Optimal Tree Structure for Key Management of Simultaneous Join/Leave in Secure Multicast

Feng Zhu, Agnes Chan, Guevara Noubir

College of Computer Science
Northeastern University
Boston, MA 02115
{zhufeng, ahchan, noubir@ccs.neu.edu}

Abstract:

As applications of secure multicast in networks continue to grow, the demand for an efficient scheme to manage group keys for secure group communication becomes more urgent. In this paper, we propose a new key tree structure for group key management. With this optimal tree structure, system resources such as network bandwidth can be saved. We devise an algorithm to generate this optimal tree and show that it can be implemented efficiently. We also design an adaptive system for group key management which consists of four components: a request receiver, a key tree update controller, a delay calculator and a request predictor. This system can maintain the optimality of the key tree dynamically. It is verified by theoretical analysis and simulation result that the performance of our scheme is better than other schemes based on traditional tree structures.

1. Introduction

Multicast communications provide efficient delivery of data from a sender to multiple recipients. Compared with unicast communications used in a large group, it consumes fewer resources, such as network bandwidth and energy. Network applications, such as the delivery of stock market information, multi-party video-conferencing, pay-per-view video programs, are based on multicast communication model to deliver data to authorized members.

In a secure group communication system, data privacy can be achieved by a shared group key. Only the authorized members know the group key. But usually group memberships vary over time. Whenever some members join or leave the group, the shared key needs to be changed. Thus there must exist an efficient group key management system to maintain group memberships such that only authorized group members are able to access the group key for that group. At the same time the group key management system must be responsible for generating, updating and delivering the keys to the authorized members.

There are two desirable properties for the shared group key: first, the BAC (backward access control) property, which means that new users should not be able to decrypt past group communications with their new keys; second, the FAC (forward access control) property, which means that the evicted users should not be able to decrypt future group communications with their old keys.

For a group with many users, join/leave requests happen frequently. Since a group key management center needs to send out a lot of re-keying messages to authorized users with limited bandwidth, the bandwidth for re-keying is the biggest bottleneck in current applications. In this paper, we will consider communication complexity, which is measured by the number of messages that need to be sent for re-keying in unit time.

There are two kinds of re-keying protocols. One is the individual re-keying protocol, in which the system updates the group key every time a request is received. Another kind of protocol is the periodic batch re-keying protocol, in which the group controller does not immediately perform updating for each request, but waits until several requests have accumulated to perform re-keying.

In [1][2], individual re-keying and the associated performance are discussed. Wong[1] proposed the key graph approach to facilitate group re-keying, and the communication complexity of group re-keying for each request is O(log N), where N is the group size. There is no extra delay to process user requests in the individual re-keying protocols. But it is difficult to control the synchronization of re-keying and it can be inefficient to do updates for a batch of requests [6]. The periodic batch re-keying protocol can significantly save the resources in communications. However the Forward Access Control property may not be guaranteed within the re-keying period (also known as the vulnerablilty windows). In many applications, such as pay-per-view TV systems, the batch-updating mode is acceptable, if the system can guaranteed that the length of re-keying period is in an acceptable range. In this paper, we only consider the batch mode.

The paper is organized as follows. In section2, we propose a special key-tree structure and use it for efficient re-keying. Then we describe the algorithm to construct the optimal tree in section 3. In section 4, we compare the performance of our protocol based on the optimal tree with other protocols. In section 5, we describe the adaptive algorithm for delay control. A complete group key management system, which implements the above ideas, is presented in section 6. In section 7, we show the simulation results of both the systems with the adaptive algorithm and the systems with the non-adaptive algorithm. Finally, section 8 concludes the paper.

2. Structure of the Optimal Key Tree.

To facilitate updating group keys, a key tree[1] is introduced. A key tree is a rooted tree with the group key at the root, the leaf nodes as U-nodes corresponding to group members and internal nodes as K-nodes. The group controller gives each member an individual key that is stored at the U-node and auxiliary keys stored in the K-nodes located along the path from the U-node to the root, including the group key stored at the root.

The key tree proposed in [1] is regular in that every K-node has a fixed number of child nodes (K-nodes or U-nodes). The authors recommended the use of a binary tree. However such a key tree may not be efficient for batch re-keying. For example, if the system needs to process many requests from members, then the keys contained in the higher level K-nodes always need to be updated but may not be essential in the re-keying process. In this section, we propose a tree structure that is optimal with respect to bandwidth usage. Such a tree is called an optimal tree.

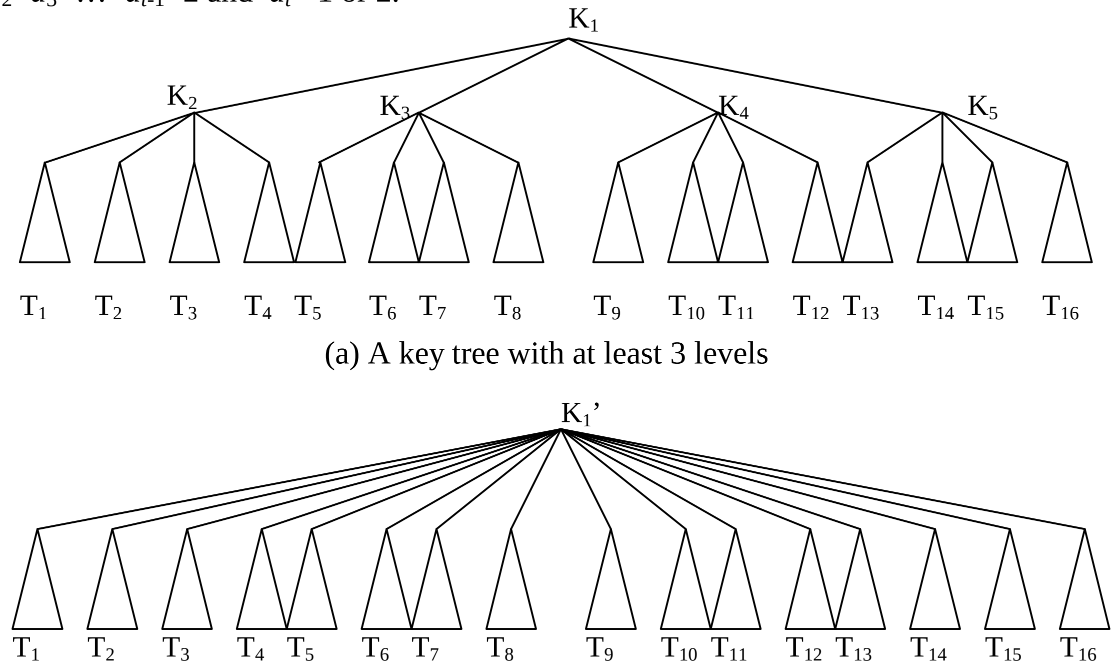
For simplicity, we restrict the key-tree in our discussion to trees where the number of branches of each node is 2^i and the number of users is $N=2^k$. With this structure, a tree could be converted to another tree by adding or removing a level of nodes whenever needed, as shown in Figure 1.

The performance of a re-keying scheme depends on the structure of the key tree. The performance can be improved by organizing the tree structure with respect to the re-keying probability of users [3][4]. We define the re-keying probability p_i of a user U_i as the probability that the system should re-key some users' keys including U_i 's key in one unit time. To simplify the analysis, we assume the re-keying probabilities of all users are equal to p. We also assume that the re-keying probability of each user is independent of the others.

First let us define some notation.

- Let T(a_1, a_2,…a_t; p) denote a tree with t-levels, such that the top (first) level has 2^{a_1} branches, the i-th level has 2^{a_i} branches .. and the bottom(last) level has 2^{a_t} branches, the re-keying probability of each user is equal to p.

We will show that if a tree T(a_1, a_2,…a_t; p) is an optimal tree, then $a_1 \geq 2$, $a_2 = a_3 = \ldots = a_{t-1} = 2$ and a_t= 1 or 2.



(a) A key tree with at least 3 levels

(b) A key tree obtained from (a) by removing level 1

Figure 1 Conversion of a key tree

Lemmas 1, 2 and 3 will establish the optimality for trees with only two levels. Lemma 4 show how optimal trees can be established recursively from smaller trees. Detail proof of the lemmas and theorems can be found in the appendix.

For p=0, no user joins/leaves and so no re-keying process is needed. For p=1, all users join/leave, clearly the optimal tree is a flat tree with one level. Henceforth, we can assume 0<p<1 in following discussion.

Lemma 1 For a given tree T(k-j, j; p), if j >2, then this tree T(k-j, j; p) is not an optimal tree.

From Lemma 1, we know that the only two level optimal trees are T(i,1; p) and T(i,2; p).

Lemma 2 The tree T(1,1; p) is not an optimal tree.

Lemma 3 The tree T(1,2; p) is not an optimal tree.

Combining with Lemma 1, these two lemma show that the two-level optimal trees must be T(i, j; p), where i>1, j = 1 or 2.

Lemma 4 If a tree T(a_1, a_2,…a_t; p) is an optimal tree, then, for any i and j, the subtree T'(a_i, a_{i+1},…a_j; p_j) is also an optimal tree, where $p_j = 1-(1-p)^{|T_j|}$, $|T_j| = \prod_{k=j+1}^{t} 2^{a_k}$ is the number of leaves on the subtree whose root is on j level of T(a_1, a_2,…a_t; p) .

Lemma 4 shows that any subtree of an optimal tree is also an optimal tree. Using Lemma 1, 2, 3 and 4, we can prove the following theorem.

Theorem 1: If a tree T(a_1, a_2,…a_t; p) is an optimal tree, then, either it is a flat tree or a_1≥2, a_2=a_3=…=a_{t-1}=2 and a_t= 1 or 2

Proof:

If the optimal tree T(a_1, a_2,…a_t; p) is not a flat tree, in other words, it has more than one level, since it is an optimal tree, by Lemma 4, all subtrees T_i(a_i,a_{i+1}; p_{i+1}), for 1 ≤ i <t, are optimal. From Lemma 1, a_i=1 or 2 for 2 ≤ i ≤ t. From Lemma 2 and Lemma 3, a_i=2 , for 2 ≤ i ≤ t-1), because T_i(a_i,a_{i+1}; p_{i+1}) is optimal. Similarly, a_1 can not be 1, because T_1(a_1,a_2; p_2) is optimal. Thus a_1≥2.

In the next section, we will give an efficient algorithm to generate the optimal tree**.**

3. Algorithm for Finding the Optimal Tree.

In the previous section we have established that the optimal tree must be a flat tree or be of the form -- T(a_1, 2,…2, a_t; p), with a_1≥2 and a_t= 1 or 2. Next we will use T(a_1; t; a_t ; p) to denote T(a_1, 2,…2,a_t; p) and we will use T(a_1; 1; 0 ; p) to denote a flat tree.

A simple naïve method to determine the values a_1 and t for an optimal tree is to computer the expected number of messages $M(N,p)_{T(a_1;\ t;\ a_t;\ p)}$ needed to be transmitted for each tree T(a_1; t; a_t ; p).

Lemma 5. Given a tree T(a_1; t; a_t ; p),

$$M(N,p)_{T(a_1;\ t;\ a_t;\ p)} = 2^{a_1}[1-(1-p)^N] + \sum_{i=2}^{t-1} 2^{a_1+2i-4}[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}] + N[1-(1-p)^{2^{a_t}}]$$

By compare all these values, the values $a_1$and t that correspond to the minimal value $M(N,p)_{T(a_1;\ t;\ a_t;\ p)}$ provides the optimal tree for a given probability p. I this section, we present a more efficient algorithm using binary search.

First, we consider the trees with one or two levels. If we use a flat tree, then the expected number of message needed to send is $2^k(1\text{-}q^N)$, where q = 1- p, k = $\log_2 N$. If we use a tree with two levels, the expected number of messages sent is $2^{N/a_t}[1-(1-p)^N]+N[1-(1-p)^{2^{a_t}}]$ for a_t = 1 or 2.

By comparing these quantities and solving the inequalities:

$2^k(1-q^N)<2^{N/a_t}[1-(1-p)^N]+N[1-(1-p)^{2^{a_t}}]$ for a_t = 1 or 2

we can identify the probability $p_1(k)$ of user joining/leaving the group under which a one-level tree is better than a two-level tree. Let $q_1(k) = 1 - p_1(k)$.

Lemma 6. Let the number of user be N=2^k and $q_1(k)$ be defined as above. Then the values of $q_1(k)$ are given in Table.

k	$q_1(k)$
3	0.7373527057593127
4	0.7085316342724187
5	0.7071121772808317
6	0.7071067812696629
≥7	0.707106781187345

In general, for a given k, q>$q_1(k)$, then a two-level tree is better than a one-level tree. We can build an optimal tree inductively from this result. Intuitively, if we consider each subtree T_J of T(a_1, a_2, …, a_t) beyond the first level as a single user J (see Figure1(a)), then T(a_1) is a one-level tree. Thus to compute a_1 is to find j such that the probability of each subtree T_J joining/leaving is 1- $q_1(j)$, where J consist of all leaves in T_J, $|J| = 2^{k-j}$. However, the probability of user J joining/leaving given by $q'(j)= (1-p)^{2^{k-j}}$ may not equal to 1-$q_1(j)$. In order to find a_1, we can find j by Algorithm 1 such that

$q'(j+1)>q_1(j+1)$, and $q'(j) \le q_1(j)$. …..(1)

```
Algorithm1
1. Given $N=2^k$ and $q= 1-p$,
2. Use by binary search to find, $j$ in [2,k] that satisfies the inequalities
   in (1).
3. If $j$ exist, then compare the expect number of messages based on the
   trees, T($j$, t, $a_t$; $p$) with T (j-1, t, $a_t$+1; $p$) (if $a_t$=2, then use T ($j$-1, $t$+1,
   $a_t$-1; $p$) instead), i.e. $M(N,p)_{T(a_1;\,t;\,a_t;\,p)}$ and $M(N,p)_{T(a_1-1;\,t;\,a_t+1;\,p)}$ ,
   the optimal tree is the one with the smaller values.
 Otherwise, the optimal tree is T(2; $\log_4 N$; $a_t$; $p$)).
```

Theorem 2. The tree found by Algorithm 1 is the optimal tree.

Proof:

We only need to show that if j exist in Algorithm 1 Step 3, then only two trees T_1(j; t; a_t; p) and T $_2$(j-1; t ; a_t+1; p) can be optimal. Without lose generality, we assume a_t = 1.

We will first consider any other tree T'(m; t_m;1; p) which is not T_1(j; t; 1; p) and show that T'(m; t_m;1; p) can not be an optimal tree.

Case 1: m > j , Since q'(m)>q'(j+1)> q1(j+1), a two-level tree T(m-2;2;1; 1-q'(m)) is better than a one-level tree T(m; 1;1;1-q'(m)).

Thus the tree T”(m-2; t_m +1; 1; p) is better than the tree T’(m; t_m ; 1 ; p). So T’(m; t_m; 1; p) is not an optimal tree.

Case 2: $m < j$, Since $q'(m)<q'(j)\leq q1(j+1)$, a one-level tree T(m+2; 1;0; 1-$q'(m)$) is better than a two-level tree T(m; 2; 1; 1-$q'(m)$).

Thus the tree T”’(m+2, t_m-1,1 ; p) is better than tree T’(m; t_m; 1; p). So T’(m; t_m; 1; p) is not an optimal tree.

Next for any other tree T’(m; t_m; 2; p), where m≠ j-1, it can not an optimal tree, we compared it with T_2(j-1; t ; 2; p) and similarly show that it can not be an optimal tree.

So the optimal tree must be either T_1(j; t;1; p) or T_2(j-1; t; 2; p). By comparing the corresponding expected number of messages. We can construct the optimal tree as given in Step 3 of Algorithm 1.

Next, we will show that the computation complexity of Algorithm 1 is very low.

Theorem 3. The complexity of Algorithm 1 is O(loglog N).

Proof:

For a given number of users $N=2^k$, the height of the tree is at most m=$\lceil \log_4 \mathrm{N} \rceil$= k/2 because the tree is a 4-ary tree. Since the main body of this algorithm is a binary search algorithm for searching in interval [1, k], so the complexity of this algorithm is O(loglog N).

4. Performance of Batch Processing using an Optimal Tree

In this section, we will compare the performance of our scheme with other commonly used schemes. Namely

(1) Processing every key update request as it is received using a fixed 4-ary tree as the key tree structure.
(2) Batch processing of several key update requests over a fixed 4-ary tree
(3) Batch processing of several requests using a flat tree, which is using unicast.
(4) Batch processing of several requests using an optimal tree T(a_1, a_2, .., a_t)

Let $M(N,p)$ denote the expected number of keys needed to be updated for each processing. We consider an N-user optimal key tree with the probability of re-keying requests from each user to be p. To simplify the analysis, we assume that the key tree is balanced and complete.

For the first scheme, the expected number of requests is Np. Each request is processed one by one, and every time it needs to send out $\log_4 N$ messages, thus,

$$M(N,p) = Np\log_4 N\,.$$

The second scheme uses simultaneous processing for several key update requests over a fixed 4-ary tree. At level i, there are 4^i keys stored in the nodes. For each key in this level, the probability that this key needs to be changed is $[1-(1-p)^{\frac{N}{4^{i-1}}}]$. Thus,

$$M(N,p) = \sum_{i=1}^{\log_4 N} 4^i[1-(1-p)^{\frac{N}{4^{i-1}}}]\,.$$

For the third scheme, which uses simultaneous processing for several requests over a flat tree, the probability of the group key to be updated is 1–(1-*p*)^N. Every time the group key is updated, *N* messages have to be sent, thus,

$$M(N,p) = N[1-(1-p)^N].$$

For the fourth scheme using simultaneous processing for several requests over an optimal tree, then *M*(*N*,*p*) can be computed with the algorithm given in [5]. If tree T(a_1; *t*, a_t; p) is the optimal tree, then by Lemma 5

$$M(N,p)_{T(a_1;\ t;\ a_t;\ p)} = 2^{a_1}[1-(1-p)^N] + \sum_{i=2}^{t-1} 2^{a_1+2i-4}[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}] + N[1-(1-p)^{2^{a_t}}]$$

These analysis results are shown in Figure 3 for *N* = 65536 and for *p* ranges from 0 to 0.4. From Figure 2, we observe that the simultaneous processing for several requests over an optimal tree outperforms the other algorithms. The difference will become even larger as N becomes larger. The system with the optimal tree structure requires less system resources, such as CPU time and bandwidth, since fewer messages need to be encrypted and to be sent.

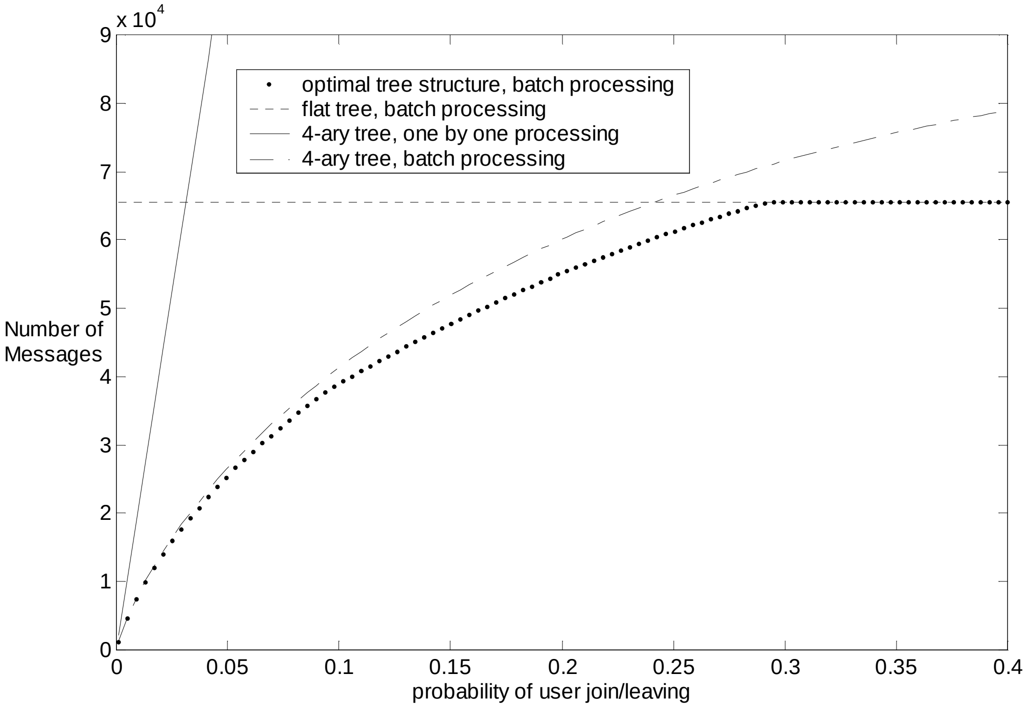


Figure 2. Comparison of performance of the re-keying operation

5. Adaptive algorithm for delay control

The optimal tree provides a key management structure to reduce bandwidth usage whenever an update is needed. The bandwidth may be further reduced if we increase the access control granularity [5]. But, in most applications, making the time interval long is not a good solution because the vulnerability window [7] will be large. In this section, we propose an adaptive algorithm for delay control to find a good tradeoff between the bandwidth and access control granularity.

Assume that key update is processed at time intervals t_0, t_1, t_2,…, where t_i is the i-th time interval. Since the length of messages broadcasted by the system used to process the updating group key requirements at a given time interval is an important parameter in our algorithm, we will first discuss two different methods of computing the length of messages broadcasted by the system used to process the updating group key requirements at t_i. One method uses the length of the current time interval and the probability of user modification in unit time to predict the length of messages needed to process the requests in the i-th time interval. Another method is to scan the key tree after the current time interval to get the length of messages for processing the requests in the i-th time interval. Using the first method, the result can be pre-computed before the end of the time interval. The Key Management System can submit to the application server to request for bandwidth beforehand. Since the computation is based on the probability of user modification, the result may not be exact. On the other hand, the computation from the second method is exact, since the result is not based on prediction, but on scanning the key tree for actual requests at real time. So the outputs (B_i and t_i) from Algorithm 2 are computed in real time. But the result of the function $f(t_i)$ can not be obtained before the end of time interval t_i. So the Key Management System has to submit to the application server for bandwidth request at real time.

Let $f(t_i)$ denote the length of messages broadcasted by the system to update group key requirements at t_i. Let B be the default bandwidth for group key control, B_i be the real bandwidth used in the i-th time interval, t_{max} be the maximum length of time interval that the application will tolerate.

Algorithm 2 given below computes time for delay to reach a good tradeoff between bandwidth usage and access control granularity.

```
Algorithm 2
    Input $t_{i-1}$
        $t = f(t_{i-1}) / B$
        if $t > t_{max}$, set $t_i = t_{max}$, else $t_i = t$
        $B_i = f(t_{i-1}) / t_i$
        output $B_i$ and $t_i$
```

The algorithm always tries to reduce the time interval by using the default bandwidth whenever possible. However, when the length of the time interval t obtained from using the maximal bandwidth still exceeds the application’s maximal acceptable length of time interval, the group key management system will then request more bandwidth from the system control of the application.

6. A system for group key management

In this section, we will construct a group key management system to process the update of group keys. This system is based on the optimal tree structure and the adaptive algorithm to reduce system resources required and the size of vulnerability windows. The group key manager processes the requests from users, changes keys and distributes new keys to users.

A group key management system has four components: a request receiver, a key tree update controller, a delay calculator and a request predictor. Figure 3 depicts such a system. If the client request rate is high, multiple request receivers may be set up in the network to communicate with the key tree update controller.

When a user wants to join or leave the group, it sends a request with a time stamp to the key server housed in the Request Receiver. After the user and the key server mutually authenticate each other, the request receiver generates an individual key shared with the user to protect communication between the user and the key management system. The shared key is provided to the Key Tree Manager to be stored at a leaf node.

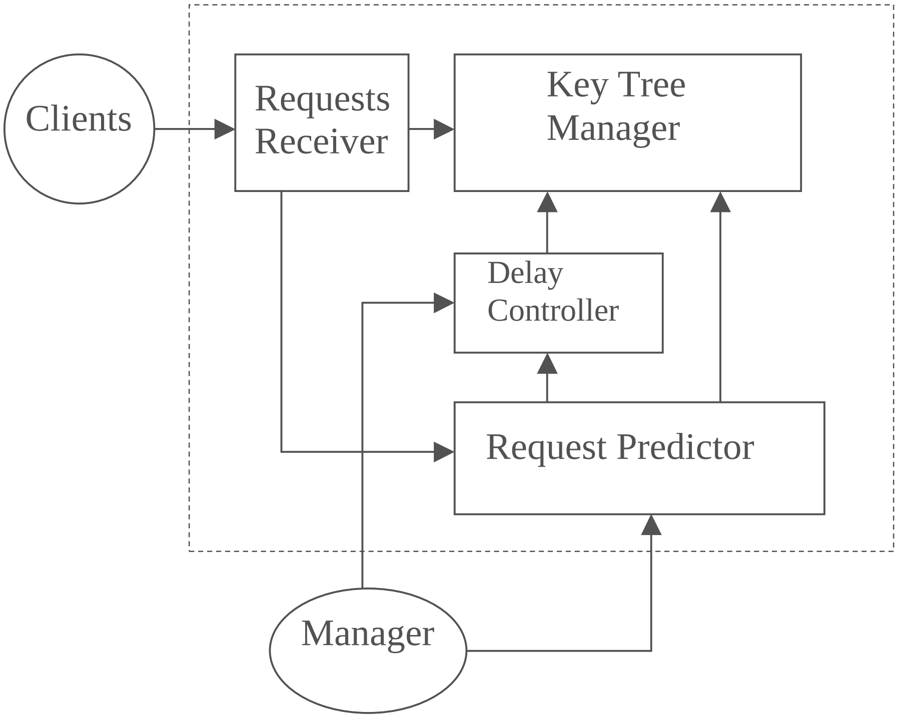


Figure 3. A group key management system

After the Request Receiver generates the user's individual key, the user's request is forwarded to the Request Predictor. The predictor will predict the probability of each leaf node of the key tree that is to be modified in the next time interval. These probabilities are then given to both the Key Tree Manager and the Delay Controller. The Key Tree Manager and the Delay Controller will maintain the optimal key tree and compute the length of time interval based on this information.

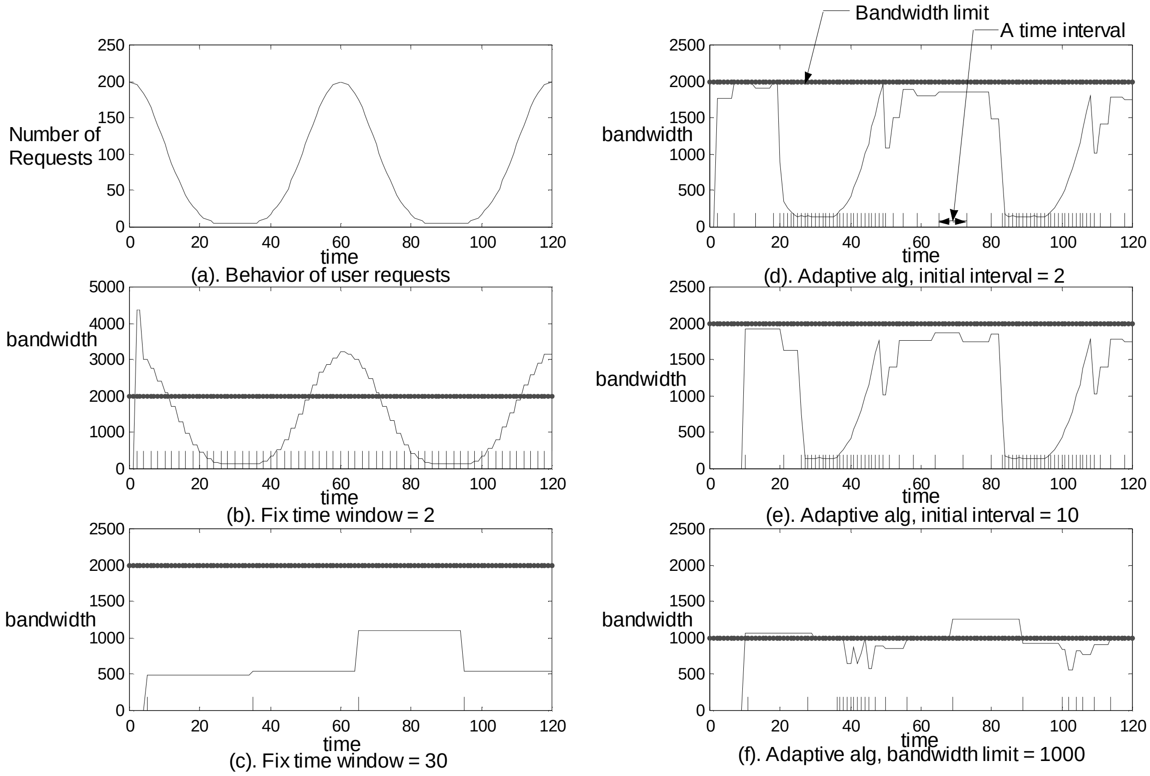
The Key Tree Manager is responsible to manage the key tree and to generate new group keys. It assigns a position to a newly added node and marks the keys on the key tree that need to be updated. The Key Tree Manager also changes the key tree structure based on the probabilities received from the Request Predictor. It may change the number of branches at the top level of the key tree in order to maintain the optimal key tree structure according to the predicted probabilities. This can be easily accomplished by deleting or adding one level immediately below the top level.

The Delay Controller computes and controls the size of the vulnerability window. By controlling the time delay, the system can perform the synchronized key update with variable time periods. It has been shown in [6] that the bandwidth required for securing the multicast group communication can be reduced if the time period of the key update can be dynamically changed. The Delay Controller sends its computed delay time to the Key Tree Manager, which will check if the time out is reached. If so, the Key Tree Manager will broadcast the updated keys to prepare the broadcast data. The Key Tree manager checks the nodes along each path from the new user node at the leaf to the root. If it finds a node that is marked, it will encrypt the new key stored in this node by using the keys stored in its child node.

In addition to the four components described above, the Group Key Management Scheme has a Manager that allows human input to modify the parameters that effect the computation of probability of user joining/leaving and the time delay.

7. Results and comparison

We have carried out experiments to test bandwidth usage by the adaptive algorithm described in section 6, with the aim to minimize length of the time interval for batch processing under some given bandwidth constraint. The simulation results are shown in Figure 4(a) - Figure 4(f).



(a). Behavior of user requests

(d). Adaptive alg, initial interval = 2

(b). Fix time window = 2

(e). Adaptive alg, initial interval = 10

(c). Fix time window = 30

(f). Adaptive alg, bandwidth limit = 1000

Figure 4

For ease of simulation, we assume the behavior curve of user requests is smooth. Figure 4(a) represents the behavior of a sinusoidal-like curve. We use a 4-ary tree as the key tree.

In Figure 4(b), a fixed short time interval for batch processing is used. From this figure, we see that at times an unusually high bandwidth is needed to process the requests. The fluctuation of the bandwidth requirement is quite large. For real time applications this translates that at times insufficient bandwidth is available while at other times much bandwidth remains unused. In Figure 4(c), a fixed long time interval is used. In that case, minimal bandwidth is needed, but processing for requests are delayed for quite a long time providing a vulnerability window that can be exploited by attackers.

In Figure 4(d), the adaptive algorithm is used. In that case, when the requests come frequently, the time interval is controlled to be relatively long and the bandwidth is kept lower than the maximum bandwidth that the application can accept. If the requests come infrequently, the time interval is controlled to be quite short. At that time, the requests can be accommodated very quickly.

In the simulation shown in Figure 4(e), we increase the initial time interval to 10 time units. By comparing Figure 4(d) and Figure 4(e), we can see that the performance of the system is not sensitive to the initial time interval if the requests do not come very frequently. It shows that our adaptive algorithm is robust with respect to changes in the initial time interval.

If the maximum bandwidth that the application can accept is quite low, as shown in Figure 4(f), the time interval has to be relatively long. Sometimes a little more bandwidth is required from the application system to accommodate the requests. But this happens less frequently than in the case where the fixed short time interval is used as shown in Figure 4(b). The adaptive algorithm always tries to decrease the delay by using the limited bandwidth efficiently and always tries to keep the usage of bandwidth below the default bandwidth provided by the application systems.

8. Conclusion

In this paper, we have shown that batch processing for group key updates provides a more efficient means of utilizing resources than single request update. The new adaptive algorithm for group key management provides a tradeoff between bandwidth usage and vulnerability window size. Our scheme can handle the re-keying processing for secure multicast in a dynamic group. We have carried out simulations to examine the behaviors of bandwidth usage under such an adaptive system. We are currently building the system to be deployed for wireless multicast.

Reference:

1. Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure group communications using key graphs. In *Proceedings of ACM SIGCOMM '98,* September 1998.
2. R. Canetti, T. Malkin, K. Nissim. Efficient Communication-Storage Tradeoffs for Multicast Encryption. Advances in Cryptology, EUROCRYPT 1999, May 1999.

3. A. Selcuk, C McCubbin, D. Sidhu, Probabilistic Optimization of LKH-based Multicast Key Distribution Schemes. Internet-Draft drafe-selcuk-probabilistic-lkh-01.txt
4. Radha Poovendran, John S. Baras An Information Theoretic Analysis of Rooted-Tree Based Secure Multicast Key Distribution Schemes. Advances in Cryptology - CRYPTO'99
5. G. Noubir, F. Zhu, A.H. Chan Key Management for Simultaneous Join/Leave in Secure Multicast. 2002 IEEE International Symposium on Information Theory
6. Y. R. Yang, X.S. Li, X. B. Zhang, S. S. Lam Reliable Group Re-key: A Performance Analysis In *Proceedings of ACM SIGCOMM '01*.
7. X.S. Li, Y.R. Yang, M.G. Gouda, S.S. Lam Batch Re-keying for Secure Group Communications *WWW10,* May 2-5, 2001, Hong Kong

Appendix

Lemma 1 For a given tree T(k-j,j; *p*), if j >2, T is not an optimal tree.

Proof:

Let q = 1-p. For tree T(k-j,j; p), the number of messages needed to be sent to update the key is $2^k(1-q^{2^j}) + 2^{k-j}(1-q^{2^k})$.

For tree T(k; p), the number of messages needed to be sent to update the key is $2^k(1- q^{2^k})$

For tree T(k-j, j-1, 1; p), the number of messages needed to be sent to update the key is

$2^{k-1}(1- q^{2^j}) + 2^{k-j}(1- q^{2^k}) + 2^k(1-q^2)$

When $0 \leq q < 2^{(-j/2^j)}$

$1-2^j q^{2^j} > 0$

$\Rightarrow 1-2^j q^{2^j} + (2^j-1) q^{2^k} > 0$

$\Rightarrow 2^j(1- q^{2^j}) + (1- q^{2^k}) > 2^j(1- q^{2^k})$

$\Rightarrow 2^k(1- q^{2^j}) + 2^{k-j} (1- q^{2^k}) > 2^k(1- q^{2^k})$

This means that the tree T(k; p) is better than the tree T(k-j,j; p),

When $1 > q \geq 2^{(-j/2^j)}$, we define $f(q) = 1 + q^{2^j} - 2q^2$,

Then $d^2 f(q) / dq^2 = (2^j) (2^j-1) q^{(2^j-2)} - 4$

So $d^2 f(q) / dq^2 \geq (2^j) (2^j-1) q^{2^j} - 4 \geq (2^j-1) - 4 > 0$

Thus $f(q)$ is a convex function.

And since, $f(2^{(-j/2^j)}) = 1+2^{-j}-2^{(-j/2^{j-1}+1)} < 1+2^{-3}-2^{1/4} < 0$, $f(1) = 0$

so $f(q) < 0$ for any q where $1 > q \geq 2^{(-j/2^j)}$

Thus when $1 > q \geq 2^{(-j/2^j)}$

$1 + q^{2^j} - 2q^2 < 0$

$\Rightarrow (1- q^{2^j}) > 2(1-q^2)$

$\Rightarrow 2^k(1- q^{2^j}) > 2^{k-1}(1- q^{2^j}) + 2^k(1-q^2)$

$\Rightarrow 2^k(1- q^{2^j}) + 2^{k-j}(1- q^{2^k}) > 2^{k-1}(1- q^{2^j}) + 2^{k-j}(1- q^{2^k}) + 2^k(1-q^2)$

This means that the tree T(k-j, j-1, 1; p) is better than the tree T(k-j,j; p),

So for any $q \in (0,1)$, the tree T(k-j,j; p), j >2, is not an optimal tree.

Lemma 2 For the tree T(1,1; *p*), it is not an optimal tree.

Proof:

Let q = 1-p.

For tree T(1,1; p), the number of messages (keys) needed to be sent is $4(1-q^2) + 2(1- q^4)$
For tree T(2; p), The number of messages (keys) needed to be sent is $4(1-q^4)$,
For any $0 < q < 1$, $2>(1+q^2)$, so
$2(1-q^2) > (1-q^2)(1+q^2) =(1-q^4)$
$\Rightarrow 2(1-q^2) + (1-q^4) > 2(1-q^4)$
$\Rightarrow 4(1-q^2) + 2(1-q^4) > 4(1-q^4)$
Thus T(1,1; p) is not an optimal tree.

Lemma 3 For the tree T(1,2; p), it is not an optimal tree.

Proof:

Let $q = 1-p$.
For the tree T(1,2; p), the number of messages (keys) needed to be sent is $8(1-q^4) + 2(1- q^8)$
For the tree T(3; p), the number of messages (keys) needed to be sent is $8(1-q^8)$
For the tree T(2,1; p), the number of messages (keys) needed to be sent is $8(1-q^2) + 4(1- q^8)$
When $1>3q^4$, $8(1-q^4) + 2(1- q^8) < 8(1-q^8)$, thus T(3) is better than T(1,2; p)
When $1< 3q^4$, $q^4+2q^2-1>0$, so
$(q^2-1)(q^4+2q^2-1)<0$
$\Rightarrow 1-3q^2+q^4+q^6<0$
$\Rightarrow 8-8(1+q^2)+2(1+q^2)(1+q^4)<0$
$\Rightarrow 8(1-q^2)-8(1-q^2)(1+q^2)+2(1-q^2)(1+q^2)(1+q^4)<0$
$\Rightarrow 8(1-q^4)+2(1-q^8)>8(1-q^2)+4(1-q^8)$
so, T(2,1; p) is better than T(1,2; p)
thus, T(1,2; p) is not optimal.

Lemma 4 If a tree T(a_1, a_2,…a_t; p) is an optimal tree for modification then, for any i and j, the subtree T'(a_i, a_{i+1},…a_j; p') is also an optimal tree, where $p'=1-(1-p)^{|T_j|}$, $|T_j|=2^{\sum_{s=j+1}^{t} a_s}$

Proof:

Let $q = 1-p$.

If the subtree T'(a_i, a_{i+1},…a_j; p') is not an optimal tree, there is an optimal tree T''(a'_i, a'_{i+1},…$a'_{j'}$; p') , where a_i+a_{i+1} …+ a_j= $a'_i+a'_{i+1}+\ldots+a'_{j'}$

When we replace the subtrees T'(a_i, a_{i+1},…a_j; p') with tree T''(a'_i, a'_{i+1},…$a'_{j'}$; p') to obtain T'''(a_1..a_{i-1},a'_i, a'_{i+1},…$a'_{j'}$, a_{j+1}…, a_t; p).

For levels between i and j, we can reduce the number of messages to be transmitted is reduced.

For other levels, the number of messages transmitted remains unchanged.

So the total number of messages needed to be sent is reduced.

Thus the T'''(a_1..a_{i-1},a'_i, a'_{i+1},…$a'_{j'}$, a_{j+1}…, a_t; p) is better than T(a_1, a_2,…a_t; p),

So the tree T(a_1, a_2,…a_t; p) is not an optimal tree, which contradicts the original assumption.

Lemma 5. Given a tree T(a_1; t; a_t ; p),

$$M(N,p)_{T(a_1;\,t;\,a_t;\,p)} = 2^{a_1}[1-(1-p)^N]+\sum_{i=2}^{t-1} 2^{a_1+2i-4}[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}]+N[1-(1-p)^{2^{a_t}}]$$

Proof:

For the top level, if the group key needs to be updated, 2^{a_1}messages need to be transmitted. The probability of the group key needing to be updated is 1-(1-*p*)^N. So the expected number is $2^{a_1}[1-(1-p)^N]$.

Similarly, for the level *i* (1< *i* <*t*) , if a key needs to be updates, 4 messages need to be transmitted. The probability of a key needing to be updated is $[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}]$. There are 2^{a_1+2i-4} keys at level *i*. So the expected number is $2^{a_1+2i-4}[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}]$

For the bottom level, if a key needs to be updates, 2^{a_t} messages need to be transmitted. The probability of a key needing to be updated is $[1-(1-p)^{2^{a_t}}]$. There are $N/2^{a_t}$ keys at this level. So the expected number is $N[1-(1-p)^{2^{a_t}}]$.

Thus the expected total number is

$$M(N,p)_{T(a_1;\ t;\ a_t;\ p)} = 2^{a_1}[1-(1-p)^N] + \sum_{i=2}^{t-1} 2^{a_1+2i-4}[1-(1-p)^{\frac{N}{2^{a_1+2i-2}}}] + N[1-(1-p)^{2^{a_t}}]$$