

Distributing Key Updates in Secure Dynamic Groups¹

Sandeep S. Kulkarni Bezawada Bruhadeshwar
Department of Computer Science and Engineering
Michigan State University
East Lansing MI 48824 USA

Abstract

We focus on the problem of distributing key updates in secure dynamic groups. Due to changes in group membership, the group controller needs to change and distribute the keys used for ensuring encryption. However, in the current key management algorithms the group controller broadcasts these key updates even if only a subset of users need them. In this paper, we describe a key distribution algorithm for distributing keys to only those users who need them. Towards this end, we propose a descendent tracking scheme. Using our scheme, a node forwards an encrypted key update only if it believes that there are descendants who know the encrypting key. We also describe an identifier assignment algorithm which assigns closer logical identifiers to users who are physically close in the multicast tree. We show that our identifier assignment algorithm further improves the performance of our key distribution algorithm as well as that of a previous solution. Our simulation results show that a bandwidth reduction of upto 55% is achieved by our algorithms.

Keywords : Secure Multicast, Key Distribution, Descendent Tracking, Identifier Assignment

1 Introduction

In group oriented applications, such as conferencing, networked gaming and news dissemination, it is necessary to secure the data from intruders as the data is confidential or it has monetary value. In the algorithms for secure group communication (e.g., [1–6]), a group controller distributes a cryptographic key, called the group key, to all the group users. The group key is used to encrypt data to secure the group communication. The group membership is dynamic. To protect privacy of the current users, the group controller changes and securely distributes the group key at each membership change. This rekeying is especially needed when a user leaves the group and should no longer understand the group communication.

In the algorithms in [1–4], the group controller distributes additional keys which are shared by different subsets of users. These shared keys reduce the number of group key update messages the group controller needs to transmit. The group controller encrypts the new group key with a minimum subset of the shared keys and transmits it to the current users. To reflect current group membership, the group controller changes and securely transmits the shared keys known to the leaving user. Although each of the new shared keys needs to be transmitted to only a subset of the users, the current key management algorithms assume a broadcast primitive. Hence, *all* the current users receive *all* the key update messages. Of course, users

¹Email: sandeep@cse.msu.edu, bezawada@cse.msu.edu
Web: <http://www.cse.msu.edu/~{sandeep, bezawada}>,
Tel: +1-517-355-2387, Fax: 1-517-432-1061

This work is partially sponsored by NSF CAREER 0092724, ONR grant N00014-01-1-0744, DARPA contract F33615-01-C-1901, and a grant from Michigan State University.

cannot decrypt the key update messages that are not intended for them since they do not have the necessary keys.

From the above discussion, we observe that the current key management algorithms only focus on *what* key update messages are sent but do not emphasize on *how* they are distributed. This results in wastage of bandwidth as users receive key update messages for keys that they do not need. This wastage increases further if retransmission is required for any lost messages and such retransmission is done using a broadcast. Although solutions for reducing the number of retransmissions in secure group communication have been proposed in [7, 8], the group controller still needs to broadcast them. Thus, efficient distribution of key updates is an important problem in secure dynamic groups.

A more general scenario of the problem we discussed above occurs in the *security of interval multicast* [9]. The objective of the security of interval multicast algorithm [9] is to securely transmit data messages to a selected interval of users within the group. The group controller encrypts the data messages using shared keys known only to the selected interval of users and broadcasts them. We note that, the percentage of bandwidth wasted in this scenario is far more critical as the size and frequency of the data messages can be arbitrarily large in many multicast applications.

One possible solution to the key distribution problem, based on [10], is to create a different multicast group for each key update message and inform the users to join a particular group if they need the key being sent to that group. However, in this solution, the overhead of group setup and tear down for each key update message is not suitable for large groups. Also, the delay incurred during the rekeying is not desirable as the group controller interrupts the group communication until the rekeying is complete. Other possible solutions are the router level filtering technique proposed in [11] or the internet labeling and addressing architecture proposed in [12]. However, both these algorithms suffer from increased delay and increased complexity.

In this paper, we propose an algorithm for the distribution of key update messages in secure dynamic groups. In our key distribution algorithm, we integrate the key management algorithms in [1, 2] with appropriate forwarding functionality. We assume that the users are arranged in a multicast tree which can be built using any of the IP [13–16] or overlay [17–19] multicast protocols. Depending on the multicast protocol used, an intermediate node in the multicast tree can be a router (IP multicast) or an overlay node. Hence, we only focus on the actions of an intermediate node as the implementation details are beyond the scope of this paper. The contributions of our paper are as follows:

- We describe a compact descendant tracking scheme to track the descendants of the intermediate nodes. The memory required at the intermediate nodes for our scheme is small and scales logarithmically in the size of the group. The advantage of our descendant tracking scheme is that this information can either be updated periodically or in the *background*, i.e., after the group communication has resumed.
- We describe the forwarding mechanisms used by the intermediate nodes to forward the key update messages.
- We describe a user identifier assignment algorithm. Using our assignment algorithm, the group controller assigns closer logical identifiers to users who are located close to each other in the multicast tree. We show that our assignment algorithm improves the performance of our key distribution algorithm as well as that of a previous solution in [20].
- We discuss the application of our key distribution algorithm to the problem of distributing data messages in the security of interval multicast [9].

Organization of the paper. The paper is organized as follows. In Section 2, we describe the notations used in our key distribution algorithm and describe a previous solution. In Section 3, we describe our key distribution algorithm and our user identifier assignment algorithm. In Section 4, we present the simulation results. In Section 5, we discuss the application of our key distribution algorithm to the problem of secure interval multicast. Finally, in Section 6, we conclude and discuss some future work.

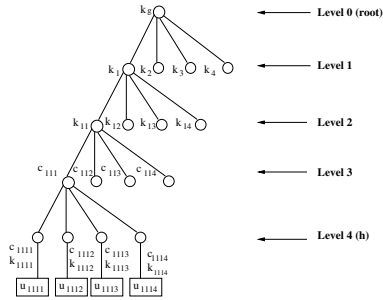


Figure 1: Partial View of Key Tree

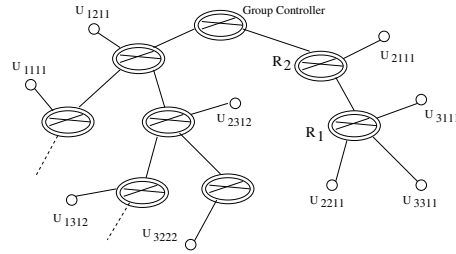


Figure 2: Partial View of Physical Multicast Tree

2 Notations

In Section 2.1, we describe the key management algorithms from [1, 2]. In Section 2.2, we describe the components of the multicast tree. Any multicast protocol IP [13–16] or overlay [17–19] can be used to build the multicast tree; our approach is independent of the protocol used to build the multicast tree. In Section 2.3, we describe the problem of key distribution and a previous solution.

2.1 Key Management Algorithms

In the key management algorithms from [1, 2], the group controller arranges the users and the keys in a key tree. The leaf nodes in this key tree correspond to the users in the group (cf. Figure 1). Based on this arrangement of users and keys, we number them according to their location in the tree. For example, in a key tree of height h , the user $u_{i_1 i_2 \dots i_h}$ denotes the user obtained by taking the i_1^{th} child at level 1, i_2^{th} child at level 2, and so on. The keys are also numbered likewise. Each node in the key tree is associated with a key from the logical key hierarchy [1] or a key from the complementary key hierarchy [2] or both. We use $k_{i_1 i_2 \dots i_l}$ to denote the logical key at node $i_1 i_2 \dots i_l$. And, we use $c_{i_1 i_2 \dots i_l}$ to denote the complementary key at node $i_1 i_2 \dots i_l$. For the levels where the group controller uses logical keys, the user obtains the keys on the path to the root. For the levels where the group controller uses complementary keys, the user gets the keys associated with the siblings of its ancestors. For example in Figure 1, user u_{1112} gets the group key (k_g), the logical keys (k_1, k_{11} and k_{1112}), and the complementary keys ($c_{112}, c_{113}, c_{114}, c_{1111}, c_{1113}$ and c_{1114}). Finally, we use $k(m)$ to denote that message m is encrypted using the key k and, hence, only users that have the key k can obtain m .

2.2 Multicast Tree

In our key distribution algorithm, we assume that the group controller distributes data and key update messages to the users and, hence the root of the multicast tree is the group controller. We define the *parent* of a user x , say $parent.x$, as the next hop node on the path from x to the group controller. For an intermediate node, we consider its children and all other nodes reachable through its children as its descendants. As an illustration, in Figure 2, we show a part of the multicast tree corresponding to the key tree in Figure 1. Each intermediate node performs reverse path forwarding, i.e., a message from the group controller is replicated on all outgoing links except on the link on which the message was received. We define a Key Distribution (KD) aware intermediate node as a node which supports our key distribution algorithm. Unless otherwise specified, in our paper, we assume that all intermediate nodes in the multicast tree are Key Distribution (KD) aware.

2.3 Problem of Key Distribution and a Previous Solution

In the current key management algorithms [1, 2], when group membership changes, the group controller changes the keys in the key tree and securely broadcasts the new keys. Since all users do not need all the keys, this mode of key distribution is not efficient. Hence, we focus on the key distribution in algorithms where each user receives only a small subset of keys that includes all the keys it needs. Towards this end, we modify the forwarding mechanism at the intermediate nodes; an intermediate node forwards a key update message only if it believes that there are descendant users who need this key update. In our approach, an intermediate node performs this check by verifying that any of its descendants know the key with which the key update message is encrypted.

Previous solution. In [20], to distribute the changed keys in the key tree, the group controller encrypts the keys at the higher levels using changed keys at the lower levels. For example, when u_{1111} leaves, to distribute a new key k'_1 , the group controller generates the messages $k'_{11}(k'_1)$, $k'_{12}(k'_1)$, $k'_{13}(k'_1)$ and $k'_{14}(k'_1)$, where the key k'_{11} is already distributed using keys further down in the key tree. Before transmitting these messages, the group controller broadcasts the identifier of the leaving user to the current users. Using this information and knowledge of the structure of the key tree, each user calculates the level numbers of the changed keys it needs. This information is propagated towards the group controller. Upon reception of replies, the group controller transmits the key update messages and includes the level number, at which the key is changed, in each message. When an intermediate node receives a key update message with a level number l , it forwards the message to its descendants only if some descendant of l had requested that key. The main shortcoming of this key distribution algorithm is that, it is executed for every membership change which causes delay at the users for receiving key updates. To address this deficiency, in Section 3, we describe two approaches to improve bandwidth usage without increasing delay in rekeying.

3 Proposed Improvements for Key Distribution

In this section, we identify our approach for reducing the cost of key distribution. Towards this end, in Section 3.1, we describe our descendant tracking scheme that enables the intermediate nodes to approximately track its descendants. In Section 3.2, we describe our algorithm for assigning identifiers to users. Using our assignment algorithm, the group controller arranges users close to each other in the key tree if they are close to each other in the multicast tree. Our identifier assignment algorithm can be used to improve the performance of the key distribution algorithm in Section 3.1 as well as that of the previous solution in [20].

3.1 Descendant Tracking Scheme

A simple method to track descendants is to store the identity of each descendant user and forward the key update message only if any descendant user needs this key. However, this straightforward solution requires each intermediate node to store a large amount of information, especially in large groups. Thus, there is a need for an efficient descendant tracking scheme which can solve the key distribution problem. To describe our scheme, first, we define the steady state configuration of the multicast tree. Then, we describe the technique used at the intermediate nodes to update the descendant tracking information to account for group membership changes. In Sections 3.1.1-3.1.2, we describe how keys encrypted with logical and complementary keys are forwarded by the intermediate nodes using the descendant tracking information. Our key distribution algorithm consists of the descendant tracking scheme and the forwarding mechanisms used by the intermediate nodes to forward keys encrypted with logical and complementary keys.

Steady State Configuration. At each intermediate node, we store a $h \times d$ matrix called DT (Descendant Tracker), where h and d are, respectively, the height and degree of the key tree. Each element in DT is a single bit. Thus, the information kept in DT is very small, even for large groups. For example, for a group of 1024 users, where the group controller maintains a key tree of degree 4 and height 5, each intermediate node stores only 20 bits of information in DT . To track a descendant user with identifier $u_{i_1 i_2 \dots i_h}$, at an

intermediate node, we set the elements, $DT[1, i_1], DT[2, i_2], \dots, DT[h, i_h]$, to 1.

When the group is initialized, the group controller assigns each user a unique identifier based on its location in the key tree. The users who are leaves in the multicast tree record their identifiers in their DT matrices. The intermediate nodes request their children for the DT information. The DT entries of a parent are the disjunction (binary OR) of the corresponding entries of its children. As an illustration, in Figure 3, we show the DT entries at the leaf node, R_1 with users U_{2211}, U_{3311} and U_{3111} , and the disjunction of these entries by its parent node, R_2 with user U_{2111} . We note that, the descendants identified in a DT matrix are a superset of the actual descendant users. However, the DT matrix achieves the required functionality accurately, i.e., it provides sufficient information about the descendants of an intermediate node.

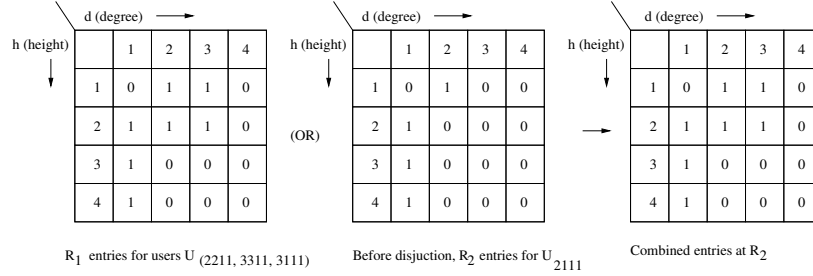


Figure 3: DT entries at intermediate nodes R_1 and R_2

Tasks for Join. When a new user joins the group, the group controller distributes any keys, the new user needs, using a secure unicast channel. Also, the group controller distributes the new keys to the current users, where the intermediate nodes perform appropriate forwarding (cf. Sections 3.1.1-3.1.2), using the existing DT entries. The new user receives its identifier from the group controller and provides this information to its local intermediate node. The local intermediate node updates its DT matrix. Further, if the DT matrix has changed, it propagates this information to its ancestors in the multicast tree.

Tasks for Leave. When a user leaves the group, we do not update the DT entries at the intermediate nodes immediately. We perform an update of the DT entries only when the number of group membership changes exceeds a threshold level. Although this could cause some messages to traverse extra links, updating the DT matrix periodically allows us to achieve a tradeoff between the processing overhead and the amount of bandwidth reduced.

3.1.1 Forwarding Key Update Messages Encrypted with Logical Keys

In a key tree with only logical keys [1], each user knows all the keys associated with its ancestors. Thus, based on the naming scheme of the key tree from Section 2, the label of every key a user knows is a prefix of the user's identifier. Now, consider the case when some user, say u_{1112} , leaves the group. The group controller changes all the keys known to u_{1112} and distributes them to the remaining users who need these keys. The new keys generated by the group controller are k'_{111}, k'_{11}, k'_1 and k'_g . To send these keys, the group controller sends, $k_{1111}(k'_{111}), k_{1113}(k'_{111}), k_{1114}(k'_{111}), k'_{111}(k'_{11}), k_{112}(k'_{11}), k_{113}(k'_{11}), k_{114}(k'_{11}), k'_{11}(k'_1), k_{12}(k'_1), k_{13}(k'_1), k_{14}(k'_1), k'_1(k'_g), k_2(k'_g), k_3(k'_g)$ and $k_4(k'_g)$. The approach for a joining user is similar.

To determine if a key update is needed by any descendants of an intermediate node, we need to determine whether the label of the encrypting logical key is a prefix of at least one descendant in the intermediate node's DT matrix. To allow the intermediate nodes to make this identification, the group controller includes the label of the encrypting logical key in the key update message and transmits it to the users. For example, to send $k_{12}(k'_1)$, the group controller appends the label 12, to the encrypted message. To identify if the label l_1, \dots, l_k of the encrypting logical key is a prefix to any descendant user, an intermediate node checks whether the DT elements, $DT[1, l_1], \dots, DT[k, l_k]$ are all set to 1.

3.1.2 Forwarding Key Update Messages Encrypted with Complementary Keys

We note that, in a key tree with only complementary keys [2], each user knows the keys associated with the siblings of its ancestor. Thus, the labels of these keys differ in the last position from any prefix of the user's identifier. For example, a user u_{1113} , knows the complementary keys, c_{1111} , c_{1112} , c_{1114} , c_{112} , c_{113} , c_{114} , c_{12} , c_{13} , c_{14} , c_2 , c_3 , c_4 , the group key k_g and the personal key k_{1113} . Now, we consider the case when this user leaves the group. The group controller generates new keys for all the keys known to u_{1113} and distributes them to the other users who need them. The rekeying method used by the group controller for distributing complementary keys is similar to the method for distributing logical keys. The group controller encrypts the changed complementary keys at the higher levels using the changed complementary keys at the lower level. For example, to distribute c'_{12} , the group controller generates the messages, $c'_{112}(c'_{12})$, $c_{113}(c'_{12})$.

To determine if a key update is needed by a descendant of an intermediate node, we need to determine whether the label of the encrypting complementary key differs in the last position from a prefix of a descendant's identifier. As in the case of transmitting keys encrypted with logical keys, the group controller appends the label of the encrypting complementary key to the key update message. Thus, to distribute $c'_{112}(c'_{12})$, the group controller appends the label 112 to the message. To verify that the label l_1, l_2, \dots, l_k of the encrypting complementary key is matched, an intermediate node checks whether the entries, $DT[1, l_1]$, $DT[2, l_2]$.., $DT[k-1, l_{k-1}]$ and any entry $DT[k, l_p]$ where $p \neq k$, are set to 1. As an illustration, the intermediate node U_{1211} , on receiving the message $[c'_{12}(c'_{12}), 12]$ forwards it, as the entries, $DT[1, 1]$, $DT[2, 3]$ are set to 1. These entries correspond to the prefix 13, of a descendent user 1312, which differs in the last position from the encrypting complementary key's label 12.

3.2 User Identifier Assignment Algorithm

The key distribution algorithms we described in Sections 2.3 and 3.1, route the key update messages based on the identity of the descendants. The performance of these algorithms can be improved if the distribution of leaf nodes (group users) in the multicast tree corresponds to the distribution of the leaf nodes in the key tree. In this ideal scenario, users close to each other in the multicast tree will need almost the same key update messages and hence, the cost of key distribution would be low. While such a scenario is not always possible, one heuristic to achieve a near ideal scenario is to assign a joining user a logical identifier that is closer to the logical identifiers of users who are close to this user in the multicast tree. In this section, we use this heuristic to describe a user identifier assignment algorithm.

When a user sends a join request, the group controller communicates with this user using a secure unicast channel and learns about the location of the user in the multicast tree. Now, the group controller can use a program such as *mtrace* [21] to identify other nearby users in the multicast tree and record their logical identifiers. The group controller selects a user $u_{i_1 i_2 \dots i_t}$, at the first intermediate node which is closest to the joining user. To assign an identifier to the joining user, the group controller selects an identifier $u_{i_1 i_2 \dots i_p}$ such that $i_p \neq i_t$ and $1 \leq p, t \leq d$, i.e., the identifiers differ in the last position. The group controller assigns this identifier to the joining user, if it is not already assigned to any other user. If this attempt fails, the group controller repeats this process with another user at the first intermediate node. As an illustration, consider that the group controller selects the user u_{1111} at the first intermediate node. The group controller generates the identifiers 1112, 1113, 1114, which differ in the last position with 1111. If any of these identifiers are still available, the group controller assigns one of them to the joining user. If no more users exist at the first intermediate node, the group controller selects users at the second intermediate node and so on.

In our assignment algorithm, as the intermediate nodes are further away from the joining user, the group controller successively moves up the position at which the identifiers differ. For example, the group controller selects a user, $u_{i_1 i_2 \dots i_k i_t}$, at the second intermediate node, and tries to assign the joining user, $u_{i_1 i_2 \dots i_p i_l}$, such that $i_p \neq i_k$, i.e., the identifiers now differ in the second last position. As an illustration, consider that the group controller selects the user u_{2111} at the second intermediate node. The group controller generates the identifiers, 2121, 2131, 2141, which differ in the second last position with 2111 and tries to assign one of these identifiers to the joining user. If the users are found at the third intermediate node, the group controller

tries to assign identifiers which differ in the third last position and so on. The group controller repeats this process until it successfully assigns an identifier or stops, if the only users found are very close to itself.

In case the group controller finds that the only intermediate nodes with users are very close to itself, the group controller assigns the next available identifier to the joining user. We do not select the logical identifiers of users close to the group controller due to two reasons. The first reason is that these users are not in the proximity of the joining user. The second reason is that these users, being close to the group controller, receive almost all of the key update traffic anyway and, thus, there would be no performance gain if the joining user is logically closer to these users.

4 Simulation Results and Analysis

We simulated our algorithms using the NS2 network simulator [22]. We performed experiments on randomly generated network topologies for groups of 256, 512 and 768 users. We used the CBT [13] protocol to build the multicast tree with the group controller as the root node. For each experiment, we selected a random set of users to join or leave the group and recorded the number of messages in the multicast tree over the entire multicast session. We define the following metrics to measure the performance of our algorithms:

- *Key update Bandwidth.* This is the number of links traversed by each key update message, i.e., this is the network bandwidth used by the key update messages.
- *Total Bandwidth.* This is the number of links traversed by all messages exchanged over the multicast tree. This metric includes the key update bandwidth and the control messages exchanged among the intermediate nodes.
- *Key update Bandwidth Reduction Ratio(KBRR).* This ratio is the measure of key update bandwidth saved using our algorithms as against using broadcast. We calculate it as follows:

$$KBRR = \frac{(KeyupdateBandwidth_{broadcast} - KeyupdateBandwidth_{optimised})}{KeyupdateBandwidth_{broadcast}} * 100$$

- *Total Bandwidth Reduction Ratio(TBRR).* This ratio is the measure of total bandwidth saved using our algorithm as against using broadcast. We calculate it as follows:

$$TBRR = \frac{(TotalBandwidth_{broadcast} - TotalBandwidth_{optimised})}{TotalBandwidth_{broadcast}} * 100$$

- *Per Hop Bandwidth Reduction Ratio(PBRR).* This ratio is the hopwise breakup of the *TBRR* for a given network topology. This metric gives an overview of the performance of our algorithms as function of the network distance away from the group controller.

We conducted experiments on three key management algorithms. The first algorithm is the logical key hierarchy algorithm (LKH) in [1]. The second and third algorithms are from our earlier work which appears in [2]. In the second algorithm, the group controller uses complementary keys at every level in the key tree. We refer to this algorithm as complementary key hierarchy (CKH). In the third algorithm, the group controller uses both logical and complementary keys at every level in the key tree. We refer to this algorithm as logical+complementary key hierarchy (LKH+CKH). For comparison purposes we also simulated the previous key distribution solution we described in Section 2.3. We use the following notation to refer to the various key distribution algorithms:

- *Id-based:* Our key distribution algorithm from Section 3.1.
- *Id-based-cluster:* Combination of algorithms in Sections 3.1 and 3.2.
- *Level-based:* The key distribution algorithm from [20] that is described in Section 2.3.
- *Level-based-cluster:* Combination of algorithms in [20] and 3.2.

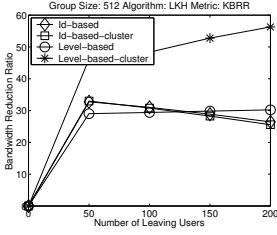


Figure 4: KBRR for 512 Users using LKH

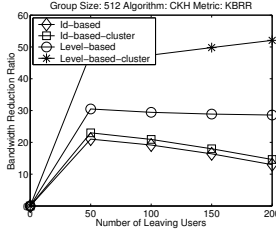


Figure 5: KBRR for 512 Users using CKH

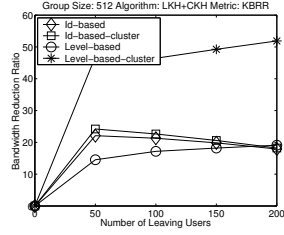


Figure 6: KBRR for 512 Users using LKH+CKH

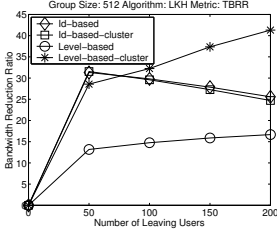


Figure 7: TBRR for 512 Users using LKH

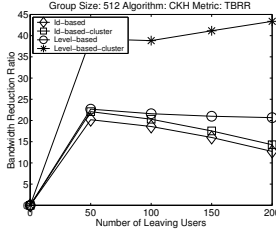


Figure 8: TBRR for 512 Users using CKH

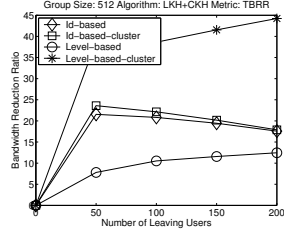


Figure 9: TBRR for 512 Users using LKH+CKH

In Figures 4-6, we plot the $KBRR$ for a group of 512 users. From these figures, we observe that the $KBRR$ achieved in our key distribution algorithm is in the range of 20-55%. We observed similar simulation results for groups of 256 and 768 users. However, for reasons of space we omit these results.

In Figures 7-9, we plot the $TBRR$ for a group of 512 users. The $TBRR$ achieved is in the range of 20-45%. We observe that using our identifier assignment algorithm improves the $KBRR$ and $TBRR$ of our key distribution algorithm as well as that of the level based key distribution algorithm.

In Figures 10-12, we plot the $PBRR$ for a group of 768 users. A random set of 200 users leave this group during the session. The leaf nodes in this group are at a distance of upto 6 hops from the group controller. The $PBRR$ value for Hop Number 1 indicates the $TBRR$ observed between hop 0 (group controller) and hop 1 (immediate children of group controller) and so on. From Figures 10-12, we observe that the *Level-based* algorithm causes more link stress near the group controller due to the responses by the users for each membership change. We note that, this problem is remedied by using our identifier assignment algorithm which reduces the link stress near the group controller in this algorithm. Due to limitations of space and similarity of other results, we have presented only one case study for $PBRR$.

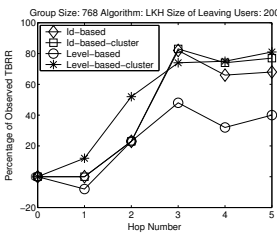


Figure 10: PBRR for 768 Users using LKH

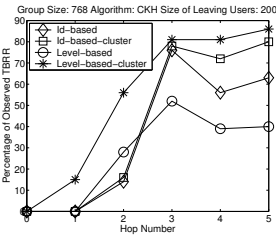


Figure 11: PBRR for 768 Users using CKH

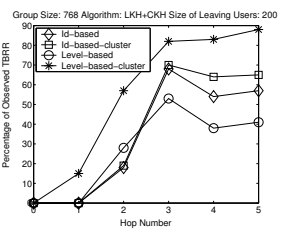


Figure 12: PBRR for 768 Users using LKH+CKH

5 Application to Security of Interval Multicast

The concepts used in our key distribution algorithms can also be used in other secure multicast applications, especially in applications that require data to be securely transmitted to only a subset of group users. One such application is *the security of interval multicast*, described in [9]. In this application, the group controller securely multicasts data to a selected interval (subset) of users. To send a message to the interval of users, the group controller identifies the key(s) that are shared among these users in the key tree. The group controller encrypts the message with each of the keys and broadcasts it. As the keys selected to encrypt the message are known only to the users in the interval, only those users can decrypt that message(s).

We note that, in the secure interval multicast algorithm [9], *all* the group users receive the encrypted message(s) due to the broadcast even if they do not need that message(s). To reduce bandwidth utilization, we reuse the techniques from Section 3.1. We have found that the reduction in bandwidth utilization for secure interval multicast is similar to the simulation results in Section 4. However, for reasons of space we omit these simulation results.

6 Conclusion

In this paper, we addressed the problem of distributing key updates to users in secure dynamic groups. Towards this end, we described a descendant tracking scheme to track the descendants of the intermediate nodes in the multicast tree. In our descendant tracking scheme, each intermediate node stores a small information about its descendants. Next, we described the forwarding mechanisms used by the intermediate nodes based on the descendant tracking information. Each intermediate node forwards an encrypted key update message only if it believes that its descendants know the encrypting key. Using simulation results we showed that our key distribution algorithms reduce the bandwidth needed for distributing key updates in the key management algorithms in [1,2] by upto 55% when compared to the broadcast of key updates.

Also, we described an algorithm for assigning identifiers to group users so that users who are physically close in the multicast tree are assigned logically close identifiers. We showed that our assignment algorithm improves the performance of our key distribution algorithm as well as that of the previous solution in [20]. Using our key distribution algorithm, we described a solution to distribute data messages in secure interval multicast [9]. Interval multicast occurs so often in secure multicast applications that reducing bandwidth during this phase is critical for improving performance.

For overlay multicast protocols [17–19], where the end hosts attempt to reduce bandwidth usage, the use of our key distribution algorithm results in better performance. The processing overhead of users in overlay multicast protocols [17–19] is high as the users need to constantly monitor and reconfigure the overlay links and route multicast data. Our key distribution algorithm reduces the processing overhead of the users by reducing the key update traffic that the users need to process.

In this paper, we have not addressed the correctness and performance of our key distribution algorithm in the face of multicast tree failures and repairs. We note that, this is not a serious issue as we perform periodic updates of the *DT* matrix which handles any changes in the descendant information for an intermediate node. If any users fail to receive key updates because of multicast tree reconfiguration, the group controller simply retransmits them to the users once the multicast tree stabilizes and the *DT* matrices are updated.

For our key distribution algorithm it is not necessary that all the intermediate nodes store the *DT* matrices. We note that, a few selected nodes at appropriate points in the multicast tree are sufficient. We are currently exploring the selection techniques for choosing the best set of intermediate nodes which will participate in the key distribution algorithm. Also, many overlay multicast protocols maintain more links connecting the users. We are exploring methods to exploit the added connectivity to distribute the key updates more efficiently.

References

- [1] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 2000.
- [2] Sandeep S. Kulkarni and Bezawada Bruhadeshwar. Adaptive rekeying for secure multicast. *IEEE/IEICE Special issue on Communications: Transactions on Communications*, E86-B(10):2948–2956, October 2003.
- [3] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. RFC 2627.
- [4] D.McGrew and A.Sherman. Key establishment in large dynamic groups using one-way function trees. Manuscript.
- [5] H.Harney and C.Muckenhirn. Group key management protocol (GKMP) specification. RFC 2093, July 1997.
- [6] S.Mitra. Iolus: A framework for scalable secure multicasting. In *Proc. ACM SIGCOMM'97*, pages 277–288, 1997.
- [7] Sanjeev Setia, Sencun Zhu, and Sushil Jajodia. A comparative performance analysis of reliable group rekey transport protocols for secure multicast. In *Performance Evaluation, special issue on the Proceedings of Performance 2002*, volume 49, pages 21–41, Rome, Italy, 2002.
- [8] Y. Richard Yang, X. Steve Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: A performance analysis. In *Proceedings ACM SIGCOMM 2001*, San Diego, August 2001.
- [9] Mohamed G. Gouda, Chin-Tser Huang, and E.N.Elnozahy. Key trees and the security of interval multicast. In *22nd International Conference on Distributed Systems*, pages 467–468, 2002.
- [10] Sneha Kumar Kasera, Gísli Hjálmtýsson, Donald F. Towsley, and James F. Kurose. Scalable reliable multicast using multiple multicast channels. *IEEE/ACM Trans. Netw.*, 8(3):294–310, 2000.
- [11] Manuel Oliveira, , Jon Crowcroft, and Christophe Diot. Router level filtering for receiver interest delivery. In *Proceedings of NGC 2000 on Networked group communication*, pages 141–150. ACM Press, 2000.
- [12] B. Levine and J. Aceves. Improving internet routing with routing labels. In *Proc. IEEE International Conference on Network Protocols*, October 1997.
- [13] A.J.Ballardie, P.F.Francis, and J.Crowcroft. Core based trees. In *Proceedings of the ACM SIGCOMM*, October 1993.
- [14] T.Pusateri. Distance vector multicast routing protocol. IETF Draft, update to RFC 1075, draft-ietf-idmr-dvmrp-v3-06.txt, June 1998.
- [15] S.Deering et al. Protocol independent multicast, sparse mode protocol: Specification. IETF Draft, work in progress, 1995.
- [16] S.Deering et al. Protocol independent multicast (pim), dense mode protocol: Specification. IETF Draft, work in progress, 1995.
- [17] Y.-H.Chu, S.G.Rao, S.Seshan, and H.Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, October 2002.
- [18] B.Zhang, S.Jamin, and L.Zhang. Host multicast: A framework for delivering multicast to end users. In *IEEE INFOCOM*, March 2000.
- [19] J.Liebeherr, M.Nahas, and W.Si. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, October 2002.
- [20] Di Pietro, L. V. Mancini, Y. W. Law, S. Etalle, and P. J. M. Havinga. Lkhw: A directed diffusion-based secure multicast scheme for wireless sensor networks. In *32nd Int. Conf. on Parallel Processing Workshops (ICPP)*, pages 397–406, October 2003.
- [21] Bill Fenner and Steve Casner. A traceroute facility for ip multicast. Internet Draft, July 2000.
- [22] Ns. ucb/lbnl/vint network simulator - ns (version 2). <http://www-mash.cs.berkeley.edu/ns>.