A Scalable and Reliable Key Distribution Protocol for Multicast Group Rekeying

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

ABSTRACT

Scalable group rekeying is one of the important problems that needs to be addressed in order to support secure communications for large and dynamic groups. One of the challenging issues that arises in scalable group rekeying is the problem of delivering the updated keys to the members of the group in a reliable and timely manner. In this paper, we present a new scalable and reliable key distribution protocol for group key management schemes that use logical key hierarchies for scalable group rekeying. Our protocol, called WKA-BKR, is based upon two key ideas, weighted key assignment and batched key retransmission, both of which exploit the special properties of logical key hierarchies to reduce the overhead and increase the reliability of the key delivery protocol. We have evaluated the performance of our approach using detailed simulations. Our results show that for most network loss scenarios, the bandwidth used by our protocol is lower than that of previously proposed key delivery protocols.

1. INTRODUCTION

Many emerging Internet applications (e.g., real-time information services, pay per view, distributed interactive simulations, multi-party games) are based on a secure group communications model. In this model, authorized members of a group share a symmetric group key that is used to encrypt group communications. To provide forward and backward confidentiality [33], the shared group key is changed on each membership change and securely redistributed to the existing members of the group. This is referred to as group rekeying.

For large groups with frequent membership changes, the costs of rekeying the group can be quite substantial. The straightforward approach under which a new group key is generated on each membership change, encrypted individually and transmitted to each existing group member is not scalable since the costs of this approach increase linearly with the size of the group. Scalable rekeying is therefore an important and challenging problem that needs to be addressed in order to support secure communications for large and dynamic groups.

In recent years, several approaches for scalable group rekeying have been proposed [22, 33, 34, 32, 12, 1, 25]. One prominent approach [34, 33, 15] uses logical key hierarchies or key trees to reduce the complexity of the group rekeying operation to $O(logN)$, where N is the size of the group. Further, it has been proposed that groups be rekeyed periodically instead of on every membership change [6, 27, 37]. Periodic or batched group rekeying has been shown [37] to reduce both the processing and communication overhead at the key server, and to improve the scalability and performance of key management protocols based on logical key trees.

In group key management schemes based on logical key hierarchies or key trees (henceforth referred to as LKH), group re-keying involves two operations - key encoding and key distribution. Key encoding involves determining which keys in the logical key tree need to be changed, and encrypting the changed keys using the LKH algorithm. Key distribution involves packing the encrypted keys into packets and executing a protocol that ensures that all the packets are delivered to the members of the group in a timely fashion.

For scalable group rekeying, clearly both the key encoding and key distribution operations need to be scalable. Most previous works on scalable group re-keying have focussed on improving the efficiency and reducing the complexity of the key encoding operation; reliable key distribution has not received the same degree of attention. Most group rekeying schemes make one of the following two assumptions regarding reliable key distribution:

- Scalability is not a concern because very few packets will need to be sent when the group is rekeyed. Key update packets that are multicast to the group can simply be replicated to provide additional reliability.
- One of the scalable reliable multicast protocols that have been proposed, e.g. SRM [9], RMTP [20] can be used for key delivery.

The first assumption does not hold when the group is very large or if group rekeying is done periodically resulting in several group joins and leaves being processed in a single batch. In these cases, the number of keys that needs to be changed can be large enough that several packets need to be multicast reliably to the whole group, making it a challenging proposition. Although reliable multicast schemes such as SRM or RMTP can be used for reliable delivery of keys, these schemes are complex and (in some cases) require additional support from the network infrastructure. Further, at present, these protocols do not incorporate any security mechanisms, and opportunities exist for denial-of-service and other security attacks [13].

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

Consequently, we believe that reliable key delivery is an important problem in its own right that needs to be addressed in group key management schemes.

In a recent study, Yang et al [37, 38] investigated the reliable key delivery problem, and proposed a protocol for key delivery. They observe that the reliable key delivery problem has two features that distinguish it from the conventional reliable multicast problem. First, there is a soft real-time requirement for key delivery; group members have to buffer any encrypted data or keys they receive until the encrypting keys have arrived. To limit the size of these buffers, it is necessary to deliver the keys each member needs as soon as possible. Secondly, 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. Based on these observations, they propose a reliable key delivery protocol based on the use of proactive FEC [26]. This protocol is incorporated into their group key management system, Keystone [35].

In this paper, we present a new scalable and reliable key distribution protocol for group key management schemes that are based on hierarchical key trees. Our approach is based upon two key ideas: *proactive redundancy* and *batched key retransmission.* Like the Keystone key delivery protocol, our approach uses proactive redundancy for achieving reliability and ensuring timely delivery of keys. Unlike the Keystone protocol, we do not use FEC for redundancy; instead our protocol uses a Weighted Key Assignment (WKA) algorithm that exploits the special properties of a logical key hierarchy while assigning keys to packets that are multicast to the group. Our algorithm 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. Our protocol exploits this property by segregating keys into multiple sets depending upon the number of members who need the keys. Packets containing more valuable keys are proactively replicated to increase the probability that they will be delivered to members in the first round of the protocol.

The idea of *batched key retransmission* (BKR) is also based on the special properties of a logical key hierarchy. In a conventional receiver-initiated reliable multicast protocol, when a sender receives NACKs for packets from specific receivers, it responds by retransmitting the corresponding packets to the group. In the case of a reliable key delivery protocol, a packet will typically contain several keys most of which are not needed by a specific receiver. Instead of re-sending the whole packet to the group, our protocol determines the keys that are needed by the receivers who responded with NACKs to the initial multicast, packs these keys into new packets (again using the WKA algorithm) and multicasts them to the group.

We have evaluated the performance of our approach, which we refer to as WKA-BKR, using extensive simulations. We find that for most network loss scenarios, WKA-BKR reduces the bandwidth required at the key manager in comparison to previously proposed protocols. The bandwidth reductions achieved are significant – up to 22% improvement over FEC-based protocols and up to 45% over simpler replication based protocols. WKA-BKR is also less sensitive to changes in network loss conditions than proactive FEC-based protocols, and outperforms these protocols over a wide range of group sizes and membership dynamics. Finally, WKA-BKR is computationally inexpensive for both the key manager and group members.

The organization of the rest of the paper is as follows. In Section 2, we describe our approach in more detail. Next, in Section 3, we evaluate the performance of our approach and compare it to that of key delivery protocols based on proactive FEC. In Section 4, we discuss related work on key distribution protocols for secure multicast. Finally, Section 5 contains our conclusions.

2. THE WKA-BKR KEY DELIVERY PROTOCOL

In this section, we first present an overview of our approach. We then discuss the WKA and BKR algorithms with the help of an example.

2.1 Protocol Overview

As discussed in the introduction, group rekeying involves two operations – key encoding and key distribution. Key distribution is concerned with packing the encrypted keys into packets and delivering the packets to the members of the group in a scalable, reliable, and timely manner.

Our key delivery protocol is based on multicast and uses an application-level receiver-initiated, i.e., NACK-based, approach [30] for reliability. The operation of the protocol can be divided into two phases:

1. The key transmission phase in which packets containing encrypted keys are generated using the WKA algorithm and multicast to the group. This phase includes the first round of the multicast.

2. The retransmission phase in which the key server generates new packets using batched key retransmission in response to NACKs received from users and multicasts them to the group. This phase continues until all group members have received their keys. Note that there may be multiple rounds of multicast in this phase.

One of the features of the reliable key delivery problem that distinguishes it from a conventional reliable multicast is that a member will typically need only a small subset of the keys that are multicast to the group. Thus, in each multicast round, a receiver sends a NACK to the key manager only if it has not received the specific packets containing the keys it needs. This "sparsity" property is also exploited by the Keystone reliable key delivery protocol [37]. We note that a receiver will detect the fact that it is missing key update packets if it receives data or key packets that it cannot decrypt. If a periodic batch rekeying policy [37, 27] is in use, then receivers can also detect lost key packets if the time that has elapsed since the receipt of a key update packet is greater than the rekey period.

2.2 Weighted Key Assignment

To understand the motivation behind weighted key assignment, we need to consider the properties of a logical key tree. The discussion below assumes a familiarity with the LKH approach [33, 34].

Consider a group with N = 1024 members. Assume that the logical key tree for this group is balanced and has degree 4.

As shown in Figure 1, there are 1024 u-nodes in the keytree corresponding to the individual keys of the members of the group and 341 k-nodes corresponding to the group key and auxiliary keys.

Assume that the group is rekeyed periodically, and that in the last period, there were 64 leaves and 64 joins that have to be processed as a batch. In this scenario, the 64 departed members will be replaced by the 64 new members in the key tree. All the keys corresponding to k-nodes on the path from a changed u-node to the root of tree have to be changed during the rekey operation. During the encoding phase of the group rekeying, the keys in the logical key tree that need to be changed are identified and updated, and the key server encrypts these keys using LKH. These encrypted keys then have to be delivered to the members of the group using a key delivery protocol.

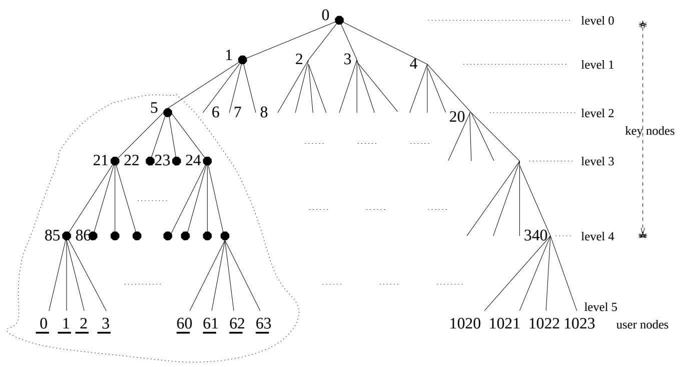


Figure 1: The Best-case Scenario for Group Rekeying for a Logical Key Tree with degree 4, Group Size = 1024, Number of Leaves = 64. Note that the new u-nodes are underlined and the updated k-nodes are denoted by black circles.

The bandwidth requirements of group rekeying depend upon the number of encrypted keys that have to be transmitted to the members of the group. This number in turn depends upon the location of the departed and newly joined members. If the departed members are clustered together as shown in Figure 1 the number of changed keys is minimized, whereas the worst case scenario occurs when the departed members are evenly distributed among the leaf nodes of the key tree. We now describe our approach using the best case scenario shown in Figure 1 as an example.

Consider the logical key tree in Figure 1. We observe that the higher a key in the key tree, the more the number of members who need the key. K_0 (the group key) is needed by all 1024 members, K_1 is needed by the 256 members who are its descendants in the key tree, K_5 is needed by 64 members, and so on. Note that an updated key has to be encrypted using each of the keys corresponding to its child nodes in the key tree. Thus, to be more accurate, each encryption of the root key using one of its children, e.g., $\{K_0\}_{K_1}$, is needed by 256 members, each encryption of K_1, e.g., $\{K_1\}_{K_5}$, is needed by 64 members, and so on. Note that all the keys in the subtree with K_5 as its root are needed only by the 64 members corresponding to u-nodes in that subtree.

From these observations, it follows that the level at which a key resides in the key tree is an indication of its importance from the viewpoint of reliable delivery (Perrig et al [25] have previously made the same observation while motivating the design of ELK, their multicast key distribution protocol.) The basic idea behind Weighted Key Assignment (WKA) is that the packets that contain more valuable keys are proactively replicated during the multicast phase. Thus packets containing keys residing at higher levels of the keytree are multicast several times, while packets containing keys at lower levels are multicast just once.

The key transmission phase involves three steps:

1. In the first step, the encrypted keys are divided into two sets based on the level of the key in the key tree. All keys that reside at or above a certain level (say k) are placed in the first set, S1. All keys below level k are placed in the second set, S2. Finally, some of the keys in S2 that are at higher levels of the key tree relative to the other keys in S2 (e.g., keys at levels $k+1$ and $k+2$) may be placed in a third set, S3. Each set is associated with a different weight that determines how many times packets that contain the keys in that set will be transmitted during the multicast; hence, the name Weighted Key Assignment (WKA).

2. In the second step, the keys in the sets S1, S2, and S3 are separately assigned to packets. The algorithm used for packing a set of keys into packets can have a significant impact on the performance of the protocol [37]. For example, if the keys are assigned to packets in a breadth first fashion, the keys needed by a specific receiver may be distributed among several packets, implying that that receiver will need to receive all those packets in order to obtain its keys. Our key assignment algorithm (which is discussed in more detail in Section 2.2.1) assigns the keys to packets in such a way that each user will need to receive at most two packets – one packet from S1 and one packet from S2 or S3 – to obtain all its keys.

3. In the third step, the packets are multicast to the group. Packets containing the keys in S1 and S3 (if any) are typically multicast several times, while packets containing the keys in S2 are multicast just once. Note that during this phase, the replicated packets can be transmitted in an interleaved manner in order to increase the probability that they will be received even if the network is experiencing correlated packet losses.

There are three decisions that have to be made while executing the steps listed above: (i) At what level, k, of the keytree, should the keys be split into S1 and S2? (ii)What keys (if any) from S2 should be included in S3? (iii) What should be the proactive replication factor for packets in S1 and S3? These decisions are made based on a cost-benefit analysis that is outlined in Section 2.2.2 below.

In our example, in the first step of the algorithm, the encrypted keys corresponding to K_0 and K_1 (which are at level 0 and 1 of the keytree) are placed in set S1, whereas the remaining keys are placed in set S2. In this example, there is no need for placing any keys in S3 [1]. In the second step, keys in each set are separately assigned to packets. Assuming that up

[1]See Appendix A for an example in which some keys in S2 are also included in S3.

to 40 keys can fit into one packet, the 8 encrypted keys corresponding to K_0 and K_1 fit into one packet (say P0), whereas three packets (say P1, P2, and P3) will be needed for the 84 encrypted keys corresponding to the remaining 21 keys. In the next step, these packets are multicast to the group using different levels of redundancy. Packet P0 is multicast multiple times, whereas P1, P2, and P3 are multicast just once.

2.2.1 Key Assignment Algorithm

The keys in sets S1, S2, and S3 are assigned to packets in the second step of the transmission phase of our protocol. There are several possible algorithms that can be used for assigning keys to packets, e.g., breadth first assignment, depth first assignment, etc. As mentioned earlier, these algorithms can have a significant impact on the performance of the key delivery protocol since they determine the number of packets each receiver needs during the rekey operation.

Previously, Yang et al [37] have compared the performance impact of various key assignment algorithms for their key delivery protocol. One of the algorithms they have proposed and investigated is User-oriented Key Assignment (UKA). This algorithm assigns all the keys needed by a user (i.e., all the keys on the path from the u-node corresponding to that user to the root of the key tree) into a single packet; thus, this algorithm ensures that any group member needs to receive only one packet during the rekeying operation. The disadvantage of this method, however, is that some encrypted keys (corresponding to the keys at the higher levels of the keytree) tend to be duplicated into several packets, and these duplications can dominate bandwidth overhead in some situations.

In our approach, we use a variant of UKA to assign keys to packets. The goal of our key assignment scheme is to ensure that a user will need to receive at most two packets – one containing the keys it needs from the upper part of the keytree (i.e., from S1) and, if necessary, one containing the keys it needs from the lower part of the keytree (i.e., from S2 or S3). Thus, our key assignment scheme first divides the keys into sets S1, S2, and S3 and then applies UKA separately to the keys in S1, S2, and S3.

Applying UKA to a set of keys is straightforward. It involves traversing a rekey subtree (produced by the LKH algorithm) in a depth first order, and sequentially assigning the encrypted keys to packets. When a new packet is started, however, this new packet will first include all the keys along the path from the root node to the next key node to be visited in the depth first traversal. Hence, the duplication overhead of this scheme increases with the height of the keytree. Since our approach applies UKA separately to sets S1, S2, and S3, the duplication overhead is negligible.

For example, consider the subtree rooted at K_5 in Figure 1. As discussed above, the keys in this subtree are included in the set S2. There are 84 encrypted keys (corresponding to the 21 updated k-nodes) that have to be assigned to packets. As discussed above, we will need three packets P1, P2, and P3 for these keys. Given that there are 64 u-nodes in the subtree, we can apply UKA such that the keys needed by members 0-20, 21-42, and 43-63 are in packets P1, P2, and P3 respectively. In this example, $\{K_5\}_{K_{22}}$ will be duplicated in P1 and P2, and $\{K_5\}_{K_{23}}$ will be duplicated in P2 and P3.

2.2.2 Weight Assignment

The weight (i.e., the degree of replication) for each group of packets has to be selected based on the keys it contains as well as the observed packet loss properties of the network. Replicating a packet several times in the first round of the protocol has the following effects:(i)it *increases* the bandwidth consumed in the first round of the protocol, (ii)it increases the probability that that every member will receive the keys it needs in the first round of the protocol, and therefore *decreases* the bandwidth consumed due to NACKs in the first round, and due to retransmission of keys in subsequent rounds. Thus there is a tradeoff between the benefits and costs of replicating a packet, and it is crucial that the weights assigned to each group of packets be selected judiciously.

To illustrate this point, let us continue our example corresponding to Figure 1. Consider a network in which each receiver experiences independent packet losses with probability p, and that the losses of different packets are independent, i.e., a homogeneous packet loss scenario used in analyses of reliable multicast protocols [30, 23]. Let us assume that $p = 0.1$. To determine how many times a packet P0 should be replicated, we need to take into consideration both p and the number of members who need P0. Since P0 contains the encrypted group key it is needed by all 1024 members. If P0 is not proactively replicated, the expected number of members who will not receive P0 in the first round is $1024 \times 0.1 \approx 102$. In a conventional receiver-initiated reliable multicast protocol, this will lead to 102 NACKs, followed by a second round of transmission of P0, followed by NACKs, and so on until each member has received the packet. On the other hand, if in the first round, we multicast P0 three times, the expected number of members who will not receive P0 in the first round goes down to $1024 \times 0.1^3 \approx 1$. The total bandwidth used (which is equal to the sum of the bandwidth used in each round of the multicast as well as the bandwidth consumed by NACKs) if P0 is replicated three times in the first round is clearly less than the bandwidth used if it is not replicated; thus, in this case, a degree of replication of 3 is appropriate for P0.

Let us now consider the remaining packets – P1, P2, and P3. These packets are needed by the 64 members in the subtree rooted at K_5. More specifically, each member needs one of these three packets since our algorithm for packing keys into packets ensures that all the keys that a particular member needs from the set S2 will be in one packet. If the keys for the 64 receivers are divided evenly among the three packets, each of P1, P2, and P3 will be needed by approximately $64/3 = 21$ receivers.

As pointed out earlier, our reliable key transport protocol differs from a conventional reliable multicast protocol in that a receiver responds to a multicast with a NACK only if it does not receive the packets that contain keys that it needs. A receiver whose keys are in packet P1 does not care if it does not receive P2 and P3. Thus it will respond with a NACK after the first round of transmission only if it has not received P1. If the packets P1, P2, and P3 are sent without replication, the expected number of receivers who will respond with NACKs is $64 \times 0.1 \approx 6$, which is a small fraction of the 1024 members of the group. Thus there is no danger of NACK implosion at the key server if these packets are not replicated, justifying the decision to send them just once.

In Appendix B, we present an approximate analytical model that computes the expected bandwidth requirements of our protocol, given a set of keys that are to be updated. Note that the keys that need to be updated are known at the end of the key encoding phase of the LKH algorithm. Using this analytical model, we can select the level k at which the keytree is split into S1 and S2, and the duplication factors for the sets S1 and S3, such that the bandwidth consumed at the key server is minimized.

2.3 Batched Key Retransmission

The idea of batched key retransmission is important for reducing the bandwidth consumption during the retransmission phases of our protocol. In a conventional receiver-initiated reliable multicast protocol, a sender retransmits a packet to the group if it receives a NACK from any receiver for the initial transmission. In our example, if the sender received NACKs after having sent the packets P1, P2, and P3 in the first round of the protocol, it would multicast the packets again in the second round. Our protocol avoids the overhead of multicasting all the packets again by using batched key retransmission.

BKR is based on the observation that each packet contains several keys, most of which are not needed by a specific receiver. On receiving a NACK from a receiver, it is not necessary to retransmit the corresponding *packet* to the receiver; instead, we only need to re-send the *keys* that that particular receiver needs. Further, in order to minimize the bandwidth used, the keys needed by all the receivers who have missed a packet in the first round can be packed together and multicast to them. Note that there will typically be some overlap between the sets of keys needed by different receivers; thus processing the NACKs sent by the receivers as a batch is more efficient than processing them one by one. Finally, we note that the WKA algorithm is also applied to the retransmission packets. However, proactive replication is typically not necessary at this stage since the number of members who are waiting for keys at this stage is usually a small fraction of the group size.

Continuing our example from Section 2.2.2, we expect a single NACK for P0 if it is replicated three times in the first round, and we expect around 6 NACKs collectively for packets P1, P2, and P3. We observe that the member who responded with a NACK for P0 is only interested in two encrypted keys (corresponding to K_0 and K_1 of the key tree). Secondly, the six members who responded with NACKs for packets P1, P2, and P3, each need at most three keys (one each from levels 2, 3, and 4 of the key tree). Thus, in the worst case, the total number of encrypted keys needed by the seven members who responded with NACKs in the first round is $6 \times 3 + 2 = 20$. These 20 keys can be packed into a single packet and multicast to the group in the second round. By using batched key retransmission, we are able to reduce the number of encrypted keys that need to be retransmitted by more than a factor of 4 (from 92 to 20) In this manner, batched key retransmission can be very effective in reducing the bandwidth requirements of the reliable key delivery protocol.

2.4 Implementation Issues

Figure 2 summarizes the basic protocols used by the key server and group members in our approach. Additional details of our protocol including the format of the various packets used are specified in Appendix C.

As discussed in Section 2.2.2, our protocol uses an analytical model for selecting the parameters that minimize the expected bandwidth used during a rekey operation. The input parameters for this analytical model (see Appendix B) include an estimate of the network packet loss rate for the members of the group. Clearly, obtaining an estimate of the packet loss rate for each member of the group poses a scalability problem for the key server, and any estimate of the network packet loss rate will probably not be very accurate given the dynamic and fluctuating nature of network traffic conditions. We propose a simple heuristic technique, whereby each member can independently estimate its rekey packet loss rate by keeping track of how many rekey packets it received in a multicast round out of the total number of rekey packets multicast by the key server. Each member can periodically report its network packet loss rate to the key server, piggybacking this information on a NACK. The key server can use this feedback to estimate the network loss rate and divide the receivers into high loss and low loss categories. The analytical model described in Appendix B could be then be used for obtaining a reasonable initial set of parameters for the WKA algorithm. These proactive replication factors can then be dynamically adjusted in each round based on the number of NACKs received using additive-increase-multiplicative-decrease or stochastic control with hysteresis.

Protocol for key server:

1. Divide keys into sets S1, S2, and S3. Assign weights to each set as discussed in Section 2.2.2.
2. Pack keys in sets S1,S2, and S3 into packets using the key assignment algorithm discussed in Section 2.2.1
3. Multicast packets using weights assigned in step 1 to decide proactive replication factor. Replicated packets from S1 are interleaved with packets from S2 and S3 in this step.
4. Repeat until all members have received their packets
   (a) Collect NACKs from receivers
   (b) Generate retransmission packets using BKR (Section 2.3)
   (c) Multicast packets

Protocol for a user:

Repeat until all desired packets are received:

1. Scan packets received from key server.
2. If the desired packets from the upper part (set S1) and lower part (sets S2 or S3) of the keytree are not received then send a NACK to the server indicating which sets of keys (upper or lower) is needed

Figure 2: The basic protocols used by the key server and group members in the WKA-BKR approach.

3. PERFORMANCE EVALUATION

In this section, we use a detailed simulation to compare the performance of the WKA-BKR key delivery protocol to two other approaches.

3.1 Key Distribution Protocols

The three key distribution protocols evaluated are:

Multi-send Approach This is our baseline protocol. Under this protocol, key update packets are multicast to the group in several rounds until all receivers have obtained their keys. In the first round of the multicast, each key update packet is replicated a fixed number of times to increase the probability of its delivery to each member. In subsequent rounds, corresponding to retransmissions of packets in response to NACKs accumulated in the previous round, replication is not used. We note that the use of multi-send protocols for reliable rekey transport has been discussed in the IETF Multicast Security forum [16] .

Proactive FEC-based Approach This approach was proposed by Yang et al [37, 38] for reliable key delivery. The protocol evaluated utilizes a user-oriented key assignment (UKA) method to form rekeying packets. We use FEC(ρ), where ρ is the proactivity factor, to denote this protocol. Appendix D contains a brief description of this protocol.

WKA-BKR We use WKA(k, r, m) to denote our key delivery protocol. Here k denotes the level at which the logical keytree is divided into S1 and S2, and r denotes the level of replication for the packets containing the keys in S1. Finally, m denotes the number of levels immediately below s from which keys in S2 are duplicated in S3. In other words, keys in S2 that are at levels $k+1, \ldots, k+m$ are also included in the set S3. In all our simulations, keys in S2 and S3 are transmitted once. We also consider an adaptive version of our protocol (referred to as WKA(adaptive)) in which the parameters k, r, and m for a rekey operation are selected using the procedure described in Section 2.2.2.

3.2 Metrics

The following metrics are used for comparing the performance of the protocols:

Key Server Communication Bandwidth This includes the bandwidth for key transmission and NACKs for each round of the multicast. Note that communication bandwidth is considered the bottleneck resource for the scalable group rekeying [37, 25].

Number of key delivery rounds The number of rounds taken to deliver the updated keys to the all the receivers reflects the latency of group rekeying.

Key Server Computational Costs For the WKA-BKR protocol, we report the cost of running the optimization procedure discussed in Section 2.2.2 and Appendix B.

We note that another metric of interest is the receiver communication bandwidth, i.e., the bandwidth consumed at a receiver for all the rekeying packets. In most scenarios, this is proportional to the key server communication bandwidth. Hence, we do not show any results for this metric in this section.

3.3 Simulation Model

The network topology we employ in our simulations is similar to that used in other performance studies [23, 37] where the server is connected to a backbone through a source link and all receivers are connected to the backbone through independent receiver links. Measurement studies [36, 11] have shown that packet losses occur mainly on source and receiver links; hence we assume that the backbone is loss-free.

We evaluate the performance of the protocols for several network packet loss scenarios:

- Heterogeneous independent loss on receiver links. Here we assume a fraction of the receivers ($0\% \leq \alpha \leq 100\%$) will experience high loss (p_h), while the rest of the receivers will experience low packet loss (p_l). Internet measurement studies [11] have shown that heterogeneous loss scenarios are more realistic than homogeneous scenarios where all receivers experience the same level of packet loss.

- Shared source link loss. This scenario models packet losses close to the source in the multicast delivery tree resulting in correlated losses at multiple receivers.

- Temporally correlated packet loss. This scenario models bursty link losses. We model bursty losses in our simulations using a discrete Markov chain $\{X_n\}$ with a countable state space I and stationary transitions P, where $\{X_n\} \in I = \{0, 1\}$ and $P = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix} = \begin{bmatrix} u_0 & 1-u_0 \\ u_1 & 1-u_1 \end{bmatrix}$. A packet is lost if the link is in state 0 and forwarded if the link is in state 1. Given a link loss rate (p) and the average burst length (b), we can compute the state transition probabilities u_0 and u_1 for the Markov chain $\{X_n\}$. We note that previous studies [18] have used the same approach for modeling temporally correlated packet losses.

We examine the performance of key distribution protocols for batched group rekeying operations for different group sizes (N) and number of leaves (L). For a fixed group size, the number of keys that are updated on a rekey operation depends upon the number of joins (J) and leaves (L) that are being processed (in addition to the location of the departing and joining members in the keytree). For a given number of membership changes (C), where $C = J + L$, the number of keys that will need to be updated is largest if $C = L$, i.e., $J = 0$. Hence, in our simulations, for simplicity, we assume that the number of joins being processed (J) is 0. By varying the number of leaves (L) that are being processed, we can model a wide range of group dynamics. Each simulation run consists of several rounds of group rekey events in which the departing members are uniformly distributed among the leaf nodes of the keytree. Based on the results reported in [34], we used a logical keytree with degree 4 in our experiments.

In our simulations, all FEC packets contain 40 keys and have a fixed payload size of 1171 bytes (we assume TripleDES is used for encryption with key sizes of 24 bytes). Unless otherwise stated, we use a FEC block size of 10 for the proactive FEC-based key delivery protocol based on the results reported in [38]. WKA packets can contain up to 40 keys; however, some packets may contain fewer keys. When computing bandwidth overhead, we also include the size of the network headers. Unless otherwise stated, the default parameter settings for our simulations are as follows: Group size (N) = 65,536, Number of member leaves (L) = 256, percentage of receivers experiencing high loss (α) = 20%, high receiver link packet loss rate (p_h) = 0.2, low receiver link packet loss rate (p_l) = 0.02, shared source link loss rate (p_s) = 0.01, average packet loss burst length (b) = 2.

Our simulation programs were written using the CSIM [21] simulation library. We use the method of independent replications for our simulations and all our results have 95% confidence intervals that are within 1% of the reported value.

3.4 Results

3.4.1 Heterogeneous Loss Scenario

Comparison with Multi-send Protocol: We first examine the performance of our baseline protocol, i.e., the multi-send protocol in which all key update packets are replicated a fixed number of times. The goal of this comparison is to quantify the performance benefits of the other protocols, FEC and WKA, and to make a case for using these protocols in preference to the simpler multi-send protocol.

In Figure 3, we examine the key server bandwidth as a function of α, the fraction of receivers experiencing a high level of packet loss. In Figure 3, the plots labeled Multi(x) correspond to the multi-send protocol, where the number x in parentheses is the degree of replication of the key update packets. Thus, Multi(1) is really a policy in which the key update packets are not replicated while being multicast to the group. Note that the total bandwidth used at the key server includes the bandwidth used in the transmission and retransmission phases of a protocol as well as the bandwidth consumed by NACKs.

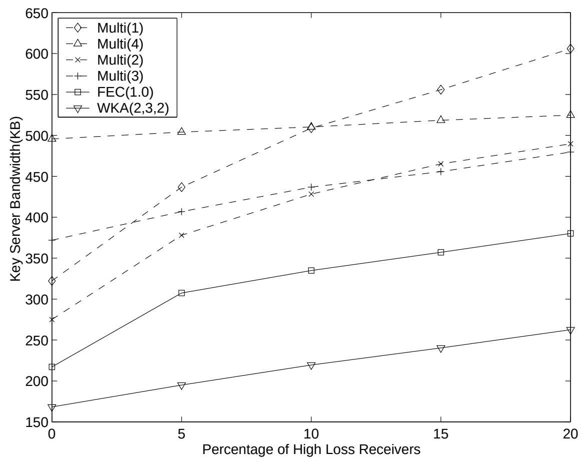


Figure 3: Key server bandwidth as a function of α for a heterogeneous loss scenario for the Multi-send, FEC(1.0), and WKA(2,3,2) protocols. Here $N = 65536$ and $L = 256$.

We can make several observations from Figure 3. First, we observe that Multi(2) outperforms Multi(1). This illustrates the importance of proactive replication of packets in the first round of a key distribution protocol. Second, as the percentage of high loss receivers increases beyond 10% Multi(3) becomes competitive and slightly outperforms Multi(2). Third, and more importantly, we observe that WKA and FEC both significantly outperform the Multi-send protocol, and the performance benefits of these approaches increase as the fraction of high loss receivers increases. In Figure 3, the bandwidth reductions achieved by WKA(2,3,2) over the best performing multi-send policy range from 39-45%. We conclude that the superior performance of WKA and FEC justifies their use in scalable group key management systems despite their higher implementation complexity.

Key Server Bandwidth: We now examine the relative performance of WKA and proactive FEC-based protocols in more detail. To evaluate the tradeoff between the costs and benefits of using proactive redundancy, we consider two parameter settings for WKA-BKR, one with a high proactivity factor, WKA(3,4,2) and one with a low proactivity factor, WKA(2,3,2). Similarly, we consider three parameter settings for FEC: FEC(1.0), FEC(1.2), and FEC(1.6). We also consider an adaptive version of WKA-BKR in which the proactivity factors are determined dynamically for each rekey event.

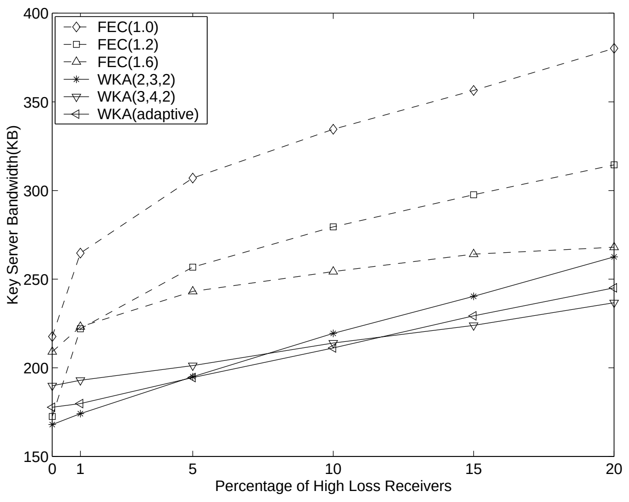


Figure 4: Key server bandwidth as a function of α for a heterogeneous loss scenario for the FEC and WKA-BKR protocols. Here $N = 65,536$ and $L = 256$.

In Figure 4, we plot the key server bandwidth as a function of α. We can make five observations. First, all other schemes outperform FEC(1.0), illustrating the importance using proactive redundancy for key distribution. Second, the protocols with a higher redundancy have the best performance for higher α, while the reverse is true for low α. For example, WKA(3,4,2) outperform WKA(2,3,2) for $\alpha > 10\%$, and FEC(1.6) outperforms FEC(1.2) for $\alpha > 1\%$. Third, the bandwidth used by WKA(adaptive) closely tracks the best WKA policy for a given α illustrating its adaptive nature. Fourth, the WKA protocols generally consume lower key server bandwidth than the FEC-based protocols. In Figure 4, for $\alpha \geq 1\%$, the bandwidth reductions achieved by WKA over FEC range from 15-22%. Fifth, we observe that the FEC-based protocols are much more sensitive to heterogeneity in the receiver loss rate than WKA-BKR. When $\alpha = 0$, FEC(1.2) and WKA(2,3,2) use approximately the same key server bandwidth. However, if the fraction of high loss receivers increases to 1%, the key server bandwidth increases by 50 KB (30%) for FEC(1.2), whereas it only increases by 6 KB (3.5%) for WKA(2,3,2).

In Table 1, we report the bandwidth used at the key server for transmitting packets in the first and subsequent rounds, as well as the total number of NACKs received by the key server. To better understand the impact of α on the relative performance of FEC and WKA, we consider the components of the key server bandwidth for $\alpha = 0$ and $\alpha = 1\%$. From this table we can make several observations

- The policies with lower proactivity factors use a smaller amount of bandwidth in the first round but have higher re-transmission costs. Further, the lower the proactive redundancy in the first round, the more the bandwidth consumed for receiving NACKs.
- The number of NACKs received by the key server is much higher for the WKA protocol than it is for proactive FEC. However, the bandwidth consumed in the retransmission phase is lower for WKA than it is for FEC. This result illustrates the benefits of Batched Key Retransmission.
- Increasing α from 0% to 1% has a greater impact on the bandwidth consumed in the retransmission phase for FEC-based protocols than it does for WKA. For example, for FEC(1.2) increasing α from 0 to 1% results in the bandwidth for retransmissions increasing from 14 KB to 65 KB, whereas for WKA(2,3,2) it increases from 30 KB to 36 KB. The explanation for this behavior is that in the FEC-based protocol for key delivery (see Appendix D), the number of parity packets transmitted by the key server at the end of each round is based on the *maximum* number of parity packets required by a receiver. Thus, a small number of high loss receivers can have a big impact on the performance of the system. In the case of WKA-BKR, however, the key server uses the batched key retransmission to retransmit the keys needed by the receivers. Most of the keys that need to be retransmitted are from the set S2 (corresponding to the lower levels of the tree), which is transmitted without duplication in the first round. It can take several rounds for all the high-loss receivers to obtain all their keys; however, the impact of a few high-loss receivers on the overall bandwidth is not as dramatic as it is for FEC-based protocols.

Overall, Figure 4 shows that WKA-BKR outperforms FEC-based protocols from the viewpoint of keyserver bandwidth, which is typically the bottleneck resource for group rekeying [37, 25].

Rekey Latency:
Next, we examine the latency of key delivery for the various protocols. Figure 5 shows the fraction of group members who have not yet received all their keys at the beginning of a certain round. In the case of all the protocols, close to 99% of the members receive their keys in the first round itself (Note that the Y-axis is in log scale). We observe that the more proactive a protocol, the fewer rounds required before all members have

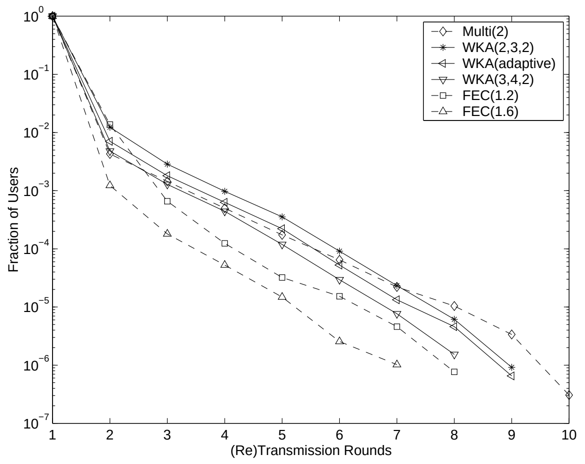


Figure 5: The fraction of members who have not yet received their keys at the beginning of a round. Here $N = 65536$, $L = 256$, and α = 20%.

received their keys. From this figure one can conclude that at the end of two or three rounds of multicast, it would be a good strategy to switch to unicast delivery for the remaining members.

3.4.2 Correlated Loss Scenarios

We now discuss the impact of shared source link loss and temporally correlated loss on the performance of FEC and WKA.

Impact of Shared Source Link Loss:
Packet losses on the shared source link will lead to correlated packet losses at the receivers. To investigate the impact of shared source link loss, we varied the source link loss rate in our simulations and examined its impact on the key server bandwidth for WKA and FEC.

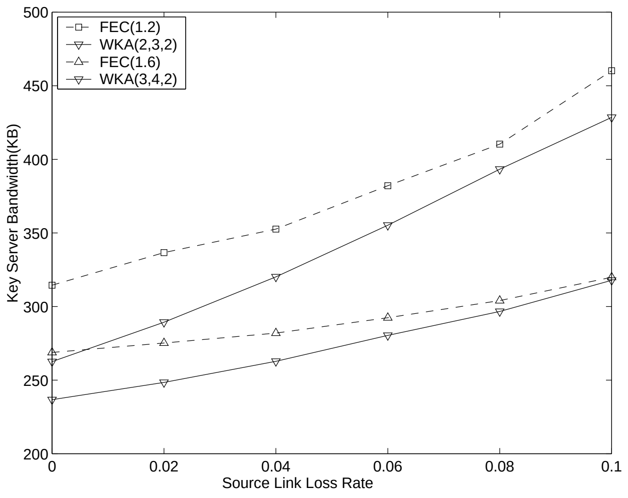


Figure 6: Key server bandwidth as a function of shared source link loss rate. Here $N = 65536$ and $L = 256$ and $\alpha = 20\%$.

	WKA(2,3,2)	WKA(3,4,2)	WKA(adaptive)	FEC(1.0)	FEC(1.2)	FEC(1.6)
Key transmission bandwidth (1st round)	138.5	180.7	165.2	130.7	156.8	209.1
Bandwidth in subsequent rounds ($\alpha = 0\%$)	29.8	9.7	12.5	80.0	14.2	0
Bandwidth in subsequent rounds (α =1%)	35.7	12.2	14.6	139	65.3	13.7
Number of Nacks ($\alpha = 0\%$)	312.5	91	98.8	1279.9	23.9	0
Number of Nacks (α =1%)	381.3	119.6	156.5	1393.5	114.1	10.1

Table 1: The components of the key server bandwidth for various FEC-based and WKA-BKR protocols under the heterogeneous network loss scenario.

In Figure 6, we plot the key server bandwidth as a function of the shared link packet loss rate for our default scenario. We observe that losses on the source link have a greater impact on the key server bandwidth used under WKA than FEC. This result is not surprising; one of the powerful properties of FEC-based protocols is that a single parity packet can repair the loss of multiple packets in its FEC block. Thus if a packet is lost on the source link, any receiver who needed that packet can recover it as long as it has received a sufficient number of packets in the same FEC block. In the case of WKA, packets containing keys in set S2 are not replicated; thus the loss of these packets on the source link impacts multiple receivers leading to a larger number of NACKs and higher retransmission bandwidth. One way to reduce the impact of shared source link loss on WKA would be use smaller packets for keys in the set S2, thus reducing the number of keys contained in a packet and hence, the number of receivers affected by a packet loss. We note, however, that in Figure 6, WKA(3,4,2) outperforms FEC(1.6) because the retransmission traffic for $\alpha = 20\%$ is dominated by packet losses on the receiver links. In scenarios where the retransmission traffic starts being dominated by shared losses on the source link, FEC-based protocols will outperform WKA-BKR.

Impact of Temporally Correlated Packet Loss:
We next consider the impact of temporally correlated (i.e., bursty) packet losses on the receiver links. Note that bursty losses are considered to be the Achilles' Heel of protocols that utilize replication or redundancy for reliability [25]. To increase the probability that a packet will be received despite correlated losses, a protocol that employs redundancy can interleave the duplicated packets with other packets. In the case of WKA-BKR, packets corresponding to the keys in the set S1 are interleaved with packets containing keys in S2 and S3. Similarly, a FEC-based key distribution protocol can interleave packets from different FEC blocks at the time of transmission.

To examine the effect of packet interleaving, we implemented two versions of WKA-BKR and FEC in our simulations – one with packet interleaving and one without interleaving. Figure 7(a) shows the key server bandwidth for the two versions of WKA and FEC. We observe that while interleaving packets is beneficial for both protocols, the reduction in key server bandwidth is much larger for FEC. In FEC-based protocols, each packet is part of an FEC block; thus a burst of packet losses within a FEC block can result in a large number of NACKs. In the case of WKA-BKR, however, only packets corresponding to keys in S1 are replicated. Packets in S2 and S3 are not replicated, and so bursty losses of these packets are no different from uncorrelated losses.

The main conclusion we can draw from Figure 7 is that interleaving packets (in a protocol-specific fashion) can be beneficial in reducing the key server bandwidth for both FEC and WKA, but the impact is more significant for FEC. We next examine the impact of increasing the average burst length (b) for correlated packet losses on FEC and WKA. Figure 7(b) plots the key server bandwidth for these protocols as a function of the average (packet loss) burst length. We observe that increasing the burst length results in a significant increase in the bandwidth for FEC(1.2) but for WKA-BKR the impact is negligible.

We note that results discussed above have been obtained for a scenario in which the number of key update packets generated in a rekey operation is substantial (more than 100). For rekey events in which the number of packets generated is smaller, the effect of correlated losses is likely to be larger.

3.4.3 *Impact of Changing the Group Size and Number of Leaves*

To study the impact of changing the group size, we considered the performance of the protocols in our default scenario for a fixed number of member leaves ($L = 256$) and different group sizes (N = 1K, 4K, 16K, 64K). We observe that the key server bandwidth for both policies increases with the group size. There are two reasons for this. First, the height of the keytree increases with group size. For the same L, the total number of keys that need to be transmitted will increase with the height of the keytree. Second, with a larger receiver population, the number of keys that will need to be retransmitted because of packet losses and the number of NACKs are also larger, leading to a higher communication cost.

Figure 8(a) shows that when the group size is small (e.g. $N =$ 1K), WKA-BKR consumes much lower key server bandwidth than FEC. In FEC-based policies, if the number of packets generated for rekeying is not an integral multiple of the FEC block size, some additional padding packets have to be generated in the last FEC block. In the case of small groups, the number of packets containing keys is small and the padding overhead can be quite substantial. One way to reduce the padding overhead is to reduce the FEC block size. However, as shown in [38], if the FEC block size is reduced, the effectiveness of FEC in recovering from packet loss decreases, and the overall bandwidth used may actually increase.

In Figure 8(b), we show the impact of changing the number of leaves (L) on the key server bandwidth. In these experiments, we kept the group size fixed at $N = 64$K and used our default packet loss scenario. The trends observed here are similar to those observed when we change the group size. Finally, from both Figures 8 (a) and (b), we can observe

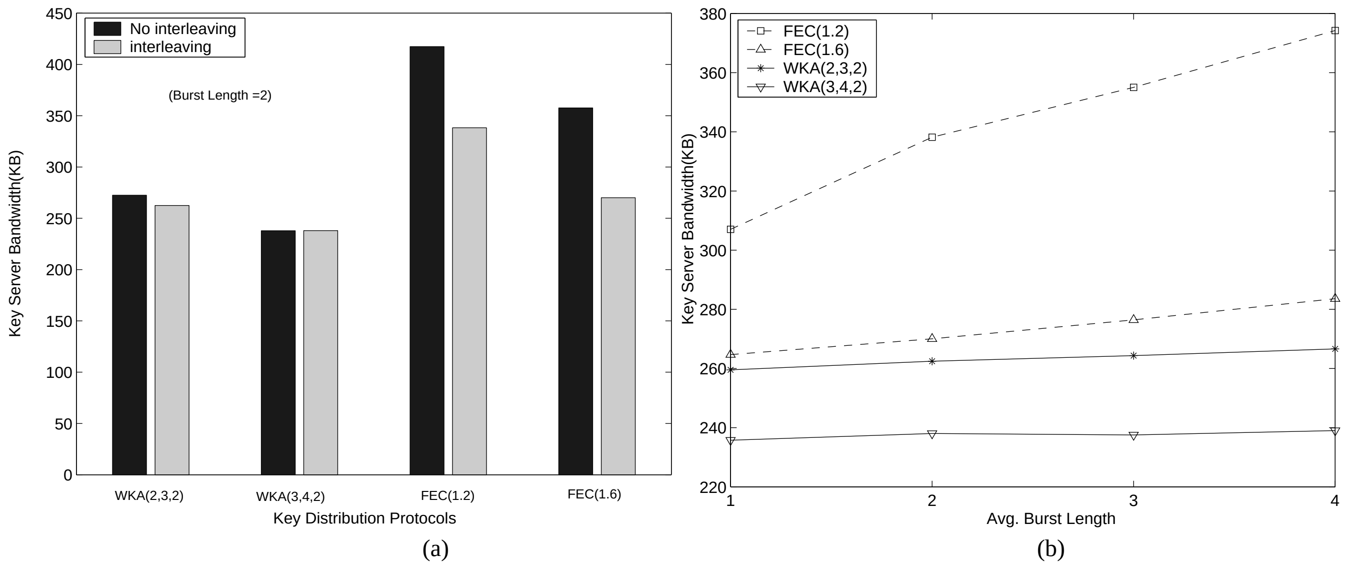


Figure 7: Impact of (a) packet interleaving and (b) average packet loss burst length on the key server bandwidth for FEC and WKA-BKR in a network scenario with bursty packet losses. Here $N = 65536$, $L = 256$, and α = 20%.

that WKA(adaptive) is effective in adapting to changes in the group size and number of leaves.

Overall, Figure 8 (a) and (b) show that WKA-BKR outperforms FEC-based protocols over a wide range of group sizes and membership dynamics, and the relative performance advantages of WKA-BKR over FEC are larger for smaller and less dynamic groups.

3.4.4 Key Server Computational Costs

Another metric that merits consideration while comparing different key delivery protocols is the computational overhead at the key server. Unlike FEC-based key delivery protocols, WKA-BKR does not have any costs for generating the parity packets for the rekey payload. However, WKA-BKR incurs the overhead of executing the cost-benefit analysis procedure described in Section 2.2.2 and Appendix B. We measured the overhead of executing this procedure for group size $N = 64$K and $L = 256$ on a SUN Ultra 10 to be less than 1 millisecond on average. This is not surprising given the small parameter space that needs to be explored in the analysis. We conclude that WKA-BKR has low computational overhead.

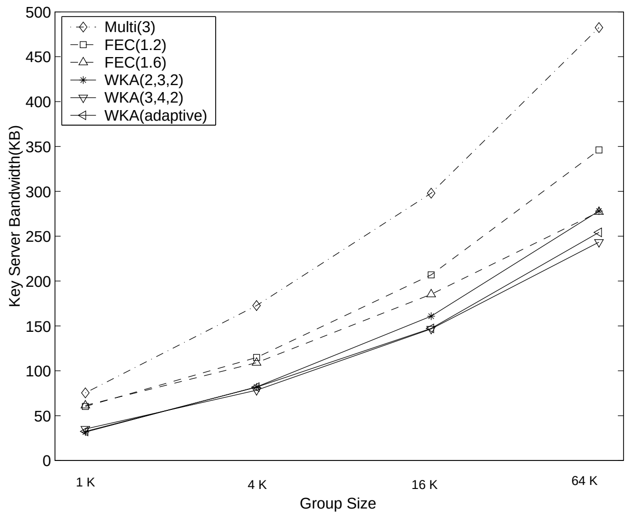
4. RELATED WORK

There has been a great deal of work on secure group communications over the last few years on topics ranging from algorithms for group key establishment to authentication for large groups [8, 2, 10, 14, 29, 28, 24]. The use of logical key trees for scalable group rekeying was independently proposed by Wallner et al [33] and Wong et al [34]. Mittra [22] proposed Iolus, a distributed framework for addressing the same problem. Periodic (batched) group rekeying was proposed and evaluated in [6, 27, 19, 37]. There have been several other works on group re-keying [4, 15, 1, 12, 7, 25, 3] that have focussed on scalable algorithms and approaches that reduce the number of encryptions at the key server and/or the number of keys that need to be securely transmitted to the group during a rekey. In contrast, the focus of this paper has been on the reliable key delivery problem. As discussed in the introduction to the paper, we believe that this an important problem in its own right that needs to be addressed for scalable group re-keying.

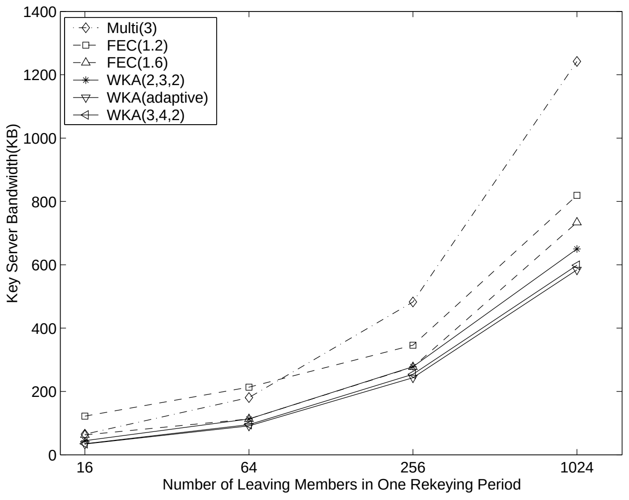
In this paper, we have discussed the WKA-BKR approach for reliable distribution of keys in the context of the LKH approach. Other approaches for scalable rekeying such as one-way function trees [1] and ELK [25] also involve the use of a hierarchical key tree in which keys at higher levels of the tree are needed by more members than keys at lower levels. As such, we believe that the basic ideas behind our approach are also applicable for group key management systems based on one-way function trees.

Reliable Group Key Transport: Other than the work of Yang et al [37, 38] there have been very few studies that have examined the performance of reliable key delivery protocols for scalable multicast group rekeying. An interesting approach that has been proposed by some studies [17, 31] is to send the key updates in the same stream as data packets. The main advantage of this approach is that key updates are synchronized with the encrypted data, and a separate protocol is not needed for reliable key delivery. However, this approach is only feasible for single-source applications in which the data source and the key server are colocated.

The ELK protocol [25] uses a similar idea to increase the reliability of key delivery. In ELK, group members receive key updates via messages multicast by the key server. In addition, “hints” that enable group members to recover lost key updates are embedded in data packets. Since these hints correspond to “maximum impact keys”, i.e., keys at the higher levels of the logical key hierarchy, a large fraction of the receivers can recover from lost key updates using this mechanism. We note, however, that the hint mechanism is not applicable in a multi-sender environment, where the key server is separate from the data sources. Second, in order to use ELK’s hint technique, it



(a)



(b)

Figure 8: Impact of changing (a) the group size, and (b) the number of leaves on the key server bandwidth.

is also necessary to use ELK's key construction technique. In other words, ELK's key construction technique (and hence its security) and hint mechanism are tightly coupled. In contrast, WKA-BKR is a reliable key distribution protcol that does not impose any restrictions on the key encoding algorithm or the size of the keys used by a key management system. Indeed, WKA-BKR can be adopted for increasing the reliability of the key update messages in ELK.

5. CONCLUSIONS

In this paper, we have presented WKA-BKR, a new scalable and reliable key delivery protocol for group key management schemes based on the use of logical key hierarchies. We have evaluated the performance of our protocol using extensive simulations. The main conclusions of our performance study are:

- For most network loss scenarios, WKA-BKR reduces the bandwidth required at the key manager in comparison to previously proposed protocols. The bandwidth reductions achieved are significant ranging from approximately 10-20% improvement over FEC-based protocols and 40-45% over simpler replication based protocols. FEC-based protocols perform better than WKA-BKR for scenarios with high levels of packet loss on the shared source link.

- WKA-BKR outperforms the other key distribution protocols over a wide range of group sizes and membership dynamics.

- WKA-BKR is less sensitive to changes in network loss conditions and to temporally correlated losses than proactive FEC-based protocols.

- WKA-BKR has low computational overhead.

We believe that the basic ideas behind WKA-BKR are also applicable for reliable key delivery in group key management systems that use other approaches based on logical key trees, e.g., OFT[1] and ELK [25]. We are currently exploring this issue in our research. We are also exploring the design of a more general version of the WKA algorithm in which the number of sets into which the updated keys are partitioned and the proactive replication factors for each set are a function of the height of the keytree.

6. REFERENCES

[1] D. Balenson, D. McGrew, and A. Sherman. Key Management for Large Dynamic Groups: One-way Function Trees and Amortized Initialization. Internet Draft, IETF, February 1999.

[2] A. Ballardie and J. Crowcroft. Multicast-specific threads and counter-measures. *Proc. of NDSS '95*, 1995.

[3] S. Banerjee and B. Bhattacharjee. Scalable Secure Group Communication over IP Multicast. *Proc. of ICNP 2001*, November 2001.

[4] R. Briscoe. MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences. Proc. of First Intl. Wkshp. on Networked Group Comm., Nov. 1999.

[5] R. Canetti, J. Garay, G. Itkis, D. Miccianncio, M. Noar, B. Pinkas. Multicast security: A taxonomy and some efficient constructions. *Proc. of IEEE Infocom 1999*, 1999.

[6] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha. Key Management for secure Internet multicast using boolean function minimization techniques. *Proc. of IEEE Infocom 1999*, 1999.

[7] L. Dondeti, S. Mukherjee, A. Samal. A Dual Encryption Protocol for Scalable Secure Multicasting. Proc. of IEEE Symp. on Computers and Communication, July 1999.

[8] A. Fiat and M. Naor. Broadcast Encryption. *Proc. of Crypto 93*, 1993.

[9] S. Floyd, V. Jacobson, C. Liu, S. McCanne and L. Zhang. A Reliable Multicast Framework for Lightweight Session

and Application Layer Framing. *IEEE/ACM Trans. on Networking*, December 1997.

[10] L. Gong and N. Shacham. Elements of trusted multicasting. *Proc. of IEEE ICNP '94*, 1994.

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

[12] T. Hardjono, B. Cain, and N. Doraswamy. A Framework for Group Key Management for Multicast Security. Internet Draft, IETF, July 1998.

[13] T. Hardjono and G. Tsudik. IP Multicast Security: Issues and Directions. Technical Report, USC, 1999.

[14] H. Harney, and C. Muckenhirn. Group Key Management Protocol (GKMP) Architecture. RFC 2094, July 1997.

[15] H. Harney and E. Harder. Logical Key Hierarchy Protocol Internet Draft, IETF, March 1999.

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

[17] M. Kandansky, D. Chiu, J. Wesley, J. Provino. Tree-based Reliable Multicast (TRAM) Internet Draft, IETF, 2000.

[18] D. Li and D. Cheriton. Evaluating the Utility of FEC with Reliable Multicast. *Proc. of ICNP'99*, October 1999.

[19] X. S. Li, Y. R. Yang,M. G. Gouda, and S. S. Lam. Batch re-keying for secure group communications. *Proc. of WWW10*, May 2001.

[20] J.C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. *Proc. of IEEE INFOCOM '96*, March 1996.

[21] Mesquite Software, Inc. CSIM18 User's Guide. 1996.

[22] S. Mittra. Iolus: A framework for Scalable Secure Multicast. *Proceedings of ACM SIGCOMM'97*, Cannes, France, September 1997.

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

[24] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and Secure Source Authentication for Multicast. Proc. of NDSS 2001, Feb. 2001.

[25] A. Perrig, D. Song, and D. Tygar. ELK, a New Protocol for Efficient Large-Group Key Distribution. *Proc. of IEEE Security and Privacy Symposium 2001*, May 2001.

[26] D. Rubenstein, J. Kurose, and D. Towsley Real-Time Reliable Multicast Using Proactive Forward Error Correction. *Proceedings of NOSSDAV '98*, 1998.

[27] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos: A Scalable Group Re-Keying Approach for Secure Multicast. *IEEE Symposium on Security and Privacy* , Berkeley,CA, May 2000.

[28] C. Shields and J.J. Garcia-Luna-Aceves. KHIP - A Scalable Protocl for Secure Multicast Routing. Proc. of SIGCOMM '99, 1999.

[29] M. Steiner, G. Tsudik, M. Waidner. CLIQUES: A New Approach to Group Key Agreement. *Proceeding of ICDCS'98*, May 1998 Amsterdam, The Netherlands.

[30] D. Towsley, J. Kurose, S. Pingali. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE J.Select Areas Commun.*, vol.15, pp.398-406, April 1997.

[31] W. Trappe, J. Song, R. Poovendran, K. J. R. Liu. Key Distribution for secure multimedia multicast via data embedding. *IEEE ICASSP 2001.*

[32] N. Waldvogel, G. Caronni, D. Sun, N. Weiler, B. Plattner. The VersaKey Framework: Versatile Group Key Management. *IEEE J. Select Areas Comm.*, 17(9), Sept. 1999.

[33] D.M. Wallner, E.G. Harder and R.C. Agee. Key Management for Multicast: Issues and Architecture. RFC 2627, IETF, June 1999.

[34] C.K. Wong, M. Gouda, S.S. Lam. Secure Group Communication Using Key Graphs. *Proc. of SIGCOMM '98*, 1998.

[35] C.K. Wong, S.S. Lam. Keystone:a group key managerment system. *ICT 2000*, Acapulco, Mexico, May 2000

[36] M. Yajnik, J. Kurose, and D. Towsley. Packet loss correlation in the MBONE multicast network. *Proceedings of IEEE Global Internet*, London, UK, November 1996.

[37] Y.R. Yang, X.S. Li, X.B. Zhang, and S.S. Lam. Reliable group rekeying: Design and Performance Analysis. *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, USA, August 2001.

[38] X.B. Zhang, S.S. Lam, D.-Y.- Lee, and Y.R. Yang. Protocol Design for Scalable and Reliable Group Rekeying. *Proceedings of SPIE Conference on Scalability and Traffic Control in IP Networks* Denver, CO, USA, August 2001.

Appendix A

To better illustrate our key distribution approach, we present an example in which the WKA algorithm will duplicate some keys from the set S2 in the set S3.

Consider how the WKA-BKR approach would work in the scenario shown in Figure 9. In this scenario, all the keys at levels 0, 1, 2, and 3 of the key tree need to be changed. In the first step, the key assignment algorithm divides the keys into two sets, S1, that includes all the encrypted keys at levels 0, 1, and 2, and S2, that includes the changed keys at level 3 and below in the keytree.

The next step is to determine the weights for the packets in S1 and S2. Assuming a homogeneous loss probability $p = 0.1$, the expected number of NACKs (collectively) is 1 if the packets in S1 are replicated 3 times. For the packets in S2, the keys are spread among 64 subtrees, each of size 16. If the packets in S2 are transmitted once (without replication), the expected number of NACKs is $1024 \times 0.1 \approx 102$, a relatively large number.

To reduce the number of NACKs, we can replicate the packets in S2. However, replicating all the packets is wasteful of bandwidth since many of the keys are of interest to very few members. A better strategy is to create a new set, S3, that contains the keys in S2 that are needed by a (relatively) large number of members. Clearly, keys at level 3 of the keytree such as K_{21} are needed by more members than keys at level 4 such as K_{85}. Further, from Figure 9, we observe that 768 out of the 1024 group members need just one key from S2 in addition to any keys in S1. For example, in the sub-tree rooted at K_{21}, members 4-15 are only interested in K_{21}. Thus, if

$\{K_{21}\}_{86}$, $\{K_{21}\}_{87}$, and $\{K_{21}\}_{88}$ are sent twice to the group, once in S2 and once in S3, the expected number of NACKs will be greatly reduced.

Based on this observaton, we create a new set of packets S3 that includes all the encrypted keys in S2 corresponding to keys at level 3 in the keytree encrypted using their unchanged child keys. There are 64 keys at level 3 of the key tree, each of which has 3 unchanged child keys, leading to a total of $64 \times 3 = 192$ encrypted keys that are included in the set S3.

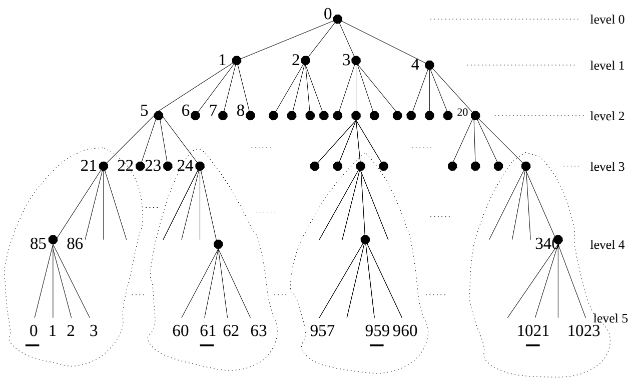


Figure 9: A scenario for the WKA-BKR approach in which it is beneficial to duplicate some keys from the set S2 in the set S3. Note that the new u-nodes are underlined and the updated k-nodes are denoted by black circles.

Appendix B

As discussed in Section 2.2.2, three decisions have to be made during the key transmission phase of our approach: (i) At what level, k, of the keytree should the keys be split into S1 and S2? (ii) What keys from S2 (if any) should be included in S3? (iii) What is the degree of replication for packets in S1 and S3?

We describe below a methodology for making these decisions based on a cost-benefit analysis. For simplicity of exposition, we first consider a homogeneous independent packet loss scenario.

Consider a group with size $N = d^h$, where d is the degree of the logical keytree. Assume that when the group is rekeyed, there are u_i keys that are changed at level $i, 0 \leq i < h$ of the keytree. Note that this information is available at the end of the key encoding phase of the LKH algorithm. Assume that the bandwidth used for transmitting a key is b, and that the keytree is divided into sets S1 and S2 at level $k < h$. Thus the number of encrypted keys in S1 is $N_{S1} = \sum_{i=0}^{k} du_i$. If these keys are replicated r times, the total bandwidth for transmitting the keys in S1 is rbN_{S1}. Note that we are ignoring any duplication overhead due to the use of User-oriented Key Assignment (UKA).

Assume that the keys in S1 needed by the N members of the group are divided into U packets using UKA. Thus, each group member needs only one of the U packets for S1. Further, assume that the number of members who only need packets from S1 (and none from S2 or S3) is N_1 (Note that this information is also available at the end of the key encoding phase). Assuming a homogeneous independent loss scenario with packet loss probability p, and , then the expected number of NACKs (collectively) from these N_1 members will be equal to $N_1 p^r$. Thus the expected bandwidth for the first round of the protocol used for the keys in set S1 is equal to

$$B_{S1} = rbN_{S1} + N_1 p^r b_n$$

where b_n is the bandwidth used by a NACK.

Moving to S2, the number of encrypted keys is is $N_{S2} = \sum_{i=k+1}^{h-1} du_i$, and the bandwidth used for sending these keys is bN_{S2}.

Consider an updated key (say K) at level $i > k$. This key will be encrypted d times. Thus, there are d encrypted keys corresponding to K in the set S2. Assume that c out of the d child keys of K are unchanged. These c encryptions may potentially be included in the set S3. Let be m be the number of levels below and including level $k + 1$ that are included in S3, and $v_i, k + 1 \leq i < k + m$ be the number of encryptions of the changed keys at level i using their unchanged child keys (for example, the c encrypted keys of K). If S3 is transmitted s times, the total bandwidth for sending the keys in S3 will be $\sum_{i=k+1}^{k+m} bsv_i$.

For each updated key at level $k + 1$ in S2, there are d^{h-k} members who need this key. Thus, the total number of group members interested in the keys in S2 is equal to $M = u_{k+1} d^{h-k}$. However, out of these M members, several members can obtain their keys from either S2 or S3. The number of members who can get their keys from a packet in either S2 or S3 is equal to $Q = \sum_{i=k+1}^{k+m} v_i d^{h-i}$. Thus a member who requires packets from S1 and either of S2 and S3 will send back a NACK if it is missing at least one of these packets. So the total expected number of NACKs is equal to $(M-Q)p_1 + Qp_2$, where $p_1 = 1-(1-p)(1-p^r), p_2 = 1-(1-p^{s+1})(1-p^r)$. Thus, the expected total bandwidth for the first round of the protocol is

$$B = B_{S1} + bN_{S2} + \sum_{i=k+1}^{k+m} bsv_i + b_n((M - Q)p_1 + Qp_2)$$

For the first retransmission round, we can compute bandwidth cost based on the expected number of NACKs in the first round. In the worst case, there will be no overlap among the keys needed by the receivers who responded with NACKs in the first round. Our analysis above gives us the expected number of NACKs for S1, S2, and S3. For each member who responds with a NACK for S1, we need to send $k + 1$ keys. For each member, responding with a NACK for S2 we need to send m keys, and for each member responding with a NACK for S3 we need to send $h - k$ keys. Thus, given the expected number of NACKs we can compute the retransmission cost in the second round.

Ignoring the NACKs for the second round and ignoring subsequent rounds, we can compute the expected total bandwidth for a particular choice of k, r, s, and m given a packet loss probability p. Since there are only a few choices of each of these variables that need to be evaluated (e.g. k= 1,2,3,4; m = 0,1,2; s = 1,2; r = 2,3,4,5), we can easily determine the choices that result in the minimal expected bandwidth.

To extend the analysis above to a heterogeneous loss scenario, we simply need to replace the expressions for the number of NACKs for S1, S2, and S3 with expressions that take into account the separate loss rates for the high loss and low loss receivers.The rest of the model is identical to that for the homogeneous loss scenario. Although our approach above makes several approximations, it gives us good results, and we have use it to guide our selection of parameters for the simulation experiments reported in Section 3.

Appendix C

In this appendix, we provide some additional details for the WKA-BKR approach.

Key Identification

We adopt the approach used by Zhang *et al* [38] in their key transport protocol for uniquely identifying each key and encryption. The nodes in the key tree are divided into two categories: user nodes and key nodes, each of which has a unique integer id. Nodes are numbered from 0 while traversing the key tree in a top-down and left-right fashion. For example, in Figure 9, the keynodes are numbered from 0 to 340 whereas the user nodes are numbered from 0 to 1023.

Given this key identification strategy, if a key node has id m, its parent's id will be $\lfloor \frac{m-1}{d} \rfloor$, where d is the degree of the tree. The same relationship also applies between user nodes and key nodes when we add a base number to the user id corresponding to the total number of key nodes. To uniquely identify an encryption $\{k'\}_k$, we assign the ID of the encrypting key k as the ID of the encryption.

During a re-key operation in the LKH approach, a user node needs to be able to identify the encrypted keys it needs, i.e. keys on the path from that node to the root. Given the key numbering scheme outlined above, a user node with ID m can uniquely identify the IDs of the encrypted keys it needs during a key update.

We refer the reader to the paper by Zhang *et al* [38] for further details on this scheme.

Packet Formats

Three types of packets are used by the key delivery protocol: WKA, KEYS, and NACK. The WKA packets are the packets transmitted by the key server in the first round of the protocol, the KEYS packets are used in subsequent rounds, whereas NACK packets are sent by a receiver to the key server, if it detects that it has not received the WKA or KEYS packets that include its keys.

The format of WKA packet is as follows:

1. Packet type: WKA (2 bits)
2. Rekey Message Id: (6 bits)
3. Total Number of Packets used for this rekey operation: (10 bits)
4. Number of keys included: 6 bits
5. MaxKID: (16 bits). Maximum Id of the current key nodes.
6. <fromId,toId>: 48 bits. Members can use this field to quickly check if the packet contains any encryptions that they need.
7. SplitLevel: (4 bits). This field specifies the level at which the keytree is split into S1 and S2/S3.
8. A list of <ID, encrypted key>: variable length

The format of a KEYS packet is as follows:

1. Packet type: KEYS (2 bits)
2. Rekey Message Id: (6 bits)
3. Number of keys included: 6 bits
4. MaxKID: (16 bits). Maximum Id of the current key nodes.
5. SplitLevel: (4 bits)
6. A list of <ID, encrypted key>: variable length

Note that a receiver will need to scan the entire packet to see if it contains any keys that it needs.

The format of a NACK is as follows:

1. Packet type: NACK (2 bits)
2. Rekey Message Id: (6 bits)
3. Member Id: (24 bits)
4. Upper/Lower Flag: (2 bit). This field is used to specify if the member is missing keys in the upper half (S1) or lower half (S2/S3) of the keytree or both halves.
5. Member Loss Rate: (7 bits)

Appendix D

In Section 3, we have compared the performance of our approach with that of the proactive FEC-based protocol proposed in [37, 38]. Figure 10 below describes the basic operation of this protocol.

Protocol for key server:

1. Divide original packets into blocks of k packets.
2. Generate $\lceil (\rho - 1)k \rceil$ parity packets for each block
3. For each block i, multicast k original packets and $\lceil (\rho - 1)k \rceil$ parity packets
4. For each round
   (a) Collect $amax[i]$ as the largest number of parity packets needed for block i
   (b) Generate $amax[i]$ new parity packets for each block
   (c) Multicast these parity packets to all users

Protocol for a user:

Repeat until success

1. Scan packets received from key server.
2. If the specific packet needed, or at least k packets in the required parity blocks are received, then success, else compute a, the number of parity packets needed for recovery. Send a to the key server using a NACK.

Figure 10: The basic protocols used by the key server and group members in the Keystone key delivery protocol.