the total number of rekey messages for a member leave is $O(d N \log N)$.

In Iolus, when a member u joins the group, the designated group security agent (GSA) needs to multicast one rekey message to all the old members in u's subgroup and unicast one rekey message to u for updating the subgroup key. In ALM, the multicast rekey message consists of $O(M)$ unicast messages, where M is the subgroup size. When a member leaves the group, the GSA has to unicast a rekey message to each of the subgroup members. For each subgroup member, the GSA sends a rekey message containing the new subgroup key encrypted using the member's individual key.[5] Hence, the total number of rekey messages for a member leave is $O(M)$. In this approach, all the M rekey messages are sent from the GSA, making the network link at the GSA a bottleneck.

In SOM, a member arrival triggers the generation of a new cluster key by the cluster leader. The cluster leader then encrypts the new cluster key using the old cluster key and multicast it within the cluster using ALM. The new cluster key is also encrypted using the new member's secret key and sent to the new member. This operation incurs an equivalence of $O(C)$ unicast rekey messages, where C is the cluster size. When a member leaves the system, the cluster leader multicasts the new cluster key within the cluster. The distribution of the new cluster key uses host-to-host encryption. This operation also incurs $O(C)$ unicast rekey messages. In this approach, the cluster leader only sends the rekey messages to its directly connected peers in the ALM tree. Furthermore, unlike Iolus, the cluster size in SOM is bounded and so is the number of rekey messages generated. To summarize, SOM incurs fewer rekey messages during membership changes.

5.3 Processing Overhead

In each rekeying for key updates, some peers need to update their keys, leading to some computation or processing overhead. In general, updating a key is equivalent to decrypting an encrypted key. In the following, we analyze the number of keys required to be updated.

In LKH, for each joining or leaving member, its key path to the root is required to be updated. As some of the subgroup keys and the group key are shared with other group members, the rekeying process also incurs computation/processing on other members as well. We take a binary key tree as an example. Due to the key tree structure, for a balanced tree, on average 50% of the group members need to update

[5]Every member has a secret key called *individual key* shared with its corresponding trusted GSA.

Table I. Cluster Characteristics

c	Average size m
16	20.88
32	38.45
64	80.27
128	161.28
256	318.72

one key, 25% two keys, 12.5% three keys, and so on. This sums up to $O(N)$ key updates in the group. Furthermore, the joining member needs to update around $\log N$ keys. On the other hand, a leaving member does not need to do any update.

In Iolus and SOM, only the members in the affected cluster (or subgroup in Iolus) need to update the cluster key. However, since the subgroup size in Iolus is not bounded, a large number of members may need to update their keys. In summary, the dynamic cluster management in SOM localizes the effect of membership changes within the affected cluster, leading to much reduced processing overhead.

6. OPTIMAL CLUSTER SIZE

In this section, we analyze the average nodal processing overhead given by the sum of the rekeying and reencryption overhead and show that an optimal cluster parameter exists to minimize the overhead in the system.

In SOM, the size of each cluster is bounded between $c/2$ and $2c$. Let m be the average cluster size. Assuming that the cluster size is uniformly distributed, we have

$$m = 1.25c.$$

Table I summarizes the characteristics of the clusters in our simulation. The average cluster size m is roughly equal to $1.25c$ which agrees with what is calculated in our analysis. For the sake of simplicity and analytic tractability, we will approximate in the following that all the clusters are of (constant) size m and all integral values as continuous. For example, given a group size of N, the number of clusters is N/m.

In this analysis, we assume members arrive according to a Poisson process with rate λ req./s and stay in the system with exponential holding time of mean $\overline{T}$ seconds. The data stream is of constant bit rate at R bits/s. Each packet is of constant size S bits where S may be thousands of bits.

We normalize each symmetric encryption or decryption (of one key-sized data) as one unit of computation overhead. (Encryption and decryption are the same operation in symmetric cryptography.) As symmetric cryptographic operations are often three to five orders of magnitude faster than their asymmetric counterparts, we define α and β as the relative CPU time spent on asymmetric encryption and decryption, respectively, for accounting their corresponding processing overhead. They are given by:

$$\alpha = \frac{\text{Time for one asymmetric } \textit{encryption}}{\text{Time for one symmetric } \textit{encryption}};$$

and

$$\beta = \frac{\text{Time for one asymmetric } \textit{decryption}}{\text{Time for one symmetric } \textit{decryption}}.$$

In other words, an asymmetric encryption costs α units of computation, while an asymmetric decryption costs β units of computation. Table II lists the major nomenclatures defined in the analysis.

Table II. Nomenclature Used in the Analysis

Symbol	Definition
λ	Average arrival rate of the users (req./s).
$\overline{T}$	Average holding time (s).
μ	Average service rate $= 1/\overline{T}$.
ρ	Average number of concurrent users $\Delta\lambda\overline{T}$ (req.).
c	Size of a cluster is bounded between $c/2$ and $2c$.
m	Average number of users in a cluster.
m^*	Optimal cluster size.
R	Data rate for an application (bits/s).
S	Size of a data packet (bits).
α, β	Speed factors between symmetric and asymmetric encryption and decryption, respectively.

Table III. Procedures for Member Join and Their Cost Accounting

Steps	New Member u	New Member's Neighbors nbr_i $\{nbr_i : i \in [1..A_J]\}$	Cluster Leader v_L	Cluster Members v_i $\{v_i : i \in [1..m]\}\backslash\{v_L\}$
1. u's neighbors encrypt their neighbor keys using u's *public key* PK_u. Then, u decrypts the neighbor keys using its *private key*.	$A_J\beta$	$A_J\alpha$	-	-
2. Cluster leader v_L multicasts the new *cluster key* encrypted using the old cluster key, i.e., $E_{k_c}\{k_c'\}$. Cluster members v_i decrypts the new cluster key.	-	-	1	$m-1$
3. u's parent encrypts k_c' using their neighbor key. u can then decrypts it.	1	1	-	-

We are now ready to derive the total overhead in the system, which is the sum of rekeying and reencryption overhead. These are given as follows:

6.1 Total Rekeying Overhead

In the steady state, there are m members in each cluster. Recall that a joining member establishes some keys with its neighbors using asymmetric encryption. Let A_J be the average number of neighbor keys established using PKI for this. Following that, the cluster leader generates a new cluster key, which is encrypted with the old cluster key before multicasting in the cluster. Every cluster member except the new one then decrypts the new cluster key. When the parent of the new member receives the new cluster key, it reencrypts the key using the symmetric key just established with the new member before sending it to the member. Recall that even if the member join triggers a cluster split process, the rekey procedure is the same as that in joining process without cluster split. This is because when the cluster splits into two smaller clusters, the two cluster leaders rekey their own clusters after split and the total number of symmetric encryption/decryption is still equal to $m + 2$ $(= m/2 + m/2 + 2)$. Therefore, we will not double count the processing overhead for cluster split in the following analysis.

Let H_{Join} be the processing overhead for a join event. Table III summarizes the total overhead in such joining event. Summing the items up, we hence have

$$H_{Join}(m) = A_J(\alpha + \beta) + m + 2. \tag{1}$$

For leaving, recall that when a member leaves the multicast group, the cluster leader generates a new key k_c'. Note that it cannot use the old cluster key k_c to encrypt k_c' (because the leaving member holds the key). Instead, the leader and the peers encrypt k_c' using the neighbor keys and forward downstream. Therefore, every cluster member needs to decrypt for k_c' and all internal nodes of the cluster tree need to

Table IV. Procedures for Member Leave and Their Cost Accounting

	Steps	Leaving Member u	Leaving Member's Neighbors nbr_i $\{nbr_i : i \in [1..A_L]\}$	Cluster Members v_i $\{v_i : i \in [1..m]\}\backslash\{u\}$
1.	u's neighbors form new links in the cluster tree. They reestablish the neighbor keys with their new neighbors.	-	$A_L(\alpha+\beta)$	-
2.	Cluster leader generates a new *cluster key* k_c'. It is then encrypted by neighbor keys and forwarded downstream within the cluster. Apart from the cluster leader, there are $(m-2)$ members need the encryption of k_c'.	-	-	$m-2$
3.	Upon receiving the encryption of k_c' from their parents, the $(m-2)$ members decrypt for k_c'.	-	-	$m-2$

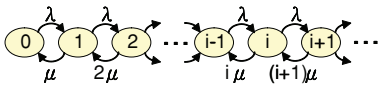


Fig. 6. State transition diagram for a $M/M/\infty$ queue.

encrypt k_c'. Since there are $(m-1)$ remaining members, there are $(m-2)$ pairs of symmetric encryption and decryption (the leader does not need to decrypt k_c'). In addition, some new connections may be formed due to the leaving member. Let A_L be the average number of neighbor keys established using PKI for this. Similarly, the rekey processing overhead due to cluster merge during member departure shares the same processing overhead accounting. Therefore, we will not double count the overhead for cluster merge in the following analysis. Let H_{Leave} be the total processing overhead for a member leaving the group. Table IV summarizes the total overhead in such leaving event. Summing up, we have

$$H_{Leave}(m) = A_L(\alpha+\beta) + 2(m-2). \tag{2}$$

The system can be modeled as a $M/M/\infty$ queue. Figure 6 shows the state transition diagram of the system. Let $i = \{0, 1, 2, \ldots\}$ be the system state indicating the number of concurrent members in the group. For the system in state i, since there are two exponential random processes with the total rate of $\lambda + i\mu$ (member join with mean arrival rate λ and member leave with mean departure rate $i\mu$) for the system to leave the state, the expected time $E[T_i]$ for the system to stay in state i is $E[T_i] = \frac{1}{\lambda+i\mu}$, and the expected rekeying overhead $E[H_{Rekey_i}(m)]$ at state i is

$$E\big[H_{Rekey_i}(m)\big] = \frac{\lambda}{\lambda+i\mu}H_{Join}(m) + \frac{i\mu}{\lambda+i\mu}H_{Leave}(m). \tag{3}$$

Let π_i be the steady-state probability that the system is in state i. Clearly, from queueing theory for a $M/M/\infty$ queue [Kleinrock 1975], we have the probabilities following a Poisson distribution; that is, $\pi_i = \frac{(\lambda/\mu)^i}{i!}e^{-\lambda/\mu}$. Thus, the expected total rekeying overhead per second, $E[H_{Rekey}(m)]$, is given by

$$\begin{aligned} E\big[H_{Rekey}(m)\big] &= \sum_{i=0}^{\infty}\pi_i\frac{E\big[H_{Rekey_i}(m)\big]}{E[T_i]} \\ &= \sum_{i=0}^{\infty}\pi_i\left[\lambda H_{Join}(m) + i\mu H_{Leave}(m)\right] \\ &= \lambda[A(\alpha+\beta)+3m-2], \end{aligned} \tag{4}$$

where $A = A_J + A_L$. From Equation (4), we can see that the rekeying overhead increases with the member arrival rate λ. This is obvious that for a higher joining rate, the rekeying procedure is performed more frequently, and hence the expected rekeying overhead is also higher. Moreover, the overhead also increases with the cluster size m, because the rekeying procedure involves all the members in the cluster of interest. Interestingly, the overhead is independent of the service rate μ. This is because the total service rate μ_i $(= i\mu)$ is proportional to the number of concurrent users in the system, while the probability that the system in state i, π_i, is smaller for larger i. These two factors cancel each other, leading to the independence of μ in the result.

6.2 Total Reencryption Overhead

Packets are forwarded across clusters via egress-ingress pairs. Recall that each egress node reencrypts the key appended in the data packets using the symmetric key established with its downstream peer, an ingress node. Each ingress peer in turn reencrypts them using its cluster key. Clearly, this reencryption requires two extra symmetric encryptions for each data packet transmitted between an egress-ingress pair. Given i members in the group, there are $i/m - 1$ egress-ingress pairs. Let the data rate be R bps and packet size be S bits. Then, the packet arrival rate is R/S packets per second. Let $E[H_{ReEncrypt_i}(m)]$ be the expected reencryption processing overhead per second in the group given that the system is in state i. Then,

$$E[H_{ReEncrypt_i}(m)] = 2\left(\frac{i}{m} - 1\right)\frac{R}{S}. \tag{5}$$

Thus, the expected total reencryption overhead per second, $E[H_{ReEncrypt}(m)]$, is

$$\begin{aligned} E[H_{ReEncrypt}(m)] &= \sum_{i=0}^{\infty} \pi_i \left[2\left(\frac{i}{m} - 1\right)\frac{R}{S}\right] \\ &= 2\left(\frac{\rho}{m} - 1\right)\frac{R}{S}. \end{aligned} \tag{6}$$

From Equation (6), we observe that the reencryption overhead increases with the packet rate R/S. This is clear that for a higher packet rate, ingress/egress nodes need to perform reencryption more frequently. Furthermore, since the reencryption process is performed across clusters, the overhead is also proportional to the number of clusters in the system ρ/m. Therefore, a larger cluster size leads to a higher overhead, which contrasts to the result obtained in Equation (4).

6.3 The Sum of Processing Overheads

Adding Equations (4) and (6), the total nodal processing overhead per second, $E[H(m)]$, is hence given by

$$\begin{aligned} E[H(m)] &= \frac{1}{\rho}\left\{E[H_{Rekey}(m)] + E[H_{ReEncrypt}(m)]\right\} \\ &= \frac{1}{\rho}\left\{\lambda[A(\alpha+\beta) + 3m - 2] + \frac{2\rho R}{mS} - \frac{2R}{S}\right\} \\ &= \frac{A(\alpha+\beta)}{\overline{T}} + \frac{1}{\overline{T}}(3m-2) + \frac{2R}{mS} - \frac{2R}{\rho S}. \end{aligned} \tag{7}$$

As a special case, for host-to-host encryption, we have

$$E[H_{host-to-host}] = \frac{A(\alpha+\beta)}{\overline{T}} + \frac{2R}{S}\left(1 - \frac{1}{\rho}\right).$$

For whole group encryption, we have

$$E[H_{whole\_group}] = \frac{1}{\overline{T}}\left[A(\alpha+\beta)+(3\rho-2)\right].$$

From Equation (7), we note that the first and the last terms are constants independent on m. In particular, the values that we choose for α and β do not affect our results on optimal cluster size m^*. To minimize $E[H(m)]$, it is equivalent to minimize $f(m)$ given by the two terms in the middle, that is,

$$f(m) = \frac{1}{\overline{T}}(3m-2) + \frac{2R}{mS}. \tag{8}$$

To get the optimal cluster size m^*, we set

$$\partial f(m)/\partial m = 0$$

and obtain

$$f'(m) = \frac{3}{\overline{T}} - \frac{2R}{m^{*2}S} = 0,$$

that is,

$$m^* = \sqrt{\frac{2R\overline{T}}{3S}}, \tag{9}$$

or

$$c^* = \frac{1}{1.25}\sqrt{\frac{2R\overline{T}}{3S}}. \tag{10}$$

From Equation (9) or (10), we clearly see that for applications with high bandwidth and/or long holding time, the cluster size should be larger in order to minimize the processing overhead. For a high packet rate application, we should reduce our reencryption overhead by having fewer clusters (i.e., having larger clusters). For a long holding time application, rekeying is not so frequently performed, so we can afford having more members in each cluster.

Figure 7 shows that, our simulation results (with exponential holding time) fit the analytic model well, and both of them agree on the same optimal cluster parameter c^*.

A final remark is in place here. $A\,(= A_J + A_L)$ in Equation (7) depends on the underlying ALM protocol. For DT protocol, a point in a DT mesh has six neighbors on average due to its geometric properties, meaning that a joining member needs to establish six neighbor keys. Therefore, we set $A_J = 6$. The six neighbors form three new triangles after a member leaves and hence, $A_L = 3$. Therefore, we have $A = 9$ for DT.

As shown above, due to DT's geometric properties, we can easily determine the values of A_J, A_L, and thus A. For this reason, though any ALM protocol can be used in SOM, in order to easily compare experimental results with our analytical results, we choose DT as the base protocol for constructing overlay meshes among peers in our simulation.

7. ILLUSTRATIVE NUMERICAL RESULTS

We compare through simulation the performance of SOM with the two basic schemes and a recently proposed ALM protocol, namely Delaunay Triangulation (DT) [Liebeherr et al. 2002; Wong and Chan 2003]. We use DT protocol in both Leader layer and Member layer in SOM.

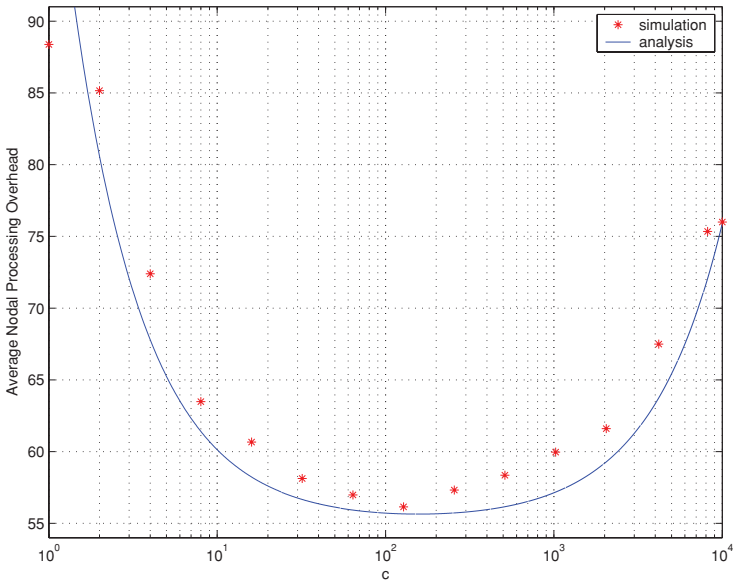


Fig. 7. Nodal processing overhead versus cluster parameter c.

7.1 Simulation Model

In our experiments, we first generate a network topology using GT-ITM (Georgia Tech Internetwork Topology Models generator) [Zegura et al. 1996]. We use the Transit-Stub model with 2,000 routers. Members arrive according to a Poisson process with rate λ req./s and stay in the system with exponential holding time of mean $\overline{T}$ seconds. Each arrival is randomly attached to one of the routers. As GNP is used, 15 landmarks are introduced, randomly connected to different routers in the network. The source is randomly chosen among the peers and generates a data stream of constant bit rate at R bits/s. Each packet is of constant size S bits where S may be thousands of bits.

As our baseline system, we use the parameters $R = 256$ kbps, $S = 8{,}000$ bits, $\overline{T} = 30$ min., $\rho = \lambda\overline{T} = 10,000$ and $c = 128$ (i.e., a typical Internet TV application). For α and β, we use the values given by experimental results tested on OpenSSL utility program (http://www.openssl.org/). In our experiment, we measure the speed of commonly used cryptographic operations on a 800MHz Pentium III Linux workstation using the optimized implementations of the OpenSSL utility program, and find the following values to be reasonable: $\alpha = 1{,}000$ and $\beta = 10{,}000$. Note that these values of α and β are for demonstration purpose and do not affect our final result on how to choose an optimal system parameter c. This means that the optimal cluster size is the same for different values of α and β. The reason is clear from Equation (7). These values only offset the overhead, but do not change the minimal point of the equation as derived in Section 6. In our simulation, we vary most of the parameters beyond their base values while keeping the remainder the same in order to study the system performance.

In our simulation, we study the following three performance metrics:

Average nodal processing overhead. We define the average nodal processing overhead $\overline{H}$ as:

$$\overline{H} = \frac{\sum_{i \in G} H_i}{\rho},$$

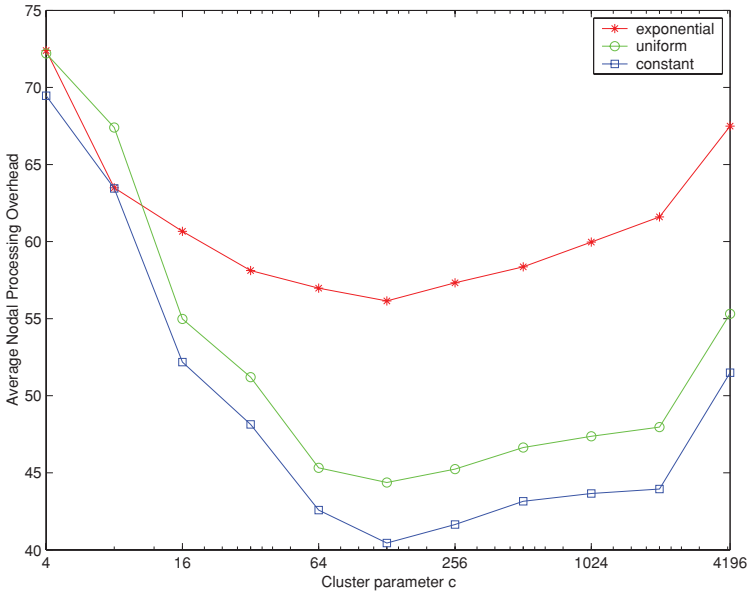


Fig. 8. Nodal processing overhead versus cluster parameter c ($\lambda = 4$ req./s, $\overline{T} = 1800$s).

where H_i is the processing overhead per second of peer i in the group G. Recall that the nodal processing overhead is incremented by one when a node performs a symmetric encryption/decryption, by α for each asymmetric encryption and β for each asymmetric decryption.

Physical link stress (PLS). PLS is defined as the number of identical packets transmitted over a physical link.

Relative delay penalty (RDP). RDP is defined as the ratio of the delay in the overlay path between a host and the source to the delay in the unicast path.

7.2 Nodal Processing Overhead

In Figure 8, we plot $\overline{H}$ against c with different holding time distributions, namely, exponential, uniform ($[0.5\overline{T}, 1.5\overline{T}]$) and constant ($\overline{T}$). $\overline{H}$ first decreases (due to a reduction in reencryption overhead) and then increases again (due to an increase in rekeying messaging overhead). There is an optimal cluster parameter c^* to minimize $\overline{H}$ for the system. (For our baseline parameter, c^* is around 100.) The curve around the minimal point is quite flat, which means that keeping the cluster size to be the exact value of c^* is not required. Hence, our scheme roughly maintains the cluster size and does not need complicated clustering protocol. On the other hand, the curve shows that blindly using either of the basic schemes (host-to-host or whole group encryption at the left and right extreme points in the curve respectively) leads to substantially high overhead as compared to our cluster-based approach. c^* depends on the application parameters. The figure also shows that randomness in holding time leads to higher nodal processing overhead. However, it does not affect the optimal cluster parameter c^* much. In Table 5, we illustrate the optimal c^* for various applications (based on Equation (10)). As we can see from the table, c^* ranges from as few as tens to as many as hundreds. The cross among lines on the left of the graph is mainly due to statistical fluctuation of the collected data.

We compare SOM in terms of $\overline{H}$ with the two basic schemes (host-to-host and whole group encryption) in Figure 9, given the value of c ($c = 128$). The overhead for host-to-host encryption is independent of λ

Table V. Optimal Cluster Size of Various Multimedia Applications ($S = 8{,}000$ bits)

Typical multimedia applications	Average holding time ($\overline{T}$)	Data rate (R)	Optimal cluster parameter (c^*)
stock quote system	30 min.	5 kbps	20
Internet radio	2 hrs.	16 kbps	80
Internet TV	30 min.	256 kbps	160
high-quality video	2 hrs.	1 Mbps	620

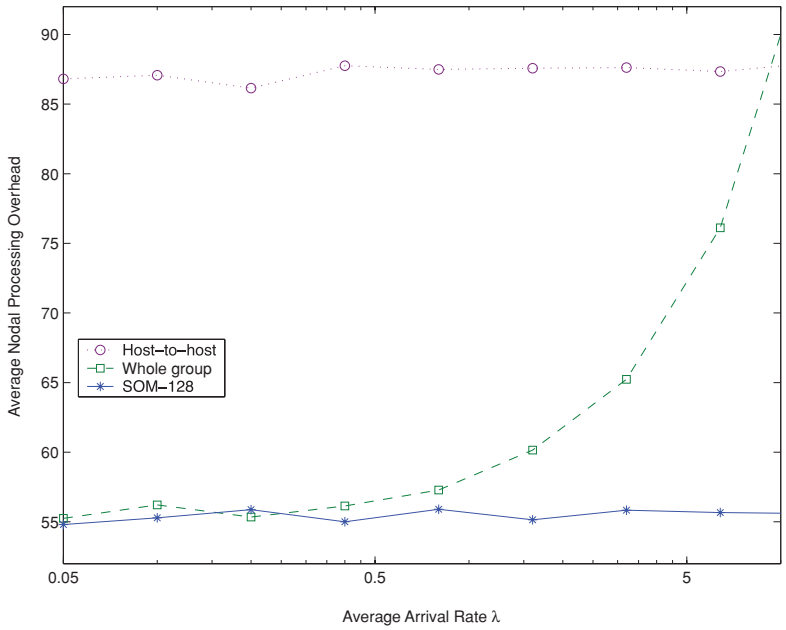


Fig. 9. Average nodal processing overhead $\overline{H}$ versus λ ($c = 128$, $\overline{T} = 1800$ s).

and stays at a level depending on the data rate. This is because each peer needs to continuously reencrypt data packets no matter how often the members join, and the reencryption is obviously proportional to the packet arrival rate. On the other hand, the overhead for whole group encryption increases with λ because i) a larger λ implies a larger ρ (since $\rho = \lambda\overline{T}$), which increases the rekeying overhead in the group. This is because the rekeying procedure in whole group encryption involves all the members in the group. Each time when a new member joins or an existing member leaves the system, each of the group members needs to decrypt the new group key. When there are more members in the group, a higher rekeying overhead is incurred; ii) a larger λ also leads to more frequent join or leave events. This means that more rekeying procedures are required, also incurring higher average rekeying overhead. It is shown in the figure that, the curve for whole group encryption goes up, and its value finally becomes larger than that of host-to-host encryption. The reason is that its rekeying overhead exceeds the overhead saving from the reencryption overhead. Finally, it is clear from the figure that SOM achieves the minimum overhead and remains at a substantially lower level when λ increases even to a large value, as join or leave events trigger rekeying processes only within the affected clusters but not in the whole group. The gain in nodal processing overhead is about 30%.

Figure 10 shows the distribution of processing load among all the peers. We find that a large fraction (over 80%) of them have nodal processing overhead lower than the average processing load (around 55

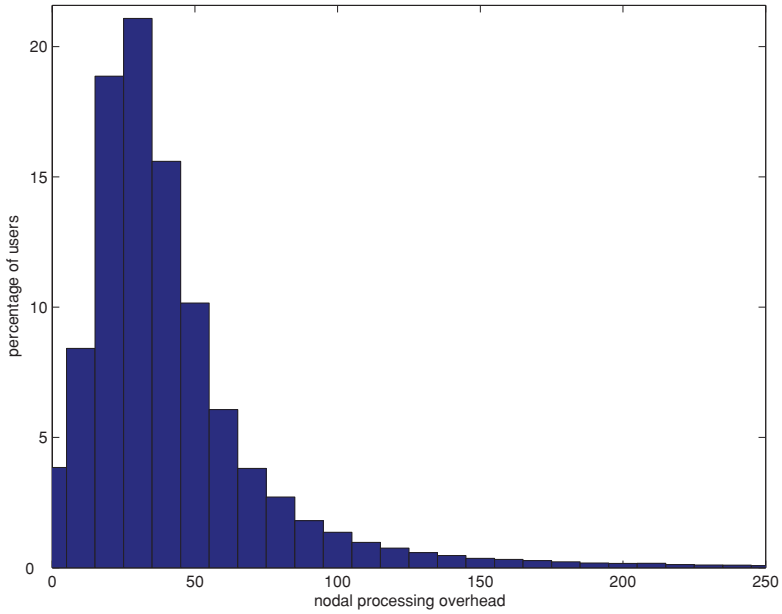
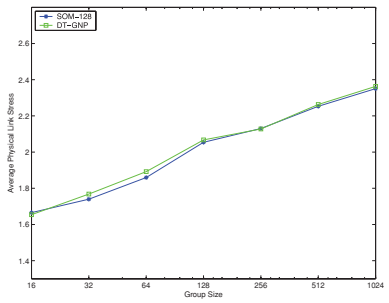
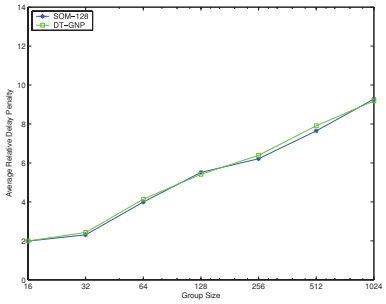


Fig. 10. Overhead distribution among peers ($c = 128$, $\overline{T} = 1800$ s).



(a) Average physical link stress versus number of members in the group.



(b) Average relative delay penalty versus number of members in the group.

Fig. 11. Network performance of SOM ($c = 128$).

as shown in Figure 9). The peers at the tail of the figure are the ingress and egress peers, which need to do packet-by-packet reencryption, and hence have higher processing overhead than the average.

7.3 Network Performance

In Figures 11(a) and 11(b), we show the network performance of SOM in terms of PLS and RDP versus the number of members in the group. Also shown is the corresponding DT performance. The figures show that SOM does not sacrifice network performance to achieve low processing overhead. When the group size is small, SOM performs like whole group encryption and hence its performance is the same as DT. When the group size increases, it is obvious that both RDP and PLS increase. The figures show that SOM achieves similar level of performance as compared to traditional ALM protocols. Note that the crosses among lines are solely due to statistical fluctuation of the collected data.

During a multicast session, if the network condition changes (e.g., some overlay links are broken), the peers may deliver data using an alternate path according to the underlying ALM protocol. In this case, network performances like PLS or RDP may change. However, unless new overlay links are formed, no rekeying is needed. Hence, the nodal processing overhead will not be affected. For example, DT protocol still works even if some of the mesh links are broken.

8. CONCLUSIONS

In order to provide confidentiality in large-group multimedia communications using application layer multicast, packets are encrypted before delivery. As the encryption keys need to be changed upon membership changes, this rekeying leads to processing overhead on each peer. There are two basic approaches to perform data encryption. Host-to-host encryption leads to high reencryption overhead when the data rate (or packet rate) of the application is high, while whole group encryption incurs large rekeying overhead when the user pool is large and dynamic. We propose a solution, Secure Overlay Multicast (SOM), to compromise on the overhead between the two schemes so as to achieve the minimum nodal processing overhead.

SOM is a framework based on clustering multicast group members into subgroups according to their network locations. A cluster key is only shared within the same cluster, hence localizing the rekeying inside the cluster. We describe SOM framework and present a simple analysis of SOM which agrees with our simulation well.

We show that, for a given application, there exists an optimal cluster size for minimizing the nodal processing overhead. SOM is scalable to large and dynamic group. It saves processing overhead substantially (by as much as 30% in our simulation) as compared with the two basic approaches. On the other hand, SOM achieves similar network performance (in terms of physical link stress and relative delay penalty) as compared to an improved version of DT.

REFERENCES

BADISHI, G., KEIDAR, I., AND SASSON, A. 2006. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. *IEEE Trans. Depend. Secure Comput. 3,* 1, 45–61.

BALLARDIE, A., FRANCIS, P., AND CROWCROFT, J. 1993. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. In *Proceedings of ACM SIGCOMM Data Communications Festival*. 85–95.

BANERJEE, S., BHATTACHARJEE, B., AND KOMMAREDDY, C. 2002. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM Data Communications Festival*. 205–217.

BASAGNI, S. 1999. Distributed clustering for ad hoc networks. In *Proceedings of IEEE International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*. 310–315.

CANETTI, R., GARAY, J., ITKIS, G., MICCIANCIO, D., NAOR, M., AND PINKAS, B. 1999. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 2. 708–716.

CHAN, K.-C. AND CHAN, S.-H. 2002. Distributed servers approach for large-scale secure multicast. *IEEE J. Select. Areas Comm. 20,* 8, 1500–1510.

CHAN, K.-C. AND CHAN, S.-H. G. 2003. Key management approaches to offer data confidentiality for secure multicast. *IEEE Netw. 17,* 5, 30–39.

CHANG, I., ENGEL, R., KANDLUR, D., PENDARAKIS, D., AND SAHA, D. 1999. Key management for secure internet multicast using boolean function minimization techniques. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 2. 689–698.

CHU, Y.-H., RAO, S. G., SESHAN, S., AND ZHANG, H. 2002. A case for end system multicast. *IEEE J. Select. Areas Comm. 20,* 8, 1456–1471.

DABEK, F., COX, R., KAASHOEK, F., AND MORRIS, R. 2004. Vivaldi: A decentralized network coordinate system. In *Proceedings of ACM SIGCOMM Data Communications Festival*. Portland, Oregon, USA.

DEERING, S. E., ESTRIN, D., FARINACCI, D., JACOBSON, V., LIU, C.-G., AND WEI, L. 1994. An architecture for wide-area multicast routing. In *Proceedings of ACM SIGCOMM*. 126–135.

GANJAM, A. AND ZHANG, H. 2004. Connectivity restrictions in overlay multicast. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. Cork, Ireland, 54–59.

HUANG, J. AND MISHRA, S. 2003. Mykil: A highly scalable and efficient key distribution protocol for large group multicast. In *Proceedings of IEEE Global Telecommunications Conference*.

JAIN, A. K. AND DUBES, R. C. 1988. *Algorithms for Clustering Data*. Prentice Hall Inc.

KIM, Y., PERRIG, A., AND TSUDIK, G. 2004. Tree-based group key agreement. *ACM Trans. Inform. Syst. Secur. 7,* 1, 60–96.

KLEINROCK, L. 1975. *Queueing Systems: Volume I—Theory*. Wiley Interscience, New York.

KRANAKIS, E., SINGH, H., AND URRUTIA, J. 1999. Compass routing on geometric networks. In *Proceedings of the Canadian Conference on Computer Geometry (CCCG'99)*. 51–54.

LEE, P. P. C., LUI, J. C. S., AND YAU, D. K. Y. 2006. Distributed collaborative key agreement and authentication protocols for dynamic peer groups. *IEEE/ACM Trans. Netw. 14,* 2, 263–276.

LIEBEHERR, J., NAHAS, M., AND SI, W. 2002. Application-layer multicasting with delaunay triangulation overlays. *IEEE J. Select. Areas Comm. 20,* 8, 1472–1488.

LIM, H., HOU, J. C., AND CHOIPCA, C.-H. 2005. Constructing internet coordinate system based on delay measurement. *IEEE/ACM Trans. Netw. 13,* 3, 513–525.

LIN, C. R. AND GERLA, M. 1997. Adaptive clustering for mobile wireless networks. *IEEE J. Select. Areas Comm. 15,* 7.

LIN, H.-C. AND CHU, Y.-H. 2000. A clustering technique for large multihop mobile wireless networks. In *Proceedings of IEEE Vehicular Technology Conference*. Vol. 2. 1545–1549.

MATHY, L., BLUNDELL, N., ROCA, V., AND EL-SAYED, A. 2004. Impact of simple cheating in application-level multicast. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*.

MITTRA, S. 1997. Iolus: A framework for scalable secure multicasting. In *Proceedings of ACM SIGCOMM Data Communications Festival*. 277–288.

NG, T. S. E. AND ZHANG, H. 2002. Predicting internet network distance with coordinates-based approaches. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 1. 170–179.

SAHNI, S. 1998. *Data Structures, Algorithms and Applications in C++*. McGraw-Hill.

SHIELDS, C. AND GARCIA-LUNA-ACEVES, J. J. 1999. KHIP - A scalable protocol for secure multicast routing. In *Proceedings of ACM SIGCOMM Data Communications Festival*. 53–64.

SRIPANIDKULCHAI, K., GANJAM, A., MAGGS, B., AND ZHANG, H. 2004. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proceedings of ACM SIGCOMM*. Portland, OR.

TRAN, D. A., HUA, K. A., AND DO, T. 2003. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*. San Francisco, CA.

WAITZMAN, D., PARTRIDGE, C., AND DEERING, S. E. 1988. Distance vector multicast routing protocol (DVMRP). RFC 1075.

WONG, C. K., GOUDA, M., AND LAM, S. S. 2000. Secure group commmunications using key graphs. *IEEE/ACM Trans. Netw. 8*, 16–29.

WONG, W.-C. AND CHAN, S.-H. G. 2003. Improving delaunay triangulation for application-level multicast. In *Proceedings of IEEE Global Telecommunications Conference*.

YIU, W.-P. K. AND CHAN, S.-H. G. 2004. SOT: Secure overlay tree for application-layer multicast. In *Proceedings of IEEE International Conference on Communications (ICC)*. Paris, France, 1451–1455.

YIU, W.-P. K., WONG, K.-F. S., CHAN, S.-H. G., WONG, W.-C. W., ZHANG, Q., ZHU, W.-W., AND ZHANG, Y.-Q. 2006. Lateral error recovery for media streaming in application-level multicast. *IEEE Trans. Multimed. 8,* 2.

ZEGURA, E. W., CALVERT, K. L., AND BHATTACHARJEE, S. 1996. How to model an internetwork. In *Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 2. San Francisco, CA, 594–602.

ZHU, Y., LI, B., AND GUO, J. 2004. Multicast with network coding in application-layer overlay networks. *IEEE J. Select. Areas Comm. 22,* 1, 107–120.

Received September 2007; revised December 2007; accepted March 2008