This is an extended version of the article that will appear in a special issue of *Performance Evaluation* on the proceedings of the Performance 2002 conference held in Rome, Italy, Sept. 2002.

A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast

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 multicast 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 and the group rekey transport payload to reduce the bandwidth overhead of the reliable key delivery protocol. Using both analytic modeling and simulation, we investigate the factors that affect the bandwidth overhead of reliable key delivery protocols. We compare the performance of WKA-BKR with that of other rekey transport protocols, including a recently proposed protocol based on proactive FEC. Our results show that for most network loss scenarios, the bandwidth used by WKA-BKR is lower than that of the other 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 [21], 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.

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

In recent years, several approaches for scalable group rekeying have been proposed [13, 21, 22, 20, 5, 1, 14]. One prominent approach [22, 21] 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 [2, 16, 24]. Periodic or batched group rekeying has been shown [24] 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 reliably delivered to the members of the group in a timely fashion. We note that 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.

For scalable group rekeying, clearly both the key encoding and key distribution operations need to be scalable. While there has been a great deal of research that has focused on improving the efficiency and reducing the complexity of the key encoding operation, reliable key distribution has not received the same degree of attention. We note that that the reliable key delivery problem is particularly challenging when group rekeying is done periodically, i.e., when several membership changes are processed in a single batch. In this situation, the number of keys that need to be changed can be large enough that a large number of packets need to be multicast reliably to the whole group. Although reliable multicast transport protocols such as SRM [3] and RMTP [12] can be used for reliable delivery of keys, these protocols are complex and (in some cases) require additional support from the network infrastructure. Moreover, the reliable key delivery problem has some characteristics that can be exploited to design custom protocols that are more light-weight in nature.

In this paper, we make two contributions. First, we present WKA-BKR, a new scalable and reliable rekey transport protocol for group key management schemes that are based on logical key hierarchies. Second, we present analytical models for evaluating the bandwidth overhead of our protocol and other protocols that have been proposed for reliable group rekey transport. Using both analysis and simulation, we investigate the factors that affect the bandwidth overhead of reliable key delivery protocols and compare the performance of WKA-BKR with that of other rekey transport protocols. In particular, we compare the performance of our protocol to that of a protocol [24, 25] recently proposed by Yang *et al* that makes use of proactive FEC [15].

Our reliable key delivery protocol is based upon two ideas: *proactive redundancy* and *batched key retransmission.* Our approach uses proactive redundancy for achieving reliability and ensuring timely delivery of keys. We note that Yang *et al*'s protocol is also based on proactive redundancy. Unlike their 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, 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 setting the degree of replication of each key based upon its position in the logical key tree, i.e. more valuable keys have a larger degree of replication.

The idea of *batched key retransmission* (BKR) is also based on the special properties of the rekey transport payload. 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.

In our performance evaluation, we find that for most scenarios, WKA-BKR has lower bandwidth overhead in comparison to previously proposed protocols. The difference in bandwidth overhead is significant – up to 26% lower than FEC-based protocols and up to 60% lower than simpler replication based protocols for the default scenario considered in our performance study. 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, based on insights provided by our results, we propose modifications that can lead to improved performance for group rekey transport protocols based on proactive FEC.

The organization of the rest of the paper is as follows. In Section 2, we provide background on scalable group rekeying and discuss related work. In Section 3, we describe the WKA-BKR protocol in more detail. In Section 4, we present a bandwidth overhead analysis of three reliable key delivery protocol. Next, in Section 5, we compare the performance of the protocols using both the analysis presented in the previous section and simulation. Finally, Section 6 contains our conclusions.

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

The use of logical key trees for scalable group rekeying was independently proposed by Wallner et al [21] and Wong et al [22]. 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 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.

In this scheme, each user owns all the keys on the path from its individual leaf node to the root of the key tree. As a result, when a user leaves the group, in order to maintain forward data confidentiality, all those keys have to be changed and re-distributed.

An example key tree is shown in Figure 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 subtrees rooted at these keys. If user U4 leaves the group, the keys K_{456} and $K_{1\text{-}9}$ will need to be changed. Assume these keys are replaced with keys K'_{456} and $K'_{1\text{-}9}$ respectively. To distribute these new keys efficiently to all remaining users, the key server can encrypt $K'_{1\text{-}9}$ with K_{123}, K'_{456} and K_{789} separately, encrypt K'_{456} with K_5 and K_6 separately, and then multicast these 5 encrypted keys to the group. Each user can extract the keys it needs independently.

In this paper, we discuss 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 [14] 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.

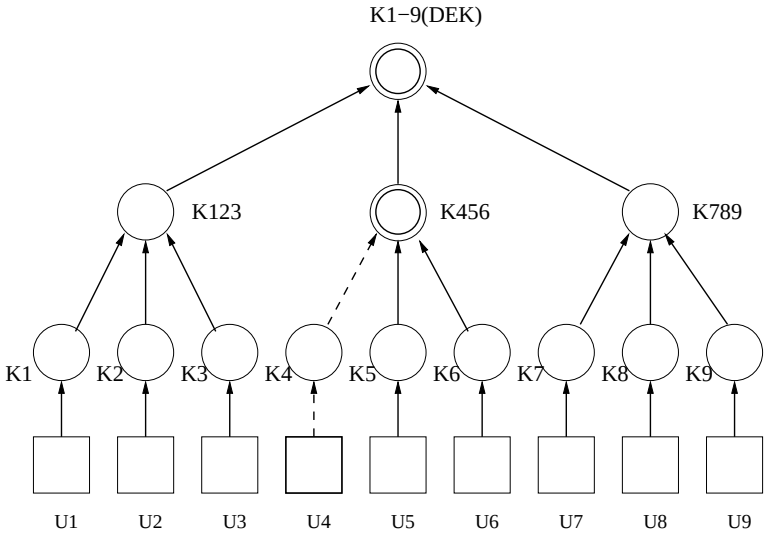


Figure 1: An example logical key tree.

2.1.1 Periodic Batch Rekeying

Periodic batched group rekeying has been shown [2, 16, 11, 24] 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. By processing several membership changes in a batch, periodic rekeying reduces the number of group rekey events in a given period of time at the expense of increasing the join and leave latency, i.e., the time that elapses before a new member joins the group or a current member leaves the group.

Further, the number of encrypted keys that need to be multicast to the group under batch rekeying can be much smaller than the number of encrypted keys that would be generated if each membership change were to be processed individually. This is because when several leaf nodes (corresponding to user keys) in a keytree are changed, there is typically some overlap in the paths from the leaf nodes to the root key. For a group rekey event, the number of encrypted keys that will need to be multicast to the group depends upon both the number of member joins and leaves that need to be processed and the location of the departed and newly joined members. If the departed members are clustered together as shown in Figure 2 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. In [24], Yang *et al* have analyzed the bandwidth requirements for batch rekeying under worst-case and average case scenarios

2.2 Rekey Transport Protocols

As discussed in the introduction, group rekeying involves two operations – key encoding and key distribution. The key encoding phase of group rekeying involves executing the batched LKH algorithm to generate a set of encrypted keys that have to be transmitted to the members of the group. The key distribution phase is concerned with packing these encrypted keys into packets and delivering the packets to the members of the group in a scalable, reliable, and timely manner.

The reliable group rekey transport problem has some characteristics that differentiate it from the con-

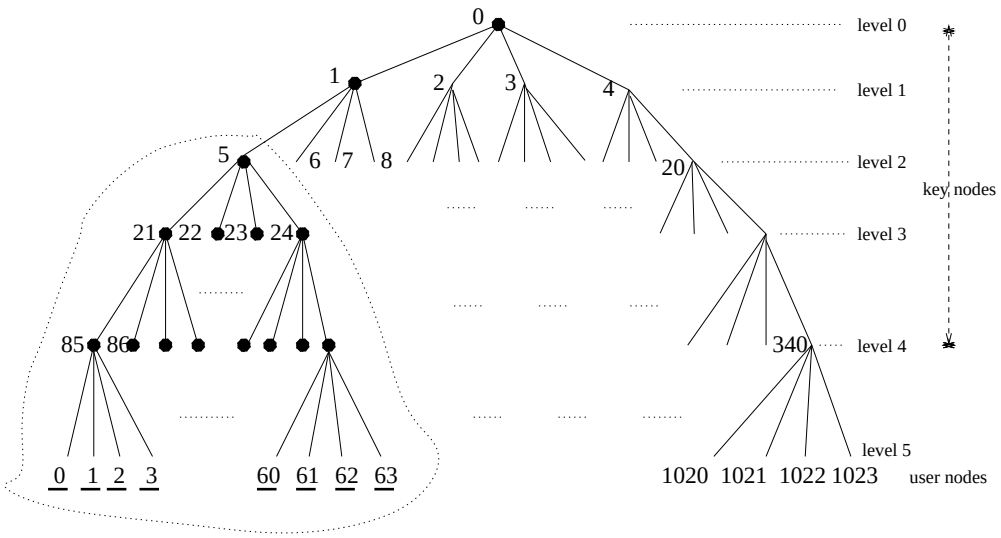


Figure 2: The Best-case Scenario for Group Rekeying for a Logical Key Tree with degree 4, Group Size = 1024, Number of Leaves = 64. Note that the new u-nodes are underlined and the updated k-nodes are denoted by black circles.

ventional reliable multicast problem. First, as pointed out in the introduction, there is a soft real-time requirement for key delivery. To address this issue, i.e. to reduce the latency of key delivery, group rekey transport protocols can make use of proactive redundancy. For example, the protocol proposed by Yang *et al* [24] 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.

We now briefly describe two protocols that have previously been proposed for reliable rekey transport.

2.2.1 Multi-Send 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 the first round. Receivers respond with NACKs only if they detect they are missing a packet containing keys of interest to them. 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 [6] .

2.2.2 Proactive FEC-based Rekey Transport

In this approach, the key transport payload is divided into several FEC blocks. A fixed number of parity packets (based on the proactivity factor ρ) are transmitted along with key update packets for each block. Figure 3 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 packets 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 3: The basic protocols used by the key server and group members in proactive FEC-based key delivery protocol.

2.2.3 Other Related Work

Reliable Group Key Transport: Other than the work of Yang et al [24, 25] 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 [7, 19] 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 [14] 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. Under ELK, 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 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, the protocols evaluated in this paper do not impose any restrictions on the key size or the key encoding algorithm.

Analysis of Reliable Multicast Protocols: Several papers [18, 9, 10] have presented analytical models for evaluating the throughput and bandwidth overhead of reliable multicast protocols. Our bandwidth analysis in Section 4 uses many of the techniques first presented in these studies.

3 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.

3.1 Protocol Overview

Our key delivery protocol is based on multicast and uses an application-level receiver-initiated, i.e., NACK-based, approach [18] 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.

Like other group rekey transport protocols, our protocol exploits the sparseness property of the rekey payload. 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. 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 re-keying policy [24, 16] 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.

3.2 Weighted Key Assignment

To understand the motivation behind weighted key assignment, we need to consider the properties of a logical key tree. 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 2, 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.

Consider the logical key tree in Figure 2. We observe that the higher a key in the key tree, the more the 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 [14] 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 keys that are needed by a large number of group members are proactively replicated in multiple packets during the key transmission phase of the protocol. The degree of replication of a key is selected based on the level at which it resides in the key tree; thus keys at higher

levels of the key tree tend to have a larger degree of replication whereas keys at the lower levels may not be replicated at all.

The key transmission phase involves three steps. In the first step, for each level in the keytree, we compute the weight that determines how often the changed keys at that level will be replicated during the multicast; the algorithm used for this is described in Section 3.2.1. In the second step, keys are assigned to packets based on the weights computed in the previous step. Thus a key with a weight w will be packed into w different packets. The algorithm used for packing a set of keys into packets can have a significant impact on the performance of the protocol [24]. 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 the receiver will need to receive all those packets in order to obtain its keys. Our key assignment algorithm is discussed in Section 3.2.2. In the third step, the packets are multicast to the group.

3.2.1 Weight Assignment

Proactively replicating a key 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 a member will receive the keys it needs in the first round of the protocol, and therefore *decreases* the latency of key delivery (iii) it *decreases* the number of NACKs received by the key server from members in the first round, and thus reduces the possibility of NACK implosion. For each key, there exists an optimal degree of replication such that using a larger degree of replication consumes greater bandwidth while having a negligible impact in terms of reduced key delivery latency or reduction in the number of NACKs.

We now sketch our approach for selecting an appropriate degree of replication for each key. Consider a receiver-initiated reliable multicast protocol based on ARQ (automatic repeat request) in which a key is multicast to a group in several rounds until all the receivers who are interested in that key have received it. Let M be the number of times a key needs to be transmitted by the key server before all the members who are interested in that key have received it.

Using proactive replication in the first round of the protocol will reduce the expected *number of multicast rounds* of transmission needed for delivering the key to all members; however, if packet losses are independent, it does not reduce the expected *number of times a key is transmitted* (including replications) before all interested receivers have received it. If a key is replicated R times in the first round of multicast, the multicast bandwidth overhead, $E[O]$, for delivering that key is given by

$$E[O] = \max(R, E[M]), \text{where } E[M] \text{ is the expected value of } M$$

and the expected number of multicast rounds $E[T]$ for delivering that key is

$$E[T] = \begin{cases} 1, & \text{if } R > E[M] \\ E[M] - R + 1, & \text{if } E[M] \geq R \end{cases}$$

From the equations above it is clear that selecting $R = \lfloor (E[M] \rfloor$ will result in a low key delivery latency while ensuring that the bandwidth overhead does not exceed $E[M]$. We use this approach in our weight assignment algorithm for each key.

In Section 4.3, we show that $E[M]$ depends upon two factors: (i) the number of members who need the key, and (ii) the packet loss properties of the network. For any given key, the number of members who need it is a function of the level at which it resides in the logical key tree. In Section 3.4, we outline a heuristic procedure that can be used for estimating the packet loss properties of the network. Thus, given a packet loss model, we can compute $E[M]$ for each level of the key tree, and use it to decide the appropriate proactive replication factor for keys at that level of the key tree.

3.2.2 Key Assignment Algorithm

The next step in our protocol is to pack the changed keys into packets that are then multicast to the group. Previously, Yang et al [24] have examined the difference between different algorithms for assigning keys to packets, e.g., breadth first assignment, depth first assignment, etc. They show that the key assignment algorithm has a significant impact on the number of different packets each receiver needs to obtain all its keys during the rekey operation.

Under our protocol, we cannot directly use breadth first assignment or depth first assignment, because keys at different levels will potentially have a different degree of replication based on our weight assignment algorithm. We investigate two different key assignment algorithms - Weighted Breadth-First Assignment (WBFA) and Weighted Depth First Assignment (WDFA) for our protocol.

Weighted BFA Under this algorithm, we divide the keys that need to be transmitted into different sets on the basis of the level at which they reside in the key tree. Next we simply pack the keys in each set into packets using breadth-first assignment. Since all the keys at the same level have the same degree of replication, each packet containing keys from a particular set is replicated the same number of times during the multicast.

Another variant of this algorithm is one in which all the keys that need to be transmitted are divided into sets based on their degree of replication, i.e. all keys that have the same degree of replication are placed in the same set. Under this algorithm, each set can contain keys residing at different levels of the keytree. However, since our weight assignment algorithm will typically assign similar weights to adjacent levels in the key tree, each set will typically contain keys at either a single level or multiple adjacent levels. Once the keys are divided into sets, the keys in each set can simply be packed into packets using DFA or BFA. We note that this algorithm is a more general version of a key assignment algorithm that we first presented in [17].

Weighted DFA Under this algorithm, a counter is associated with each changed key in the logical key tree that has to be transmitted to the group during a rekey. This counter is initialized to the replication weight associated with that key by the weight assignment algorithm. The keytree is then traversed in depth first order. If a key's counter is greater than zero, it is copied into a packet and the associated counter is decremented. This depth first traversal procedure is repeated until the counter associated with every key becomes zero. A new packet is started if there is no space remaining in the current packet, and also at the end of every depth first traversal.

The packets produced by this algorithm will satisfy the proactive replication criterion for each key. This algorithm is potentially superior to the WBFA algorithm since multiple keys needed by a member will tend to be assigned to the same packet, thus reducing the total number of packets a specific receiver will need to obtain all its keys. We examine the performance differences between WDFA and WBFA in Section 5.

3.3 Batched Key Retransmission

The idea of batched key retransmission is important for reducing the bandwidth consumption during the retransmission phases of our protocol. BKR is based on the observation that each packet contains several keys, most of which are not needed by a specific receiver. For example, using the WBFA key assignment algorithm discussed in Section 3.2.2, a packet contains several keys at the same level of the keytree. However, a specific receiver is interested in at most one of the keys in the packet since it only needs one key from each level of the key tree.

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. Under BKR, however, 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.

3.4 Implementation Issues

Figure 4 summarizes the basic protocols used by the key server and group members in our approach. For additional details of our protocol including the format of the various packets used, please see Appendix A. Note that while multicasting the packets in the first round of the protocol, replicated packets from different sets created by the key assignment algorithm (e.g. under WBFA, packets containing keys from different levels of the keytree) 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.

Protocol for key server:

1. Compute the proactive replication factor for each level of the keytree as discussed in Section 3.2.1.
2. Pack keys into packets using the key assignment algorithm discussed in Section 3.2.2
3. Multicast packets while interleaving replicated packets from different sets created by the key assignment algorithm
4. Repeat until all members have received their packets
   (a) Collect NACKs from receivers
   (b) Generate retransmission packets using BKR (Section 3.3)
   (c) Multicast packets

Protocol for a user:

Repeat until all desired packets are received:

1. Scan packets received from key server.
2. If timeout or if the desired packets containing keys of interest to the user are not received then send a NACK to the server indicating which keys are missing

Figure 4: The basic protocols used by the key server and group members in the WKA-BKR approach.

As discussed in Section 3.2.1, the input parameters for the analytical model used for determining the degree of replication for an encrypted key 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 4 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 techniques.

4 Bandwidth Analysis

In this section, we present analytical models for the bandwidth overhead of the WKA-BKR, proactive FEC-based, and multi-send key delivery protocols. In the case of proactive FEC based reliable key transport, we show that the analytical model presented by Yang *et al* in [24] does not completely take the sparseness property of the rekey transport payload into account and that their bandwidth overhead analysis gives us an upper bound on the bandwidth of their protocol. In Section 4.4, we present an extension to their analysis that gives us more accurate results.

4.1 System Model

Our model is similar to the one used in previous analyses of reliable multicast protocols [18, 9, 10] and Yang's analysis [24] of reliable rekey transport. We assume that all packet loss events for all transmissions are mutually independent. For ease of exposition, in this section, we assume a homogeneous network scenario where the packet loss probability, p is independent of receiver. However, it is straightforward to extend this analysis to a heterogeneous network scenario where a fraction α of the N receivers in the group have a high packet loss rate, p_h, whereas the remaining fraction of receivers have a low packet loss rate, p_l. All our results in Section 5 assume a heterogeneous network loss scenario, which has been shown to be more realistic [4, 23]

4.2 Metric

The goal of our analysis for each protocol is to compute the bandwidth overhead for delivering the group rekey payload to all the members of the group. This metric is defined slightly differently for the three protocols. Let the total number of packets in the rekey payload be equal to P and let the total number of encrypted keys be U (assuming no keys are replicated in the payload). Note that for a rekey event U and P will depend upon the group size N and the number of membership changes (joins and leaves) being processed. In the case of the FEC-based and multi-send protocols, let the total number of packets multicast during the rekey be Q. For multi-send, the Q packets include all the replicated packets as well as retransmitted packets, whereas for proactive FEC, the Q packets include all the parity packets transmitted proactively or in response to NACKs. The bandwidth overhead for these protocols is equal to Q/P. For WKA-BKR, let V be the total number of keys that are transmitted by the key server during the rekey event including proactively replicated keys and retransmitted keys. Then the bandwidth overhead for WKA-BKR is defined as V/U.

4.3 WKA-BKR

Our bandwidth analysis for WKA-BKR is based on the observation that the unit of replication and retransmission under this protocol is a *key* and not a *packet*. Thus, we focus on the number of times a particular key will need to be transmitted in order for it to be successfully delivered to all the receivers who are "interested" in it.

Consider a balanced logical key tree with degree d and height h. The key tree has $N = d^h$ leaf nodes corresponding to members of the group. 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 d^{h-l-1} members. Thus, from the viewpoint of reliable key delivery, an encrypted key corresponding to a key at level l of the keytree has to be delivered to $R(l) = d^{h-l-1}$ receivers. 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.

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. Since all the packet loss events for receiver r, including replicated packet and retransmissions, are mutually independent, M_r is geometrically distributed. Thus, $P[M_r \leq m] = 1 - p^m, m \geq 1$ and $E[M_r] = 1/(1-p)$. Since lost packet events at different receivers are independent, for key K, we have

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

Thus,

$$E[M(l)] = \sum_{m=1}^{\infty} P[M(l) \geq m] = \sum_{m=1}^{\infty} (1 - (1 - p^{m-1})^{R(l)}) \tag{2}$$

We can compute $M(l)$ numerically using the equation above by truncating the summation when the mth value falls below a threshold.

Under WKA-BKR, let the key K be proactively replicated W times in the first round of the transmission. Thus, the expected bandwidth used for key K is given by $\max(W, E[M(l)]$. If as discussed in Section 3.2.1, $W = \lfloor E[M(l)] \rfloor$, then the expected bandwidth used for a key at level l is equal to $E[M(l)]$.

To compute the total bandwidth used for a rekey event, we need to derive the expected number of encrypted keys that are part of the rekey payload. The number of keys in the logical key tree that are changed depends upon the number of membership changes being processed in the rekey event. Assume that the number of joins and leaves being processed is equal to J and L respectively, that $J = L$ (for simplicity), and that the L leave requests are uniformly distributed over the leaf nodes of the key tree. Previously, Yang *et al* [24] have shown that under LKH, the average number of keys at level l (denoted by $U(l)$) of the keytree that are changed when J joins and L leaves are processed as a batch and $J = L$ is equal to

$$U(l) = d^l \left(1 - \frac{\binom{N-N_0}{J}}{\binom{N}{J}} \right), \text{where } N_0 = N/d^l \tag{3}$$

Since each of these keys will be encrypted d times and the bandwidth used for each of these encrypted keys is given by $E[M(l)]$, the expected total bandwidth used in a rekey event (denoted by $E[V]$) is given by

$$E[V] = \sum_{l=0}^{h-1} d \cdot U(l) \cdot E[M(l)] \tag{4}$$

The expected number of encrypted keys that are in the rekey payload is given by

$$E[U] = \sum_{l=0}^{h-1} d \cdot U(l) \tag{5}$$

and the expected bandwidth overhead can be approximated by $E[V]/E[U]$. In Section 5, we show that our approximate analytical model gives us accurate results.

4.4 Proactive FEC based rekey transport

The main difference between Yang *et al*'s key delivery protocol and a conventional proactive FEC-based reliable multicast protocol is that a packet has to be successfully delivered only to those receivers who need one or more of the keys it contains. Consider a FEC block of size k packets, and assume that keys a receiver r needs are assigned to z_r packets in that block. Once the receiver r has received the z_r packets of interest to it, it will not need to participate in subsequent rounds of the FEC protocol.

Based on this observation, Yang *et al* [24] presented an analysis for the bandwidth overhead for proactive FEC-based rekey transport. They show that the bandwidth overhead for this protocol assuming N receivers can be derived by computing the bandwidth overhead of a conventional proactive FEC based reliable multicast for smaller number of receivers, M, where $N - M$ is the number of receivers who satisfy the condition that they have received their specific z_r packets despite having received fewer than k packets in the first round of the protocol. We refer the reader to [24] for the complete details of the analysis. We note that according to this analysis, the bandwidth overhead for FEC based rekey transport is a function of the number of receivers, i.e. the group size (N), the FEC block size (k), the proactivity factor (ρ), the maximum number of packets needed by a member from a FEC block, i.e., the maximum value of z_r, and the network packet loss rate.

While this analysis captures the impact of the sparseness property of the rekey payload *within a FEC block*, it does not take into account the impact of the sparseness property *across the different FEC blocks* in the rekey payload. As a specific example, consider the situation where the number of members in the group, N = 65536 and the number of member joins and leaves being processed in the rekey, $J = L = 512$. In this case, the expected number of encrypted keys in the rekey payload is large enough that 170 packets are needed assuming that 40 keys can fit in a single packet. Assuming a FEC block size of 10 packets, this means that there are 17 FEC blocks in the payload. Now consider the situation where a receiver r is interested in exactly one key and this key is assigned to a packet in the first FEC block. Once receiver r has received this packet, it will no longer need to participate in the FEC protocol for the remaining 16 FEC blocks of the payload. Further, based on the protocols described in [24, 25] a receiver can compute the FEC block its keys will be assigned to and need only participate in the FEC protocols for the blocks which contains its keys. Thus, the bandwidth overhead for a specific FEC block will depend upon the number of receivers who are interested in the packets in that block, which in turn depends upon the keys assigned to that block. The overall bandwidth overhead for Yang *et al*'s protocol is the average of the overheads for the different FEC blocks in the rekey payload. We note that the bandwidth analysis in [24] is appropriate when the entire payload fits in one FEC block; if there are several FEC blocks, then that analysis underestimates the impact of the sparseness property and thus overestimates the bandwidth overhead.

We now describe an approach for obtaining an estimate of the bandwidth overhead for Yang *et al*'s protocol assuming that keys are assigned to packets using BFA. To compute this overhead, we need to estimate for each FEC block the number of receivers who are interested in the keys in that block. This requires a knowledge of the keys in the block. For a particular set of encrypted keys, it is straightforward to estimate the number of interested receivers since these receivers will correspond to leaf nodes that are descendants of the keys in the logical key hierarchy. For our bandwidth analysis, we need to estimate the number of receivers interested in a specific FEC block in the average case. Assuming that Breadth First Assignment is used to assign keys to packets, Figure 5, describes the procedure we used to estimate this overhead.

Inputs: Group size (N), Number of leaves (L), Number of keys per packet , FEC block size, FEC proactivity factor, Packet loss rate

Outputs: Bandwidth overhead for each FEC block, Overall average bandwidth overhead

Procedure:

1. Use equation 3 to compute average number of changed keys at each level i, $0 \leq i < h-1$, where $N = d^h$.
2. Let X be the number of FEC blocks needed for packing the encrypted keys given the number of keys per packet and the FEC block size. Using the BFA key assignment algorithm, estimate the number of keys at level i, $0 \leq i < h-1$ in each FEC block.
3. Under the BFA algorithm, the first FEC block will contain all the encrypted keys corresponding to the root key. Hence, the number of interested receivers is equal to N. For the remaining FEC blocks, if a block only contains keys from one level l, the number of interested receivers is easy to compute. Let Y_l be the expected number of encrypted keys contained in a FEC block corresponding to key nodes at level l of the keytree. Then the number of interested receivers for that block is given by $Y_l \cdot d^{h-l-1}$. If a FEC block contains keys from multiple levels of the keytree, then the calculation is somewhat more complex. Let l and $l+1$ be the highest levels of the keys contained in the block. Let Y_l and Y_{l+1} be the number of encrypted keys in the FEC block that reside at levels l and $l+1$ respectively, and let U_l and U_{l+1} be the expected number of changed keys at level l and $l+1$ of the keytree (see equation 3). To estimate, Z_{l+1}, the number of keys at level $l+1$ that are not descendants of the Y_l keys at level l, we assume that the changed keys at level $l+1$ are uniformly distributed under those at level l. This gives us $Z_{l+1} = \min\left(Y_{l+1}, \frac{d \cdot U_l - Y_l}{U_l} \cdot U_{l+1}\right)$. Then, the number of interested receivers is given by $Y_l \cdot d^{h-l-1} + Z_{l+1} \cdot d^{h-l-2}$.
4. For each FEC block, the maximum value of z_r is equal to the number of different levels in the keytree from which keys are assigned to that block. This is because a particular receiver will need to receive at most one key from each level.
5. Given the number of receivers interested in a FEC block from step 4 and z_r from step 5, use Yang et al's model [24] to compute the bandwidth overhead for that block.
6. Compute the average overhead for the entire payload by averaging the overheads of each block.

Figure 5: The algorithm for computing the bandwidth overhead for the proactive FEC based reliable key delivery protocol assuming that BFA is used to assign keys to packets.

4.5 Multi-send Key Delivery Protocols

The multi-send key delivery protocols are similar to conventional receiver initiated reliable multicast protocols, except that in the first round of the protocol each packet is multicast multiple times. Let Multi(x) denote the protocol in which each packet is replicated x times in the first round of the protocol. Note that Multi(1) represents the conventional NACK-based reliable multicast protocol. From previous analysis [18] of bandwidth overhead of these protocols, we know that given our system model, the expected bandwidth overhead of the protocol for successfully delivering a packet to R receivers is given by

$$E[M] = \sum_{m=1}^{\infty} (1 - (1 - p^{m-1})^R) \tag{6}$$

For the Multi(x) protocol, the bandwidth overhead will be equal to

$$E[O] = \max(x, E[M]) \tag{7}$$

The complexity in the analysis of the bandwidth overhead of the Multi-send protocols arises from the sparseness property of the rekey payload. Because of this sparseness property, the number of receivers R interested in a packet depends upon the keys assigned to that packet. To compute the bandwidth overhead for the rekey payload, we have to find the number of receivers interested in each packet of the payload in the average case given the size of the logical key tree, N and the number of member leaves, L, being processed.

This problem also arose in analysis of the proactive FEC protocol. Recall that in the analysis of the FEC protocol, we needed to estimate the number of receivers interested in the packets of each FEC block in the rekey payload. By setting the FEC block size to 1 packet, the technique described in step 3 of Figure 5 will give us an estimate of the number of receivers interested in each individual packet in the rekey payload. Given this information, we can compute the average bandwidth overhead for each packet using equation 7, and then compute the average overhead for all the packets in the payload.

5 Performance Evaluation

In this section, we compare the performance of the Multi-send, WKA-BKR, and proactive FEC-based rekey transport protocols. We use two metrics to evaluate the performance of these protocols: the average bandwidth overhead (as defined in Section 4) and the latency of rekeying, which as defined as the number of multicast rounds taken by a protocol for successfully delivering the keys in the rekey payload to all the members of the group.

The network topology we employ in our simulations is similar to that used in other performance studies [9, 24] 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 [23, 4] 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 [4] 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 [8] have used the same approach for modeling temporally correlated packet losses.

We use the the analytical approach described in Section 4 to compute the average bandwith overhead for a protocol under the heterogeneous independent packet loss scenario. The average bandwidth overhead for the correlated packet loss scenarios and the latency results for all scenarios are obtained via simulation. Our simulation programs were written using the CSIM 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 values.

In our evaluation, we assume that a packet contains up to 40 keys and use a FEC block size of 10 for the proactive FEC-based key delivery protocol based on the results reported in [25]. 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). To simplify our simulations and analysis, we assume that the number of joins being processed is equal to the number of leaves, i.e., $J = L$. By varying L, the number of leaves that are being processed, we can model a wide range of group dynamics.

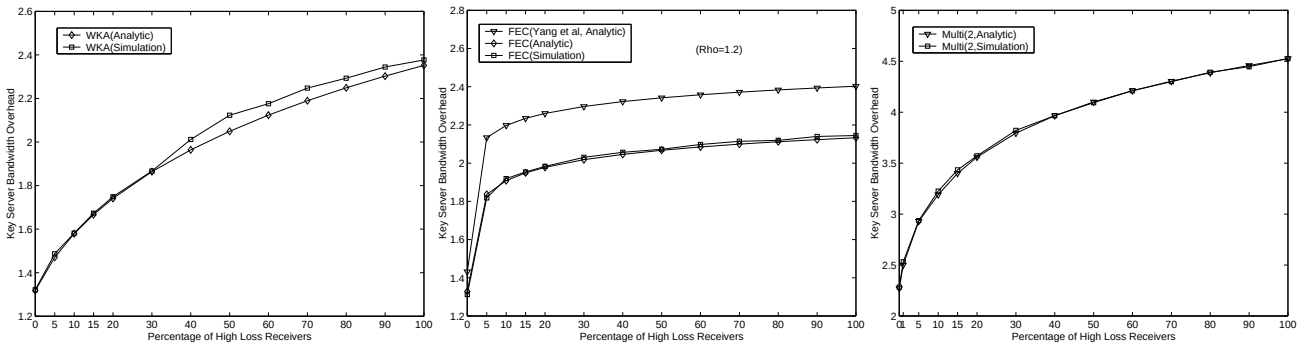
Based on the results reported in [22], we used a logical keytree with degree 4 in our analysis and simulations. Unless otherwise stated, the default parameter settings for our results are as follows: Group size (N) = 65536, Number of member leaves (L) = 256, percentage of receivers experiencing high loss (α) = 20%, high receiver link packet loss rate (p_h) = 0.2, low receiver link packet loss rate (p_l) = 0.02. In our discussion below, we use Multi(x) to denote the Multi-send(x) protocol and FEC(x) to denote the proactive FEC based key delivery protocol with proactivity factor x.

5.1 Performance Results

5.1.1 Validation of Bandwidth Overhead Analysis

The analysis present in Section 4 uses several approximations in deriving the expected bandwidth overhead for a protocol. We validated our bandwidth overhead models by comparing the numerical results obtained using the analytical expressions in Section 4 with results obtained via simulation.

In Figures 6(a), (b), and (c), we plot the predicted bandwidth overhead obtained via analysis and simulation as a function of α for WKA, FEC(1.2), and Multi(2) respectively. We observe that our analytical models give us very accurate results. In the case of WKA, the largest relative error is 3.4% when $\alpha = 0.5$. For the region of greatest interest, $\alpha < 0.3$, the maximum error is 1.1%. For FEC, our model gives us very accurate results – the largest relative error is 1.2% for $\alpha = 0$. Note that Figure 6 also plots the bandwidth overhead predicted using the model in [24]. We can see that their analysis gives us an overestimate of the bandwidth overhead. Finally, for Multi-send(2), our model is again very accurate; the maximum relative error is 1.2% for $\alpha = 0.01$.



(a) **(b)** **(c)**

Figure 6: A comparison of the expected bandwidth overhead obtained via analysis and simulation for the (a) WKA-BKR, (b) FEC(1.2), and (c) Multi-send(2) protocols. Here $N = 65536$ and $L = 256$.

5.1.2 Comparison of Protocols

In Figure 7, we plot the expected bandwidth overhead as a function of α for the three protocols in our default scenario, $N = 64K$, $L = 256$, $p_h = 0.2$ and $p_l = 0.02$. We can make the following observations. First, for the multi-send protocols, there is very little difference between the bandwidth overheads for Multi(1) and Multi(2). Second, for FEC, FEC(1.2) and FEC(1.6) have comparable overhead except for $\alpha = 0$ where FEC(1.2) has lower overhead. Third, we observe that both FEC and WKA outperform Multi-send by a wide margin, and the performance benefits of these approaches over Multi-send increase as the fraction of high loss receivers increases. Fourth, we observe that WKA uses lower bandwidth than FEC. For $\alpha \geq 1\%$, the difference in bandwidth overhead between the two protocols is around 26%. Fifth, we observe that the FEC-based protocols are much more sensitive to heterogeneity in the receiver loss rate than WKA-BKR. For $\alpha = 0$, FEC(1.2) and WKA have approximately the same key server bandwidth. However, if the fraction of high loss receivers increases to 1%, the bandwidth overhead increases by 25% for FEC(1.2), whereas it only increases by 2% for WKA. The explanation for this behavior is that in the FEC-based protocol for key delivery (see Figure 3), 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 bandwidth used in the retransmission phase of the protocol. In the case of WKA-BKR, however, the key server uses batched key retransmission to retransmit the keys needed by the receivers – the number of keys retransmitted depends upon the number of receivers who are missing packets. An individual receiver will require at most $\log N$ keys, thus a small set of high loss receivers will have a much smaller impact on the retransmission overhead than in the case of FEC.

Overall, Figure 7 shows that WKA-BKR outperforms FEC-based protocols from the viewpoint of key server bandwidth, which is typically the bottleneck resource for group rekeying [24, 14].

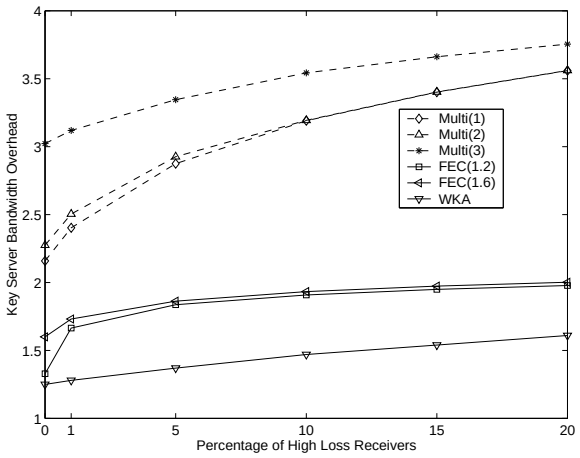


Figure 7: Key server bandwidth overhead as a function of α for a heterogeneous loss scenario for the Multi-send, FEC, and WKA-BKR protocols. Here $N = 65536$ and $L = 256$.

Next, we examine the latency of key delivery for the various protocols. Instead of plotting the average number of rounds taken by the protocols for key delivery, it is more illuminating to consider the distribution of the number of members who have received their keys after a certain number of rounds. Figure 8 (a) shows the fraction of group members who have not yet received all their keys at the beginning of a certain round. We observe that FEC-based protocols have the lowest latency among the protocols. However, with the exception of Multi(1), in the case of all the protocols, close to 99% of the members receive their keys by the end of the second round (Note that the Y-axis is in log scale). We observe that the more proactive a

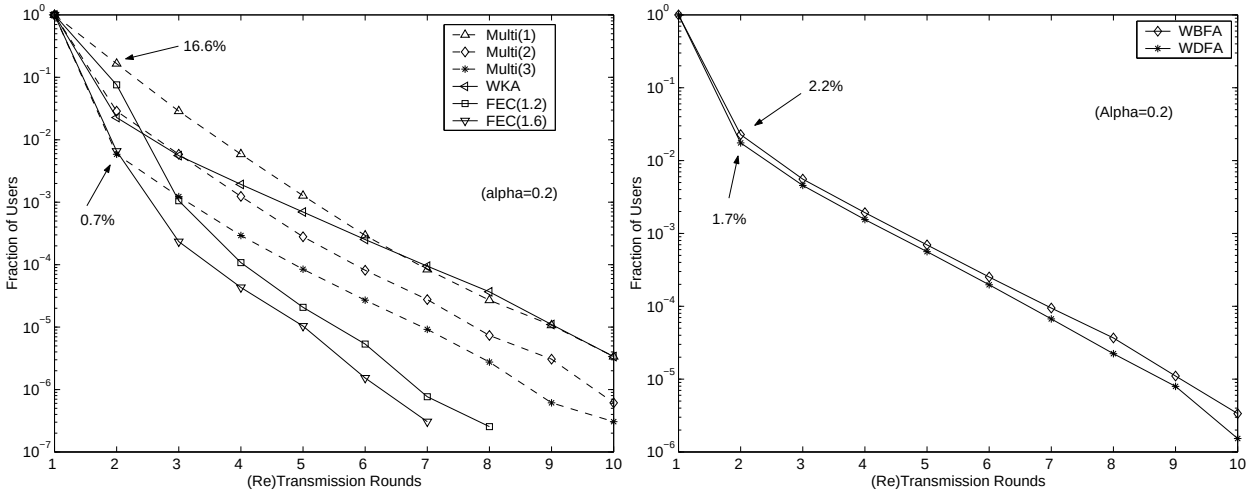


Figure 8: (a) Fraction of members who have not received their keys at the beginning of a round for the Multi-send, FEC, and WKA protocols. (b) Fraction of members who have not received their keys at the beginning of a round for the WKA-BKR protocol using WDFA and WBFA key assignment algorithms. Here $N = 65536$ and $L = 256$ and $\alpha = 0.2$.

protocol, the fewer rounds required before all members have 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.

Figure 8(b) examines the impact of the key assignment algorithm on the latency of the WKA-BKR protocol. Note that for this protocol the bandwidth overhead is independent of the packing scheme. Figure 8(b) shows that the WDFA algorithm results in slightly lower latency for WKA. This result is not unexpected - as discussed in Section 3.2.2, the keys that a specific receiver needs are likely to be distributed among a larger number of packets under WBFA than WDFA. Our results show that the larger the number of packets a receiver needs, the larger the expected number of rounds for receiving all the keys. This trend is also observed for the FEC-based protocols as discussed below. Given the fact that the average bandwidth overhead for WKA-BKR is not affected by the key assignment algorithm, from Figure 8(b) we can conclude that WDFA should be used for key assignment for WKA-BKR.

5.1.3 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 overhead for the protocols. Note that we did not adjust p_h and p_l to ensure that the average loss rates observed by receivers are either p_h or p_l; instead, we assume each receiver has an independent loss rate of p_h or p_l plus the loss rate in the source link.

In Figure 9(a), 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 shared source link result in an increase in the bandwidth overhead of all the protocols. However, the relative performance of the three protocols remains the same as

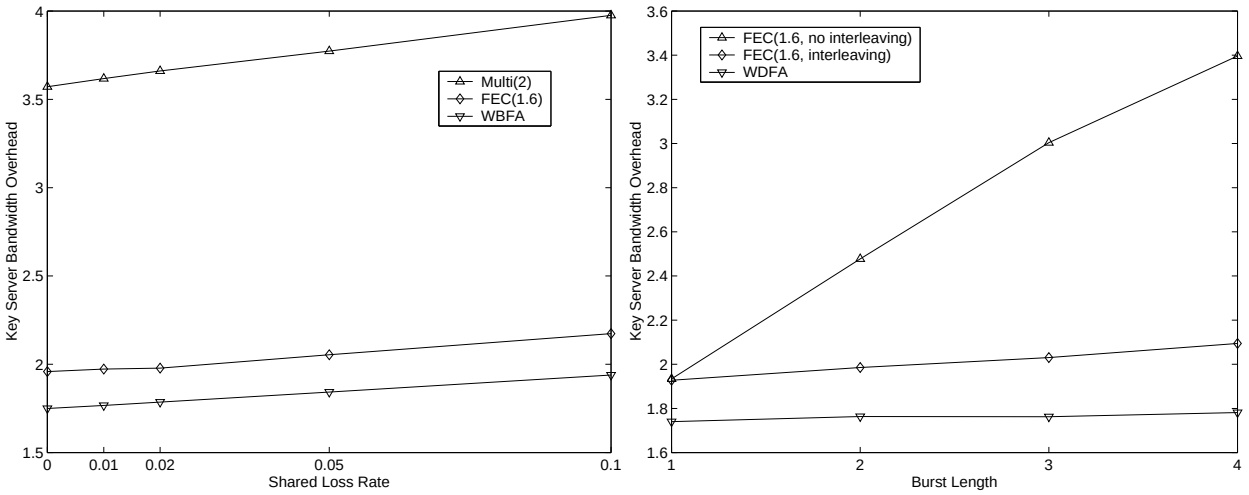


Figure 9: Impact of correlated loss on reliable key delivery protocols. (a) The average bandwidth overhead as a function of shared loss in the source link and (b) The average bandwidth overhead as a function of the length of burst loss. Here $N = 65536$, $L = 256$, and $\alpha = 0.20$

in the heterogeneous independent loss scenario. In general, in our simulations we found that although losses on shared links have a greater impact on the key server bandwidth for WKA-BKR than FEC, the relative performance of the two protocols is unchanged as long as packet losses are not dominated by losses on the shared source link.

Impact of Temporally Correlated 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 [14].

To increase the probability that a packet will be received despite time-correlated losses, a protocol that employs redundancy can interleave the packets containing redundant information with other packets. In the case of FEC-based key distribution protocols, packets from different FEC blocks can be interleaved at the time of transmission. In the case of WKA-BKR (using WDFA key assignment), packets containing copies of the same key are automatically interleaved by the WDFA algorithm. This is because packets containing copies of the same key are generated during different depth-first traversals of the keytree by the WDFA algorithm, reducing the probability that a set of packets transmitted consecutively contains copies of the same key.

In Figure 9(b) we plot the average bandwidth overhead for FEC(1.6) and WKA as a function of the average length of burst losses. To examine the effect of packet interleaving on FEC, we simulated two versions of the protocol – one with packet interleaving and one without interleaving. We observe from this figure that increasing the burstiness of the packet losses has a negligible impact on the average bandwith overhead of WKA and FEC with packet interleaving. In contrast, when packet interleaving is not used with FEC, the bandwidth overhead increases dramatically as the burst length increases. This result shows that FEC-based protocols are much more sensitive to temporally correlated packet loss than WKA-BKR, and transmitting packets from different blocks in an interleaved fashion is necessary for reducing the key server bandwidth for FEC-based protocols.

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.

5.1.4 Impact of Changing the Group Size & 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, 256K). We observe from Figure 10(a) that the bandwidth for all the protocols increases with the group size. WKA has lower bandwidth overhead than the other policies but its overhead increases with group size at a larger rate than FEC. The explanation for this behavior is that if the group size is increased the height of the keytree increases. Keys at the upper levels of the keytree have larger replication factors and bandwidth overhead under WKA. The proportion of keys from the upper levels of the key tree in the rekey payload also increases as the height of the keytree increases. Thus, the bandwidth overhead for the protocol is dominated by the overhead for delivering these keys, leading to a higher overall bandwidth overhead. In Figure 10(b), we plot the bandwidth overhead for the protocols for different group sizes assuming that the number of leaves is a equal to fixed fraction of the group's members, specifically $L = N/4$. We observe that the bandwidth overhead of WKA is lower than that of the other protocols, and changing the group size has no noticeable impact on the bandwidth overhead. This is because the proportion of keys in the rekey payload from the upper and lower levels of the keytree depends upon the ratio of group leaves to group size. Increasing the group size while keeping this ratio fixed as in Figure 10(b) does not change the bandwidth overhead of WKA-BKR.

Finally, Figure 10(c) examines the impact of changing the number of leaves while keeping the group size constant. We observe that under all the protocols, the bandwidth overhead decreases. As L is increased for a fixed N, the proportion of keys in the rekey payload that corresponds to keys from the lower levels of the keytree increases. The bandwidth overhead for delivering these keys is lower on average than the bandwidth overhead for the keys at the upper levels of the keytree. Thus, this results in an overall reduction in the average bandwidth overhead.

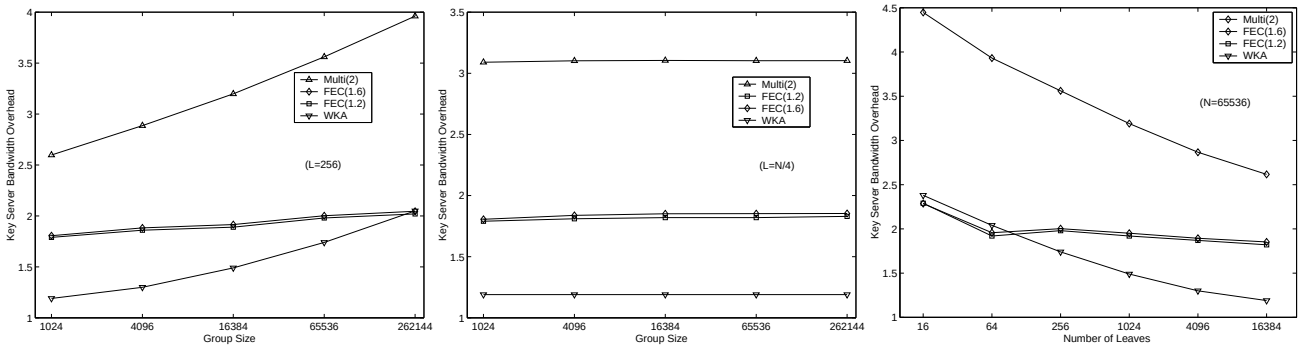
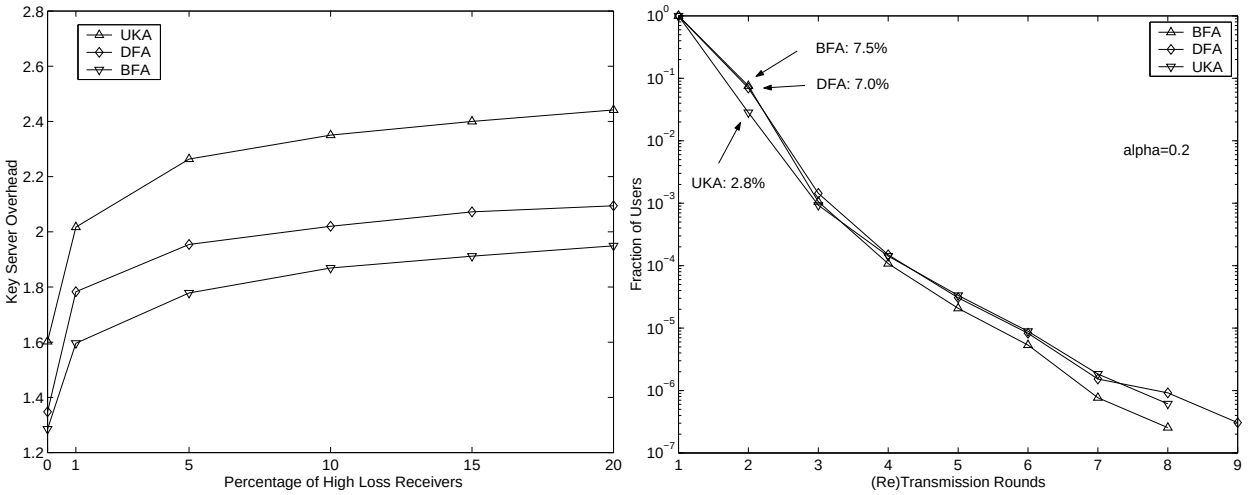


Figure 10: The impact on the key server bandwidth overhead of (a) changing the group size (N) for L = 256, (b) changing the group size (N) for $L = N/4$, and (c) changing L for $N = 65536$.

5.1.5 Impact of Key Assignment Algorithms on FEC

In Figure 11, we examine the impact of the different key assignment algorithms on the performance of FEC-based and WKA-BKR key delivery protocols. In Figure 11(a), we plot the bandwidth overhead for three different key assignment algorithms when used in conjunction with proactive FEC. The three algorithms



(a) **(b)**

Figure 11: Impact of key assignment algorithms on FEC based reliable key delivery: (a) The average bandwidth overhead for FEC(1.2) as a function of α and (b) Fraction of users who have not received their keys at the beginning of a round. Here $N = 65536$, $L = 256$, and $\alpha = 0.2$.

considered are DFA, BFA, and UKA (user-oriented key assignment). Under the UKA algorithm proposed in [25] all the keys needed by a user are assigned to a single packet in the rekey payload; thus, each user needs to receive exactly one packet during a rekey event. To achieve this property, the UKA algorithm has to replicate some keys in multiple packets; this contributes to the higher bandwidth overhead of UKA in comparison to BFA and DFA.

Figure 11 (a) shows that the BFA packing algorithm has the lowest bandwidth overhead[1]. The reason for this behavior is that among the three algorithms considered, BFA can take maximum advantage of the sparseness property of the rekey payload. As discussed in Section 4.4, the bandwidth overhead for reliably delivering a FEC block depends upon the number of receivers interested in the keys in that block. The larger the number of receivers interested in the keys of a block, the larger the bandwidth overhead. The BFA algorithm typically packs keys from the highest levels of the keytree into the first FEC block, whereas the remaining blocks contain keys from lower levels. Thus, for BFA, the number of interested receivers will decrease for consecutive FEC blocks. In contrast, UKA and DFA tend to distribute keys from all levels of the keytree among the FEC blocks since they partition the key tree vertically. This results in each FEC block containing keys that are of interest to a large fraction of the group. We can see this clearly in Table 1 which shows the number of receivers interested in a particular FEC block for BFA and DFA for our default scenario. Consequently, BFA has the lowest average bandwidth overhead for the entire rekey payload among the three packing algorithms.

On the other hand, Figure 11(b) shows that UKA and DFA have lower latency than BFA. UKA has the lowest latency among the policies since each user needs to successfully receive only the single packet that contains it keys, while BFA has the largest latency because a user's keys are distributed over several packets. Since the differences in latency between the different key assignment algorithms are small, from Figure 11 we can conclude that BFA key assignment has the best performance.

We note that our results do not agree with those in Yang *et al's* study [24]. Yang *et al* evaluated the performance of various packing algorithms for proactive FEC based key delivery and recommended the use

[1]The results for DFA and UKA in Figure 11 were obtained via simulation.

Block	0	1	2	3	4	5	6	7	8	9
BFA	65536	25600	15068	6400	5051	1600	1600	642	400	299
DFA	16384	9920	19060	8936	21480	7304	10112	20116	9600	5152

Table 1: Number of receivers interested in a specific block for proactive FEC-based key delivery protocols. There are 100 packets in the rekey transport payload leading to 10 FEC blocks of size 10. Here N=65536, L=256, α = 20%

of DFA and RDFA (a variant of DFA) packing algorithms in preference to BFA. As discussed in Section 4.4, however, their performance analysis did not take into account the impact of sparseness property of the rekey payload *across the different FEC blocks* in the payload. If the entire rekey payload fits within one FEC block, then DFA and RDFA do perform better than BFA. However, as discussed above, for rekey payloads consisting of multiple FEC blocks, BFA results in better performance because it can take advantage of the sparseness property of the rekey payload across FEC blocks.

Finally, we observe that a way to reduce the bandwidth overhead for FEC is to use BFA key assignment in combination with a weighted scheme for the proactivity factor, i.e., assign different proactivity factors to different FEC blocks on the basis of the number of receivers interested in each block. Note that it is easy to estimate the number of members who are interested in the keys being assigned to a FEC block during the key assignment process. Given the number of receivers interested in the keys in a FEC block, the analytic model proposed by Yang et al [24] can be used to guide the selection of an appropriate proactivity factor for that FEC block.

6 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 also present analytical models for evaluating the bandwidth overhead of our protocol and other protocols that have been proposed for reliable group rekey transport. Using both analysis and simulation, we investigated the factors that affect the bandwidth overhead of reliable key delivery protocols and compare the performance of WKA-BKR with that of other rekey transport protocols.

The main conclusions of our performance study are:

- For most scenarios, WKA-BKR has a lower bandwidth overhead in comparison to previously proposed protocols. The difference in bandwidth overhead is significant – it is up to 26% lower than that of FEC-based protocols and up to 60% lower than the overhead of simpler replication based multi-send protocols for the default scenario considered in our performance study.

- FEC-based protocols have lower latency on average than the WKA-BKR and the multi-send protocols. However, for all the protocols considered, almost 99% of the members of the group receive their keys by the end of the second round of the protocol.

- 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 than FEC-based protocols.

- The performance trends listed above hold up for network scenarios with temporally and spatially correlated packet losses.
- The WDFA key assignment algorithm should be used with WKA-BKR since it results in lower latency while having no impact on the bandwidth overhead. For FEC-based protocols, however, BFA can result in a significant reduction in bandwidth while leading to slightly higher latency. The bandwidth of FEC-based protocols can be further reduced by using a weighted scheme for selecting the proactivity factors of different FEC blocks in the rekey payload.

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] 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.

[3] 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.

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

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

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

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

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

[9] M. Lacher, J. Nonnenmacher, E. Biersack. Performance Comparison of Centralized vs Distributed Error Recovery for Reliable Multicast. *IEEE/ACM Trans. on Networking*, April 2000.

[10] B. Levine and J. Garcia-Luna-Aceves. A Comparison of Known Classes of Reliable Multicast Protocols. In *Proc. of ICNP '96*, 1996.

[11] 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.

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

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

[14] 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.

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

[16] 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.

[17] S. Setia, S. Zhu, S. Jajodia. A Scalable and Reliable Key Distribution Protocol for Multicast Group Rekeying Technical Report, George Mason University, January 2002.

[18] 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.

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

[20] 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.

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

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

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

[24] 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.

[25] 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

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

Key Identification

We adopt the approach used by Zhang *et al* [25] 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 integral id. Nodes are numbered from 0 while traversing the key tree in a top-down and left-right fashion. For example, in Figure 2, 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 relation also holds between user nodes and key nodes if we add a base number (equal to the total number of key nodes in the keytree) to the id of a user node.

During a rekeying operation in the LKH approach, a receiver needs to be able to identify the encrypted keys it needs, i.e., the updated keys on the path from its corresponding node in the keytree to the root. Given

the key numbering scheme outlined above, a receiver 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* [25] for further details on this scheme.

Packet Formats

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

The format of WKA packet is as follows:

1. Packet type: WBFA (2 bits)
2. Rekey message Id: (8 bits)
3. Number of keys included: (6 bits)
4. Total mumber of packets used for this rekey operation: (8 bits)
5. <fromId,toId>: 48 bits. Members can use this field to quickly check if the packet contains any encryptions that they need.
6. 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: (8 bits)
3. Number of keys included: (6 bits)
4. Total number of packets used for this rekey operation: (8 bits)
5. 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: (8 bits)
3. Member Id: (24 bits)
4. Member loss rate: (6 bits)
5. Bitmap of losses: (16 bits). Each bit is mapped to a level in the keytree, and is used to indicate if the receiver is missing a key at that level.