Scalable Secure Group Communication over IP Multicast

Suman Banerjee, Bobby Bhattacharjee
Department of Computer Science, University of Maryland, College Park, MD 20742

Abstract

We introduce and analyze a scalable re-keying scheme for implementing secure group communications IP multicast. We show that our scheme incurs constant processing, message, and storage overhead for a re-key operation when a single member joins or leaves the group, and logarithmic overhead for bulk simultaneous changes to the group membership. These bounds hold even when group dynamics are not known *a-priori*.

Our re-keying algorithm requires a particular clustering of the members of the secure multicast group. We describe a protocol to achieve such clustering and show that it is feasible to efficiently cluster members over realistic Internet-like topologies. We evaluate the overhead of our own re-keying scheme and also of previously published schemes via simulation over an Internet topology map containing over 280,000 routers. Through analysis and detailed simulations, we show that this re-keying scheme performs better than previous schemes for a single change to group membership. Further, for bulk changes, our algorithm outperforms all previously known schemes by several *orders of magnitude* in terms of actual bandwidth usage, processing costs and storage requirements.

1 Introduction

IP multicast enables scalable wide-area multi-party applications over the Internet. In this paper, we describe a new algorithm for scalable, secure group communication over IP multicast. Our algorithm can be implemented over base IP multicast and does not require router support beyond best-effort forwarding. Since our scheme is completely end-host based, it can be used to implement group security over IP multicast-enabled untrusted, insecure networks.

1.1 Group Keys and Re-keying Groups

Many secure group communication systems [9, 1, 16, 6, 3, 12], including ours, rely on the notion of a "group key" — a secret known only to the members of the secure communication group[1]. Once a group key is distributed to all current members of the multicast group, secure messages can be sent encrypted with the group key. The overall security of the group depends wholly on the secrecy and the strength of the group key. Since every group member has the group key, sending a message involves only a single encryption at the sender and a single decryption at each receiver. The routers on the way treat each message no different than any other IP multicast datagram. The only problem left to solve is to scalably and securely establish a group key known to all (and only) the members of the secure multicast group.

Since we assume the network infrastructure is insecure, it is possible for non-members to eavesdrop on the multicast group and store encrypted messages (that they cannot decrypt). It is also possible for members

[1] It is possible to construct secure group communication systems without using a secret shared between all members. In such systems, trusted intermediaries, e.g. secure routers, must encrypt and decrypt messages en-route.

who have left the group to continue to decrypt messages and for new members to decrypt messages they had stored previously. Therefore, during each membership change, a *new* group key must be distributed and all subsequent communication must use this new key. This is the process of *group re-keying*: establishing a new group key upon a membership change in the secure multicast group. Note that depending on the requirements of the application, it may or may not be necessary to re-key the group when a new member joins; but it is almost always necessary to re-key the group when a member leaves.

Clearly, the overhead of the re-keying can be reduced (at the cost of reduced security) by *batching* the re-key operations, e.g. not re-keying every time there is a membership change but re-keying periodically or when a the group membership has changed sufficiently. Batching is important since re-keying operations affect every single member in the group, and can potentially be very expensive for large groups. Thus, re-keying schemes designed for large groups must be efficient when handling bulk simultaneous changes to the group membership.

The simplest solution for re-keying involves a pair-wise secure exchange of the group key between a central key server and each group member [9]. Unfortunately, this scheme incurs a $O(N)$ overhead, where N is the number of group members, and is not viable for large groups. A particularly elegant protocol using hierarchical "key graphs" was introduced in [16]. This is the first protocol that describes a scheme that incurs sub-linear overhead for single membership changes to the group. In [6], a different scheme using boolean minimization techniques is described that is efficient for bulk membership changes. Both these schemes reduce the overhead for the group re-keying operation for single membership change to $O(\log N)$. If more information is known about group dynamics, then it is possible to do better. MARKS [3] is a scheme that assumes that the duration over which a member stays attached to the group is known at the time the member joins. Using this information a constant overhead solution is presented in [3]. In this paper, we present a constant processing, message, and storage overhead solution for the general problem when the membership durations are not known; to the best of our knowledge, ours is the first scheme with a provable constant bound. Further, we show that our scheme can handle $O(N)$ simultaneous changes to the group membership in $O(\log N)$ processing, and bandwidth complexity; this is a significant improvement on the previously known $O(N)$ bounds [6]. Obviously, the reduction of re-keying costs for single membership change from $O(\log N)$ to $O(1)$ is not very important unless the size of the group is very large. In simulations, our re-keying scheme performs better than previous schemes for single group changes. However, for bulk group changes, our algorithm outperforms all previously known schemes by *orders of magnitude* in terms of actual bandwidth usage, processing costs and storage requirements.

1.2 Overview of Our Approach

Our scheme is based on a particular sized bounded, non-overlapping, clustering of multicast group members, which we call "spatial clustering". Spatial clustering assures that members in the same cluster are near each other in the multicast tree. The constant overhead re-keying scheme is implemented using a hierarchy of spatial clusters. Our re-keying scheme has low communication overhead precisely because it is based upon a spatial partitioning of the multicast group which allows the key distribution scheme to exploit the parallelism inherent in different parts of a multicast tree.

This paper has two main contributions:

- We describe the an efficient re-keying algorithm for implementing group security over IP multicast. Our analysis shows that this scheme has constant overhead for a single group re-key even when the group dynamics are not known *a-priori*. Further, we show that the overhead for bulk simultaneous changes is logarithmic in the number of group members. Both these results improve on previously known bounds.
- We present detailed simulation of our own scheme, and also of previously published schemes including [16, 6]. Compared to previously published order analysis, our simulation results better measure the per router packet and bandwidth overheads inherent in implementing secure group communication schemes over IP multicast.

We also describe an algorithm based on IP multicast for creating bounded-size non-overlapping clusters. Such clustering is potentially useful beyond secure multicast; however, we do not explore this topic further in this paper.

1.3 Roadmap

In the next section, we describe existing approaches for key distribution for secure multicast. In Section 3, we describe our multicast key distribution scheme. In Section 4, we describe the underlying clustering algorithm required to implement our re-keying scheme. In Section 5, we present simulation results comparing our scheme versus existing approaches and conclude in Section 6. We present a formal protocol specification and asymptotic complexity analysis for the re-keying algorithm in an Appendix.

2 Existing Approaches for Secure Multicast

Group Key Management Protocol (GKMP) [9] is a simple group management protocol in which a *Group Key Controller* is responsible for generating group keys. In GKMP, static keys —pairwise shared between the *Group Key Controller* and each group member— are used to establish a group key. The Scalable Multicast Key Distribution (SMKD) [1] works in conjunction with the Core Based Tree (CBT) [2] multicast protocol to securely distribute the multicast group key. The CBT root router initially operates as the entity responsible for generation and distribution of the group key. This responsibility is delegated to other routers as they join the delivery tree. SMKD requires explicit router support, and does not scalably solve the problem of group re-keying.

In Iolus [12], the scalability of re-keying is handled by dividing the secure multicast group into multiple sub-groups. Security in each sub-group is managed by a Group Security Agent (GSA). The GSAs (and by consequence, the sub-groups) are *statically configured* and located in different parts of the Internet. Each sub-group has its own sub-group key that is managed by the relevant GSA. Membership changes in any sub-group require local re-keying of the sub-group key. If a global shared group key is used for the data, this key needs to be distributed to members in each sub-group by sub-group specific means. Alternatively, if no global group key is used, changes in one sub-group do not affect the others, but expensive data-path encryptions and decryptions are necessary. Since Iolus does not define sizes of subgroups etc., it is difficult to provide analytic bounds on Iolus' performance (and hence it is not included in the simulation results). It is, however, possible to use an Iolus-like infrastructure to implement our scheme, and we discuss the details of such an implementation in Section 4.3.

As mentioned in Section 1, the MARKS [3] scheme defines a constant overhead key distribution protocol. More precisely, members in MARKS incur a one time cost which depends on the length of the time they stay in the group. MARKS is based upon the premise that many applications, e.g. pre-paid or subscription pay-TV or pay-per-view, do not (or rarely) require premature eviction. Thus, the protocol assumes that the duration over which a member stays in the group is known when the member joins. For single changes to the group, our scheme provides better performance bounds without the known *a-priori* membership duration requirement.

The two protocols that are of most interest to us are the Key Graphs scheme by Wong, Gouda, and Lam [16], and the Boolean Minimization scheme by Chang, Engel, Kandlur, Pendarakis, and Saha [6]. Both these protocols and our protocol solve the re-keying problem under the same assumptions (unlike MARKS), and we outline their operation in the next section. We are aware of two extensions to the Key Graphs scheme: the scheme in [5] uses input doubling functions to reduce number of re-keying messages in half, and the scheme in [11] describe how to effectively re-key during bulk membership changes. In our simulations, we have implemented the protocols described in [11].

2.1 Key Graphs and Boolean Minimization

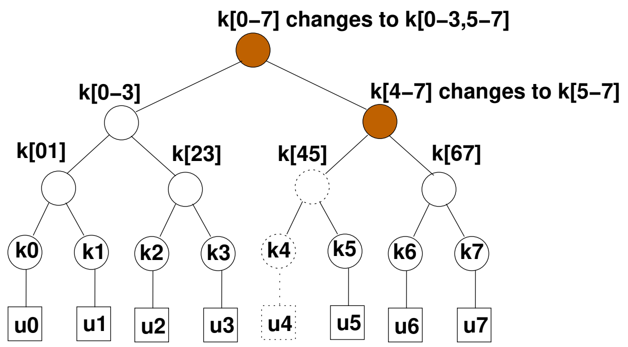

Figure 1: Key Graphs Scheme

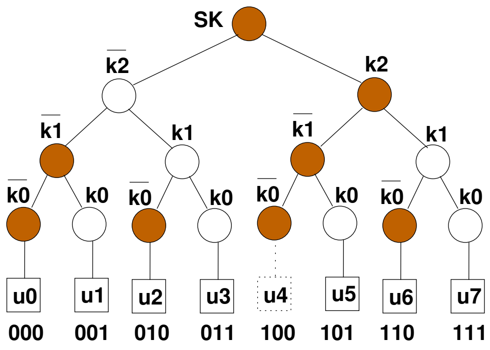

Figure 2: Boolean Minimization Scheme

The first secure multicast protocol that did not incur linear re-keying cost for single membership change, was proposed by Wong et al [16]. They define a tree hierarchy *of keys* distributed between different sets of members, as shown in Figure 1. The leaf nodes on the tree represent the different members, while the circular nodes represent the different keys. Each member u_i possesses all the keys on its path to the root (from $k_i \rightarrow k[0-7]$ in Figure 1). The root key serves as the group key. A central key server is responsible for generating and distributing new keys as required. When a member (e.g. u_4 in the figure) leaves the group, all keys on the path from this member to the root needs to be changed. However, the updated keys can be multicast by the key server to the sub-groups instead of being unicast to individual members separately, and thus the re-keying cost is $O(\log N)$. In [16] different re-keying —key-, user-, and group-oriented— schemes are described, each with different processing, and message overheads. We have implemented all of these schemes in our simulation, and in our comparisons, we use the re-keying scheme that is most favorable to the Key Graphs scheme for the particular experiment.

The boolean minimization technique [6] is another scheme that uses virtual key hierarchy-based scheme. There are $2 \log N$ auxiliary keys, namely $\{k_0, k_1, \ldots\}$ and $\{\overline{k_0}, \overline{k_1}, \ldots\}$ [2] and one group key SK. If $b_R \ldots b_1, b_0$ denotes the identifier of a member in binary notation, then that member has key SK and the auxiliary keys given by the following rule: For each i, if $b_i = 0$, then the member has key $\overline{k_i}$, else it has key k_i. This set of auxiliary keys held by each member can be represented using a binary key tree as shown in Figure 2. When a member leaves the group, all group key along with the auxiliary keys held by the member need to be changed. A new group key is generated by a group controller and is encrypted with the keys that are *complementary* to the keys held by the departing member. In the example in Figure 2, member u_4 leaves: the new group key SK' are therefore encrypted as $\{SK'\}_{\overline{k_2}}$, $\{SK'\}_{k_1}$, $\{SK'\}_{k_0}$ and distributed to the entire group. All and only the remaining members are able to decrypt the new group key. Further, a one-way hash function on the new group key is used to update the auxiliary keys known to u_4. Using boolean minimization techniques to determine how the group key is encrypted for distribution, this scheme is able to handle bulk membership changes more scalably than Key Graphs.

3 Secure Multicast using Clustering

In this section, we present our secure key distribution algorithm. We assume a member clustering protocol that maps multicast group members to clusters with the following properties:

> *Each cluster has between k and $2k-1$ members, for some fixed k, and consist of connected sets of tree members. Cluster topologies are non-overlapping, and no two clusters share a member.*

The actual clustering protocol we describe in Section 4 does not ensure zero overlap, but instead guarantees that two clusters share no more than one member. This is an artifact of pathological configurations of multicast trees for which it is not possible to maintain the both the size bound and the zero overlap condition. Even this weaker condition, however, is enough to guarantee the clustering properties as required in our analysis of the re-keying algorithm. (We do require that the maximum degree of the multicast delivery tree be independent of the group size).

3.1 Member Hierarchy for Key Distribution

Our key distribution scheme creates a member hierarchy as shown in the leftmost panel of Figure 3. A "layer" comprises of a set of members of the secure multicast group in the same level of the hierarchy. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (denote by L_0). An instance of the clustering protocol is executed at each layer of the hierarchy to create a set of clusters, and all members of the secure multicast group are part of the lowest layer (L_0). The cluster leaders of all the clusters in layer

[2] Note that the keys k_x and $\overline{k_x}$ are not bitwise complements of each other, instead they represent two completely different keys.

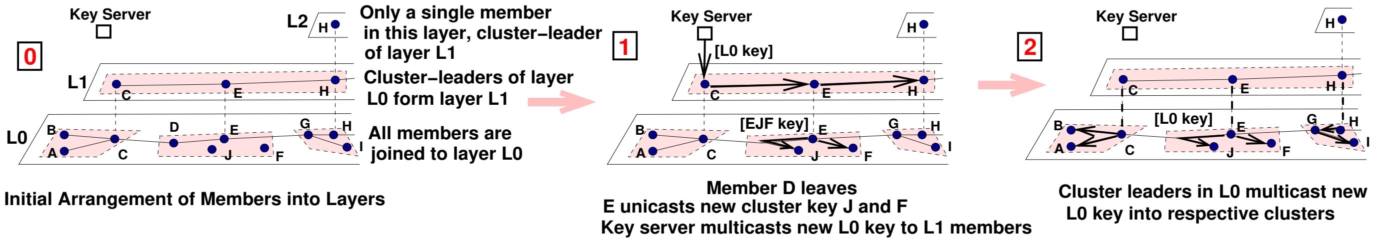

Figure 3: Key distribution on a three layer hierarchy of members on the multicast delivery tree.

L_i join layer L_{i+1}. For example, consider the arrangement of members in the initial configuration (Panel 0) of Figure 3. All ten members A–J are part of layer L_0. The clustering protocol has partitioned L_0 into three clusters: [*ABC*], [*DEFJ*], and [*GHI*][3]. The cluster leaders, C, E and H join layer L_1. Another instance of the clustering protocol executes at layer L_1 to create the single cluster [*CEH*]. The leader, H, of the layer L_1 cluster joins layer L_2 — the highest layer in this example. The procedure terminates when there is only a single member in any layer.

When a new member joins any layer, the clustering protocol places it into one of the clusters in that layer. Occasionally, arrival of a new member or the departure of an existing member from a layer can split or merge clusters. This decision is part of the clustering protocol.

3.2 Layer Keys, Cluster Keys, Key Servers and Group Communication

A secret *layer key* is associated with each layer of the hierarchy. A group member possesses a layer key for a specific layer if and only if it is a member of a cluster in that layer. Layer keys are generated, on-demand, by a *key server* whenever a new member joins or an existing member leaves any layer. A secret *cluster key* is associated with each cluster. Once again, a group member possesses a cluster key for a specific cluster if and only if it is a member of that cluster. The leader of each cluster is responsible for generating the cluster key for that cluster. Finally, in all clusters, a pair-wise key is shared between the cluster-leader and each cluster member. Since all members belong to L_0, the key for L_0 is used as a shared key for secure communication.

3.3 Key Distribution Protocol

The key distribution protocol ensures that *the layer key of each layer is only available to members joined to that layer.* Therefore, whenever a member leaves (or joins) a layer, a new layer key is required for that layer. This ensures that the layer key of layer L_0 (which is the group key for the entire secure multicast group) is available only to members of L_0, i.e. all and only the current members of the multicast group. In the rest of this section, we use three examples to illustrate how our protocol efficiently changes layer keys and maintains security guarantees. In the examples, we assume that the cluster sizes must be bounded between 3 and 5. A formal specification for the protocol is given in the Appendix.

[3] In referring to clusters in Figures 3 and 4 , we use the notation [*XY*...*Z*] to refer to a cluster with members X, Y, ..., Z.

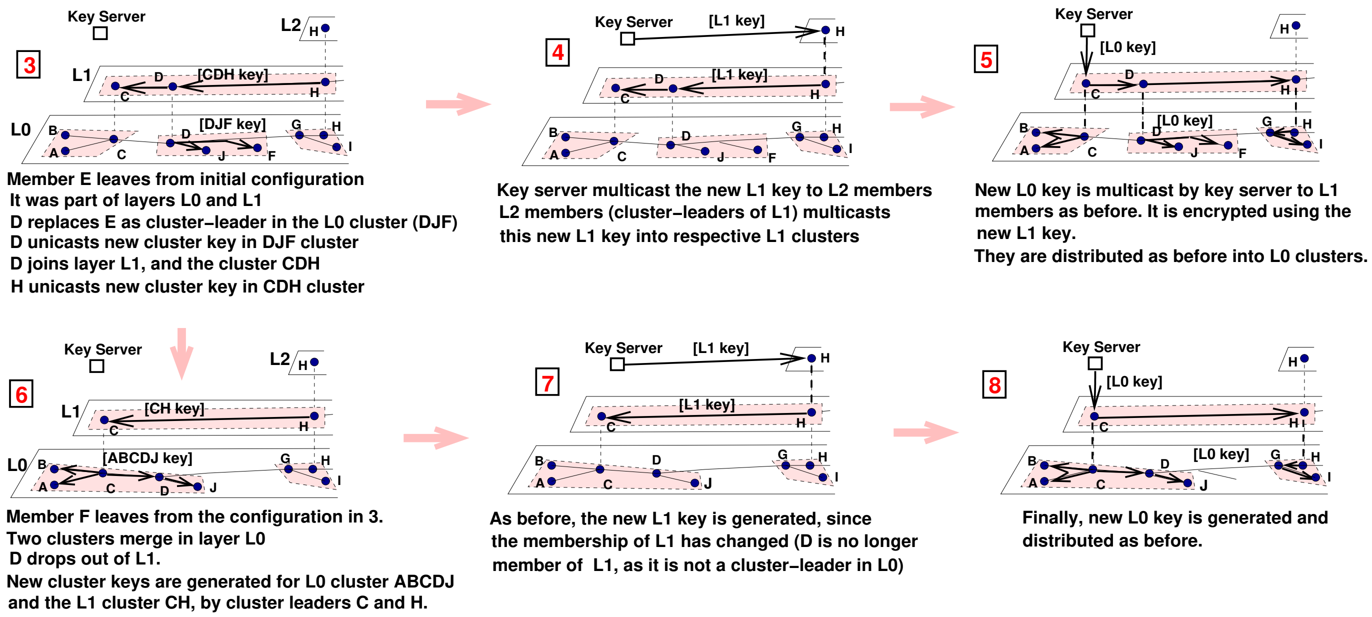

Figure 4: Examples illustrating the key distribution scheme.

Example I: Member departure In Figure 3, we first consider the case that member D leaves the secure multicast group. Since D belongs only to L_0, only the L_0 layer key needs to be changed. The following are the re-keying operations required :

1. A new L_0 layer key request is made by E, the cluster leader of [*DEFJ*][4].Simultaneously, E sets up a new cluster key by pair-wise communication with each of the remaining members of this cluster (in this case F and J), as shown in Panel 1.

2. The key server generates the new L_0 layer key, and multicasts the new L_0 layer key to all members in the immediate higher layer, i.e. L_1, encrypted by the current L_1 layer key (Panel 1). (Recall that the members of layer L_1 are the cluster leaders of the clusters in layer L_0.)

3. On receiving the new L_0 layer key, the members of layer L_1 (i.e. C, E and H) extract the new layer L_0 key, and multicast it to the other members of their clusters in layer L_0, encrypted by the respective current cluster keys. Note that when member E transmits the new L_0 layer key in its L_0 cluster (now comprising of E, F and J), it uses the new cluster key that it had set up thus ensuring D cannot decrypt the new layer key. *The non-overlapping nature of the clusters ensure that the cluster multicasts occur in parallel and traverse disjoint parts of the delivery tree.*

[4]If E leaves the group at the same time as D, then E's departure would cause H, the cluster leader of cluster [*CEH*] in layer L_1, to initiate new layer key requests for both layers L_0 and L_1.

Member Join A joining member is assigned to a L_0 cluster by the clustering protocol at layer L_0. The cluster leader in this cluster generates and distributes a new cluster key. Finally, the new L_0 layer key is distributed, as explained steps 2 and 3 in the previous example.

Example II: Cluster Leader Departure In our next example, we refer to Figure 4, and consider the case when E, a cluster leader in layer L_0 and a member of both L_0 and L_1, leaves the original multicast group. The re-keying operations proceeds as follows :

1. Some member, D, in this example, is chosen as the new cluster leader of the L_0 cluster ([*DFJ*]). Since it is a leader in L_0, D joins layer L_1. The two affected clusters [*DFJ*] in layer L_0 and [*CDH*] in layer L_1 require new cluster keys. As shown in Panel 3, these keys are established by the cluster leaders by pair-wise unicast communication, using the respective pair-wise keys.

2. New layer keys, for both layers L_0 and L_1, are requested by the cluster leader (H) of the highest affected cluster ([*CDH*]). The new L_1 key is unicast to other members of [*CDH*] by H, since it is the leader of an affected cluster. The new L_0 key is multicast to layer L_1 encrypted using the new L_1 key. It is then multicast, using corresponding cluster keys, onto clusters [*ABC*], [*GHI*], and [*DJF*] by C, H, and D, respectively

It should be clear that whenever there is a change in membership in any specific layer L_j for $j > 0$, there is a corresponding change in all lower layers L_i, $0 \leq i < j$. Thus, in this example, H, which is the leader of the affected cluster in the highest layer, immediately requests a new layer key for L_1 and all lower layers.

Example III: Cluster Reconfiguration As a final example, we consider the case, when member F leaves the group from the configuration in Figure 4, Panel 3 (member E has already left). In this case, the DFJ cluster in layer L_0 shrinks to two members (D and J), violating size lower bound of 3. As a consequence, the clustering protocol merges the clusters DJ and ABC to create a cluster within the required size bound. Since, D is no longer a leader of a L_0 cluster, it must be removed from layer L_1. Both layers L_0 and L_1 are rekeyed as in the previous example.

3.4 Complexity Analysis

Along with the formal protocol specification, we have presented the complexity proofs for our algorithm in the Appendix. We show that the processing costs of our scheme for single membership change to the group and the number of keys stored at each member is of constant order. Leveraging the non-overlapping property of the clusters, we show that the communication cost per link on the multicast delivery tree is also constant for a single membership change. More importantly, when we batch process multiple leaves (or equivalently, there are multiple simulataneous leaves in the group), the processing cost at the key server and the messaging overheads have logarithmic bounds, which is a significant improvement over previously known bounds. We summarize our results and previously known results in Table 1.

Scheme	Storage		Processing (single change)		Processing ($O(N)$ change)		Net. Load (Bytes/router)	
	Member	Key Server	Member	Key Server	Member	Key Server	Single change	$O(N)$ change
GKMP	2	$N+1$	1	N	$O(1)$	$O(N)$	$O(N)$	$O(N)$
Key Graphs	$\log_d N + 1$	$Nd/(d-1)$	≤ 2	$d(\log_d N - 1)$	$O(\log N)$	$O(N)$	$O(d \log N)$	$O(N)$
Bool. Min.	$\log_2 N + 1$	$2\log_2 N + 1$	1	$\log_2 N$	$O(1)$	$O(N)$	$O(\log N)$	$O(N)$
Spatial Clustering	≤ 4	$\log_k N$	≤ 2	≤ 2	$O(1)$	$O(\log N)$	$O(1)$	$O(\log N)$

Table 1: Average costs for different secure multicast key distribution schemes. For $O(N)$, i.e. bulk, changes to the group membership the overhead costs are typically maximized when $N/2$ members simultaneously leave the group.

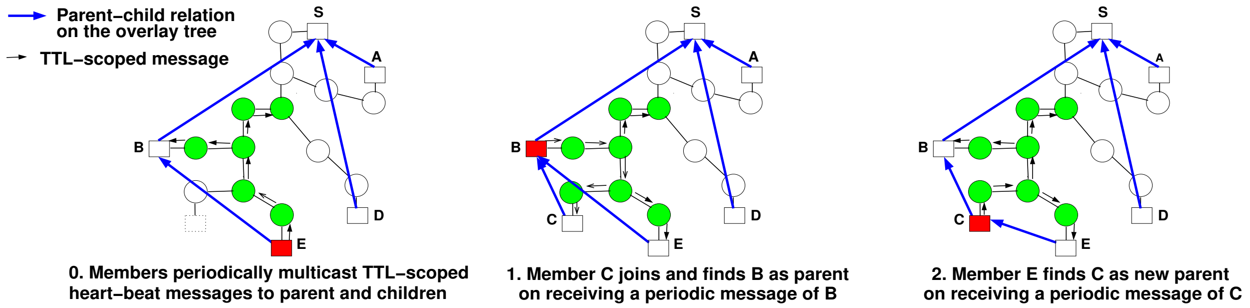

Figure 5: Member discovery protocol example

4 Spatial Clustering

In this section, we outline an algorithm to partition the members of a multicast group into fixed size, (mostly) non-overlapping clusters, as required by the re-keying algorithm. The input to the clustering algorithm is a *member overlay tree* which contains only the multicast group members as nodes. As members join the multicast group, we use a *member discovery protocol* to establish a parent for each new member in the member overlay tree. We describe the member discovery protocol next and the clustering protocol in Section 4.2.

4.1 Member Discovery Protocol

The member discovery protocol takes a multicast topology as input, and outputs a member overlay tree. The member discovery protocol defines parent-child relationships among the different members of the multicast tree. This is the only component that is inherently tied to the network layer multicast scheme being used. Network layer multicast schemes create data delivery trees, which are broadly classified to be either source-based (DVMRP [15]) or shared (CBT [2]) trees, each of which can be unidirectional (PIM-SM [7]) or bi-directional (CBT). We have defined different member discovery protocols tailored for each of these different network layer multicast schemes. In this paper, we focus on a network layer multicast scheme that creates shared bi-directional trees (e.g. Core Based Tree protocol [2]) and only describe the appropriate member discovery protocol. This member discovery protocol uses mechanisms similar to the low overhead technique of fault isolation in multicast trees [13]. Let $d(u, v)$ denote the distance, in router hops, between the members x and y along the multicast delivery with source S. A member y is considered to be a parent of member x, if and only if the following two conditions hold:

C1: $d(S, y) \leq d(S, x)$. This condition ensures that the parent is closer to the source than the child.

C2: $\forall\, z$ that satisfy **C1**, $d(y, x) \leq d(z, x)$. This condition chooses the closest member that satisfies condition **C1**.[5]

The protocol uses two periodic messages: the root S periodically multicasts a heartbeat packet to all the members of the group, from which each member infers their distance to S. Periodically each member, x, multicasts a TTL-scoped heartbeat message to parent and all of its children on the overlay tree. This message carries the tuple $\langle d(S, x), P(x) \rangle$, where $P(x)$ denotes x's parent in the overlay tree.

We explain the tree construction using an example: Consider the network in Figure 5. In Panel 0, *E* sends out a message with TTL five to reach its parent B. Assume a new member *C* joins the multicast group (as shown in Panel 1). Since *C* is part of the multicast group, the TTL-scoped message from *B* reaches *C*. Node *C* is able to infer the multicast distance between *B* and *C* and hence, using the two specified rules, concludes that *B* is its parent. If, however, the original messages from *B* does not reach *C*, then after a timeout, *C* initiates an Expanding Ring Search to locate a parent. Upon receiving a query from *C*, *B* updates its TTL-scope value such that its next heartbeat reaches *C*.

Finally, when *C* sends its periodic heartbeat scoped to reach *B* (Panel 2), it also reaches *E*. From condition **C1**, *E* realizes that *C* is its new parent on the overlay tree and adjust its TTL-scoping accordingly. *B* stops getting heartbeats from *E* and concludes that *E* is no longer its child. Note that even if the TTL-scoped message from *C* did not reach *E*, the heartbeat message from *E*, scoped to reach its current parent *B*, is guaranteed to reach *C*. *C* would realize that it is a better candidate to be *E*'s parent and would adjust the TTL-scope of its heartbeat to reach *E*.

4.2 Clustering Protocol

The clustering protocol takes an integer k and a member overlay tree as input, and outputs clusters which are (mostly) non-overlapping, connected subsets of the tree, each between size k and $2k$. To create the clusters, the tree is logically traversed from the leaves, upwards to the root. In this traversal, whenever a set of members that fall within the size bounds is detected, they are grouped into a cluster. These members are considered pruned from the overlay tree for the remaining traversal. When this procedure terminates, it is possible that a single cluster, located at the root of the overlay tree, may have size less than k. Additionally, this protocol also guarantees that two clusters share no more than one member.

[5]For a set of members that are equidistant from the source and each other, multiple members may satisfy both the conditions. In such cases, we use a simple lexicographic symmetry breaking technique to prevent loops.

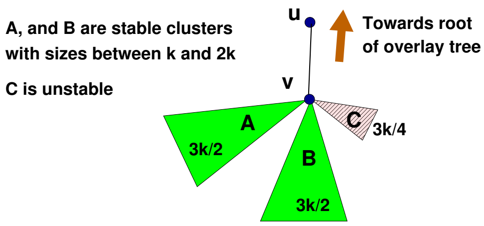

Figure 6: Stable clusters and unstable subtrees.

Procedure : $\textsc{MergeSubtrees}(u, STRootList, k)$
$TempClusters \leftarrow \perp; ThisCluster \leftarrow \perp$
for $v \in STRootList$
  $ThisCluster \leftarrow ThisCluster \cup T(v)$
  if $|ThisCluster| \geq k-1$
    Add $\{u\}$ to $ThisCluster$
    Add $\{ThisCluster\}$ to $TempClusters$
    $ThisCluster \leftarrow \perp$
if $ThisCluster \neq \perp$
  $\tau_u \leftarrow ThisCluster$
return $TempClusters$

Figure 7: Procedure $\textsc{MergeSubtrees}$.

Procedure: $\textsc{ProcessChildMessage}(u, v, |\tau_v|_{new})$
$C_u \leftarrow \perp;\ \Delta_v \leftarrow |\tau_v|_{new} - |\tau_v|_{old}$
if v already belongs to some cluster *C* rooted at u
  Update cluster size of *C* to be $|C| + \Delta_v$
  if $|C| < k$ or $|C| \geq 2k$ { *C is disbanded* }
    $C_u \leftarrow \{w | w \in Children(u) \wedge w \in (C \cup \tau_u)\}$
else { *v was not part of any cluster rooted at u before, but may have been part of τ_u before* }
  { *Add τ_v to the unstable subtree, τ_u at u* }
  Update size of τ_u to be $|\tau_u| + \Delta_v$
  if $|\tau_u| \geq k$
    $C_u \leftarrow \{w | w \in Children(u) \wedge w \in \tau_u\}$
  else
    Merge τ_u with an existing cluster if possible
if $C_u \neq \perp$
  New Clusters rooted at $v \leftarrow \textsc{MergeSubtrees}(u, C_u, k)$

Figure 8: Procedure $\textsc{ProcessChildMessage}$. C_u is a list of u's children s.t. the unstable subtrees rooted at these children need to be merged to create new clusters at u. Δ_v is the change in the subtree size between the previous message from child v and this message.

4.2.1 Protocol Description

We refer to clusters of size between k and $2k$ as *stable* clusters. Clusters of size less than k or greater than $2k$ may occur before the protocol stabilizes: we call these *unstable* clusters. In figure 6, A and *B* are stable clusters rooted at v.

Let τ_v denote the subtree, rooted at some node v which cannot be joined to any cluster rooted at v (doing so would render these clusters unstable). This subtree, which we call the *unstable subtree*, has to be joined to a cluster that is rooted at a node upstream of node v. In Figure 6, $\tau_v = C$. When the protocol stabilizes, *C* would be part of a cluster rooted at a node upstream of v or be part of the single unstable cluster rooted at the root of the overlay tree.

Protocol Operation The clustering protocol proceeds as follows:

- Initially when a member u joins the multicast group, it creates an unstable cluster, comprising only of itself. This cluster is also an unstable subtree, i.e. $\tau_u = \{u\}$.
- Each member u periodically sends a message to its parent containing the value $|\tau_u|$.
- The periodic message from child v to parent u is either a notification of a new unstable subtree rooted at v or of an existing unstable subtree rooted at v.

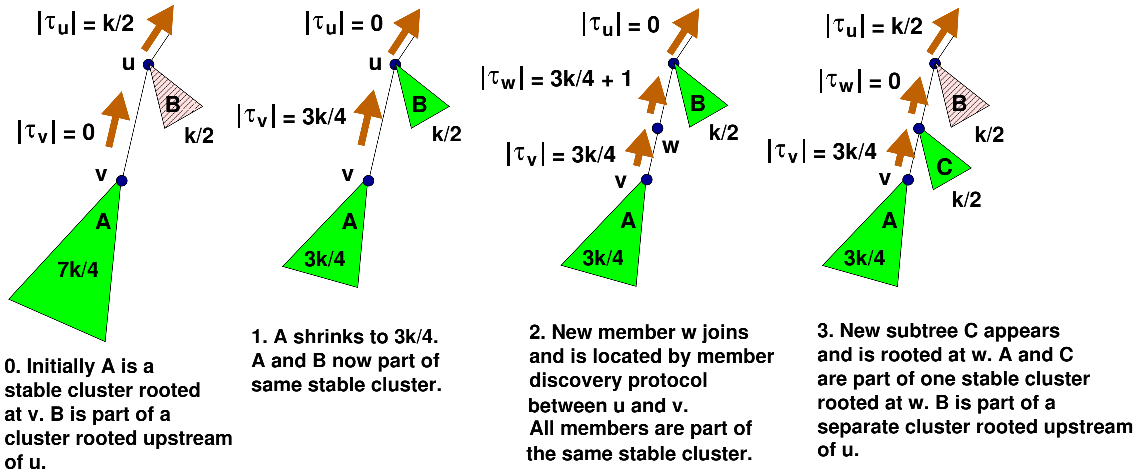

Figure 9: Clustering protocol example

If τ_v is an previously known subtree, then it part of some existing cluster (or the unstable subtree) rooted at u. In this case, u checks to see if the size of τ_v has changed. If the size has changed sufficiently, u may have to split or merge a cluster that τ_v is part of. If τ_v is part of the unstable subtree, u may now be able to create a new stable cluster.

If the message is for previously unknown subtree, this new subtree is added to the unstable subtree rooted at u. Node u tries to merge the its new upstream cluster with its existing clusters. This procedure may cause a new stable cluster, rooted at u to be formed. All subtrees that cannot be put into any cluster form the new unstable subtree rooted at u.

We list the exact procedure for handling child messages in Figure 8.

Example We illustrate the operation of the clustering protocol with the example shown in Figure 9.

- Initially, the overlay tree has a cluster A rooted at the member v. Member v has no unstable subtree, and sends $|\tau_v| = 0$ to its parent u. Member u has an unstable subtree, *B*, which is part of a cluster rooted upstream from u.

- At a later time (Panel 1), the cluster A reduces below size k, making it unstable and is advertised by v as an unstable subtree. As a consequence, at u, $|\tau_u| = 3k/4 + k/2 \geq k$. The procedure MERGESUBTREES is called on the two subtrees A and *B* to create one single cluster. No unstable cluster is created rooted at u. So τ_u becomes empty.

- Next (as shown in Panel 2), a new member w joins the overlay tree. The member discovery protocol locates it between u and v but there is no change in the clusters.

- Eventually (see Panel 3), an entire subtree *C* of size $k/2$ joins w. This causes $|\tau_w|$ to increase beyond k and leads to the creation of a single cluster including subtrees A and *C*. The new τ_w becomes empty, and the subtree *B* again becomes a unstable subtree at u.

4.3 Implementing Secure Multicast over Clustering

We have implemented, in detailed simulations, our re-keying scheme over the clustering protocol described here. Clearly, there is an added overhead for clustering that our solution incurs that is not present in other schemes. Further, in our solution, certain members have to act as cluster leaders, potentially leading to added trust and security issues. We address these two concerns in the rest of this section.

Clustering Overheads The main overhead of clustering is the time it takes for clusters to stabilize, which is, in turn depends on the heartbeat period for the member discovery protocol. In Section 5.3.5, we show that clusters stabilize relatively quickly, e.g. in less than 2 seconds for clusters of size 20 for the topologies we consider. The stabilization time can be reduced further by using an adaptive heartbeat period algorithm that sends control traffic more frequently while clusters are forming or changing. Lastly, we note that our experiments show that the cluster stabilization time is independent of group size and depends primarily on the size and depth of clusters. We believe it is possible to control the heartbeat period and the cluster size to keep cluster stabilization time manageable for most applications.

Trust Since we implement the re-keying algorithm on an overlay tree, there is a certain amount of inter-member cooperation that we assume, e.g. some members may have to incur higher costs than other members since they are a cluster leader. It is possible to implement our scheme more efficiently using infrastructure support, e.g. by using designated in-network agents that act as cluster leaders, much like "Group Security Agents" in Iolus. However, our end-point based implementation is not any less secure. Cluster leaders do not have any special advantages; specifically, it is easy to ensure that cluster leaders do not distribute an incorrect or old key to cluster members during a layer re-key. This is ensured by requiring the key server to digitally sign the layer keys. The cluster leaders re-distribute these digitally signed keys within their clusters. Additionally, the key server periodically multicasts the identity of the most current layer key of a layer that it has distributed, to all members of the layer. This allows the members to verify that they have the correct layer key.

5 Simulation Experiments

In Table 1, we presented theoretical bounds comparing various key distribution schemes for secure multicast. In this section, we present simulation results to precisely quantify and compare the actual overheads for the different schemes.

5.1 Experimental Setup

We begin with a description of the topology we used to conduct our experiments, and describe the implementation of the various schemes.

Network Topology and Multicast Support We experimented the key distribution performance of the different schemes on a large Internet map obtained by the SCAN project[6]. This map, created using the Mercator [8] tool, contains about $280,000$ IP routers discovered using traces on the Internet. About 50% of all the routers in the map were edge routers, to which we attached between $10,000$ and $500,000$ hosts routers uniformly at random for different experiments.

We have simulated network infrastructures that do and do not support *directed multicast*. Directed multicast allows a sender multicast a packet to individual subtree(s) rooted at a specific router on the multicast delivery tree. This is not currently a part of the IP multicast standard, but is extremely useful for many network services, e.g. NAK suppression for reliable multicast. It has been proposed in research, e.g. AIM [10], and is currently being considered by the IETF as part of the PGM [14], and GRA [4] efforts. In our experiments, many re-keying schemes show marked performance improvements when implemented over a directed multicast capable network.

In the absence of directed multicast, the same effect can the achieved by using a different multicast address for each subset of nodes that have to be addressed. For our scheme, which requires addressing clusters individually, and for some key graph schemes, using a different address for each addressable subgroup is not viable since the number of multicast addresses required would be on the order of the number of receivers. Instead, we use TTL-scoping to limit packets to a certain part of the multicast tree. The network load for using such *scoped multicast* is often significantly higher than pure directed multicast.

Scoped multicast using a single multicast address and directed multicast are two extremes in multicast addressing capabilities. It is possible to approximate the effects of directed multicast by using multiple multicast addresses, and using TTL-scoped multicast on each of these addresses. In our experiments, we consider directed multicast, scoped multicast with only one address, and scoped multicast with a small, fixed number of addresses.

5.1.1 Implementing Key Graphs and Boolean Minimization

We have implemented both the Key Graph and Boolean minimization schemes along with our own scheme. The key graphs scheme described in [16] proposes three different re-keying techniques: user-, key-, and group-oriented re-keying. The bandwidth requirements of the user- and key-oriented re-keying are of the same order; therefore, we present results from key-oriented and group-oriented re-keying only. When directed multicast is available, key-oriented re-keying has the lowest overhead, while group-oriented re-keying has lowest overhead when scoped multicast has to be used. In our simulations, we use key oriented re-keying for the directed multicast scenario and the group oriented re-keying scheme for scoped multicast. For all implementations of the key graphs scheme, we used 4-ary key tree graphs, as was proved to be optimal in [16].

[6]More detail about SCAN are available at `http://www.isi.edu/scan`

Along with the different re-keying strategies, we implemented two variants of the key graph algorithm, which we refer to in the results as *KG-sequential* and *KG-spatial*. These two schemes differ in the way the unique member identifier —which defines the position of a member in the key tree— is assigned to each group member. For both schemes, members join the multicast tree in a random sequence.

In the *KG-sequential* scheme, members are assigned sequentially increasing identifiers in the order they join. In the *KG-spatial* scheme, members are not assigned identifiers according to their join sequence. Instead, we assign the identifiers in sequence via a post-order traversal of the multicast topology of all the members. The identifier assignment of the *KG-spatial* scheme ensures that members that share keys are close to each other on the multicast tree. However, in order to ensure nearness on the tree, identifiers have to be reassigned as members join and leave the group: this process accrues a cost linear in the number of group members. In our performance comparison in Section 5.3, we *ignore* this cost. However, the performance of the *KG-spatial* variant is significantly better than *KG-sequential* and any real implementation of Key Graphs on a large topology must address the identifier assignment issue[7]. Lastly, as noted before, for bulk simultaneous to the key graph, we implemented the improved batch update algorithm described in [11].

For the boolean minimization scheme we implemented the key distribution scheme described in [6]. We used the publicly available logic minimization tool, *Espresso*[8] to determine the necessary boolean reductions. Since the key distribution technique in the boolean minimization scheme encrypts all the necessary keys to be updated and multicasts them in a single message, we report the network load only for scoped multicast.

5.2 Experimental Methodology

For each experiment:

- We generated a random set of group members attached to the leaf routers of the Mercator Internet map;
- Next we create the multicast delivery tree, using the CBT protocol [2].
- For our scheme, we compute the member overlay tree and execute the protocol described in Section 4 to identify the clusters using cluster size between 8 and 15.
- We implement the key distribution protocol for our scheme, Key Graphs, and the Boolean Minimization on the same set of members
- Next we choose, uniformly at random, a set of members to leave the multicast group simultaneously and record the storage, processing, message, and byte cost overheads at each node and each link of the tree.

In Section 5.3, we will present results as we vary two parameters: the group size and the number of members that simultaneously leave the group. For each parameter, we repeated each experiment 100 times, each with a randomly chosen set of departing receivers, and obtained 95% confidence intervals. In all cases, our confidence intervals were extremely tight and we do not report them in the results.

[7] It should be noted that our assignments of identifiers to members does not optimally solve the assignment problem; unfortunately, we found that the problem is NP-complete for all key trees with tree degree greater than two.

[8] *Espresso* is publicly available at `http://www-cad.eecs.berkeley.edu/Software/software.html`

Scheme	Single member leaves (varying group size) 3000	6000	12000	24000	48000	96000	1% of group leave (varying group size) 3000	6000	12000	24000	48000	96000
KG-sequential	4.9	5.6	5.0	6.0	5.3	6.3	23.2	32.8	36.3	49.5	57.1	76.8
KG-spatial	1.9	2.1	**1.6**	2.0	**1.5**	**1.8**	10.3	13.7	14.9	18.7	20.8	27.0
Spatial clustering	**1.6**	**1.6**	1.7	**1.7**	1.7	**1.8**	**3.2**	**3.3**	**3.4**	**3.4**	**3.5**	**3.6**

Table 2: Comparison of key normalized byte count per router on a directed multicast-capable network.

Performance Metrics In our results in the next section, we report the following metrics:

- *Key-normalized byte count*: This is the network overhead for re-keying at a single router assuming unit (1 byte) key size. The actual byte overhead can be obtained by simply scaling the key-normalized byte count by the key size, and accounting for packet headers, etc. This metric is a measure of the total bandwidth requirements of a scheme.

- *Packet load*: This is a count of the number of packets processed by the routers on the multicast tree. To compute the packet load, we assume a key size of 512 bits, and the maximum IP packet size of 576 bytes.

- *Storage and Processing Overhead*: These numbers refer to the number of keys stored at each node and the number of cryptographic operations at each node. These numbers are independent of the particulars of the topology.

5.3 Simulation Results

We simulated the various key distribution schemes over multiple topologies and different network configurations. We begin with the results for the case when the underlying network is capable of directed multicast.

5.3.1 Directed multicast

In Table 2 we present the results for two different Key Graph implementations and compare them to our scheme. (Recall that since Boolean Minimization sends all messages to the entire multicast group, its performance is equivalent under directed and scoped multicast. We defer results for Boolean Minimization till the next section when we consider scoped multicast.) For each scheme, we have tabulated the average byte overhead at each router (assuming one byte keys) when a single member leaves and when member leaves are batch processed after 1% of the members leave the group. We analyze the actual number of bytes and IP packets in Section 5.3.4, and other batch sizes in Section 5.3.3.

It is clear from Table 2 that for a single leave, both the *KG-spatial* and our algorithm perform on par (and that they both have lower costs than the *KG-sequential* scheme). However, as members leave simultaneously, or if number of member departures are processed in batch (e.g. shown by the 1%-leave results), our clustering-based scheme significantly outperforms both Key Graph implementations. Finally, we note that our *KG-spatial* variant of the Key Graph scheme outperforms the originally published scheme by a factor of $2.5-3$. (Obviously, this is without accounting for the member–identifier assignment overhead of *KG-spatial* scheme.)

Scheme	Single member leaves (varying group size) 3000	6000	12000	24000	48000	96000	1% of group leave (varying group size) 3000	6000	12000	24000	48000	96000
Key graphs	22.2	23.9	26.8	28.5	31.0	32.4	394.2	792.7	1578.2	3092.7	6222.7	12609.6
Boolean minimization	11.0	12.3	13.5	14.2	15.0	16.2	53.5	94.8	169.9	302.6	567.3	1077.0
Spatial-1	162.5	125.7	121.1	117.5	130.0	117.7	296.4	221.4	190.8	191.2	198.1	208.1
Spatial-16	8.7	11.0	13.9	16.5	19.3	20.6	10.3	12.8	16.2	19.1	22.4	24.1
Spatial-24	6.3	8.1	10.5	12.1	14.4	15.3	8.2	10.3	12.5	14.7	17.1	18.5
Spatial-directed	1.6	1.6	1.7	1.7	1.7	1.8	3.2	3.3	3.4	3.4	3.5	3.6

Table 3: Comparison of normalized byte load per router for the different schemes using scoped multicast.

5.3.2 Scoped multicast

In Table 3 we compare the key-normalized byte load for different re-keying schemes using when TTL-scoped multicast is the only primitive available from the network. Group-oriented re-keying for Key Graphs is optimally suited for scoped multicast, and the member assignment issue is not relevant (both *KG-sequential* and *KG-spatial* have identical performance).

We repeated the scoped multicast experiments with different number of multicast addresses for the spatial clustering scheme. In Table 3, *Spatial-i* indicates that i different multicast addresses have been used by the spatial clusters for intra-cluster communication. We use a simple decentralized address assignment scheme in which each cluster picks one multicast address (among the i available) at random, independent of each other [9]. As is evident in Table 3, for a single leave, Spatial-1 (our scheme using only a single multicast address) performs far worse than all other schemes. This is because even with TTL-scoping traffic local to a cluster "spills over" to a large part of the multicast tree. As more addresses are used, the TTL-scoping becomes more effective and the spill over effect is effectively mitigated. For our scheme, the availability of directed multicast represents the best possible scenario since it effectively provides logical addressing for individual clusters. We have included the costs under directed multicast as a lower bound.

There are two conclusions we can draw from Table 3:

- Using a small number of multicast groups, our scheme can performs on par or better than existing schemes for single member departures.
- Our scheme has much lower overhead than all existing schemes for batched updates to the group, upto $1-3$ order of magnitude better when the group membership changes is of order N (shown by the 1% set of results in the table).

These conclusions lead to two questions:

- How many departures have to be processed in bulk before we significantly outperform existing approaches? It is clearly not always feasible or desirable to wait till $O(N)$ members have left before the group is re-keyed.

[9] Given i addresses, the optimal assignment of addresses to clusters such that the extra traffic is minimized is also NP-complete.

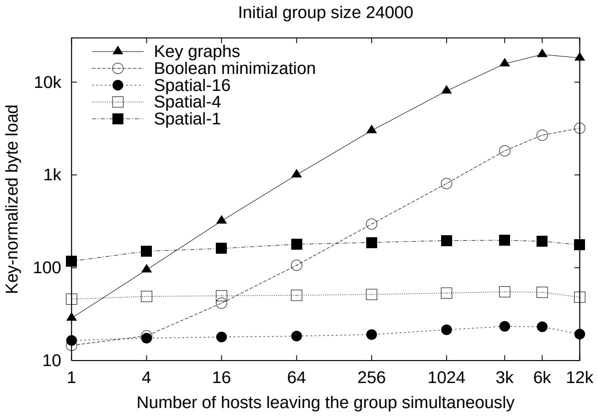

Figure 10: Varying the number of simultaneously leaving members (Scoped multicast)

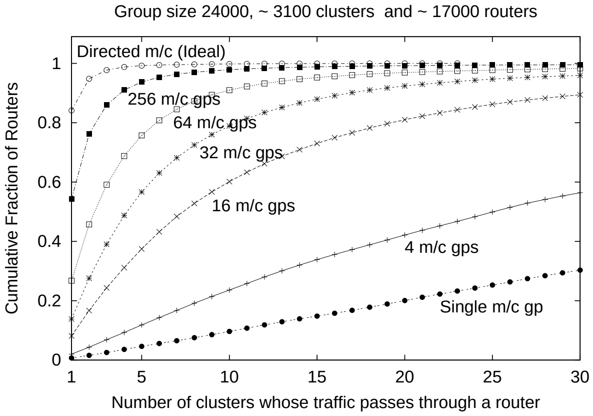

Figure 11: Cumulative distribution of the routers that handle cluster traffic for different number of clusters

- How many multicast addresses do we need to get decent performance, especially for large group sizes?

We addresses these two questions, in turn, in the next two sections.

5.3.3 Impact of batched updates and multiple multicast addresses

Batched updates are likely to be used in most realistic scenarios, especially with large group sizes. In Figure 10 we present results from an experiment in which we varied the number of members that simultaneously depart the multicast group. For each re-keying scheme, the figure plots the key normalized byte overhead for the entire group (added over all the tree routers) as different numbers of members simultaneously leave a 24,000 member secure multicast group. The effectiveness of the $O(\log N)$ bulk update provided by our scheme is clear in the plot from the shape of the *Spatial-x* curves in the plot. In comparison, all existing schemes incur $O(N)$ cost and perform worse than even *Spatial-1* when more than 128 members are processed in bulk. Using 16 addresses is enough to ensure that our scheme outperforms existing schemes for all batched updates to the group and we are a factor of two better than the previously best known scheme if we batch only 16 departures for a 24,000 member group. Lastly, we note that our scheme performs a further order of magnitude better if directed multicast is available.

Using multiple multicast addresses It is clear from Table 3 that the performance of our scheme can be improved by using multiple multicast addresses. Unfortunately, it is difficult to optimally use a given set of addresses, and was not apparent how many addresses should ideally be used for a given group size. Fortunately, directed multicast provides bounds the performance of the scoped multicast implementations: the extra overhead of the scoped multicast implementations is precisely the "spill-over" traffic because of ineffective TTL-scoping.

In order to quantify the gains from using multiple addresses, we varied the number of addresses used and

Scheme	A single member leaves KBytes	Packets	1% of group leaves KBytes	Packets	10% of group leaves KBytes	Packets	25% of group leaves KBytes	Packets
Key graphs	1.8	4	197.9	370	905.2	1689	1273.1	2376
Boolean minimization	**1.0**	**2**	12.2	37	100.2	187	184.3	344
Spatial-1	7.5	118	12.2	191	12.8	200	12.3	193
Spatial-16	**1.0**	17	**1.2**	**19**	**1.5**	**23**	**1.5**	**23**
Spatial-directed	0.1	2	0.2	4	0.5	8	0.5	8

Table 4: Comparison for key-normalized byte and packet loads per router (Group size: 24000 initial members).

randomly assigned clusters to available addresses. We then noted, for each router in the multicast tree, the number of clusters for which the router carries any traffic. In the theoretically ideal case, for perfectly disjointed clusters, all routers should carry traffic from only a single cluster. (In comparison, in the very worst case, without TTL-scoping and if only a single address is available, all routers carry traffic from every cluster!). Our results are shown in Figure 11: for different number of multicast groups, we plot the cumulative distribution of the routers for different number of clusters whose traffic pass through the routers. We see that when a single multicast address is used, more than 70% of the routers carry traffic for at least 30 different clusters. The best case is observed for directed multicast, where 95% of the routers carry traffic for at most 2 clusters. Even with only 32 addresses, the cluster overlap falls significantly (80% of the routers carry traffic 10 or less clusters). We conclude that for groups with tens of thousands of members, few addresses ($16 - 32$) approximate most of the benefits of directed multicast, and are sufficient to better all existing schemes.

5.3.4 Packets, Processing and Storage

In this section, we quantify the costs of our scheme in terms of number of packets processed at the routers and the processing and storage requirements at individual group members.

In Table 4, we present a comparison of both the byte and packet loads of the various re-keying scheme. For this experiment, we assumed the 576 bytes maximum IP packet sizes (536 bytes payload, and 512 bit key sizes). For a single member departure from a group of 24,000 initial members, the key server has to perform 28 encryptions (4-ary key tree of height 8) for the key graphs scheme, which translates to a message of size about 1.8 Kbytes (4 IP fragments). Similar computations show that 2 IP fragments are processed per router for the boolean minimization scheme. In contrast, *Spatial-1* requires more significant processing at each router, because of the high spill-over traffic. As expected, the number of packets can significantly be reduced by using a small number of multicast addresses. Like in the normalized cost case, for batched updates to the group, our scheme outperforms existing approaches with respect to the packet and byte count metrics. Depending on the level of batching and the number of addresses used, the gains from spatial clustering are quite dramatic, often using orders of magnitude less packets than previously known results.

In Table 5, we show the key storage requirements and the processing cost at the key server and individual members for a group of 24,000 members. The processing cost is due to the crytographic operations when the group membership changes. While the number of keys at each member is low for all the schemes, the number of keys stored at the server is significantly lower for both boolean minimization and spatial clustering scheme.

Scheme	Number of Keys		Processing at members			Processing at key server		
	Member	Key server	Single leave	1% leaves	10% leaves	Single leave	1% leaves	10% leaves
Key graphs	9	32002	1.7	5.5	6.36	28.0	3095.3	14310.5
Boolean minimization	15	31	**1.0**	**1.0**	**1.0**	15.0	302.9	1583.5
Spatial clustering	**3**	**6**	1.2	1.5	2.4	**1.3**	**3.5**	**4.0**

Table 5: Comparison of storage and processing costs for the different schemes for a group of 24,000 initial members

Cluster size bound	Total Num. of Clusters	Avg. Cl. Size	Avg (Max) Stab. Time (s)
4 – 7	45	5.6	1.45 (2.63)
10 – 19	17	14.7	1.85 (3.87)
20 – 39	9	27.8	2.22 (4.60)

Table 6: Cluster characteristics

	250 initial group members				
	No. of Simultaneous Changes				
	1	4	16	64	224
Join	1.0	1.8	2.4	3.6	4.3
Leave	0.5	1.6	3.3	4.0	5.1

Table 7: Cluster stabilization time (in seconds)

Our scheme leads to lower processing at the key server for single leaves. For batch updates, the processing at the key server is bounded by $O(\log N)$ for our scheme, which is a substantial improvement over the $O(N)$ costs for both Key Graph and Boolean Minimization schemes.

5.3.5 Experiments with the Clustering Protocol

We have also implemented a packet-level simulator of the spatial clustering scheme on the `ns` (version 2) simulator to study the time dependent overheads of the protocol. We present only a synopsis of the main results. Due to high processing and memory demands of `ns`, we were limited to simulations on topologies of upto 1000 routers. In Table 6, we present results from an experiment where a set of 250 members joined a single multicast group, all between simulation time 0 and 1 second. The member discovery and clustering protocols arrange the members into size bounded clusters; in all our experiments, the size bounds were met by all the clusters. The time required (in seconds) for clustering is shown in the last column of Table 6. For these experiments, we used four heartbeats per second which resulted in about 40 bytes of traffic per second, scoped within the cluster. We used only periodic heartbeats, and not the adaptive technique described in Section 4.3, which can further reduce the stabilization time.

Even though we only simulated on topologies on the order of 1000 nodes, we are confident that the stabilization times we present are representative of large topologies. This is because, in our experiments, we observed that in the vast majority of cases, independent of group size, a single join or leave to the group affects less than two clusters. As multiple simultaneous changes occur, the clustering occurs in parallel over the entire topology; the total time taken for stabilization depends primarily on the cluster size and depth, and not on the size of the group.

6 Summary and Conclusions

In this paper, we have presented a new algorithm for efficiently implementing secure group communications over IP multicast. For all metrics, our scheme provides the best analytic bound, frequently improving on previously known results. For single group changes, our improvements reduce previously known logarithmic bounds to constants, are probably only of theoretical interest. However, for bulk simultaneous group changes, our theoretical results for network load and processing cost at the key server are significant, since they reduce previously known linear bounds to logarithmic bounds.

Unlike existing approaches, our re-keying algorithm was designed to utilize the parallelism inherent in the multicast tree topology. Therefore, we expect our algorithms to perform extremely efficiently in practice. As shown by our extensive simulations on a large realistic topology, for large groups, the number of messages and encryptions required by our scheme is often orders of magnitude lower than existing approaches, especially when we consider simultaneous changes to the group. Our experiments also show that directed multicast is an useful primitive for implementing many secure multicast schemes, including ours.

Due to the extra overhead of implementing and executing a clustering protocol, it is unlikely that our scheme will be useful for small groups. However, we have shown that the time and network overhead for the clustering is relatively small, and more importantly, controllable by varying simple parameters. Further, the clustering overhead is independent of the group size, and depends only on cluster sizes.

References

[1] A. Ballardie. Scalable Multicast Key Distribution. Network Working Group, RFC 1949., May 1996.

[2] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT). *Proceedings of SIGCOMM*, September 1993.

[3] B. Briscoe. MARKS : Zero side effect multicast key management using arbitrarily revealed key sequences. In *1st International Workshop on Networked Group Communication, Pisa, Italy, November 1999.*, Pisa, Italy, November 1999.

[4] B. Cain, T. Speakman, and D. Towsley. Generic router assist (gra) building block motivation and architecture. Internet Draft, Internet Engineering Task Force, March 2000. Work in progress.

[5] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: A taxonomy and efficient constructions. In *Proceedings of INFOCOM*, New York, March 1999.

[6] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceddings of Infocom*, New York, March 1999.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, and P. Sharma. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. *RFC 2362*, 1998.

[8] R. Govindan and H. Tangmunarunkit. Heuristics for Internet Map Discovery. In *Proceedings of Infocom*, March 2000.

[9] H. Harney and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. Request for Comments (Experimental) 2094, Internet Engineering Task Force, July 1997.

[10] B.N. Levine and J.J. Garcia-Luna-Aceves. Improving Internet Multicast with Routing Labels. In *Proc. IEEE International Conference on Network Protocols*, pages 241–50, October 1997.

[11] X. Li, R. Yang, M. Gouda, and S. Lam. Batch updates for key trees. Technical report, University of Texas, Austin, September 2000.

[12] S. Mittra. Iolus: A framework for scalable secure multicasting. *Proceedings of SIGCOMM*, October 1997.

[13] A. Reddy, R. Govindan, and D. Estrin. Fault isolation in multicast trees. *Proceedings of SIGCOMM*, August 2000.

[14] T. Speakman et al. PGM reliable transport protocol. Internet Draft, Internet Engineering Task Force, April 2000. Work in progress.

[15] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. *RFC 1075*, 1998.

[16] C.K. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *Proceedings of SIGCOMM*, September 1998.

A Key Distribution Protocol

We assume a clustering protocol exists that creates and maintains clusters, as specified in Section 3. The clusters have size between k and $2k-1$, for some fixed k, are non-overlapping and proximal subsets of the multicast delivery tree. A hierarchy of members is created as shown in Figure 3 as follows:

> *All members are part of the lowest layer, L_0. Each layer, L_i, runs an instance of the clustering protocol to create a set of clusters, the leaders of which join the next higher layer, L_{i+1}.*

Each layer has a secret layer key known to only the members of the layer. Similarly, each cluster in each layer has a secret cluster key, known to only all the cluster members.

A.1 Notation

- Members and Member Sets
  - $Cluster\ (u, j)$: Cluster of layer L_j, to which member u belongs.
  - $Leader\ (u, j)$: Leader of the cluster in layer L_j to which member u belongs.
  - S : The key server for all layer keys.
- Keys and Messages
  - $\chi_G(t)$: The secret key of G at time t, where where G is a set of members. If G is a cluster, then this is the cluster key, if G is a layer, then this is the layer key. If G is a pair of members, then this is a key shared only by these two members.
  - $\{m\}_e$: Message m is encrypted by the key e.
  - $\langle Unicast :: u \rightarrow v : x\rangle$: u sends a unicast message x to v.
  - $\langle Multicast :: u \rightarrow G : x\rangle$: u multicasts message x to set of members G, where G is either a cluster or a layer.

A.2 Distributed Re-keying Operation

When a member u joins or leaves layer L_j, the following operations are performed distributedly by the key distribution protocol:

1. Cluster re-key: $Leader\ (u, j)$ generates a new cluster key $\chi_{Cluster\ (u,j)}(t+1)$ and unicasts it to each current member of the cluster $Cluster\ (u, j)$ encrypted separately by the pair-wise key of the leader with each member.

$$\forall v \in Cluster\ (u, j), \langle Unicast :: Leader\ (u, j) \rightarrow v : \{\chi_{Cluster\ (u,j)}(t+1)\}_{\chi_{\{Leader\ (u,j), v\}}}\rangle$$

Obviously, in case u is leaving this cluster, this message is not sent to u. The total communication overhead of this cluster re-key is $O(k)$ per link (due to $O(k)$ pair-wise communication between the cluster-leader and the cluster members).

2. Layer re-key: The key server,S, generates a new layer key for layer L_j, and multicasts it to all members of layer L_{j+1} — these are the cluster-leaders of the clusters of layer L_j. Each cluster leader of layer L_j then performs a cluster multicast to all the members of its cluster in layer L_j. The multicast messages are encrypted by the appropriate keys.

$$\langle Multicast :: S \ \rightarrow \ L_{j+1} : \{\chi_{L_j}(t+1)\}\chi_{L_{j+1}}(t)\rangle$$

$$\forall v \in L_{j+1}, \langle Multicast :: v \ \rightarrow \ Cluster\ (v,j) : \{\chi_{L_j}(t+1)\}_{\chi_{Cluster\ (v,j)}(t+1)}\rangle$$

The multicast message from S to layer L_{j+1} traverses each link only once. The cluster multicasts of the cluster-leaders of layer L_j traverse spatially disjoint parts of the multicast delivery tree. Hence, only one cluster multicast message traverses each link in the tree. Thus, the combined communication cost of these multicasts is $O(1)$ per link.

The total communication cost for distributed re-key operation in each layer is, therefore, $O(k)$ per link.

A.3 Re-keying algorithm for member joins and leaves

When a new member joins the secure multicast group, it is inducted into some L_0 cluster. When a member leaves the group, it leaves from all the layers it was joined to. In both cases, the clustering protocol appropriately re-clusters these affected layers. Upon a membership change, the re-keying algorithm re-keys each affected cluster and each affected layer, in turn using the cluster re-key and layer re-key primitives, respectively.

A.4 Analysis

We analyze three different metrics — key storage requirements at each member, the processing costs due to encryptions and decryptions at each member and the communication overheads per link of the multicast delivery tree for each re-key.

Assume that there are N members currently joined to the secure multicast group. Let L_R be the highest layer in the hierarchy (it contains a single member). The following properties hold :

- $R \leq \log_k N$. For all j in $[0, R]$, layer L_j has not more than $N/(k^j)$ members.
- If a member u, is present in L_j, it is present in all lower layers $L_0 \dots L_{j-1}$. If a member u is not present in layer L_j, it is not present in any of the higher layers, $L_{j+1} \dots L_R$.

Communication Cost Most members (i.e. at least $N(1-\frac{1}{k})$ of them) are joined only to the lowest layer, L_0. Hence, a majority of member joins and leaves affect only layer L_0 and the communication cost incurred for the necessary re-keying is $O(k)$, a constant (as shown before). In the worst case, the single member at the highest layer L_R leaves, all the layer keys would need to be changed. Each layer key change requires $O(k)$ communication cost per link, which makes the *worst case* communication cost of $O(k \log_k N)$ for leaves (and analogously for joins).

Under the assumption that each member of the group is equally likely to leave (and join), we now show that the *amortized cost* for joins and leaves is constant. We consider the case of a member leaving the group. Consider the case when a member u, joined only to layers $L_j \dots L_0$, leaves the secure multicast group. Each of these $j+1$ layer keys need to be changed. The communication cost per link for each layer key change is $O(k)$. Additionally, in each layer, it can be shown that at most two clusters would need to be re-clustered (if size of the affected cluster falls below the size lower bound, k). Due to the clustering protocol, this would involve communication cost of $O(k)$ per link. Hence, the total communication cost per link due to re-keying of all the layers, for member u leaving the group is $O(k(j+c))$, where c is some constant. For any layer, the number of members in layer L_j is bounded by N/k^j. Thus, the amortized cost of for a member leaving the multicast group is given by :

$$\frac{1}{N}\sum_{j=0}^{R}|L_j|k(j+c) \leq \frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}k(j+c) \;=\; c\frac{k}{k-1}+\frac{k^2}{(k-1)^2}+O(\frac{\log_k N}{N})$$

However, k and c are constants and $\frac{\log N}{N} \rightarrow 0$ asymptotically with increasing N. Hence, the amortized cost of a member leaving (and also for joining) is constant.

Storage Let us assume that the clusters are of size k each [10]. The total number of keys at a member that belongs to layer L_j and no layer above that, are as follows : $j+1$ layer keys, 1 pair-wise key with the leader of its cluster in layer L_j, and $k-1$ pair-wise keys with the other members in each of the clusters in layers $L_0,\ldots,L_{j-1}$, leading to a total of $jk+2$ keys. Then number of such members in L_j is $\frac{N}{k^j}(1-\frac{1}{k})$.

Hence, the average number of keys per member is: $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}(1-\frac{1}{k})(kj+2) \;=\; 2+\frac{k}{k-1}+O(\frac{\log N}{N})$. Asymptotically, the average number of keys at a member is, therefore, $2+\frac{k}{k-1}\leq 4$.

Processing cost at a member When a member, that belongs to only L_0 leaves, the decryption cost per member is: 1 for each member to decrypt L_0 layer key and 1 each of the $k-1$ members in the affected cluster of L_0 to decrypt the cluster key. Therefore, the average cost per member is $\frac{N+k-1}{N}$.

Similarly, when a member that belongs to layers $L_j \ldots L_0$ leaves, the average decryption cost per member is: $\chi_j = \frac{(k-1)(j+1)}{N}+\sum_{i=0}^{j}\frac{1}{k^i}$. Therefore, amortized decryption cost per member is $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}.(1-\frac{1}{k}).\chi_j = \frac{k^2}{k^2-1}+O(\frac{\log N}{N})$, which is ≤ 2 for asymptotically increasing N. Similarly, it can be shown that the amortized encryption cost per member for departure of a single member is ≤ 1.

Processing cost at key server The key server encrypts new layer keys prior to layer multicast. The amortized number of encryptions for a single departure is $\frac{1}{N}\sum_{j=0}^{\log N}\frac{N}{k^j}.(1-\frac{1}{k}).(j+1) = 1+\frac{1}{k-1}+O(\frac{\log N}{N})$, which is asymptotically bounded by ≤ 2.

Bulk simultaneous departures For a group of size N, assume xN of the groups leave, where $0 < x < 1$. In this case, the average number of layers affected is given by $\log N.(1-x)$. From layer L_j, the number of leaving members is $x.\frac{N}{k^j}.(1-\frac{1}{k})$, and the decryption cost per member due to the layer and cluster key updates at this layer is given by $\frac{1}{N}\cdot\frac{N}{k^j}.(2+x\frac{k-1}{k})$. Hence, the total cost is bounded by $\sum_{j=0}^{\log N.(1-x)}\frac{1}{k^j}.(2+x\frac{k-1}{k}) \leq 2\frac{k}{k-1}+x+O(\frac{\log N}{N})$, which is asymptotically $O(1)$.

Additionally, the processing cost at the key server is $(1-x)\log N$ (same as the number of affected layers). Similarly, the communication overheads at each router is $O(\log N)$.

[10] Although cluster sizes can vary between k and $2k-1$, this assumption does not change the order of the results.