Group Rekeying with Limited Unicast Recovery [*]

X. Brian Zhang, Simon S. Lam, and Dong-Young Lee
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{zxc, lam, dylee}@cs.utexas.edu

TR-02-36
July 2002

Abstract

In secure group communications, a key server can deliver a "group-oriented" rekey message [20] to a large number of users efficiently using IP multicast. For reliable delivery, Keystone [21] proposed the use of forward error correction (FEC) in an initial multicast, followed by the use of unicast delivery for users that cannot recover their new keys from the multicast. In this paper, we investigate how to limit unicast recovery to a small fraction r of the user population. By specifying a very small r, almost all users in the group will receive their new keys within a single multicast round.

We present analytic models for deriving r as a function of the amount of FEC redundant information (denoted by h) and the rekeying interval duration (denoted by T) for both Bernoulli and two-state Markov Chain loss models. From our analyses, we conclude that r decreases exponentially as h increases. We then present a protocol designed to adaptively adjust (h, T) to achieve a specified r. In particular, our protocol chooses from among all feasible (h, T) pairs one with h and T values close to their feasible minima. Our protocol also adapts to an increase in network traffic. Simulation results using ns-2 show that with network congestion our adaptive FEC protocol can still achieve a specified r by adjusting values of h and T.

[*]Research sponsored in part by NSF grant no. ANI-9977267.

1 Introduction

Many emerging Internet applications, such as pay-per-view distribution of digital media, restricted teleconferences, multi-party games, and virtual private networks, will benefit from using a secure group communications model [9]. In this model, members of a group share a symmetric key, called *group key*, which is known only to group users and the key server. The group key can be used for encrypting data traffic between group members or restricting access to resources intended for group members only. The group key is distributed by a group key management system which changes the group key from time to time (called group rekeying).

The design of a group key management system has had extensive research in recent years [18, 20, 8, 6, 3, 23, 5, 13, 11]. In particular, the key tree approach [18, 20] reduces the server processing time complexity of group rekeying from $O(N)$ to $O(\log_d(N))$ where N is group size and d the key tree degree. This approach was shown to be optimal in [17]. A key tree is a rooted tree with the group key as root [20]. There are two types of nodes: *u-nodes* containing users' individual keys, and *k-nodes* containing the group key and auxiliary keys. A user's individual key is shared only between this user and the key server. A user is given its individual key as well as keys contained in k-nodes on the path from its u-node to the root node. When a user joins or leaves the group, all keys on the path from the user's u-node to the root node should be changed. Rekeying after every join or leave request, however, can incur a large server processing overhead. Thus periodic batch rekeying was proposed to further reduce server processing overhead [16, 23, 10].

The key tree approach requires reliable delivery of new keys to users for group rekeying.[1] This is because the key server uses keys for one rekeying interval to encrypt new keys for the next rekeying interval. Each user however does not have to receive the entire rekey message because it needs only those new keys that are located on the path from the user's u-node to the root node (a very small subset of all new keys).

[1]To our knowledge, there are three approaches, namely, MARKS [5], TLK [13], and Subset-Difference [11] that do not require reliable delivery of a rekey message. However, each of them has its limitations. In particular, MARKS approach assumes the lifetime of each user to be pre-determined before it joins the system. TLK approach introduces "hint" information into each data packet to help users recover a new group key. Therefore, this approach introduces per-packet bandwidth overhead and also expensive computation overhead at the user side. In the Subset-Difference approach, each rekey message contains $2 \cdot e$ keys, where e is the total number of revoked users from the beginning of a session until now. Therefore e can become a very large value as the session proceeds.

For reliable delivery, [21, 23, 24] proposed the use of forward error correction (FEC) in an initial multicast, followed by the use of unicast delivery for users that cannot receive or recover their new keys from the multicast.

Each unicast packet contains encrypted keys for only one particular user. Thus, the size of a unicast packet is much smaller than that of the rekey message for the group. As a result, unicast recovery will not cause a problem at the server if the number of users who need unicast recovery is small.

In this paper we investigate how to limit unicast recovery to a small fraction r of the user population. To achieve low delay, we assume that there is only one multicast round.

With a very small r, we can achieve the following benefits. First we can significantly reduce the unicast traffic. Second, a small r can achieve low delivery latency for most users who receive or recover their new keys in a single multicast round. Last, a small r can reduce the data buffering overhead at the user side. This is because when a sending user (or the data server) uses the newly received group key to encrypt outgoing data, the users who have not yet received the new group key will have to buffer incoming encrypted data before the arrival of the new group key. If r is small, we expect that most users will receive the new group key at roughly the same time, and thus they incur only a very small buffering overhead.

To achieve a small r, we may need to increase the rekeying interval duration T. More specifically, to make r smaller, we need to increase h (the amount of FEC redundant information) and thus the number of packets in rekey traffic. To keep the sending rate of rekey traffic constant, we may have to increase T; otherwise, the sending rate of rekey traffic will increase and it may hurt the performance of other flows in the Internet [1, 2].

On the other hand, as a measure of the granularity of group access control, a small T is preferable. This is because all join and leave requests issued in the same rekeying interval are processed in a batch. Thus a new group key will not be generated and used until the end of each rekeying interval. As a result, a departed user can still read future data for up to T time units after it leaves the group. Hence, a small T is desirable to achieve tight access control.

In this paper, we investigate the tradeoffs between r, T and h. We present analytic models for deriving r as a function of T and h. We then design an adaptive FEC protocol to achieve a target value of r under dynamic network situations. Our protocol chooses from among all feasible (h, T) pairs one with h and T values close to their feasible minima. Simulation results from ns-2 show that our protocol can achieve fairly smooth traces of r when group rekeying is subjected to statistical fluctuations of a fixed set of competing flows. We also investigated the dynamic behavior of our protocol when the set of competing flows is increased. We found that with the onset of network congestion our adaptive FEC protocol can still achieve the target r by adjusting values of h and T.

The balance of the paper is organized as follows. In Section 2, we present the basic group rekeying protocol. In Section 3, we analyze the tradeoffs between r, T, and h. In Section 4, we design and evaluate our adaptive FEC protocol. Our conclusions are in Section 5.

2 An overview of group rekeying protocol

In this section, we give an overview of our group rekeying protocol. The key server protocol for one rekey message is as follows. (See [24] for a more detailed description.)

- At the end of each rekeying interval, the key server uses group-oriented rekeying strategy [20] to generate a rekey message. Each item in the message is an encrypted new key, called *encryption*. In the key tree approach, one user needs a particular encryption only when the encryption contains a key which is on the path from the user's u-node to the root node.

- The key server divides the rekey message into rekey packets. Our packet generation algorithm guarantees that all of the encryptions needed by any user will be contained in a single packet. We refer the interested readers to [24] for a detailed discussion of our packet generation algorithm.

- The key server partitions the packets into multiple blocks. Each block contains k packets. [2] We call k the block size. The key server then generates h parity packets for each block using a Reed-Solomon Erasure (RSE) coder [14].

- The key server multicasts k rekey packets and h parity packets for each block within the next rekeying interval.

- The key server collects NACKs from users, and adjusts the value of h and T for the next rekey message according to the number of NACKs received.

[2]To form the last block, the key sever may need to duplicate the packets in the last block until there are k packets.

- The key server switches to unicast recovery for users that sent NACKs. For each such user, the key server sends a single unicast packet containing encryptions needed by the user. Since it takes time for the key server to receive NACKs, unicast recovery for one rekey message has to be executed in later rekeying intervals, concurrently with the multicast of subsequent rekey messages.

At the user side, following a timeout, a user checks whether it has received or can recover its required encryptions. A user can recover its required encryptions in any one of the following three cases: 1) The user receives the specific rekey packet which contains the user's encryptions. 2) The user receives at least k packets from the block that contains its specific rekey packet, and thus the user can recover the k original rekey packets. 3) The user receives a unicast packet during subsequent unicast recovery. The unicast packet contains all of the encryptions needed by the user.

If the user cannot recover its required encryptions, it will report a NACK to the key server. The NACK specifies the number of packets needed by this user to recover its block. By the property of Reed-Solomon encoding, this value equals to k minus the number of packets received in the block containing its specific rekey packet. This information is needed by the key server's adaptive FEC scheme.

3 Analyses

In our group rekeying protocol, we care about two performance metrics: r and T.

Metric r measures the percentage of users who cannot receive the new group key on time. Our protocol uses FEC scheme to send $k + h$ packets for each block within rekeying interval T, so that most users can receive or recover their required encryptions within a single multicast round. We call the (expected) percentage of users who cannot receive or recover their required encryptions during multicast as residual error rate r. For those users, the key server will use unicast to deliver their required encryptions to them. However, those users have to buffer incoming data packets which are encrypted by the new group key until they receive the new keys. Hence, a small r is usually preferable in terms of reducing the buffering overhead at the user side as well as reducing the key server's unicast traffic.

Metric T is a performance measure of the group access control granularity. In periodic batch rekeying a new group key will not be generated and used until the end of rekeying interval. As a result, a departed user can still read future data for up to T time units after it leaves the group. Hence, a small T is desirable to achieve tight access control.

In summary, ideally we want to achieve both small r and small T; however, these two goals usually conflict with each other. For example, in order to achieve a smaller r, the key server needs to increase h. To send all of the $k+h$ packets for each block within rekeying interval T, the key server may have to increase T. Otherwise, the rekey traffic may hurt the performance of other flows in the Internet [1, 2].

In this section, we will investigate the tradeoffs between r, T, and h. To do it, we will first analyze r as a function of h and T, and then investigate the impact of rekeying bandwidth constraint on the relationships among h, T and r.

3.1 Analytic models

In this subsection, we will analyze r as a function of h and T, denoted by $r = f(h, T)$. Both Bernoulli and Markov loss models are considered. For simplicity of analyses, we assume that different users experience independent and homogeneous (same loss parameters) losses. Under this assumption, r equals to the probability that a user cannot receive or recover its required encryptions during multicast. For purpose of simplicity, we still call this probability residual error rate.

3.1.1 Bernoulli model for independent loss

We now derive the expression of $r = f(h, T)$ for the case that T is large, that is $f(h, \infty)$. In this case, packets sent consecutively can be spaced widely enough such that they will likely experience independent losses. Temporally independent losses can be simulated by Bernoulli loss model. In particular, letting p denote the packet loss rate seen by each user, we have

$$\begin{aligned} r &= f(h,\infty) && (1)\\ &= p \cdot \sum_{i=0}^{k-1} \binom{k+h-1}{i} (1-p)^i p^{k+h-1-i} && (2)\\ &= p^{k+h} \cdot \sum_{i=0}^{k-1} \binom{k+h-1}{i} (\frac{1}{p}-1)^i && (3) \end{aligned}$$

where k is the block size. Intuitively, r equals to the probability that the user does not receive its specific rekey packet, and it receives less than k packets from the block that contains its specific rekey packet.

From Equation 3, we observe that for fixed k and p, $\sum_{i=0}^{k-1} \left(\begin{array}{c} k+h-1 \\ i \end{array} \right) (\frac{1}{p} - 1)^i$ is a polynomial of h with a degree of $k-1$. Letting $\mathcal{P}^{k-1}(h)$ denote this polynomial, we have

$$r = p^{k+h} \cdot \mathcal{P}^{k-1}(h) \tag{4}$$

From this expression, we can see that the effects of h on r come from the product of two factors: p^{k+h} and $\mathcal{P}^{k-1}(h)$. When h increases, the first factor p^{k+h} decreases exponentially, while the second factor $\mathcal{P}^{k-1}(h)$ increases as a polynomial of h. Since the first factor changes at a faster rate than the second, we expect that increasing h will sharply reduce r.

3.1.2 Markov model for burst loss

In the subsection above, we consider the case that T is large, and thus derive $r = f(h, \infty)$ based on Bernoulli loss model. If T is small, however, packets sent within interval T will likely experience temporally dependent losses. To derive $r = f(h, T)$ for small T, we apply a two-state Markov chain model used in [12, 4] to investigate the correlated losses between consecutive packets.

In the Markov loss model, we use a two state continuous time Markov chain $\{X_t\} \in \{0, 1\}$ to describe the packet losses. A packet transmitted at time t is lost if $\{X_t\} = 1$ and not lost if $\{X_t\} = 0$. The generation matrix of this Markov chain is

$$\mathcal{Q} = \left[\begin{array}{cc} -\mu_0 & \mu_0 \\ \mu_1 & -\mu_1 \end{array} \right]$$

Its rate transition diagram is shown in Figure 1.

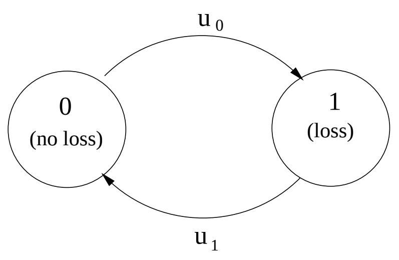

Figure 1: Transition diagram of the two state Markov chain

Let π_i, $i = 0, 1$, be the stationary distribution at state i of this Markov chain. Let $p_{i,j}(\tau)$ denote the probability that the process is in state j at time $t+\tau$ given that it was in state i at time t. That is $p_{i,j}(\tau) = P(X_{t+\tau} = j | X_t = i)$. Then we have $\pi_0 = \mu_1/(\mu_0+\mu_1)$, $\pi_1 = \mu_0/(\mu_0+\mu_1)$, and

$$\begin{aligned} p_{1,1}(\tau) &= (\mu_0 + \mu_1 \cdot \exp(-(\mu_0+\mu_1)\tau))/(\mu_0+\mu_1) && (5) \\ &= \pi_1 + \pi_0 \cdot \exp(-(\mu_0+\mu_1)\tau) && (6) \end{aligned}$$

Before deriving the expression of $r = f(h, T)$, we first need to figure out how to space packets when they are sent out within rekeying interval T. It is well known that residual error rate is sensitive to the packet spacing under burst losses [12]. Therefore, we are concerned with how to space packets so as to minimize r while they are sent out within interval T.[3]

To answer this question, we observe that in our group rekeying protocol, a particular user needs packets only from the block to which its specific rekey packet belongs. Therefore, we consider the case that the key server sends only $k+h$ packets within interval T to a particular user. We are concerned with how to space the $k+h$ packets so as to minimize the residual error rate for this user.

Let τ_i denote the interval between the times at which the i^{th} and $(i+1)^{th}$ packets are sent, $i = 1, ..., k+h-1$. Those packets tend to experience burst losses when T is small. We assume that the probability of more than one burst loss duration happening within a small interval T is low. Suppose that the specific rekey packet that this user requires is at the m^{th} position, $m = 1, ..., k$. Given that the loss duration starts from the j^{th} packet, where $j = 1, ..., m$ and $j+h \geq m$, we can derive the conditioned residual error rate for this user as

$$\begin{aligned} r_{m,j} &= \pi_1 \cdot \prod_{i=j}^{j+h-1} p_{1,1}(\tau_i) && (7) \\ &= \pi_1 \cdot \prod_{i=j}^{j+h-1} (\pi_1 + \pi_0 \cdot \exp(-(\mu_0+\mu_1)\tau_i)) && (8) \end{aligned}$$

where the right side expression represents the probability that the j^{th} packet and the following h packets are lost.

[3] [4] investigated another case of this problem in a similar way. The optimization problem in [4] was presented for a unicast telephony application, and the paper maximized the probability that at least one packet out of k packets is received. However, in our multicast based group rekeying protocol, each user needs to receive its specific rekey packet, or receive at least k packets out of the $k+h$ packets from the block to which its specific rekey packet belongs.

From Equation 8 we observe that condition probability $r_{m,j}$ is a decreasing function of τ_i for $j \leq i < j+h$. To minimize $r_{m,j}$, we should have $\tau_i = 0$ for $i < j$ or $i \geq j+h$ since $\sum_{i=1}^{k+h-1} \tau_i = T$. Then the desired values of $\{\tau_i \mid j \leq i < j+h\}$ should be the solution to the following optimization problem:

$$\begin{array}{ll} \text{minimize} & \pi_1 \cdot \prod_{i=j}^{j+h-1} (\pi_1 + \pi_0 \cdot \exp(-(\mu_0 + \mu_1)\tau_i)) \\ \text{subject to} & \sum_{i=j}^{j+h-1} \tau_i = T \end{array}$$

Solving it using the standard Lagrange multipliers method, we get $\tau_j = \tau_{j+1} = ... = \tau_{j+h-1}$. Therefore, to minimize $r_{m,j}$ for a particular user, the packets should be equally spaced.

Our eventual goal is to minimize the residual error rate for each user without conditioning on the start point of the loss duration. We observe that different users may need different specific rekey packet, and loss duration may start from different packets. That is, m and j may vary from user to user.[4] To minimize the sum of residual error rates for all users, we argue that we should have $\tau_i = \tau_j, \forall i, j = 1, ..., k+h-1$.

Thus we get our packet spacing strategy as follows. If the rekey message consists of only one block of packets, all the packets should be equally spaced in rekeying interval $[t,\ t+T]$ including both endpoints. If the key server has multiple blocks to send, the packets belonging to the same block should be equally spaced. However, the packets from different blocks should be sent in an interleaved fashion. That is, the i^{th} packets from each blocks should be sent together without spacing. Though these packets will likely experience burst losses, it is harmless since each particular user needs packet only from one specific block. Figure 2 illustrates how the key server should space packets when a rekey message is divided into three blocks, and each block contains $k+h=4$ rekey and parity packets.

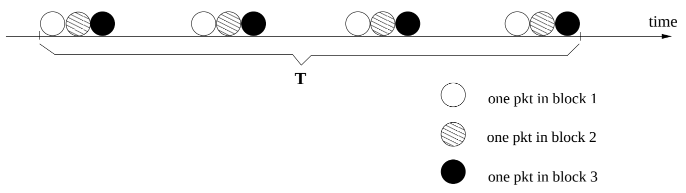

Figure 2: Spacing between packets

[4]In our packet generation algorithm presented in [24], each rekey packet contains encryptions for roughly equal number of users.

With equal spacing between packets, we can derive a lower bound of r by assuming that at most one loss duration happens during rekeying interval T, as follows:

$$\begin{aligned} r &= f(h,T) & (9) \\ &= \pi_1 \cdot \exp(-\mu_1 \cdot h \cdot \tau) & (10) \\ &= \pi_1 \cdot \exp(-\frac{\mu_1 \cdot h \cdot T}{h+k-1}) & (11) \end{aligned}$$

where $\exp(-\mu_1 \cdot h \cdot \tau)$ is the probability that the loss duration last for at least $h \cdot \tau$ time interval. Factor τ here is the spacing interval between two consecutive packets of the same block. Roughly speaking, τ equals to $T/(h+k-1)$ if we ignore packet transmission time.

3.1.3 Illustration

We now illustrate the function $r = f(h,T)$ with numerical and simulation results. We set $p = 0.06$ for our Bernoulli loss model, and $\mu_0 = 0.75$ and $\mu_1 = 11.75$ (thus $\pi_1 = 0.06$) for the two-state Markov model.

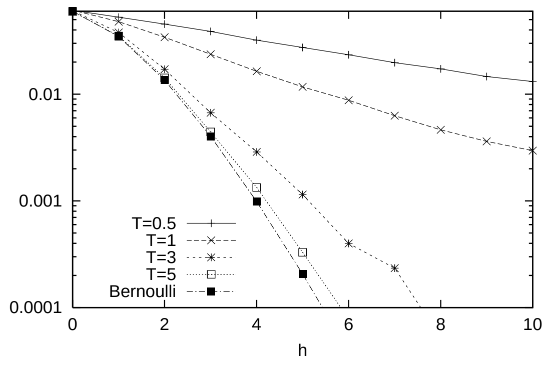

Figure 3: r as a function of h

Figure 3 shows the value of r as a function of h. The figure contains five curves. One is based on our Bernoulli loss formula (Equation 3), and the remaining four are based on our Markov loss model for different values of T , that is $T = 0.5$, 1 , 3 and 5 second(s). For the Markov loss model, we use simulations to compute the value of r for various h and T. Each point in the four curves is the average value based on 100 trials. As can be seen from the figure, as h increases, r decreases roughly linearly on a logarithmic scale. We also observe that when T is small, packets sent consecutively will likely experience burst losses, and thus our Markov model generates larger r than the Bernoulli model. When T increases, the curve of r produced by the Markov model will gradually approaches that of the Bernoulli model.

From now on, given (h, T), we use the larger value produced by Equation 3 and 11 to approximate the actual r. The value we choose is still a lower bound. However, for the evaluation in this section, the conclusions we draw based on the lower bound will usually hold for the actual r. Furthermore, we expect the value we choose will be close to the actual r if T is not very small.

3.2 Rekeying bandwidth constraint

In the previous subsections, we analyze $r = f(h, T)$ under our Bernoulli and Markov loss models. During our derivations, we assume that h and T are independent variables. However, the relationship between h and T should be constrained because the available rekeying bandwidth is usually very limited.

Rekeying bandwidth constraint requires that rekey traffic should not exceed a given sending rate $b(t)$ at any time t. This constraint arises from the fact that rekey traffic has to share bandwidth with data traffic, while the total available bandwidth is determined by the network situations and users' receiving capacities. For example, in secure group communication applications such as pay-per-view distribution of digital media, restricted teleconferences, and multi-party games, there typically exists a considerable amount of data traffic among group users. The data traffic competes for bandwidth with rekey traffic. Therefore, usually only a small percentage of total available bandwidth can be allocated to group rekeying. Let $b(t)$ denote the allowed sending rate for rekey messages at time t. In the literature, there are extensive research results on how to determine the unicast/multicast sending rate in dynamic network situations. We refer interested readers to related papers such as [7, 22, 19, 15]. In this paper, we assume that $b(t)$ is a given system parameter.

We claim that $b(t)$ will not sharply change with time t. The total available bandwidth shared by data and rekey traffic is a dynamic function of time. However we can adjust the rate of data traffic to keep $b(t)$ smooth. From now on, we assume that $b(t)$ is constant for the duration of a rekeying interval.

Before formulating the rekeying bandwidth constraint, we introduce some notation. Let n be the number of users in the system, s_m be the length of a multicast packet, s_u be the unicast packet length, n_b be the number of blocks in a rekey message,[5] and w be the expected number of unicast transmissions/retransmissions in order to deliver a unicast packet to a user.

Then at the key server side, the multicast traffic (per rekey message) equals to

$$BW_m(h) = (k + h) \cdot n_b \cdot s_m \tag{12}$$

After multicast, there are about $n \cdot r$ users who cannot receive or recover their required encryptions. The key server sends unicast packets to them to provide eventual reliability. The key server's unicast traffic (per rekey message) is

$$BW_u(r) = n \cdot r \cdot w \cdot s_u \tag{13}$$

In summary, our rekeying bandwidth constraint can be formulated as[6]

$$BW_m(h) + BW_u(r) \leq T \cdot b(t) \tag{14}$$

We now evaluate how h and T affect the amount of rekey traffic, which equals to $BW_m(h) + BW_u(r)$. As a concrete example, suppose that at the beginning of each rekeying interval the key tree (with degree 4) is balanced with 768 users. During each rekeying interval, 192 join and 192 leave requests are processed. We further assume that the leave requests are uniformly distributed over the users. We set the length of a multicast packet as 1005 bytes (including UDP and IP header sizes). The length of a unicast packet is 132 bytes. This is determined by the height of the key tree. We set block size $k = 14$ and unicast retransmission factor $w = 1.3$. We refer interested readers to [24] for a detailed discussion on how to determine packet length and block size.

Figure 4 illustrates the rekey traffic (in units of bytes per rekey message) as a function of h and T (in units of seconds). As can be seen, as a function of T, rekey traffic is very large for small T, and then it decreases and keeps flat as T increases. This is explained by the fact that burst losses produce large r when T is small. Large r requires large unicast traffic. On the other hand,

[5]n_b is in fact a function of rekeying interval T; however, it can be treated as a given value while the rekeying bandwidth constraint is formulated. In particular, at the end of one rekeying interval T_i, the key server generates a rekey message which has n_b (a function of T_i) blocks. This rekey message will be sent out within the next rekeying interval T_{i+1}. As far as the rekeying interval T_{i+1} is concerned, n_b is a given value.

[6]Since it takes time for the key server to receive NACKs, unicast recovery for one rekey message has to be executed later. However, as long as Inequality 14 holds for each rekeying interval, the rekey traffic will not exceed allocated bandwidth over a long term.

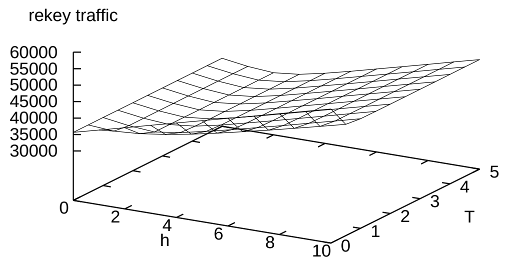

Figure 4: Rekey traffic (bytes per rekey message) as a function of h and T (seconds)

as a function of h, rekey traffic first decreases and then increases linearly when h increases. This is because when h is small, unicast traffic produced by large r will dominate the overall rekey traffic. When h increases, unicast traffic will sharply decrease and eventually diminish. And then multicast traffic will begin to dominate and increase as a linear function of h.

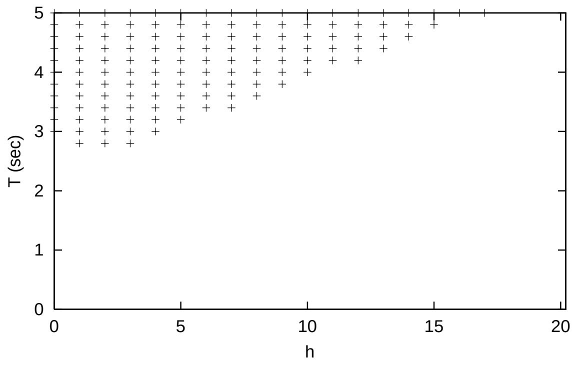

Figure 5: Feasible (h, T) pairs for $b(t) = 100$ Kbps

We next investigate the impact of the rekeying bandwidth constraint on the relationship between h and T. As a concrete example, we set $b(t) = 100$ Kbps. Because of the rekeying bandwidth constraint, we expect that some (h, T) pairs will violate the constraint. Figure 5 shows (h, T) pairs which satisfy the rekeying bandwidth constraint. We observe that when T or h is small, the bandwidth constraint will not be satisfied because of high unicast traffic, as implied by Figure 4. Furthermore, large h is not allowed either since it produces high multicast traffic. From Figure 5 we can draw another important conclusion. That is, T has to be in the order of seconds to satisfy the rekeying bandwidth constraint under the configuration of this example.

Based on this observation, we predict that our FEC scheme will be very effective for our group rekeying protocol because packets of the same block tend to experience independent losses. This is justified by two aspects. First we note that T cannot be very small because of the rekeying bandwidth constraint, as implied by Figure 5. Second, each particular user needs packets only from one specific block. Therefore, when $k+h$ packets of the same block are equally spaced within interval T, two consecutive packets may have a low probability of experiencing the same burst loss duration.

3.3 Tradeoffs between r, T and h

Up to now, we have analyzed $r = f(h, T)$ under our Bernoulli and Markov loss models, and also quantified the impact of rekeying bandwidth constraint on the relationship between h and T. We are now ready to investigate the tradeoffs between r, T and h.

As we know, r is the percentage of users who cannot receive the new group key on time, while T measures the group access control granularity. Small values of r and T are preferable. However, achieving a small r and a small T are conflicting goals.

In practice, we would like to give higher priority to r than to T. This is because r is directly related to the performance seen by each user. For this purpose, we specify a target residual error rate (denoted by r^*) as a system parameter. We aim to make sure that current (h, T) satisfies $f(h, T) \leq r^*$ as well as the rekeying bandwidth constraint. As a concrete example, we set $b(t) = 100$ Kbps and $r^* = 5/n$, where $n = 768$. Figure 6 shows feasible (h, T) pairs which satisfy $f(h, T) \leq r^*$ as well as the rekeying bandwidth constraint.

Among all feasible (h, T) pairs for a given r^*, the one with the smallest T is preferred. Let $T^* = \min\{T \mid \exists h,\ s.t.\ f(h,T) \leq r^*, BW_m(h) + BW_u(f(h,T)) \leq T \cdot b(t)\}$, $h^* = \min\{h \mid f(h,T^*) \leq r^*, BW_m(h) + BW_u(f(h,T^*)) \leq T^* \cdot b(t)\}$. Figure 7 and 8 illustrate the values of T^* and h^* for various r^*. (The curves for h' and T' will be introduced later.) First consider h^*. From Figure 7 we observe that when r^* decreases from 0.1 to 0.0001 on a logarithmic scale, h^* increases roughly at a linear speed. This confirms that our FEC scheme is very effective in reducing r. We next examine T^* as a function of r^*, as shown in Figure 8. Recall that when r^* decreases from a

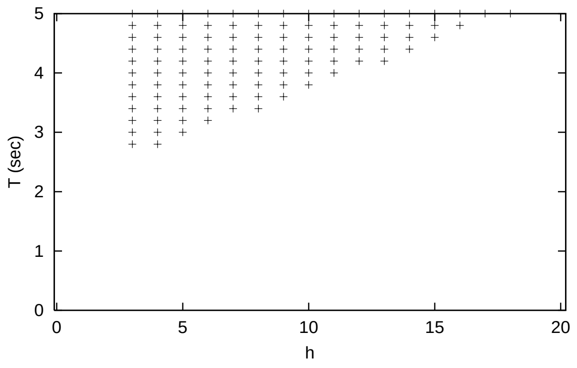

Figure 6: Feasible (h, T) pairs for $r^* = 5/768$ and $b(t) = 100$ Kbps

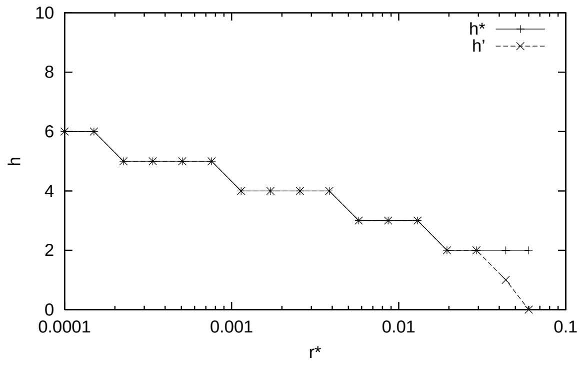

Figure 7: h^* and h' as functions of r^*

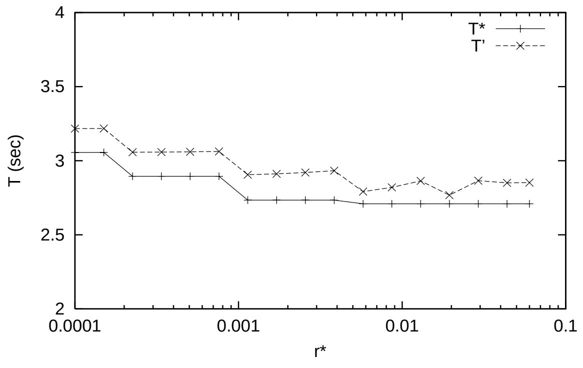

Figure 8: T^* and T' as functions of r^*

large value, unicast traffic will decrease significantly. The decrease of unicast traffic will balance the increase of multicast traffic. As a result, the curve of T^* keeps flat when r^* decreases from 0.1 to 0.001. When r further decreases, unicast traffic diminishes and multicast traffic will dominate. Consequently, T^* will increase at a similar speed as h^*. A direct conclusion from Figure 8 is that we can achieve a very small r without significantly increasing h and T.

3.4 Further discussions

In our previous analyses, we assume that loss parameters p, μ_0 and μ_1 are not functions of h and T. Then from Equation 3 and 11, we conclude that r can be made as small as possible by increasing h and T.

This conclusion seems to conflict with previous research results such as [1, 2], which observed that large FEC traffic may increase residual error rate at receivers. However, a further investigation will resolve the discrepancies. In the FEC schemes investigated in [1, 2], the sender sends $k + h$ packets for each block within fixed time interval T. Therefore, the traffic rate will increase proportionally if h is increased. When h is too large, FEC traffic will eventually overflow router buffers, and thus cause congestion. On the other hand, in our group rekeying protocol, the rate of rekey traffic is constrained by $b(t)$. In principle, the value of $b(t)$ should be updated over time and reflect up-to-date network situations. However, since rekey traffic is usually much smaller than data traffic, we expect that our rekey traffic will not hurt network performance as long as we update $b(t)$ in a smooth manner.

4 Adaptive FEC Protocol

From our analyses in Section 3, we observe that our FEC scheme is very effective in reducing r. As a result, we can achieve a very small r without significantly increasing h and T. In this section, we will discuss how to determine (h^*, T^*) for any specified r^* in a dynamic network environment.

4.1 Foundation

In practice, it seems hard to find the exact T^* and h^* for a specified r^*. This is because the general form of $f(h, T)$ is usually unknown. In particular, the expression of $f(h, T)$ depends on network loss situations as well as network topology. Therefore, it is desirable to find a near-optimal pair in a simple and efficient way.

Theorem 1 shows how to find a feasible pair (h', T') which is close to (h^*, T^*).

Theorem 1 *Given that $f(h, T)$ is a non-increasing function of h and T, let (h', T') be a solution to the following set of inequalities,*

$$BW_m(h) + BW_u(r^*) = T \cdot b(t) \tag{15}$$

$$f(h, T) \le r^* \tag{16}$$

$$f(h-1, T) > r^* \quad for \quad h > 0 \tag{17}$$

then we have $h' \le h^*$ and $T' - T^* \le \frac{BW_u(r^*) - BW_u(f(h^*, T^*))}{b(t)}$.

Proof: We first prove $h' \le h^*$. Suppose $h' > h^*$. Then we have $h' - 1 \ge h^*$, and also $T' \ge T^*$ by the definition of T^*. Thus we have $f(h'-1, T') \le f(h^*, T^*)$ since $f(h, T)$ is a non-increasing function of T and h. Therefore we have $f(h^*, T^*) > r^*$ by Inequality 17. This contradicts the definitions of h^* and T^*.

We next prove $T' - T^* \le \frac{BW_u(r^*) - BW_u(f(h^*, T^*))}{b(t)}$. In our group rekeying protocol, multicast traffic is an increasing function of h. Hence by Equation 15 we have

$$\begin{aligned} T' \cdot b(t) &= BW_m(h') + BW_u(r^*) \\ &\le BW_m(h^*) + BW_u(r^*) \\ &= (BW_m(h^*) + BW_u(f(h^*, T^*))) + \\ &\quad\ (BW_u(r^*) - BW_u(f(h^*, T^*))) \\ &\le T^* \cdot b(t) + BW_u(r^*) - BW_u(f(h^*, T^*)) \end{aligned}$$

Therefore, we have $T' - T^* \le \frac{BW_u(r^*) - BW_u(f(h^*, T^*))}{b(t)}$. $\square$

We expect that the difference between T' and T^* is very small in practice. By Theorem 1, we know that it is bounded by $\frac{BW_u(r^*) - BW_u(f(h^*, T^*))}{b(t)} = \frac{n \cdot w \cdot s_u \cdot (r^* - f(h^*, T^*))}{b(t)}$. This bound will be close to 0 if r^* is small enough. To see it, we first notice that the length of a unicast packet (denoted by s_u) is usually very small since each unicast packet contains encryptions only for one particular user. Second, we expect that $f(h^*, T^*)$ will be close to r^* when r^* is small. Figure 7 and 8 compare the values of h' with h^* and T' with T^* for various r^*. The numerical results are based on the loss models defined in Section 3. From the figures, we observe that h' is the same as h^* for a large range of r^*, and the difference between T' and T^* is very small.

4.2 Adaptation scheme

4.2.1 Framework of our scheme

Based on Theorem 1, we begin to design an iterative algorithm to find (h', T') for any specified r^*. Recall that r measures the percentage of users who send NACKs. The number of users in the system is usually a dynamic variable. Therefore, instead of specifying a fixed r^* , we define a target number of NACKs as our system parameter. Let u^* denote the target number of NACKs.

In fact, the number of NACKs directly reflects the residual error rate. Given an (h, T) pair, the number of NACKs returned to the key server is a random variable. Let U denote this random variable, and $E(U)$ be its expectation. Assuming different users have independent and homogeneous losses, we have

$$\mathbf{P}\{U = u\} = \left(\begin{array}{c} n \\ u \end{array} \right) r^u (1-r)^{n-u} \tag{18}$$

$$E(U) = n \cdot r \tag{19}$$

Therefore, we can think that $u^* = n \cdot r^*$ if n is a constant.

The framework of our adaptation scheme is as follows:

```
• if $E(U) > u^*$
      then $h \leftarrow h + \Delta h$
           $T \leftarrow \frac{BW_m(h)+BW_u(u^*/n)}{b(t)}$
• if $E(U) \leq u^*$
      then $h \leftarrow \max(h-1, 0)$
           $T \leftarrow \frac{BW_m(h)+BW_u(u^*/n)}{b(t)}$
```

The beauty of this scheme lies in the fact that it does not require knowledge of the mathematical expression for $r = f(h, T)$.

This scheme works as follows. If $E(U) > u^*$ (and thus $r > r^*$), the key server increases h by a certain value, denoted by Δh, so that hopefully Inequality 16 will hold for future rekey messages. On the other hand, if $E(U) \leq u^*$ (and thus $r \leq r^*$), the key server will reduce h by 1 to make sure that current h satisfies Inequality 17. At any time, whenever h is updated, T will be updated according to Equation 15. Finally the values of h and T will be around h' and T'. Now the remaining issues are how to determine Δh, and how to tell whether $E(U) > u^*$ or $E(U) \leq u^*$.

4.2.2 When to update h

We first consider when the key server should increase h in our adaptive FEC scheme. From the framework above, we know that the key server should increase h if $E(U) > u^*$. To estimate $E(U)$, it seems that the key server should collect a large number of sample values of U from consecutive rekey messages. However, this will significantly slow down the system's responsiveness to sudden network congestion, and thus may cause poor performance in terms of r metric. On the other hand, a hasty estimation of $E(U)$ may let the key server increase h unnecessarily, and thus hurt the system performance in terms of T metric. (Recall that T will increase proportionally with h in our framework above.) As a tradeoff, we argue that r metric may be more important than T metric. Therefore, it is desired for our protocol to have quick responsiveness to network congestion.

To achieve quick responsiveness to network congestion, the key server will decide whether to increase h by checking the number of NACKs (denoted by u) for the corresponding rekey message. In particular, the key server will increase h whenever $u > u_\alpha$, where u_α is defined as $\mathbf{P}\{U > u_\alpha\} \leq 1 - \alpha$ by assuming $E(U) = u^*$. Confidence level α can be specified by the owner of the key server. In this way, whenever event $u > u_\alpha$ happens, we have a confidence level of α to tell that $E(U) \neq u^*$ (and thus $E(U) < u^*$ possibly). To derive u_α, we notice that random variable U follows binomial distribution with parameters (n, r), as shown in Equation 18. Binomial distribution can be approximated as normal distribution when n is large. In particular, $\frac{U-nr}{\sqrt{nr(1-r)}}$ can be approximated as a standard normal random variable. Then for any specified α, we can derive u_α by solving $\mathbf{P}\{\frac{U-nr}{\sqrt{nr(1-r)}} \leq u_\alpha\} = \alpha$ and $n \cdot r = u^*$. For example, letting $\alpha = 99.9\%$ and $n = 768$, we have $u_\alpha = 11.9$, 19.7 and 33.6 for $u^* = 5$, 10 and 20 respectively.

We next consider when the key server should decrease h. An inappropriate decreasing of h may significantly increase the number of NACKs. Therefore, it is desired to measure $E(U)$ based on several rekey messages before the key server decides to reduce h. We use exponentially weighted average of u to approximate $E(U)$. Let $\bar{u}$ be the estimate value of $E(U)$. The key server executes $\bar{u} \leftarrow v \cdot \bar{u} + (1 - v)u$ whenever a new sample value u is available for current (h, T) pair. In our simulations, we use $v = 0.8$ and the average value should be based on at least three sample values for each updated (h, T) pair.

We further specify a lower threshold (denoted by h_l) on h. That is, the value of h should be larger than or equal to h_l at any time. This prevents the key server from reducing h to a very small value because of inaccurate estimation of $E(U)$. In fact, as can be seen from Figure 7, it is possible for h' to reach 0 where h^* is 2. Given u^*, h_l can be determined by

$$h_l = \min\{h \mid p \cdot \sum_{i=0}^{k-1} \left(\begin{array}{c} k+h-1 \\ i \end{array} \right) (1-p)^i p^{k+h-1-i} \leq u^*/n\}$$

where the value of p can be chosen based on experience. Intuitively, h_l is the minimum h which makes the value of r computed by our Bernoulli loss formula (Equation 3) no less than $r^* = u^*/n$. Here we consider only Bernoulli loss model since packets of the same block will likely experience independent losses, as observed in Section 3. However, the Markov loss model can also be considered if T is very small.

In our simulations, we set $p = 6\%$, and then derive $h_l = 3$, 3 and 2 for $u^* = 5$, 10 and 20 respectively.

4.2.3 Determining Δh

From our previous discussions, we know that the key server should increase h by Δh whenever $u > u_\alpha$. We now investigate how to determine the value of Δh.

We use a heuristic to determine Δh as follows. After multicast, the key server collects NACKs from users. Each NACK specifies the number of parity packets needed by a user in order to recover its required encryptions. Let a^i be the i^{th} (i starting from 1) largest one among collected NACKs. Then we let $\Delta h = a^{u^*+1}$.

To explain why we let $\Delta h = a^{u^*+1}$, let us consider a simple example. Assume that $u^* = 2$. Suppose there are 10 users $\{u_1, u_2, ..., u_{10}\}$ who send NACKs for the current rekey message. Assume user u_i sends NACK a_i, $i = 1, ..., 10$. For illustration purpose, we also assume $a_1 \geq a_2 \geq ... \geq a_{10}$. According to our protocol, for the next rekey message, the key server will multicast $a^{u^*+1} = a_3$ additional packets for each block. When the next rekeying interval starts, those flows (including data flows in our group communication application) which share the same bottlenecks as our rekey traffic should have backed off. As a result, users $\{u_3, u_4 ..., u_{10}\}$ will have high probabilities to receive a_3 more packets. Then hopefully those users can receive or recover their required encryptions within a single multicast round.

4.2.4 Proposed A-FEC scheme

We propose our adaptation scheme named A-FEC as follows:

```
• $u \leftarrow$ the number of NACKs received

• $a^{u^*+1} \leftarrow$ the $(u^*+1)^{th}$ largest NACK

• $counter \leftarrow counter + 1$

• if $counter = 1$

      then $\bar{u} \leftarrow u$
      else $\bar{u} \leftarrow v \cdot \bar{u} + (1-v)u$

• if $(u > u_\alpha)$ or $(counter \geq 3$ and $\bar{u} > u^*)$

      then $h \leftarrow h + a^{u^*+1}$
           $T \leftarrow \frac{BW_m(h)+BW_u(u^*/n)}{b(t)}$
           $counter \leftarrow 0$

• if $counter \geq 3$ and $\bar{u} \leq u^*$ and $h > h_l$

      then $h \leftarrow h - 1$
           $T \leftarrow \frac{BW_m(h)+BW_u(u^*/n)}{b(t)}$
           $counter \leftarrow 0$
```

The key server runs it after it collects NACKs from users.

4.3 Performance evaluation

We use simulations to evaluate the performance of our A-FEC scheme. We run our simulations using network simulator ns-2. To simulate the Internet topology, we use Georgia Tech Internetwork Topology Generator (GT-ITM) to generate a Transit-Stub graph with 10 Mbps of link bandwidth. The graph contains 592 stub domains. We let the key server reside in one stub domain, and then create 591 edge networks in each of the remaining stub domains. Each edge network has an access link connected to the internetwork. For purpose of simplicity, we did not simulate data traffic for our group communication application. Instead, we set the bandwidth of each access link to a relatively small value. More specifically, the bandwidth of each access link is uniformly distributed between 0.1 and 1 Mbps. To simulate the background traffic, we let each of 514 edge networks have 30 outgoing and 30 incoming FTP flows, and each of the remaining 77 edge networks have 40 outgoing and 40 incoming FTP flows. For purpose of simplicity, we assume that when the key server updates h during one rekeying interval, the updated h will be applied for the next rekey message. We assume that at the beginning of each rekeying interval the key tree (with degree 4) is balanced with 768 users. During each rekeying interval, 192 join and 192 leave requests are processed. We set block size as $k = 14$. Our simulations show that the average packet loss rate observed by each user is about 6%. Therefore, we set $u_\alpha = 11.9$, 19.7 and 33.6, and $h_l = 3$, 3 and 2 for $u^* = 5$, 10 and 20 respectively, as calculated earlier.

For the purpose of comparison, we define two additional heuristics to compare with our A-FEC scheme:

- Heuristic 1: The key server increases h by a^{u^*+1} whenever $u > u^*$, and decrease h whenever $u \leq u^*$. No lower threshold is specified for h.

- Heuristic 2: It is the same as Heuristic 1 except that h should be larger than or equal to h_l at any time.

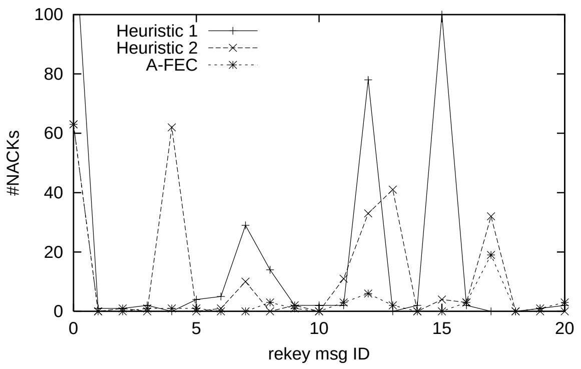

Figure 9: Traces of the number of NACKs for $u^* = 5$

Figure 9 to 11 demonstrate traces of the number of NACKs for Heuristic 1, 2 and our A-FEC scheme. For Heuristic 1, the system starts with $h = 0$. For Heuristic 2 and A-FEC scheme, the initial value of h is h_l. From the figures, we have the following observations:

- An increase of h by a^{u^*+1} upon $u > u_\alpha$ (or $u > u^*$) can effectively control the number of NACKs. In particular, whenever the number

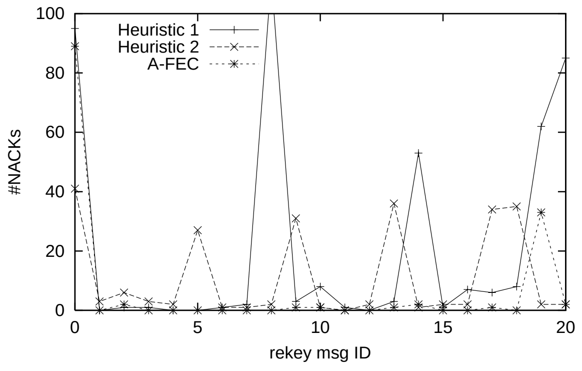

Figure 10: Traces of the number of NACKs for $u^* = 10$

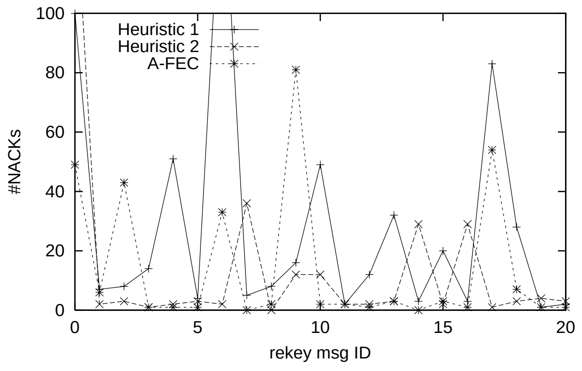

Figure 11: Traces of the number of NACKs for $u^* = 20$

of NACKs is larger than u_α (or u^*), it usually takes only one rekey message for the key server to make $u \leq u_\alpha$ (or $u \leq u^*$).

- With lower threshold h_l specified, we can effectively prevent the case that h is reduced to a very small value. As a result, with h_l, we can significantly reduce the peak point values on the curves of the number of NACKs.

- A-FEC scheme can further reduce the fluctuations of the number of NACKs by making the conditions to update h more strict. However, this is achieved by trading our protocol's responsiveness to network traffic change.

- When u^* is large, it is hard to control the fluctuations of the number of NACKs, as seen in Figure 11. Therefore, it is desired to specify a small u^* in practice.

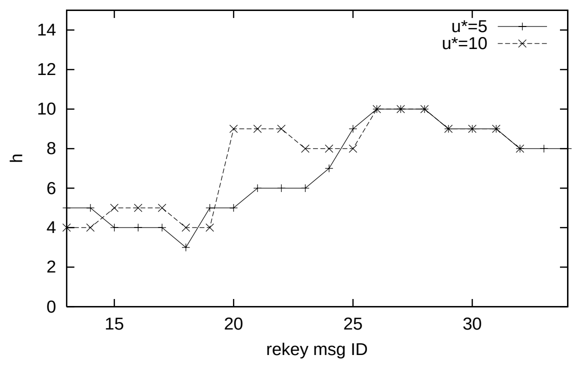

Figure 12: Traces of h when background traffic is doubled

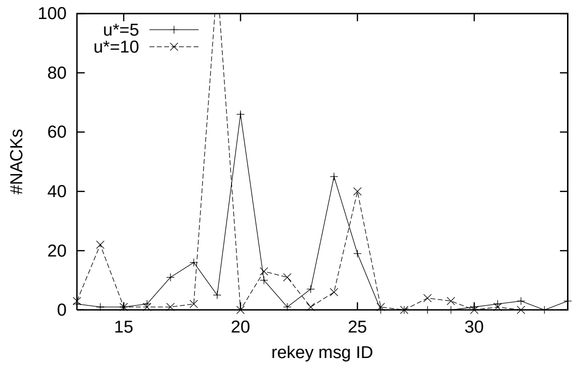

Figure 13: Traces of the number of NACKs when background traffic is doubled

We further evaluate the responsiveness of our A-FEC scheme to network traffic change. To simulate a changing network, we increase the number of background FTP flows until the total number is doubled. Each added FTP flow starts randomly during the rekeying intervals of rekey message 13 to 23. Figure 12 and 13 demonstrate the traces of h and the number of NACKs. As can be seen, when the network becomes loaded, shared loss will cause a lot of users to send NACKs. Our A-FEC scheme can quickly increase h upon network congestion, thus significantly reducing the number of NACKs for the next rekey message.

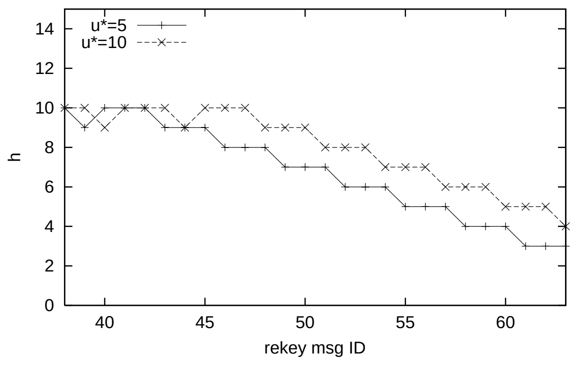

Figure 14: Traces of h when background traffic is reduced

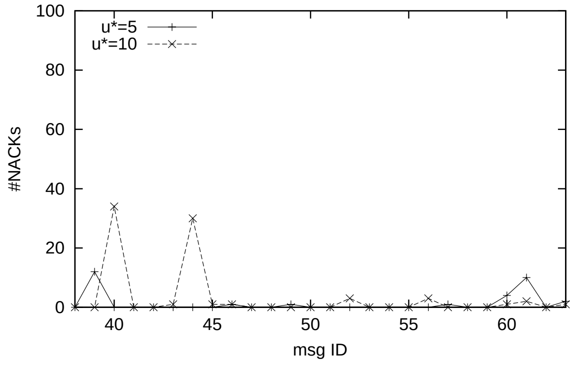

Figure 15: Traces of the number of NACKs when background traffic is reduced

To simulate a network with decreasing traffic, we let each added FTP flow stop randomly during the rekeying intervals of rekey message 38 to 48. As seen from Figure 14 and 15, our A-FEC scheme gradually reduces h (and T) to adapt to the improved network situation.

5 Conclusion

In our group rekeying protocol, we care about two performance metrics: r and T. Metric r measures the percentage of users who cannot receive the new group key on time, while T is a measure of group access control granularity. Ideally, we want to achieve both small r and T. However, to achieve a smaller r, the key server has to increase h, and thus increase T to send more traffic. Therefore, there are tradeoffs between r, T and h.

To investigate the tradeoffs between r, T and h, we analyzed $r = f(h, T)$ under Bernoulli and Markov loss models. We also examined the impact of rekeying bandwidth constraint on the relationship between h and T. The rekeying bandwidth constraint arises since we do not want rekey traffic to hurt the Internet performance. We observed that with the rekeying bandwidth constraint, the value of T cannot be very small. Then with our packet spacing strategy, packets of the same block will likely experience independent loss. As a result, an increase of h can effectively reduce r; also decreasing r will not significantly increase h and T. In conclusion, we can achieve both small r and T.

We designed an adaptive FEC scheme to determine (h', T') for any specified u^*. We proved and also demonstrated that (h', T') is close to the optimal (h^*, T^*). Our scheme does not require knowledge of the mathematical expression for $r = f(h, T)$. Simulation results from ns-2 show that our protocol can achieve fairly smooth traces of the number of NACKs when group rekeying is subjected to statistical fluctuations of a fixed set of competing flows. We also found that with the onset of network congestion our adaptive FEC protocol can still achieve the target u^* by adjusting values of h and T.

References

[1] Omar Ait-Hellal, Eitan Altman, Alain Jean-Marie, and Irina A. Kurkova. On loss probabilities in presence of redundant packets and several traffic sources. *Performance Evaluation*, 1999.

[2] Eitan Altman, Chadi Barakat, and Victor Ramos. Queueing analysis of simple fec schemes for ip telephony. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[3] David Balenson, David McGrew, and Alan Sherman. *Key Management for Large Dynamic Groups: One-way Function Trees and Amortized Initialization, INTERNET-DRAFT*, 1999.

[4] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Don Towsley. Adaptive FEC-based error control for Internet Telephony. In *Proceedings of IEEE INFOCOM '99*, March 1999.

[5] Bob Briscoe. Marks: Zero side effect multicast key management using arbitrarily revealed key sequences. In *Proceedings of NGC 1999*, Pisa, Italy, November 1999.

[6] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key management for secure Internet multicast using boolean function minimization techniques. In *Proceedings of IEEE INFOCOM '99*, volume 2, March 1999.

[7] Sally Floyd, Mark Handley, Jitendra Padhye, and Jörg Widmer. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

[8] Hugh Harney and Eric Harder. *Logical Key Hierarchy Protocol, INTERNET-DRAFT*, March 1999.

[9] Internet Research Task Force (IRTF). The secure multicast research group (SMuG). http://www.ipmulticast.com/community/smug/.

[10] X. Steve Li, Y. Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Proceedings of Tenth International World Wide Web Conference (WWW10)*, Hong Kong, China, May 2001.

[11] Jeff Lotspiech, Moni Naor, and Dalit Naor. *Subset-Difference based Key Management for Secure Multicast , INTERNET-DRAFT draft-irtf-smug-subsetdifference-00.txt*, July 2001.

[12] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM '97*, September 1997.

[13] Adrian Perrig, Dawn Song, and Doug Tygar. Elk, a new protocol for efficient large-group key distribution. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2001.

[14] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review*, April 1997.

[15] Luigi Rizzo. pgmcc: A tcp-friendly single-rate multicast congestion control scheme. In *Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, August 2000.

[16] Sanjeev Setia, Samir Koussih, Sushil Jajodia, and Eric Harder. Kronos: A scalable group re-keying approach for secure multicast. In *Proceedings of IEEE Symposium on Security and Privacy*, Berkeley, CA, May 2000.

[17] Jack Snoeyink, Subhash Suri, and George Varghese. A lower bound for multicast key distribution. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska, April 2001.

[18] D. Wallner, E. Harder, and Ryan Agee. *Key Management for Multicast: Issues and Architectures, RFC 2627*, June 1999.

[19] Jorg Widmer and Mark Handley. Extending equation-based congestion control to multicast applications. In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[20] Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of ACM SIGCOMM '98*, September 1998.

[21] Chung Kei Wong and Simon S. Lam. Keystone: a group key management system. In *Proceedings of ICT 2000*, Acapulco, Mexico, May 2000.

[22] Y. Richard Yang and Simon S. Lam. General AIMD congestion control. In *Proceedings of the 8th International Conference on Network Protocols '00*, Osaka, Japan, November 2000.

[23] Y. Richard Yang, X. Steve Li, X. Brian Zhang, and Simon S. Lam. Reliable group rekeying: A performance analysis. In *Proceedings of ACM SIGCOMM 2001*, San Diegao, CA, August 2001.

[24] X. Brian Zhang, Simon S. Lam, Dong-Young Lee, and Y. Richard Yang. Protocol design for scalable and reliable group rekeying. In *Proceedings of SPIE Conference on Scalability and Traffic Control in IP Networks*, Denver, CO, August 2001.