A Peer-Based Recovery Scheme for Group Rekeying in Secure Multicast

Qingyu Zhang and Kenneth L. Calvert
(Corresponding author: Qingyu Zhang)

Laboratory for Advanced Networking, University of Kentucky, 301 Rose Street, Lexington, KY 40506, USA
(Email: {qingyu, calvert}@netlab.uky.edu)

(Received Jan. 1, 2006; revised and accepted May 25, 2006)

Abstract

The Logical Key Hierarchy (LKH) provides a scalable and efficient way to distribute session keys to authorized group members in secure group (multicast) communication. However, because multicast is a best-effort service, it requires additional mechanism to ensure that every member receives the rekey information. To solve this problem, several FEC-based schemes have been proposed. Although these schemes significantly enhance the reliability of the rekey transmissions, they require significant additional resources at the key server. In this paper we propose a distributed scheme for recovering lost rekey packets. Our scheme allows each member to contact other members to get the missing packets during rekey events, thus avoiding the extra computation and bandwidth resources at the key server. We compare our scheme with a well-known FEC scheme with respect to latencies experienced by receivers in reasonably large groups. Results show that our scheme yields lower average and 95th-percentile latencies than the FEC-based scheme across several operating regimes.

Keywords: Rekey, reliability, secure multicast

1 Introduction

The growth of the Internet has led to the development of many group-oriented applications such as video distribution, stock quote delivery, and news broadcast. Some of these applications require that only *authorized* group members be able to access group data. Although technologies such as multicast [5] provide an efficient way to deliver data to a group of recipients, they do not provide access control—anyone can join the multicast group to receive group data.

The Secure Multicast working group (MSEC) of the Internet Engineering Task Force (IETF) has developed a general architecture for secure group communication [1, 6]. The basic idea of the approach is that group members share a symmetric key called a *session key*, and group data is encrypted with that key. Access to transmitted data is controlled by ensuring that only authorized members know the current session key. When the set of group members changes (i.e. when current members leave or new members join), the session key needs to be changed to prevent unauthorized access. The update of session key is called a *rekey event*. The central problem of secure group communication is how to make rekey events efficient; the design of a scalable rekey scheme has been studied extensively in recent years [8, 9, 13, 14]. In the MSEC architecture, a logical entity called the Group Controller and Key Server (GCKS) is responsible for managing the rekey events.

The Logical Key Hierarchy (LKH) [13, 14] approach has so far received the most attention. In LKH, the GCKS maintains a tree of auxiliary keys and assigns to each member a subset of these keys; the basic idea is that a new session key can be securely and efficiently transmitted to the group by encrypting it with certain auxiliary keys. When membership in the group changes, the session key and some of the auxiliary keys need to be changed. The GCKS encrypts the new keys in such a way that only currently-authorized members have the proper auxiliary keys to decrypt the necessary new auxiliary keys then get the new session key. This approach (described in more detail in the next section) reduces the computational complexity (i.e. number of encryptions) at the GCKS to $O(log_d N)$, where N is the group size and d is the degree of the *key tree*. Nevertheless, a rekey operation can consume a significant amount of system resources (computation and bandwidth) at the GCKS.

It is crucial for the LKH approach that each authorized member receive the packets containing the session key and the auxiliary keys he needs. Otherwise, he won't be able to access the group data encrypted with the new session key. Furthermore, these auxiliary keys are used in future rekeying to protect new session keys. Any authorized member who does not have the proper auxiliary keys will be excluded from access to future group. Therefore, the reliability of rekey messages is an important issue for LKH.

To address this problem, several schemes based on the use of forward error control (FEC) have been proposed [15, 16]. In these schemes, error-correction codes are used to produce extra redundant packets that are multicast along with the original rekey packets. Any member who receives an adequate subset of the transmitted packets can recover the necessary auxiliary keys. This scheme significantly enhances the reliability of the rekey transmissions; however, it requires significant additional computational resources at the GCKS.

In this paper we propose a *distributed* scheme for recovering lost rekey packets. In our approach, each member maintains information about a small number of other group members, called *peers*. At rekey time, a member who fails to receive all required multicast packets will contact his peer(s) to ask for the missing packets. The basic idea behind our solution is that multicast losses are generally scattered throughout the multicast tree, and it is unusual for a packet to be delivered to *no* receivers; a judicious assignment of peers can greatly increase the likelihood that some peer always has the needed packet. The advantage of our scheme is that it avoids the extra computation and bandwidth resources at the GCKS required by FEC-based schemes. This makes it possible for ordinary members to act as GCKS. By reducing the overhead at the GCKS, the group key management system is more scalable, i.e, it can support more and larger groups.

The main contributions of this paper are the introduction of the peer-based recovery method for rekeying, and a careful comparison of its performance characteristics to those of the best known FEC scheme. We perform simulations under different conditions to compare the distribution of latencies experienced by receivers in each scheme. Our results show that the peer recovery scheme yields lower average and 95th-percentile latencies than the FEC-based scheme across several operating regimes.

The rest of this paper is organized as follows. In the next section we present a more detailed description of the LKH key management approach, along with a discussion of alternative approaches to reliability. Section 3 describes our peer-based approach, while Section 4 compares the expected latency of our approach and that of AFEC, a published FEC-based rekey approach. Section 5 presents the results of our simulation study. Section 6 concludes the paper.

2 Background and Related Work

As noted in the Introduction, the problem of efficiently rekeying a secure group has received a good deal of attention. The Logical Key Hierarchy method was independently developed several years ago by Wallner et al. [13] and by Wong, Gouda and Lam [14]. Since then, other related approaches have been proposed, including one-way function trees [8] and a method based on subset differences [9]. All of these methods assume a Group Controller and Key Server (GCKS) that is responsible for enforcing the *group policy*, which defines the set of authorized members. At random times, the set of authorized members may change when new members join, or existing members leave the group. Therefore from time to time the GCKS initiates a *rekey event*.

In this paper we focus on the LKH approach, because it is very efficient overall, and has been well-studied.

2.1 The Logical Key Hierarchy

In LKH, the GCKS organizes the session key and auxiliary keys into a regular tree structure called the *key tree*. The session key is the root. Each member holds a key known only to itself and the GCKS, which corresponds to a leaf node in the tree; that key need not change throughout the user's membership. Each member knows all the keys on the path from its own leaf node to the root node. When a member joins or leaves the group, the keys on the corresponding path have to be changed to prevent the departing member from accessing subsequent data, or the new member from accessing old data. At the time of rekeying, the GCKS encrypts each updated key with each of its "child keys" respectively then puts the encrypted keys into rekey packets. For example, in Figure 1, when M_9 leaves the group, the key server constructs the encryptions by traversing the key tree in a bottom-up fashion: $\{k_{7-8}\}_{k_7}$, $\{k_{7-8}\}_{k_8}$, $\{k_{1-8}\}_{k_{1-3}}$, $\{k_{1-8}\}_{k_{4-6}}$, $\{k_{1-8}\}_{k_{7-8}}$. Here $\{k_i\}_{k_j}$ denotes key k_i encrypted with key k_j. Upon receiving rekey packets, each member extracts and decrypts the needed auxiliary keys. In our example, M_1 needs $\{k_{1-8}\}_{k_{1-3}}$.

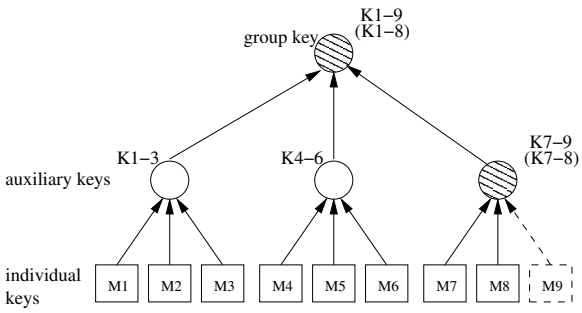

Figure 1: An example of key tree

To reduce overhead in situations where the group membership changes relatively rapidly, *batched-LKH* was proposed by Yang and others [15]. The idea of batched-LKH is to accumulate a number of membership changes in a batch before changing the session key. By amortizing auxiliary key changes over more membership changes, this method requires fewer encryptions per membership change and thus saves system resources at the GCKS. In exchange for this increased efficiency, the data may be accessible to unauthorized participants (or inaccessible to authorized participants), for some periods of time.

2.2 Reliable Transmission Mechanisms

The problem of the reliable transport of rekey messages has been discussed in the MSEC working group of the IETF. Baugher et al. [1] identify general categories of solutions to address loss of rekey messages. One general approach is to use an existing reliable multicast protocol solution. Different types (NACK-based, ACK-based, etc.) of reliable multicast protocols have been specified; each has different properties and is suitable for different situations. However, as Yang et al. have observed [15], the rekey transport workload in LKH has a *sparseness property*: although the GCKS generates a lot of encryptions at each rekey event, any particular member needs only a relatively small fraction of them. In our previous example (Figure 1) when M_9 leaves the group, M_1 needs only $\{k_{1-8}\}_{k_{1-3}}$. If the encryptions are carefully packed into packets [17], a member may need as few as *one* packet among all the transmitted rekey packets. Therefore it is unnecessary and inefficient to use a reliable multicast protocol designed to ensure that *all* members to receive *all* rekey packets.

The other general approach is to transmit redundant information along with the rekey messages, to enable receivers to reconstruct the required (encrypted) keys. For example, the rekey messages themselves might be transmitted several times. However, more efficient use of bandwidth can be achieved with modern Forward Error Control (FEC) coding techniques (erasure codes), which reduce the amount of redundancy required to reconstruct the desired packets in the face of a modest number of packet losses.

Generally, erasure coding techniques group a fixed number (k) of packets together, and then apply an algorithm to generate additional, redundant packets to form a *block*; the total number of packets in a block is n, $k < n$. If a receiver obtains *any* k out of n packets in a block, he will be able to reconstruct the original k data packets. Several FEC-based reliable transport protocol have been proposed for Batched-LKH [15, 16].

Yang et al. [15] presented a round-based proactive FEC protocol. In their protocol, the GCKS first packs the encryptions into rekey packets, then uses the FEC encoding technique to generate some redundant packets and multicasts all the packets to the entire group. At the end of the multicast round, the GCKS collects negative feedback from members. The feedback of a member is a NACK packet containing the number of packets a_r that the member needs to reconstruct the required packets. The GCKS keeps track of the largest value of a_r, generates that many new FEC repair packets, then multicasts the repair packets at the next round. This process continues until all members recover their required rekey packets.

One potential problem of this scheme is *feedback implosion*, i.e., the GCKS may receive a lot of NACKs at the end of first round. Zhang et al. [16] proposed an Adaptive FEC-based scheme (AFEC) which limits NACK feedback to a small fraction of the member population. Their protocol runs only one multicast round. At the end of multicast round, the members who couldn't reconstruct their rekey packet from the multicast will contact GCKS for unicast recovery. Based on the feedback from previous rekey events, the GCKS dynamically adjusts the redundancy factor of the encoding to achieve a target residual error rate at the next rekeying. To minimize the effect of burst loss on a single block, the authors designed a packet spacing strategy in which the packets in the same block are sent equally spaced during the sending interval while packets from different blocks are sent in an interleaved fashion. For example, if a rekey message is divided into three blocks b_1, b_2, and b_3, the GCKS will send the first packet in b_1, b_2, and b_3, then send the second packet in b_1, b_2, and b_3, and so on so forth. In this way, the packets in the same block are spaced out as far as possible and the possibility of burst loss destroying an entire block is reduced.

The main problem with FEC-based schemes is that FEC encoding is computationally expensive and introduces additional bandwidth overhead. Thus the load on the GCKS is significantly higher than that on the receivers. Also, the overhead increases with the loss rate experienced by receivers.

3 Recovery from Peer Members

The goal of our protocol is to let members in an LKH scheme get required rekey packets in a timely manner while reducing the overhead at the GCKS. We observe that each member and its sibling members in the key tree share all the keys except for the individual key; therefore they need the same rekey packets. Based on this observation, we propose a distributed scheme to use these sibling members to retransmit the missing rekey packet(s) in case of packet loss. This scheme does not require FEC encoding, and therefore saves computation power at the GCKS, as well as multicast bandwidth. It also avoids the potential "NACK implosion" problem at the GCKS. Instead, it requires that members retain the rekey packets they receive for a short time, and assist other members by forwarding the packets upon request.

3.1 Definitions

Remember that a key tree is a tree structure in which each node represents a key and a leaf node also represents a member. The leaf members who share the same parent node in the key tree form a *cluster*. Let $c(m)$ represent the cluster member m belongs to. Then the *sibling peers* of m, $sp(m)$ are the other members in the cluster, i.e., $sp(m) = c(m) \setminus \{m\}$. For example, in the key tree in Figure 3, M_1's sibling peers are M_2 and M_3.

When m is the only member in $c(m)$, $sp(m)$ is null. In this case, we find *indirect peers*, $ip(m)$, for m. We assume that the key tree is balanced[1]. Indirect peers of m are

[1] Zhang et al. [17] suggested an algorithm to construct a balanced

```
parent = get_parent(m);
//first_sibling() get the first sibling
// in sequence of id on the tree.
uncle = first_sibling(parent);
ip = children(uncle);
```

Figure 2: Find indirect peers $ip(m)$ of m

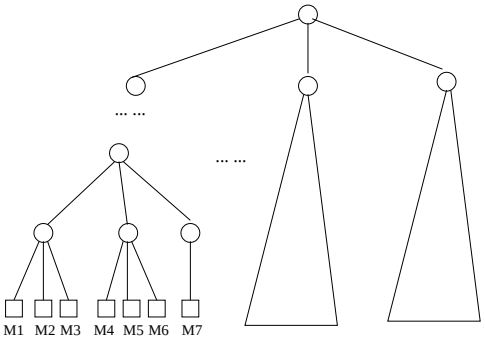

Figure 3: Part of key tree

m's cousins (children of m's parent's sibling). Figure 2 illustrates the process to find indirect peers. Given the above definition, in Figure 3 M_7's indirect peers are M_1, M_2 and M_3.

We define m's *logical peers* $lb(m)$ as the union of sibling peers and indirect peers. If the degree of the key tree is d, each member will know up to $d-1$ sibling peers and up to d remote peers. The peer information maintained at each member is $O(d)$, and is independent of group size. Briefly speaking, the protocol operates in the following way. The GCKS knows the entire logical tree; when a member joins, the GCKS informs it of its logical peers (and may optionally provide an additional shared secret key with which members of the peer group can authenticate each other). The joining member establishes a TCP connection with each of its logical peers. When a member detects the loss of a required rekey packet, it asks one of its logical peers to retransmit the packet. We explain our protocol in more detail in the following subsections.

3.2 Recovery for Required Packet

We assume that given a member's ID in key tree, any neighbor can easily determine which rekey packet is necessary to that member. In the LKH algorithm [17], for example, the header of a rekey packet has a field that specifies the members who will need this packet. Therefore our assumption is trivial.

During a rekey event, each member looks for the packet he needs by checking the header of rekey packets. When a member sees a gap in the sequence number of rekey packets, it is likely that the missed packet was dropped on the way. If the dropped packet is required by the member, it sends a request for the packet to one of its peer(s). When a member detects loss of a required packet, it does not wait until the end of a multicast "round" to ask for retransmission; instead, it uses the following fast initiation strategy: Assume m requires the packet with sequence number p_i. When m receives any packet with sequence number greater than $p_i + b$ without having received p_i, it requests retransmission from its peer. The parameter b is intended to keep (slightly) reordered packets from being mistaken for lost, and is set to 3 in our simulation. Handling the case where a packet is lost within b packets of the end of the sequence, however, does require a timeout.

To recover from the loss, the member sends a retransmission request to the first peer in its peer list, and waits for the reply. When a peer receives a retransmission request, it simply sends back the rekey packet the requesting member needs. If the peer does not have the packet, it sends a negative reply, and the requesting member contacts the next peer in its list. In case none of the peers have the requested packet, the member has to ask the GCKS to retransmit the packet.

A more aggressive strategy is to contact all the peers at the same time when a member needs retransmission. Then the member processes the first packet retransmitted from the peers and ignores the rest of replies. This strategy helps a member get a required packet earlier if it has to contact more than one peer for retransmission. The disadvantage of this strategy is that it introduces extra bandwidth overhead when a member sends requests to all the peers. However, the request packet is small, and each of the peers retransmits only one packet. So the extra bandwidth overhead will have little impact on the network.

3.3 Logical Peer Assignment

Our scheme is based on the assumption that a member's logical peers receive its required packets, even if it doesn't. This assumption may be violated if either (i) losses occur close to the GCKS, and thus affect all members; or (ii) logical peers are close to each other in the multicast tree, and are therefore likely to be affected by the same losses. The former problem is common to any multicast-based approach. The latter problem can be avoided by placing members in the key tree in such a way that logical siblings are not close together in the network topology.

Recall that the GCKS is responsible for assigning members to the key tree. The simplest approach is to replace the leaf node of a departed member with a joining member randomly. The advantage of this strategy is that it is used by existing Batched-LKH schemes, so no change is needed to the key tree assignment algorithm. The disadvantage is that it may not guarantee that logical siblings are far

key tree.

apart in the multicast tree, particularly for small groups. If the GCKS can determine members' physical location, it can try to maximize the distance between members in the same cluster when it assigns them to the key tree. However, the problem of determining nodes' physical location in the network is beyond the scope of this paper. We note, however, that one promising approach is to use a Network Coordinate System [3, 7, 10, 11] to determine each member's location in a space that corresponds in a known way to physical space. Given this information, it is possible to assign members so as to maximize the distance between logical neighbors. However, it is likely to be computationally difficult; similar problems, such as k-mean clustering, are known to be NP-hard [4]. We therefore leave this aspect as future work. In our performance evaluations, members are assigned to the key tree randomly.

3.4 Logical Peer Maintenance

As group membership changes dynamically, the peer list of a member may change from time to time. A straightforward way for members to update neighbor list is to let GCKS inform the affected members (by either multicast or unicast) at each rekeying. However, this method will cause a lot of communication overhead at the GCKS. Our protocol avoids the bottleneck at GCKS by having joining/leaving members inform their peers. To achieve this, two additional messages are introduced, namely, *HELLO*, and *GOODBYE*.

We assume that each new member will establish a secure reliable channel with GCKS when it registers. A new member x sends a JOIN request to the GCKS over this channel and receives the IP addresses of its logical peers in response. Then x makes a TCP connection and sends a HELLO message to each of its peers. When a member m receives a HELLO from x, m adds x to its list of established peers. When a member leaves the group, it may send GOODBYE to its peers. Upon receiving the GOODBYE message from m, x removes m from the list. A member can also know a peer has left by detecting the broken TCP connection.

4 Evaluation

In this section and the next, we compare our scheme to the Adaptive FEC-based scheme of Zhang et al. [16], referred to hereafter as "AFEC". Our goal is to produce an apples-to-apples comparison of the latency distributions achievable under realistic conditions with the two methods. In this section, we first consider the relationship between the degree of redundancy (a parameter of the FEC code) and the target residual error rate—that is, the fraction of receivers who still cannot reconstruct their required packets after all blocks have been transmitted by the GCKS. Then we derive and compare expressions for the delay experienced by a receiver under each scheme.

4.1 Degree of Redundancy in AFEC

As described above, AFEC improves multicast reliability by adding redundant packets. For any given amount of redundancy—i.e. ratio between n, the overall block size, and k, the number of data (rekey) packets—enough losses may still occur to prevent some receivers from reconstructing all the data they need. To limit the number of retransmission requests (number of NACKs) to a given target value, the GCKS will dynamically change the amount of redundancy in response to network conditions. In practice, the target number should be small to reduce the fluctuations of number of NACKs. In this section we analyze the amount of redundant packets the GCKS will generate to achieve the target NACKs.

Let r denote the fraction of group members who send NACK to the GCKS, h denote the number of redundant packets for each block. The GCKS will send $k + h = n$ packets for each block. For simplicity, receivers are assumed to experience same end-to-end loss rate, denoted by p. Zhang et al. studied the relationship between r and h at given p using both Bernoulli model and Markov model [16]. Bernoulli model assumes that the packets experience independent losses while Markov model considers correlated losses between consecutive packets. When the GCKS applies the packet spacing strategy suggested by Zhang et al. to send rekey packets, the packets of the same block will likely experience independent loss. Therefore we analyze the amount of redundancy based on the Bernoulli model.

Intuitively, r is the probability that the user does not receive its required rekey packet and receives fewer than k packets from the block. In [16], Zhang et al. derived r as a function of h and p as follows:

$$\begin{aligned} r &= p \cdot \sum_{i=0}^{k-1} \binom{k+h-1}{i} (1-p)^i p^{k+h-1-i} \\ &= p^{k+h} \cdot \sum_{i=0}^{k-1} \binom{k+h-1}{i} (\frac{1}{p}-1)^i. \end{aligned}$$

From this equation, we try to find the smallest h which can achieve the target residual rate r^* at given loss rate p. We calculate r with the increase of h at a specific p, find the closest r to the target then get the corresponding h. Figure 4 shows the result for h when r^* is 1% and 3%. Here we set k to 10.

From the figure we find that the number of redundant packets in a block increases nearly linearly with the end-to-end packet loss rate. The redundant packets can take up significant fraction of total bandwidth when loss rate is medium to high. For example, when receivers experience 15% of packet loss, h will be 30% to 50% of block size (3 to 5 packets per block in the figure). In the simulations described in the next section, we use a target NACK rate of 1.2%.

Table 1: Notation

Symbol	Description
L	average rekey latency of all members
p	the probability that a member will receive the required packet from multicast
q	the probability that a member can reconstruct the required packet in AFEC
r	the probability that a member has to contact the GCKS for retransmission
D_{mcast}	the average multicast packet delay
$D_{tcp}(i,j)$	the average TCP delay between i and j
k	the number of rekey packets per block in AFEC
h	the number of redundant packets per block in AFEC
T_A	the time to send out all packets at the GCKS in AFEC
T_P	the time to send out all packets in peer-based recovery scheme
Δ_A	the time a member waits before he sends NACK in AFEC
Δ_P	the time a member waits before he contacts his peers
c	the average number of peers a member needs to contact to get the required packet
δ_i	the delay in Case i, $i = 1, 2, 3, 4$

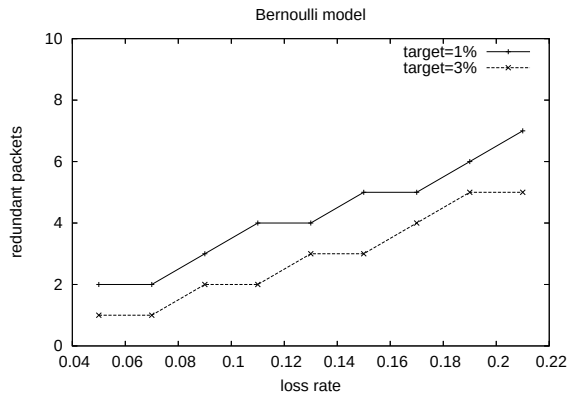

Figure 4: h as a function of p

4.2 Rekey Message Delay

We define rekey latency seen at a group member as the time from when the rekey event starts (i.e. the first packet is transmitted by the GCKS) until the member receives its required rekey packet. Table 1 lists the notations to be used in our analysis.

In AFEC scheme, a group member can get its required packet in three ways: (1) from multicast transmission directly; (2) from FEC reconstruction if it receives enough packets; or (3) after contacting the GCKS to request retransmission of missing packets. The fraction r of members that require GCKS retransmission, is the target NACK rate discussed in the previous subsection. Letting δ_i denote the delay for Case i, the rekey latency in the AFEC scheme is given by:

$$L_{AFEC} = p \cdot \delta_1 + q \cdot \delta_2 + r \cdot \delta_3$$

where $q = 1 - p - r$.

In our peer-based recovery scheme, a group member either (1) gets the required packet from multicast delivery, or (4) gets it from TCP retransmission:

$$L_{peer} = p \cdot \delta_1 + (1-p) \cdot \delta_4.$$

The average delay in the first case of both schemes should be the same, so we take a closer look at the other cases. For simplicity, we let L' denote the latency not including Case (1), i.e.,

$$L'_{AFEC} = (1 - p - r) \cdot \delta_2 + r \cdot \delta_3$$

$$L'_{Peer} = (1-p) \cdot \delta_4.$$

In Case (2) for AFEC, a member didn't receive the required packet, but received at least k other packets to perform reconstruction. According to the spacing strategy for packet sending, it will take at least $\frac{k+1}{k+h}T_A$ time to send $k+1$ packets at the GCKS.

$$\delta_2 = 1/2(\frac{k+1}{k+h}+1)T_A + D_{mcast} = \frac{T_A}{2} \cdot \frac{2k+h+1}{k+h} + D_{mcast}.$$

In Case (3) of AFEC scheme, the member waits for awhile at the end of rekeying, then sends retransmission request to the GCKS. The waiting time Δ_A usually is the RTT. So the latency in this case is:

$$\delta_3 = T_A + \Delta_A + 2D_{tcp}(m, GCKS).$$

Therefore,

$$\begin{aligned} L'_{AFEC} &= (1-p-r)[\frac{T_A}{2}\cdot\frac{2k+h+1}{k+h}+D_{mcast}] \\ &\quad +r[T_A+\Delta_A+2D_{tcp}(m,GCKS)] \\ &= (\frac{1-p-r}{2}\cdot\frac{2k+h+1}{k+h}+r)T_A \\ &\quad +(1-p-r)D_{mcast}+r\cdot\Delta_A \\ &\quad +2r\cdot D_{tcp}(m,GCKS). \end{aligned}$$

In Case (4) for the peer recovery scheme, when the fast recovery initiation strategy is used, a member can detect the loss of required packet before all packets have been transmitted. The detection time Δ_P is just the time to receive a few packets after the lost required packet. The delay in this case is:

$$1/2T_P + D_{mcast} + \Delta_P + c\cdot 2D_{tcp}(m_i, m_j).$$

Therefore,

$$\begin{aligned} & L'_{Peer} \\ = \; & (1-p)[1/2T_P + D_{mcast} + \Delta_P + c\cdot 2D_{tcp}(m_i,m_j)] \\ = \; & \frac{1-p}{2}T_P + (1-p)D_{mcast} \\ & \quad +(1-p)\Delta_P + 2c(1-p)\cdot 2D_{tcp}(m_i,m_j). \end{aligned}$$

Because AFEC scheme will send more packets in a rekey event, T_A is larger than T_P. Given a small residual rate r, the second terms of L'_{AFEC} and L'_{Peer} will be similar. Δ_A is bigger than Δ_P because the time to receive a few more packets to initiate the recovery is less than the RTT. Based on the first three terms of the two expressions, we can expect that delays will be similar when losses are low. For members who fail to receive their required packet in the "initial" transmission (i.e. in the first k packets), we can expect that the peer-based approach will have lower latency, due to elimination of the need to wait until all multicast packets have been transmitted.

Finally, the magnitude of the "residual" D_{tcp} term for each method will depend heavily on the delay experienced by TCP, in particular due to TCP's congestion control mechanism. To characterize the effect of that term, and to verify the above analysis of the effect of the other terms on the distribution of latency, we resort to simulation.

5 Simulations

We compare the performance of our peer-based recovery scheme with that of AFEC using ns2 simulation [12]. We generated several transit-stub graphs with 2500 routers using the GT-ITM software [2]. Each graph consists of 5 transit domains, with each transit node having 7 stub domains attached; each domain has 10 (on average) nodes. The capacity of a link between two transit nodes is 100 Mbps. All other links have capacity 10 Mbps. The GCKS resides on one stub node. The rekey bandwidth is limited to 100 Kbps. The reason for a limiting rekey bandwidth is that rekey messages will share bandwidth with group data and we do not want rekey traffic to hurt the data traffic. The 100 Kbps limit was also used in the simulation by Zhang et al. [16]. Given the bandwidth constraint of rekey traffic at the GCKS, the time to send out all rekey packets is on the order of a few seconds. Group members are randomly scattered on remaining stub nodes. The group size varies from 512 to 2048. Initially, the key tree (with degree 4) is balanced with members. At each rekey event, $n/4$ members depart and $n/4$ members join, where n is the group size. The actual processing time of a rekey operation in AFEC—in particular the time taken to encode rekey packets—will depend on the computational capacity of the GCKS. For simplicity, we assume that capacity is adequate and the encoding takes zero time. The target number of NACKs in AFEC is set to slightly more than 1%: 6, 12, 18 and 24, for group sizes 512, 1024, 1536 and 2048, respectively.

In all simulations, we assume that losses are due entirely to congestion. We first test the scenario of light background traffic on the network. In this scenario, we assume that the traffic causes limited number of congested links and has no impact (other than longer queueing delay) on other links. We randomly select some links in the graph as bottleneck links[2]. For each selected bottleneck, we add 30 TCP flows and 5 UDP flows to cause congestion. On each TCP connection, an ns2-provided FTP application generates traffic. On each UDP flow, we use a traffic generator which generates traffic with an exponential inter-arrival distribution. The mean rate of each such exponential flow is 0.4 Mbps. The observed average loss rate of multicast packets on the bottleneck links is 27%.

Figures 5 to 7 show scatter plots of rekey latencies with each method, for 1024 group members when the graph has 20, 40, 100 congested links, respectively.

From the above results, we can see that, as expected, most members in our scheme receive the required rekey packets earlier than those in AFEC. The reason is twofold: First, members in our scheme detect the loss faster and initiate the retransmission process earlier. Second, it takes less time in our scheme to send all the rekey packets at the GCKS, because no redundant packets are transmitted. However, from the scatter plot we also find that occasionally some members may experience very long delay to get rekey packet. This is due to TCP congestion control behavior in the case of repeated packet losses.

Next, we introduce more background traffic to test the performance of our scheme. To save simulation running time, we introduce background traffic only on the links of the multicast tree. Every tree node has 20–30 incoming FTP flows and 20–30 outgoing FTP flows. Each source node is randomly connected to one destination node. To introduce some heterogeneity, we add extra UDP flows on between 100 and 200 tree nodes. Each of those nodes has five incoming and five outgoing Exponential traffic flows. The mean rate of each Exponential flow is 0.4 Mbps. How-

[2]Only transit-stub and intra-stub links are selected as a bottlenecks.

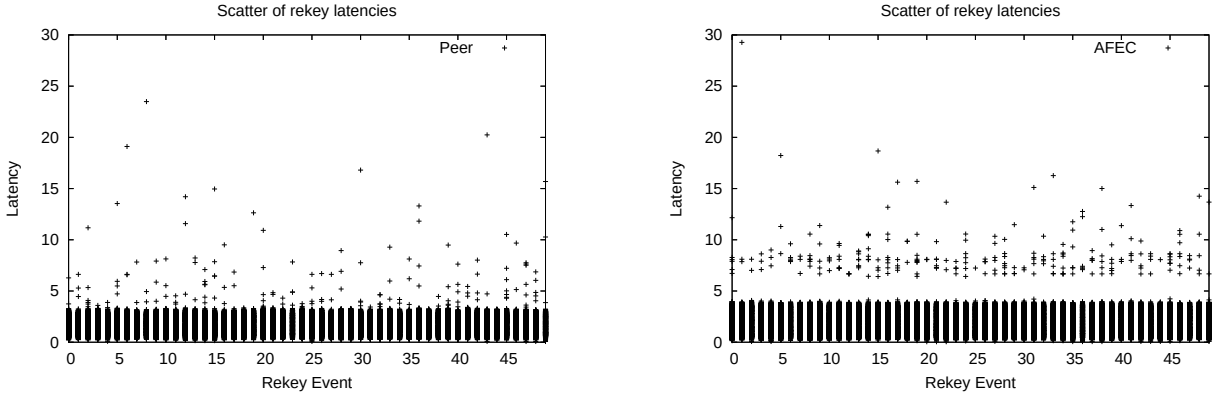

Figure 5: 20 bottlenecks

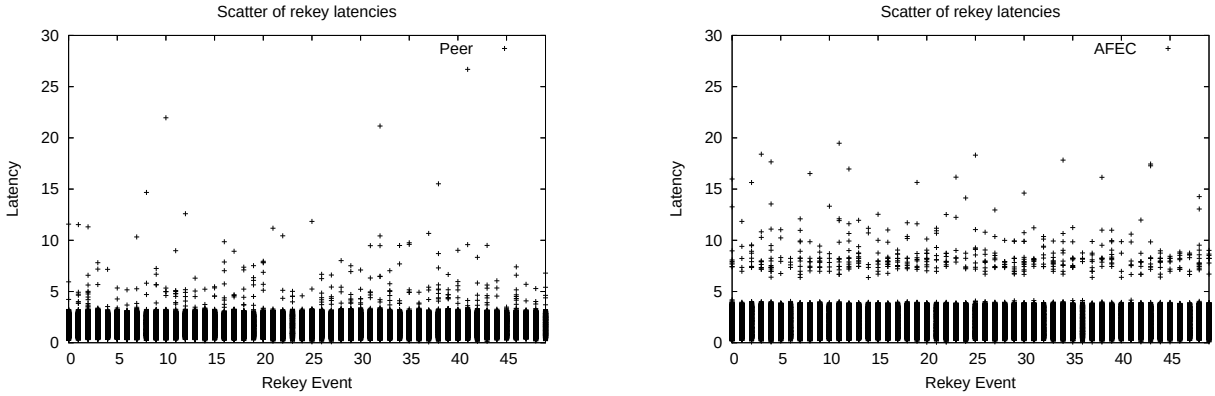

Figure 6: 40 bottlenecks

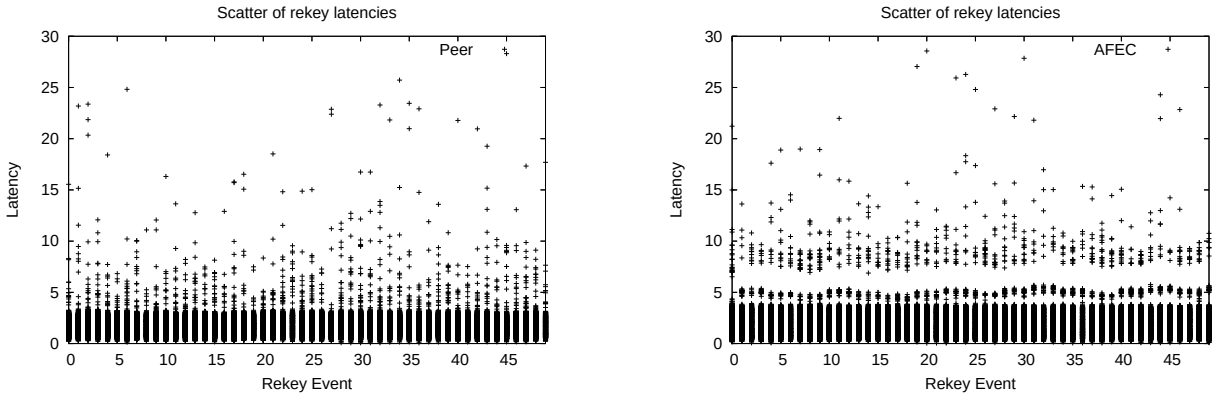

Figure 7: 100 bottlenecks

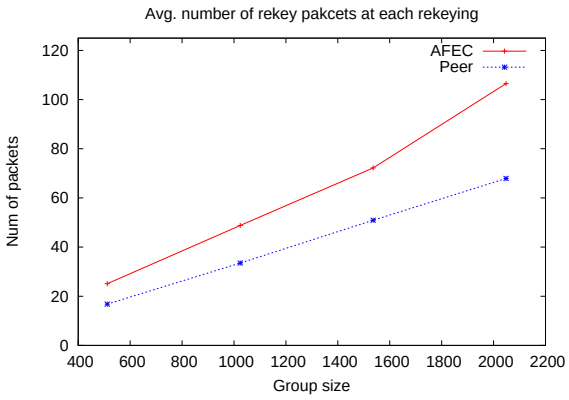

Figure 9: Avg num of rekey packets (ts2500-1)

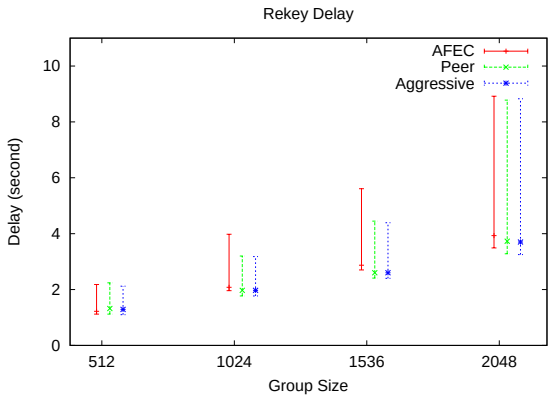

Figure 10: Rekey delay(ts2500-1)

Bottlenecks	AFEC			Peer-Recovery		
	avg delay	95%-ile	median	avg delay	95%-ile	median
20	2.0	3.5	1.9	1.8	2.9	1.8
40	2.0	3.5	1.9	1.8	2.9	1.8
100	2.1	3.6	1.9	1.8	2.9	1.8

Figure 8: 1024 group members with light background traffic

Group		AFEC	Peer	Aggressive-Peer
512	avg pkt	25.1	16.8	16.8
	avg delay	1.22	1.32	1.29
	95%-ile	2.18	2.24	2.12
	median	1.12	1.12	1.10
1024	avg pkt	48.8	33.5	33.5
	avg delay	2.08	1.97	1.97
	95%-ile	3.98	3.20	3.18
	median	1.96	1.77	1.77
1536	avg pkt	72.2	50.9	50.9
	avg delay	2.87	2.60	2.60
	95%-ile	5.61	4.45	4.39
	median	2.70	2.41	2.40
2048	avg pkt	106.5	67.9	67.9
	avg delay	3.93	3.73	3.70
	95%-ile	8.92	8.78	8.83
	median	3.49	3.28	3.25

Figure 11: Rekey delay (ts2500-1, more background traffic)

ever the node the GCKS is on has no background traffic. We run simulations on two graphs, ts2500-1 and ts2500-2. On both graphs, the end-to-end loss rate experienced by each user varies from 6% to 10 % at different group sizes. On both graphs, we test AFEC and two recovery strategies of peer-based recovery scheme.

We show our results in Figures 9–13. To show the numbers clearly, we also give the results of ts2500-1 in Figure 11. Because the results of ts2500-2 are similar, we omit the table. Figures 9 and 12 compare the average number of rekey packets transmitted in a rekey event between AFEC and peer-based recovery scheme. It is clear that AFEC scheme sends many more rekey packets than peer-based scheme. Figures 10 and 13 show the median, average and 95th-percentile of rekey latencies for different groups. The lowest point of each bar represents the median. Again, we find that most members in our scheme can receive required rekey packets earlier than those in AFEC: For larger group sizes, the median and 95th-percentile latencies are at least 10% lower for the peer-based recovery scheme. The figures also show that the "aggressive" request strategy offers very little improvement over the non-aggressive strategy. The reason is that most members can get required packet from the first peer contacted, so the aggressive strategy does not help too much.

6 Conclusions

We have presented a peer-based approach to reliability in group rekeying. Our approach is suitable for use whenever a logical tree structure is imposed on the group, and members who are "close" in the logical tree need to receive similar information during rekey events. The idea of the scheme is to have each member contact other members who are nearby in the logical tree (but—ideally—*not* nearby in the multicast tree) to obtain information that is missing due to loss of multicast packets.

The advantages of our approach are that it is simple, and it reduces the resource requirements at the GCKS by removing the need to process rekey messages using

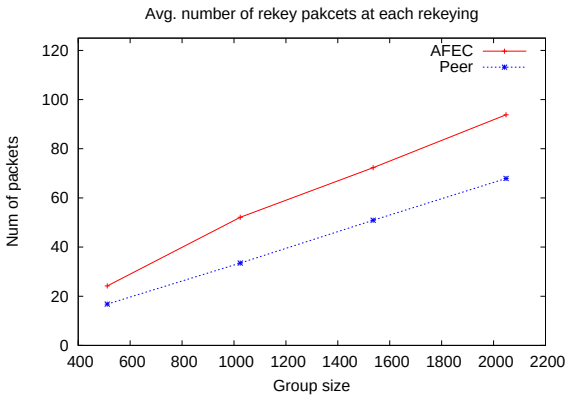

Figure 12: Avg. num of rekey packets (ts2500-2)

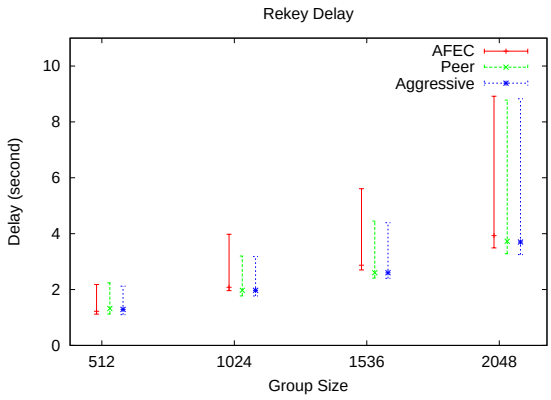

Figure 13: Rekey delay(ts2500-2)

Forward Error Control Coding. Analysis and simulation confirm that our approach also reduces the latency experienced by most receivers who need to recover packets. Reduced latency is important for reducing the likelihood that an authorized member will fail to have access to group data because of missing keys. The approach requires that members rely on other members to provide some rekey information, but does not require any additional authentication or other mechanisms beyond what were already required.

In future work, we will investigate methods of assigning members to the key tree so that nodes that are nearby in the key tree are far apart in the multicast tree.

Acknowledgements

The support of the US National Science Foundation under grant ANI-9977292 is gratefully acknowledged. The authors would also like to thank X. B. Zhang for sharing his ns2 simulation scripts with us.

References

[1] M. Baugher, R. Canetti, L. Dondeti, and F. Lindholm, *Multicast Security (MSEC) Group Key Management Architecture*, Request for Comments, draft-ietf-msec-gkmarch-08.txt, Apr. 2005.

[2] K. Calvert, M. Doar, and E. W. Zegura, "Modelling Internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160-163, June 1997.

[3] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical internet coordinates for distance estimation," in *International Conference on Distributed Systems*, Mar. 2004.

[4] P. Crescenzi and V. Kann, *A Compendium of NP Optimization Problems.* (http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html)

[5] S. E. Deering, "Multicast routing in internetworks and extended LANs," in *Symposium proceedings on Communications Architectures and Protocols*, pp. 55-64, Stanford, California, ACM Press, 1988.

[6] T. Hardjono and B. Weis, *The Multicast Group Security Architecture*, IETF, RFC 3740, Mar. 2004.

[7] H. Lim, J. Hou, and C. Choi, "Constructing Internet coordinate system based on delay measurement," in *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement*, pp. 129-142, Oct. 2003.

[8] D. A. McGrew and A. T. Sherman, *Key Establishment in Large Dynamic Groups Using One-Way Function Trees*, TIS Labs at Network Associates, Inc, Technical Report, no. 0755, Glenwood, MD, may 1998.

[9] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers," *Advances in Cryptology - CRYPTO'01*, LNCS 2139, pp. 41-62, 2001.

[10] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordinates-based approaches," in *Proceedings of IEEE InfoCom'02*, pp. 170-179, 2002.

[11] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," in *Proceedings of IEEE InfoCom'03*, Apr. 2003.

[12] VINT, *The Network Simulator ns2.* (http://www.isi.edu/nsnam/ns/)

[13] D. Wallner, E. Jarder, and R. Agee, *Key Management for Multicast: Issues and Architectures*, Internet-Draft, Sept. 1998.

[14] C. K. Wong, M. G. Gouda, and S. S. Lam, "Secure group communications using key graphs," in *Proceedings of the ACM SIGCOMM'98*, pp. 68-79, Sept. 1998.

[15] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam, "Reliable group rekeying: a performance analysis," in *Proceedings of the ACM SIGCOMM'01*, pp. 27-38, ACM Press, Aug. 2001.

[16] X. Zhang, S. Lam, and D. Lee, "Group rekeying with limited unicast recovery," *Computer Networks*, vol. 44, no. 6, pp. 855-870, Apr. 2004.

[17] X. Zhang, S. Lam, D. Lee, and Y. Yang, "Protocol design for scalable and reliable group rekeying," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 6, pp. 908-922, Dec. 2003.

Qingyu Zhang is a Ph. D. candidate in the Department of Computer Science at the University of Kentucky. She received her B. S. in computer science from Beijing Institute of Technology, China in 1997. Her research interests include secure communication protocols, multicast services, and overlay networks.

Kenneth L. Calvert is Associate Professor in the Department of Computer Science at the University of Kentucky. His research deals with the design and implementation of advanced network protocols and services. Current interests include lightweight network programmability, generalized routing, network security, and models of Internet topology. He holds a Ph. D. in Computer Science from the University of Texas at Austin.