Performance Optimizations for Group Key Management Schemes for Secure Multicast

Sencun Zhu Sanjeev Setia* Sushil Jajodia
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
{*szhu1,setia,jajodia*}*@gmu.edu*

Abstract

Scalable group rekeying is one of the biggest challenges that need to be addressed to support secure communications for large and dynamic groups. In recent years, many group key management approaches based on the use of logical key trees have been proposed to address this issue. Using logical key trees reduces the complexity of group rekeying operation from $O(N)$ to $O(logN)$, where N is the group size. In this paper, we present two optimizations for logical key tree organizations that utilize information about the characteristics of group members to further reduce the overhead of group rekeying. First, we propose a partitioned key tree organization that exploits the temporal patterns of group member joins and departures to reduce the overhead of rekeying. Using an analytic model, we show that our optimization can achieve up to 31.4% reduction in key server bandwidth overhead over the un-optimized scheme. Second, we propose an approach under which the key tree is organized based on the loss probabilities of group members. Our analysis shows this optimization can reduce the rekeying overhead by up to 12.1%.

Keywords Multicast Security, Group Rekeying, Logical Key Tree, Reliable Multicast, Performance.

Technical Area(s) Security and Network Protocols

1 Introduction

Multicast is a very efficient and scalable technique for group communication. Some multicast applications, e.g., pay-per-view, online auction, teleconferencing, may require a secure communication

*also with Dept. of Computer Science, George Mason University

model. However, IP Multicast, the multicast service proposed for the Internet, does not provide security services in the network layer; indeed, anyone can join a multicast group to receive data from the data sources or send data to the group. Therefore, cryptographic techniques have to be employed to achieve data confidentiality. One solution is to let all members in a group share a key that is used for encrypting data. To provide backward and forward confidentiality [WHA98], this shared key has to be updated on every membership change and redistributed to all authorized members securely. This is referred to as group rekeying.

A simple approach for rekeying a group is one in which the group key server encrypts and sends the updated group key individually to each member. This approach is not scalable because its costs increase linearly with the group size. For large groups with very frequent membership changes, scalable group rekeying becomes an especially challenging issue.

In recent years, many approaches for scalable group rekeying have been proposed, e.g. LKH [WGL98, WHA98], OFT [BM00, CGIMNP98], ELK [PST01], MARKS [Briscoe99], Subset-Difference [MNL01], and BFM [CEKPS99]. All these schemes use logical key trees to reduce the complexity of a group rekeying operation from $O(N)$ to $O(logN)$. Further, it has been proposed that groups be re-keyed periodically instead of on every membership change [SKJ00, YLZL01]. Periodic or batched rekeying can reduce both the processing and communication overhead at the key server, and improve the scalability and performance of key management protocols based on logical key trees.

In addition to the rekeying algorithm, e.g. LKH, OFT, etc., the communication overhead of group rekeying also depends on the protocol used for reliably delivering the changed keys to the members of the group. Recently, researchers have proposed customized reliable multicast protocols [YLZL01, SZJ02] for group rekeying which take advantage of the special properties of the rekey payload for achieving reduced communication overhead in comparison to conventional reliable multicast protocols.

In this paper, we propose two performance optimizations that are applicable to many group key management schemes based on the use of logical key hierarchies. These optimizations involve simple modifications to the algorithms and data structures used by the group key server to maintain the logical key tree for a group. While all the schemes mentioned above do not take member characteristics into account while organizing the logical key tree (indeed, most of them simply try to maintain a balanced key tree), our optimizations use information about the characteristics of specific group members, e.g. the time that has elapsed since a member joined the group or the network packet loss rate experienced by a particular member, while organizing the logical key tree.

The first optimization we present exploits the temporal patterns of group member joins and

leaves to reduce the overhead of rekeying. The main idea is to split the logical key tree into two partitions - a short-term partition and a long-term partition. When a member joins the group, the key server initially places it in the short-term partition. If the member is still in the group after a certain time threshold, the key server then moves it from the short-term partition to the long-term partition. In this way, we can separate the members who participate in the group for different periods. By using separate data structures and/or algorithms for managing the two partitions, we can reduce the overhead of rekeying the group. Using an analytical model, we show that a performance improvement of up to 31.4% can be achieved when a majority fraction of members in a group have short-durations.

The second optimization we present aims to reduce the communication overhead of the previously proposed reliable rekey transport protocols [YLZL01, SZJ02]. Reliable rekey transport protocols are based on receiver-initiated reliable multicast protocols [TKP97]; however, they take advantage of the special properties of the rekey payload for achieving reduced communication overhead. In these protocols, the rekey overhead depends on two factors - the key assignment algorithm used by the key server and the packet loss rates experienced by the members. We propose a scheme in which the key server separates keys needed by high loss members from those needed by low loss members when it assigns keys into packets. More specifically, the key server maintains multiple key trees and places the members with similar loss rates into the same keytree. Our analysis shows this scheme can reduce bandwidth overhead by up to 12.1% over the one-keytree scheme used by WKA-BKR [SZJ02].

The reminder of this paper is organized as follows. In Section 2, we provide background on scalable group rekeying and discuss related work. Then in Section 3 we detail our two-partition scheme and use analytic models to evaluate its performance. Section 4 describes our loss-homogenized scheme and shows its performance. Finally, we summarize our work in Section 5.

2 Background and Related Work

In this section, we briefly review the main ideas underlying the LKH approach and the previously proposed protocols for reliable rekey transport.

2.1 Logical Key Hierarchies (LKH)

The use of logical key trees for scalable group rekeying was independently proposed by Wallner et al [WHA98] and Wong et al [WGL98]. The basis for the LKH approach for scalable group rekeying

is a logical key tree which is maintained by the key server. The root of the key tree is the group key used for encrypting data in group communications and it is shared by all users. The leaf nodes of the key tree are keys shared only between the individual users and the key server, whereas the middle level keys are auxiliary key encryption keys used to facilitate the distribution of the root key. Of all these keys, each user owns and only owns those on the path from its individual leaf node to the root of the key tree. As a result, when a user joins/leaves the group, all the keys on its path have to be changed and re-distributed to maintain backward/forward data confidentiality.

An example key tree is shown in Fig. 1. In this figure, $K_{1\text{-}9}$ is the data encryption key (DEK) shared by all users, K_1, $K_2, \ldots, K_9$ are individual keys, and $K_{123}, K_{456}, K_{789}$ are auxiliary keys known only by users that are in the sub-trees rooted at these keys. We next illustrate member joins and leaves through an example, based on group-oriented rekeying [WGL98].

Join Procedure Suppose in Fig. 1 the root key was $K_{1\text{-}8}$ and K_{789} was K_{78} before user U_9 joins the group, and they are replaced with keys $K_{1\text{-}9}$ and K_{789} respectively when U_9 joins. To distribute these new keys to all users, the key server encrypts $K_{1\text{-}9}$ with $K_{1\text{-}8}$, K_{789} with K_{78}. In addition, the key server encrypts $K_{1\text{-}9}$, K_{789} with K_9. All the encrypted keys are multicast to the group, and each user can extract the keys it needs independently.

Departure Procedure When user U_4 departs from the group, the keys K_{456} and $K_{1\text{-}9}$ need to be changed. Assume these keys are replaced with keys K'_{456} and $K'_{1\text{-}9}$ respectively. Now the key server encrypts $K'_{1\text{-}9}$ with K_{123}, K'_{456} and K_{789} separately, encrypt K'_{456} with K_5 and K_6 separately, and then multicasts these five encrypted keys to the group.

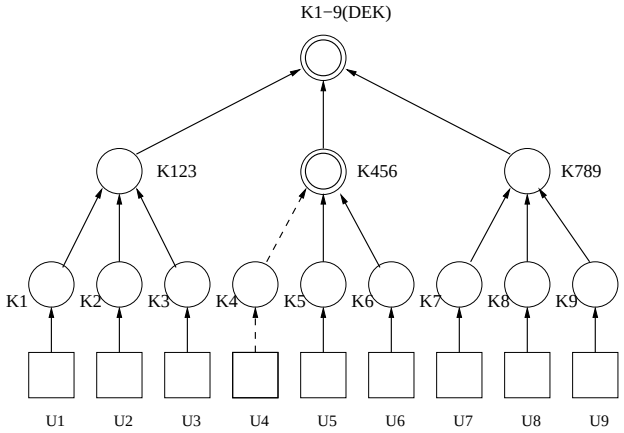

Figure 1: An example logical key tree.

LKH is a very efficient and hence scalable protocol for group rekeying when compared to a unicast-based naive approach. Let N be the group size, d be the degree of the key tree, then the

communication cost for re-keying is $O(log_d N)$, whereas the naive approach requires a communication cost of $O(N)$.

2.1.1 Periodic Batch Rekeying

For a large group with very dynamic memberships, LKH may not perform well [SKJ00] because it performs a group rekeying for every membership change. To reduce the frequency of group rekeyings, researchers [SKJ00, YLZL01] have proposed to use batched rekeying instead of individual rekeying. Batched rekeying can be done in a periodic fashion, so that the rekeying frequency is decoupled from the membership dynamics of a group and hence the processing overhead at the key server can be reduced. In addition, using batched rekeying can reduce the overall bandwidth consumption significantly. This is because when several leaf nodes (corresponding to user keys) in a key tree are changed, there is typically some overlap in the paths from these leaf nodes to the root key. For example, in Fig. 1 when user u4 and u6 both depart from the group during a same rekeying period, k_{1-9} and k_{456} only need to be changed once. In [YLZL01], Yang *et al* have analyzed the bandwidth requirements for batch rekeying under worst-case and average case scenarios.

In this paper, we discuss our optimization algorithms in the context of the batched LKH approach. Other approaches for scalable rekeying such as one-way function trees [BM00] and ELK [PST01] also involve the use of a hierarchical key tree. As such, the basic ideas behind our approaches are also applicable for these group key management protocols.

2.2 Rekey Transport Protocols

On a group rekeying, the key server first runs the batched LKH algorithm to generate a set of encrypted keys that have to be transmitted to the members of the group; then it runs a reliable key distribution protocol that packs these encrypted keys into packets and delivers the packets to the members of the group in a scalable, reliable, and timely manner.

The reliable key distribution problem has some characteristics that differentiate it from the conventional reliable multicast problem. First, there is a *soft real-time* requirement for key delivery, that is, the transport of a rekey message be finished with a high probability before the start of the next rekey interval. To address this issue, i.e. to reduce the latency of key delivery, group rekey transport protocols make use of proactive redundancy. For example, the protocol proposed by Yang *et al* [YLZL01] uses proactive FEC in which parity packets are transmitted along with payload packets in each FEC block. Second, the rekey payload has a *sparseness* property, i.e.,

while the packets containing the new keys are multicast to the entire group, each receiver only needs the subset of packets that contain the keys of interest to it. Thus, if a receiver-initiated, i.e., NACK-based, protocol is used for reliable multicast, a receiver need only provide negative feedback for packets that contain keys of interest to it.

Three main protocols have previously been proposed for reliable rekey transport. The first one is a multi-send protocol [MSEC] that repeatedly sends all keys with the same degree of replication; The second one is the above proactive-FEC based protocol [YLZL01]; and the third one is called WKA-BKR [SZJ02], which is shown to have a lower bandwidth overhead than the other two in most loss scenarios. Since we will demonstrate our optimization for reliable rekey transport protocols using the same analytic model as that of the WKA-BKR protocol, we introduce WKA-BKR in more detail below.

2.2.1 WKA-BKR Protocol

In [SZJ02], Setia *et al* proposed two ideas, weighted key assignment (WKA) and batched key retransmission (BKR), both of which exploit the special properties of logical key hierarchies. The idea of WKA is based on the observation that during a rekey operation, certain keys (typically the keys at higher levels of the logical key tree) are more valuable than other keys since they are needed by a larger fraction of the group's members. Their protocol exploits this property by segregating keys into multiple packets based upon their importance, and then proactively replicating the packets containing more valuable keys to increase the probability that they will be delivered to members in the first round of the protocol. More specifically, WKA first determines the weight, i.e., the expected number of replications, for each updated key based on the number of the members interested in this key and the loss rates of these members. Then it packs the keys (including their replicas) into packets in a breadth-first or a depth-first fashion, and multicasts these packets to the whole group.

In BKR, at the end of each multicast round, the key server collects NACKs from all the members who experience packet losses. Unlike the conventional receiver-initiated reliable multicast that simply retransmits the lost packets to the group, BKR generates and distributes new packets which only contain keys that are still needed by the members. Again, this exploits the *sparseness* property of the rekey payload.

2.3 Other Related Work

Moyer *et al* [MRR99] have suggested to keep the key tree balanced in a key tree based scheme, so that the rekey cost is fixed to be logarithmic to the height of the key tree. However, Selcuk *et al* [SMS00] show that it could be beneficial to use an unbalanced key tree in some cases. Their idea is to organize the key tree with respect to the compromise probabilities of members, in a spirit similar to data compression algorithms such as Huffman and Shannon-Fano coding. Basically, the key server places a member that is more likely to be revoked closer to the root of the key tree. If the key server can know in advance or can make a good guess of the leaving probability of each member, this probabilistic organization of the LKH tree can achieve better performance than that based on a balanced one. Banerjee and Bhattacharjee [BB01] show that organizing members in a key tree according to their topological locations would also be very beneficial, if the multicast topology is known to the key server.

3 A Two-Partition Algorithm for Group Rekeying

In this section, we first motivate the design of our two-partition rekeying algorithm, and then show the algorithm in detail. Finally, we evaluate the performance of this algorithm using an analytic model.

3.1 Motivation

Almeroth and Ammar [AA97] study the multicast group behavior in the Internet's multicast backbone (MBone) based on the collected MBone data. They observe that group members typically either join for a very short period of time or stay for the entire session. As an example, in one session they study, the average membership duration is 5 hours, while the median duration is only 6.5 minutes.

In the context of secure multicast, each member departure results in a group rekeying, no matter what the member duration is. The current rekeying schemes such as LKH and OFT simply maintain a balanced keytree. As a result, the rekeying cost, in terms of the number of encrypted keys to be redistributed, is almost the same on every member departure. For instance, in LKH, if a group has N members in its key tree and the degree of the key tree is d, the rekeying message on a member departure will contain about $d \cdot \lceil log_d N \rceil$ keys.

It is possible to reduce the rekeying cost incurred by the departures of short-duration members. For instance, when a short-duration member joins, rather than mapping it to a leaf node in the key

tree and then sending it all the keys along the path from the leaf to the root (group) key, the key server can send it the group key only; therefore, only the group key needs to be updated when this member departs. Because the new group key is encrypted by its d children nodes for distribution, the rekey message contains only d keys in this case.

While the above approach exhibits an opportunity for performance optimization over the one balanced key tree scheme, the key server needs to know in advance the membership duration of each member. For applications such as media distribution, the access pattern of a member in a particular session is unpredictable to the key server, although the access history of a member may provide some statistics. Thus a deterministic algorithm that can separate the short-duration members from the long-duration members is very desirable.

3.2 Our Approach

The approach we propose to address this issue is to divide the key tree into two partitions, a *S-partition* and a *L-partition*. The *S-partition* is used to hold short-duration members while the *L-partition* is used to hold long-duration members. We can actually view these two partitions as two sub-trees under the root key of the one key tree scheme. The operation of this algorithm can be divided into three phases:

1. When a member joins, the key server puts it in the *S-partition*, and runs a *join procedure* for this partition. In addition, the key server updates the group key and multicasts the new group key to the entire group encrypted with the previous group key. Note this member obtains one or more keys from the *S-partition* through the *join procedure*.

2. If this member departs from the group before a certain threshold time, the key server will run a *departure procedure* in the *S-partition*, i.e., updating all the keys known to this member and then distributing them to the rest of the members in the *S-partition*. Note no keys in the *L-partition* need to be updated because this departed member does not know any keys in the *L-partition*. For members in the *L-partition*, they only need to receive one key, which is the new group key encrypted with the root key of the sub-tree for the *L-partition*.

3. However, if this member stays in the *S-partition* for a time exceeding a certain threshold (we refer to it *S-period* hereafter), the key server then move it to the *L-partition*. Moving a member from the *S-partition* to the *L-partition* impacts both partitions. First, the key server executes a *departure procedure* for the *S-partition*. Second, the key server runs a *join*

procedure for the *L-partition*. Note it is not necessary to update the group key here because this member is still a legitimate member that is authorized to access the data traffic.

The performance gains of our two-partition algorithm over the one key tree scheme are from the second phase, i.e., when a member departs from the *S-partition*. For a long-duration member, the key server has to pay additional price for moving it from the *S-Partition* to the *L-partition*. Clearly, the overall performance of our algorithm depends on the composition of membership durations.

To reduce the overhead for moving a member from the *S-partition* to the *L-partition*, the key server can move multiple members together in a batched fashion. Moreover, when a periodic rekeying scheme is used, the key server can move these long-duration members into the *L-partition* at the moment it processes the departures in the *L-partition*. For the *S-partition*, the key server can also process the joining members, the departed members and the migrated members in a batch. Thus both the partitions perform batched rekeying that can reduce rekeying overhead significantly.

Note the data structures for the two partitions can be different. Below we present two constructions for the two-partition algorithm.

QT-scheme In a *QT-scheme*, the key server uses a linear queue for the *S-partition* and uses a balanced tree for the *L-partition*. The use of a queue for the *S-partition* has two opposing effects. First, a member only needs one key (the group key) on its join. Second, the key server has to encrypt the new group key individually with the secret key of each member residing in the *S-partition* when there is a member departure. Thus this scheme is advantageous when the *S-partition* has a small number of members.

TT-scheme In a *TT-scheme*, the key server uses a balanced tree for each partition. Likewise, the use of a tree for the *S-partition* also makes a tradeoff between the rekeying cost for join and the cost for departure. But this scheme is advantageous when the *S-partition* has a large number of members.

Note that in the schemes described above we do not assume the key server knows in advance the class to which a member belongs. We provide a third two-partition scheme **PT-scheme** where the key server is assumed to have this knowledge as in [SMS00]. The key server simply places a member in one of the partitions based on the class of the member; therefore, no overhead is incurred for moving members between partitions. We include this scheme since it provides an estimate of the best performance that can be achieved.

3.3 Performance Evaluation

3.3.1 Analytic Model

In this part, we analyze the cost of group rekeying when the group is in a steady state. We assume the key server performs periodic batched rekeying.

The performance of our two-partition algorithm depends on the temporal patterns of group members. The temporal results [AA97] show that the membership duration data in a session can roughly fit into an exponential distribution or a Zipf distribution. We adopt a more general model where two exponential distributions are used, one with a small mean M_s and the other one with a large mean M_l. Accordingly, we divide group members into two classes, C_s and C_l. Through varying α, the fraction of group members from each class, we can model the distributions of membership durations for various applications.

Fig. 2 shows a two-class open queueing system for the key server. Let N denote the group size, N_s the number of members in the *S-partition*, N_l the number of members in the *L-partition*, N_{cs} denote the number of members from class C_s, and N_{cl} the number of members from class C_l. Clearly,

$$N = N_s + N_l = N_{cs} + N_{cl}. \tag{1}$$

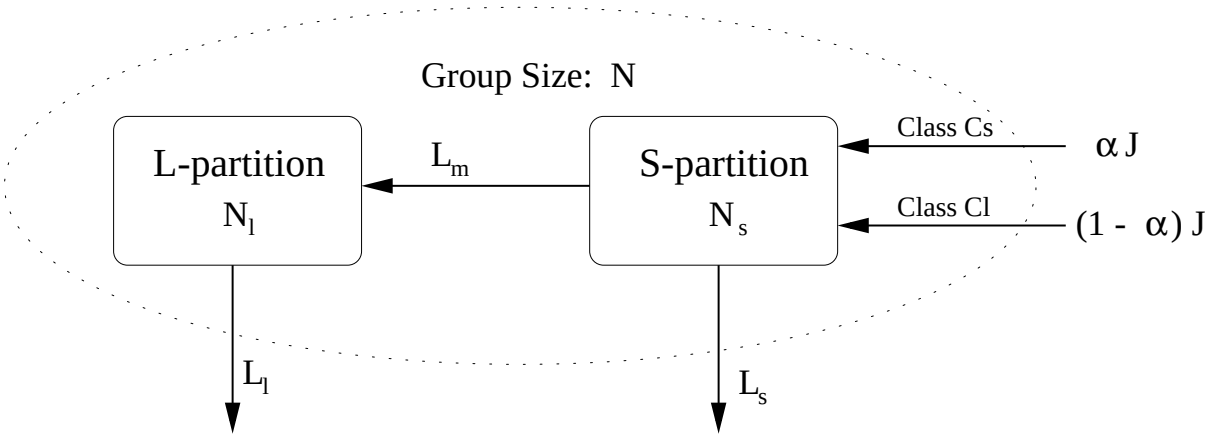

Figure 2: A two-class open queuing system for the key server

Now consider an individual member. For a member whose membership duration T is exponentially distributed with a mean M_i, the probability it departs from the group in any time period of

length t is

$$P_r(t, M_i) = P(T <= t) = 1 - e^{-t/M_i}. \tag{2}$$

Thus in each rekeying period T_p, L_{cs}, the number of departed members from class C_s, is

$$L_{cs} = N_{cs} \cdot P_r(T_p, M_s). \tag{3}$$

Suppose the key server receives J join requests in T_p and a fraction α of these joins are from class C_s members, we have

$$L_{cs} = \alpha \cdot J. \tag{4}$$

Likewise, L_{cl}, the number of departed members from class C_l in T_p, is

$$L_{cl} = (1 - \alpha) \cdot J = N_{cl} \cdot P_r(T_p, M_l). \tag{5}$$

Given N, M_s, M_l and α, we can hence compute N_{cs}, N_{cl}, L_{cs}, L_{cl} and J based on the above formulae.

Now consider the *S-partition*. Let T_s be the *S-period*, and $T_s = K \cdot T_p$, $K \in Z$ for the purpose of batched rekeying. The members in the *S-partition* have stayed for a duration of 0, T_p, $2T_p$,..., or $(K-1) \cdot T_p$. The number of members in the *S-partition* is

$$N_s = \sum_{i=0}^{K-1} (\alpha J \cdot e^{-iT_p/M_s} + (1 - \alpha) J \cdot e^{-iT_p/M_l}). \tag{6}$$

From (1) we have $N_l = N - N_s$. Next, let L_s denote the number of departures from the *S-partition*, L_l the number of departures from the *L-partition*, and L_m the number of migrated members from the *S-partition* to the *L-partition* in T_p. Then we have $L_l = L_m$ in the steady state. Note every time the key server performs a group rekeying, only the members that have stayed in the *S-partition* for T_s are moved to the *L-Partition*. Thus

$$L_m = \alpha J \cdot e^{-T_s/M_s} + (1 - \alpha) J \cdot e^{-T_s/M_l}. \tag{7}$$

In addition, we can compute $L_s = J - L_m$. In Appendix A, we show an algorithm to compute the rekeying cost $N_e(N_t, L_t)$ given N_t, the number of members in a keytree, and L_t, the number of departures from the key tree. Finally, the overall rekeying cost for the *QT-scheme* is

$$C_{qt} = N_{eq} + N_e(N_l, L_l), \tag{8}$$

where $N_{eq} = N_s$ is the rekeying cost for the queue in the *S-partition*, $N_e(N_l, L_l)$ is that for the key tree in the *L-partition*. Similarly, the overall rekeying cost for a *TT-scheme* is

$$C_{tt} = N_e(N_s, J) + N_e(N_l, L_l), \tag{9}$$

where $N_e(N_s, J)$ is the rekeying cost for the key tree in the *S-partition*. For the *PT-scheme*, there are no migration of members between partitions. Indeed, class C_s and C_l members are placed in the *S-partition* and *L-partition* directly on their joins. So the overall rekeying cost for the *PT-scheme* is

$$C_{pt} = N_e(N_{cs}, L_{cs}) + N_e(N_{cl}, L_{cl}). \tag{10}$$

3.3.2 Results

We evaluate the performance of our rekeying scheme based on formula (8), (9) and (10). Table. 1 shows the default parameters for this evaluation.

Table 1: Default Parameter values for evaluation of the two-partition algorithm

Rekeying Period T_p	60 s
Group Size N	65536
Degree of a Keytree d	4
$K = T_s/T_p$	10
Small Mean M_s	3 Minutes
Large Mean M_l	3 Hours
Fraction of Class C_s Members α	0.8

(a): Impact of S-Period

In Fig. 3, we plot the rekeying cost involved in one periodic group rekeying as a function of S-period T_s for three rekeying schemes. We vary $K = T_s/T_p$ instead of T_s. When $K = 0$, our two-partition schemes fall back into the previous one-keytree scheme. We can make the following observations from this figure. First, the TT-scheme can achieve up to 25% bandwidth reduction (at $K = 10$) over the one-keytree scheme. Second, the *TT-scheme* outperforms the *QT-scheme* for a large K. Third, the *PT-scheme* works the best, up to 40% performance gain, because it does not incur the overhead for moving members between partitions.

(b): Impact of Group Heterogeneity

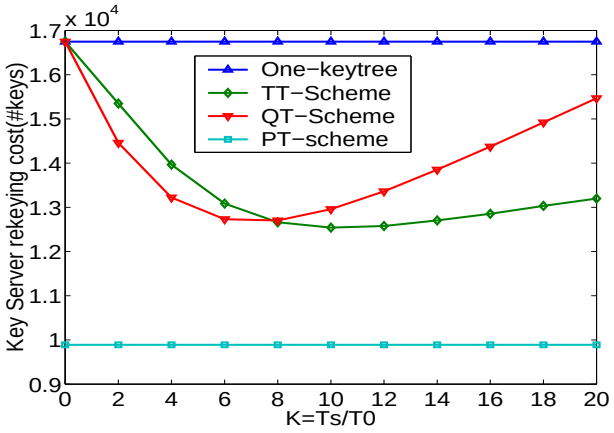

Figure 3: Impact of S-period on key server rekeying cost

To study the impact of the heterogeneity of member temporal patterns, we fix the *S-period* ($K = 10$) and vary α, the fraction of class C_s members. By varying α from 0 to 1, we can simulate a large range of group dynamics. From Fig. 4, we can observe that when α is greater than 0.6, both the *TT-scheme* and the *QT-scheme* outperform the one-keytree scheme. The performance improvement can be up to 31.4% at $\alpha = 0.9$. The *one-keytree* scheme, however, works better than the *TT-scheme* and the *QT-scheme* when $\alpha \leq 0.4$. For a small α, most of the members stay in the group longer than *S-period*, so the key server has to pay the additional overhead for moving them to the *L-partition*. Again, since the *PT-scheme* does not move members between partitions, it works always the best.

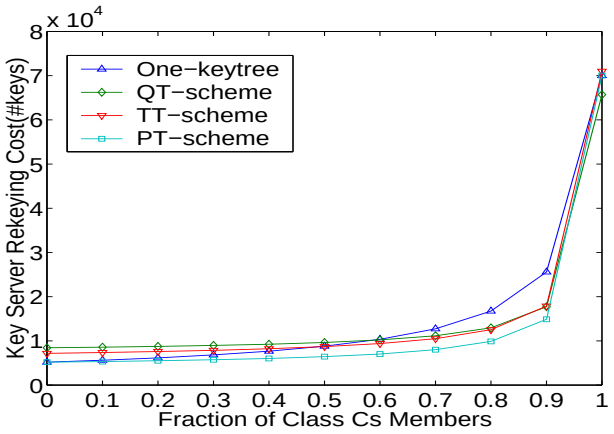

Figure 4: Impact of the heterogeneity of membership durations on key server

(c): Impact of Group Size

We investigate the scalability of our schemes with respect to the group size N. Fig. 5 shows the *reduction* of rekeying cost with N varying from $1K$ to $256K$. The *reduction* of rekeying cost as in the Y-axis is the reduction of the rekeying cost under our rekeying schemes over that under the one-keytree scheme. We can observe that the group size has little impact on the relative performance of our schemes and in average there are more than 22% bandwidth savings in the default scenarios.

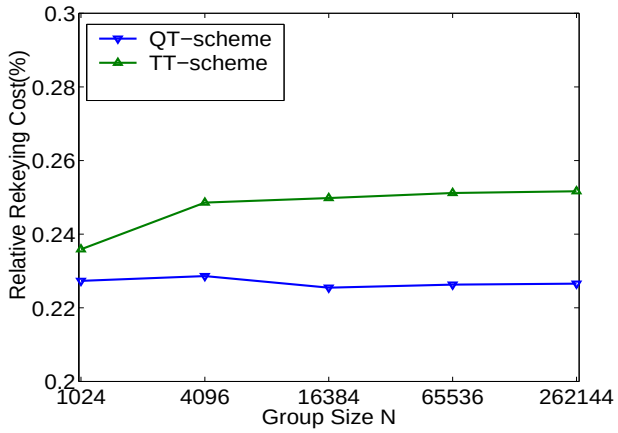

Figure 5: Impact of changing group size on key server

3.4 Discussion

From the analytic results, we can conclude that our two-partition schemes outperform the one-keytree scheme when a group has certain degree of dynamics. The measurement study [AA97] shows that majority of the members joined only for a very short period of time in some sessions; therefore, our schemes are useful for these applications. The previous one-keytree scheme is actually a special case of our schemes where the *S-period* T_s is 0. For real applications, the key server can always adjust T_s for better performance. More specifically, at the beginning of a session, the key server just maintains one key tree; later, from its collected trace data it can compute the group statistics such as M_s, M_l, and α. Then using our analytic model, the key server can choose the best scheme to use. And this process can be repeated periodically.

We note the *PT-scheme*, as well as the scheme in [SMS00], can achieve the best performance in most of the scenarios. However, these schemes depend on the departure probability of each member being known in advance.

4 A Loss-Homogenized Key Tree Organization for Group Rekeying

In this section, we discuss our second optimization in which the key server organizes key trees based on member loss characteristics.

4.1 Motivation

Loss measurements for Internet multicast [Handley97] show the heterogeneity of packet loss on the receiver links; that is, a fraction of the receivers experience relatively high loss, while the others have low loss rates. The traditional one-keytree scheme has not taken into account receivers' loss characteristics while organizing the key tree (hereafter we use the terms "receiver" and "member" interchangeably).

For proactive reliable rekeying protocols such as the WKA-BKR where one key tree is used, the key server replicates a key based on the number of receivers interested in this key and the loss rates of these receivers. Due to the heterogeneity of loss rates among receivers, the low-loss receivers only need a small degree of replication, while those high-loss receivers need a large degree of replication for the keys they are interested in. For example, consider a scenario that most of the members in a multicast group have no packet loss, but a small fraction of them have very high loss rates. If the key server has only one key tree, some updated keys have to be transmitted multiple times because of the existence of these high loss members; however, for the members without any loss, the transmission redundancy is unnecessary. Thus reducing the over-replication of the keys for the low-loss receivers could lead to the overall bandwidth saving for the key server.

4.2 Our Approach

We propose a loss-homogenized scheme, in which the key server maintains multiple key trees that correspond to different loss rates, and put members with similar loss rates in the same key tree. Thus this scheme isolates the impact of high loss members on low loss members. Again, we can view these key trees as the sub-trees under the root (group) key in the one key tree scheme.

In the above example, using our scheme the key server puts the low-loss members into a key tree, and those with high losses into a second key tree. Consequently, all the updated keys in the low-loss key tree need only to be transmitted once, while those in the high-loss key tree may have more redundancy based on the members interested in them.

For this scheme to work in realistic environments, we need first answer the following questions.

First, how does the key server know the loss rate of each member? A member can estimate its loss probability based on the number of packets it failed to receive, and piggyback this information in its retransmission request packets (or NACK) to the key server. In this way, the key server can know the approximate loss rate of each member.

Second, should the key server move a member to another key tree when the loss rate of this member has changed? Due to the overhead involved in moving members between key trees, the key server does not move a member any more once it maps the member to a key tree initially. This requires a member to provide its loss information at its joining time. A member who has participated in the group before can provide its loss information based on the past sessions; however, a member who joins for the first time will have difficulty in providing this information. The two-partition scheme we proposed in Section 3 can help solve this issue because a long-duration member can estimate its loss rate in the time period when it stays in the *S-partition*. For a short-duration member, it might not be that helpful to exploit its loss characteristics because it departs from the group soon after it joins.

Third, what if the estimated loss probability is not accurate due to network dynamics? For example, the key server maps a member with a loss rate 0.2 into a key tree that corresponds to the loss rate 0.02. Performance results from [SZJ02] shows that WKA-BKR protocol is not sensitive to loss heterogeneity; that is, the existence of a small fraction of high loss receivers has little impact on the overall bandwidth overhead. We show the impact of misclassifying receivers in Section 4.3.1.(b).

4.3 Performance Evaluation

The metric of interest for this evaluation is the total number of encrypted keys the key server has to transmit for one (periodic) group rekeying until all receivers get their interested keys. Of these keys, some are proactively replicated (if necessary) and some are retransmitted.

We utilize the WKA-BKR protocol [SZJ02] to evaluate the performance of our loss-homogenized scheme over the one-keytree scheme. The analytic model we adopt is similar to that in [SZJ02] (We provide a sketch of this analytic model in Appendix B). Basically, given N, the number of receivers in a key tree, p_i, the loss rate of each receiver, and L, the number of departed receivers from this key tree during one rekeying event, their analytic model can compute the expected number of encrypted keys that are in the rekey payload. Because in our scheme the key server has multiple key trees, we use their model to compute the rekeying cost incurred in each key tree. We let the number of departed members from a key tree be proportional to the total number of members in

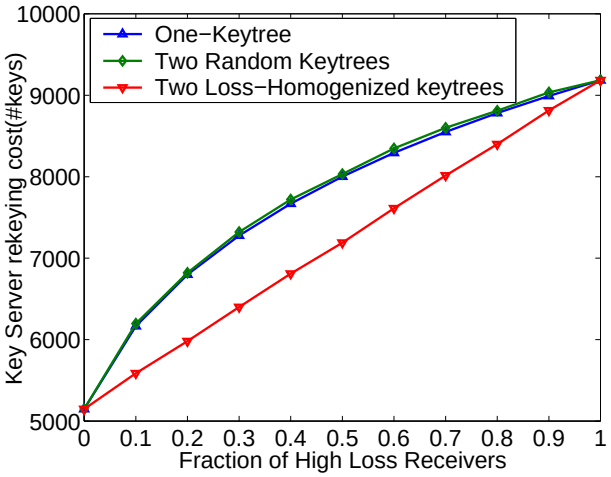

Figure 6: Impact of Group Loss Heterogeneity

the key tree. As a default, we choose $N = 65536$, $L = 256$, and the degree of the key trees $d = 4$ for our performance evaluation.

4.3.1 Performance Results

(a) Impact of Group Loss Heterogeneity

To investigate the impact of group loss heterogeneity, we assume a fraction α of members experience high loss rate $p_h = 20\%$, while the others have low loss rate $p_l = 2\%$. Varying α from 0 to 1 allows us to simulate a large range of group loss heterogeneity. In Fig. 6, in addition to the one-keytree scheme, we also show a two-random-keytree scheme where the key server also maintains two key trees as in our loss-homogenized scheme except the members are randomly placed into the key trees. This allows us to differentiate the performance gains that result from using two key trees as opposed to two loss-homogenized key trees. We can make the following observations from Fig. 6. First, the two-random-keytree scheme works even slightly worse than the one-keytree scheme, which shows random partitioning of a key tree into multiple smaller ones does not help improve the performance. Second, when the members have heterogeneous losses, our loss-homogenized scheme can outperform the one-keytree scheme by up to 12.1% (when $\alpha = 0.3$). This shows that organizing key trees based on member loss rates is beneficial. Third, when the members have homogeneous losses(when $\alpha = 0$ or $\alpha = 1$), all the schemes have the same performance. This is true because our loss-homogenized scheme falls back to the one-keytree scheme in these cases.

(b) Impact of Misplacement of Members

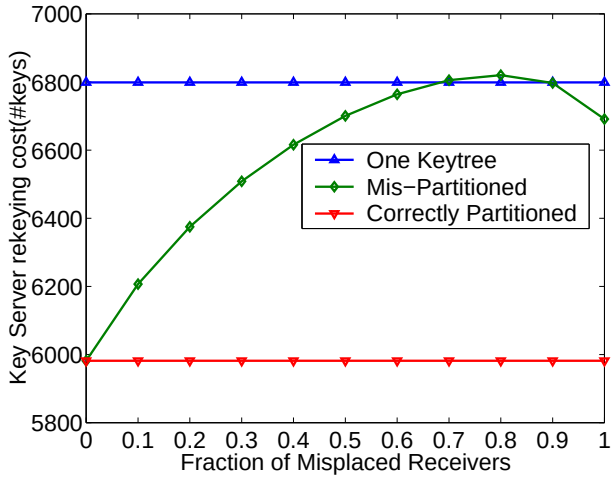

Figure 7: Impact of misplacement of members when organizing key trees

Now we show the impact of misplacing members into key trees at their join time. Note that this will happen if the key server does not have accurate information about the loss rate of a joining member. We choose $p_h = 20\%$, $p_l = 2\%$, and $\alpha = 0.2$. Under this setting, our *correctly partitioned* scheme maintains two key trees, one has all the low loss members $((1-\alpha)\cdot N)$ and the other one has all the high loss members $(\alpha \cdot N)$. In the *mis-partitioned* case, we keep the size of each key tree invariant, but change a fraction β of the high loss members into low loss in the high loss key tree and change the same number of low loss members into high loss in the low loss key tree. From Fig. 7 we can see that the performance of our loss-homogenized scheme degrades as the fraction of misplaced members increases. At $\beta = 0.8$, which means $0.8 * 0.2 * 65536 = 10486$ members are misplaced, our scheme works even slightly worse than the one-keytree scheme. But if the fraction of misplaced members is small, say β is less than 0.1 which means $0.1 * 0.2 * 65536 = 1310$ members, our scheme still outperforms the one-keytree scheme. Note when $\beta = 1.0$, the misplaced low loss members occupy the original high loss key tree, thus the rekeying cost for this loss-homogenized key tree is very small. This is the reason we observe that the performance of our scheme at $\beta = 1.0$ is even better that at $\beta = 0.8$.

4.4 Discussion

In Section 4.3, we evaluated the performance of our loss-homogenized scheme based on WKA-BKR. We have also evaluated our scheme based on proactive FEC [YLZL01] (although we do not show it here), where we find the performance gain is more significant - up to 25.7% when $p_h = 20\%$,

$p_l = 2\%$ and $\alpha = 0.1$. FEC based reliable transport protocols, however, are more sensitive to group loss heterogeneity, thus a small fraction of misplacement of members could reduce the efficiency of our scheme more than that based on WKA-BKR. In this case, the algorithm in [ZLLY01] might be useful to reduce the impact.

We note there are protocols [YSI99] using multiple multicast groups for reliable multicast, where receivers with different loss rates join in different multicast groups. If our loss-homogenized scheme is applied, the key server can maintain one key tree for each group. Using multiple groups does not affect the rekeying overhead for the key server, whereas the receivers can reduce their bandwidth consumption significantly for receiving the keying materials because of the *sparseness* property of rekey payload. Moreover, it helps achieve inter-receiver fairness because the low loss members will not receive redundant keys that are unnecessary to them.

5 Conclusions

In this paper, we have presented two schemes that utilize the characteristics of multicast group members to reduce the overhead of group rekeying operations. We also present an analytical model for evaluating the performance of the two-partition scheme. Through analytical studies, we compare our schemes with the previous schemes that simply use one balanced keytree. The main conclusions of our study are:

- Our two-partition scheme outperforms the one-keytree scheme when more than 50% of the members of a group have short membership durations. The reduction of rekeying bandwidth overhead can be up to 31.4%. For applications that have very stable memberships, the one-keytree scheme is preferred. Our two-partition scheme can adapt to membership dynamics and easily fall back to the previous one-keytree scheme when desired.

- By considering member loss characteristics while organizing the key tree, our loss-homogenized scheme can reduce the rekeying overhead by up to 12.1% over WKA-BKR.

References

[AA97] K. Almeroth and M. Ammar, Multicast Group Behavior in the Internet's Multicast Backbone (Mbone), IEEE Communications, June 1997.

[BB01] S. Banerjee, B. Bhattacharjee Scalable Secure Group Communication over IP Multicast, International Conference on Network Protocols (ICNP) 2001, Riverside, California, November 2001

[Briscoe99] B. Briscoe, MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences, in Proc First International Workshop on Networked Group Communication (NGC'99), Nov 1999.

[BM00] D. Balenson, D. McGrew, and A. Sherman, Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization. IETF Internet draft (work in progress), August 2000.

[CEKPS99] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha. Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques. In Proc. of IEEE INFOCOM'99, volume 2, March 1999.

[CGIMNP98] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In Proc. of IEEE INFOCOM 99

[Handley97] M. Handley. An examination of MBONE performance. Jan.1997.

[HCM98] T. Hardjono, B. Cain,and I. Monga. Intra-Domain Group Key Management Protocol. Internet Draft. Draft-ietf-ipsec-intragkm-00.txt, November 1998.

[HH99] H. Harney, E. Harder. Logical Key Hierarchy Protocol. Internet Draft, draft-harney-sparta-lkhp-sec-00.txt, March 1999

[MSEC] IETF Multicast Security (MSEC) Working Group. http://www.securemulticast.org

[MRR99] M. Moyer, J. Rao and P. Rohatgi. Maintaining Balanced Key Trees for secure Multicast. Internet Draft, draft-irtf-smug-key-tree-balance-00.txt, June 1999

[MNL01] D. Naor, M. Naor, and J. Lotspiech. Revocation and Tracing Schemes for Stateless Receivers, CRYPTO 2001.

[NBT98] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. IEEE/ACM Trans. Networking, vol.6, pp.349-361, August, 1998.

[PST01] A. Perrig, D. Song, D. Tygar, ELK, a new protocol for efficient large-group key distribution. in: Proceedings of the IEEE Security and Privacy Symposim 2001, May 2001.

[SKJ00] S. Setia, S. Koussih, S. Jajodia. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. IEEE Symp. on Security and Pravicy, Oakland CA ,2000

[SMS00] A. Selcuk, C. McCubbin, D. Sidhu. Probabilistic Optimization of LKH-based Multicast Key Distribution Schemes. Draft-selcuk-probabilistic-lkh-01.txt, Internet Draft Jan.2000

[SZJ02] S. Setia, S. Zhu and S. Jajodia. A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast. To appear in Performance 2002, Rome, Italy, Sep., 2002.

[TKP97] D. Towsley, J. Kurose, S. Pingali, A comparison of sender-initiated and receiver-initiated reliable multicast protocols, IEEE J.Select Areas Commun. 15(1997) 398-406.

[WGL98] C. Wong, M. Gouda,S. lam. Secure Group Communication Using Key Graphs. In Proc. Of SIGCOMM'98, 1998

[WHA98] D. Wallner, E. Harder and R. Agee. Key Management for Multicast: Issues and Architecture. Internet Draft, draft-wallner-key-arch-01.txt, September 1998

[YLZL01] Y. Yang, X. Li, X. Zhang and S. Lam. Reliable group rekeying: Design and Performance Analysis. Proc. of ACM SIGCOMM2001, San Diego, CA, USA, August 2001.

[YSI99] M. Yamamoto, Y. Sawa and H. Ikeda Layered Multicast Group Construction for Reliable Multicast Communications. Networked Group Communication 1999: 19-35

[ZLLY01] X. Zhang, S. Lam, D. Lee, Y. Yang, Protocol design for scalable and reliable group rekeying, in Proceedings of the SPIE Conference on Scalability and Traffic Control in IP Networks, Denver, Aug. 2001.

Appendix A

In this appendix, we compute the average number of encrypted keys for batched rekeying, given the total number of members, N, the number of batched revoked members, L, and the degree of the key tree, d. For simplicity, we assume the key tree is full and balanced, and the number of joins in the same rekeying period $J = L$. In addition, we assume the membership duration of each member is exponentially distributed with a same mean.

The analysis below is based on that in [YLZL01]. According to the "memoryless" property of an exponential distribution, the probability that a member will depart from a group depends not on

the time it has spent in the group, but on the mean of its membership duration. Since all members have the same mean for their membership durations, they have the same probability to depart from the group. In addition, these departed members are uniformly distributed in the leaves of the key tree. When a member departs, all the keys along the path from its leaf node to the root of the key tree (except the leaf node) are updated and then encrypted with each of its children nodes individually for secure redistribution.

For an intermediate key node K_i in level i of the key tree (i starts from 0 at the root key), the number of members under the subtree rooted at K_i is $S_i = d^{h-i}$, where $h = log_d N$ is the height of the keytree. Given totally L departures in N leaf nodes, the probability that K_i is updated because at least one departures occur under its subtree is

$$P_i = 1 - \frac{\binom{N-S_i}{L}}{\binom{N}{L}}. \tag{11}$$

Because there are d^i keys in level i, the number of updated keys in level i is $N_i = d^i P_i$. Finally, the average number of keys to be encrypted is given by:

$$N_e(N, L) = \sum_{i=0}^{h-1} d \cdot N(i). \tag{12}$$

In the above analysis, we assumed a fully balanced key tree. In real cases, a key tree may be partially full, depending on N and d. The analysis for a partially full key tree can be obtained from a simple extension to the above analysis.

Appendix B

We give a sketch of the bandwidth analysis model in WKA-BKR [SZJ02], using a balanced logical key tree with degree d and height h.

Consider an updated key, K, at level l of the logical key tree, where $0 \leq l \leq h-1$. There will be d encryptions of K in the rekey payload, each of which needs to be transmitted to $R(l) = d^{h-l-1}$ members. The probability that one of these $R(l)$ receivers (say r) will not receive K if it is transmitted once is equal to the probability of packet loss, p, for that receiver. Let M_r be the the number of key transmissions necessary for receiver r to successfully receive the key, K, then $P[M_r \leq m] = 1 - p^m, m \geq 1$, and $E[M_r] = 1/(1-p)$.

Let $M(l)$ be the number of times a key K at level l will need to be transmitted in order to be successfully delivered to all $R(l)$ receivers. Since lost packet events at different receivers are

independent, for key K, we have

$$P[M(l) \leq m] = \prod_{r=1}^{R(l)} P[M_r \leq m] = (1 - p^m)^{R(l)} \tag{13}$$

Thus,

$$E[M(l)] = \sum_{m=1}^{\infty} P[M(l) \geq m] = \sum_{m=1}^{\infty} (1 - (1 - p^{m-1})^{R(l)}) \tag{14}$$

Now that we have determined the weight(number of replications) for an arbitrary key in the key tree, we can compute the overall rekeying cost as the sum of the weights for all the updated keys in a rekey event. Formula (11) in Appendix A has shown that a key in level l will be updated with a probability P_l when J joins and L leaves are processed as a batch and $J = L$, so $U(l)$, the average number of updated keys at level l is $U(l) = d^l \cdot P_l$. Thus $E[V]$, the expected total bandwidth used in a rekey event is

$$E[V] = \sum_{l=0}^{h-1} d \cdot U(l) \cdot E[M(l)]. \tag{15}$$