An Efficient Key Management Scheme for Live Streaming

Xingyu Li and H. Vicky Zhao
ECE Dept., University of Alberta, Edmonton, AB T6G 2V4 Canada
Email: xingyu@ualberta.ca, vzhao@ece.ualberta.ca

***Abstract*— To protect multimedia content in live streaming applications, it is of critical importance to employ effective access control and key management schemes and to prevent unauthorized access. Many live streaming applications experience a large variation in the user number, high reconnection rate, and a large number of short sessions. Such frequent membership update may introduce significant rekey overhead and poses challenges to efficient key management. This paper considers applications that can tolerate a small amount of content leak, explores the unique characteristics of live broadcast streaming in membership dynamics, and investigates efficient key management mechanisms. Our proposed scheme uses the reconnection algorithm, batch rekeying and grouped user placement to reduce the communication and the computation cost associated with key update. Our analytical and simulation results show that the proposed scheme can reduce more than 50% of the rekey messages, and helps decrease the number of rekey messages that a user needs to decrypt by more than 90%.**

I. Introduction

Recent advances in communication, networking and multimedia technologies have facilitated the distribution of multimedia over networks, and live streaming has become popular. For such group applications where the same multimedia data are distributed to a group of users simultaneously, information security and digital rights management are of crucial importance, and access control is often required to prevent unauthorized users from accessing multimedia content.

Group access control is usually achieved by encrypting the data with a key that is shared by all users in the group. For secure multicast applications, the encryption key should be updated when the group membership changes. This ensures that the joining/leaving user cannot access the previous/future content. Key management schemes can be categorized into centralized and contributory approaches. In the centralized approaches, a central key server (KDC) generates and distributes the encryption keys to all group members [1]–[4]. The contributory approaches are used when there is no central server available and every participant has an equal contribution to the generation and update of the encryption keys [5]–[7].

Different from generic multicast applications, live streaming has its unique characteristics in membership update. For example, the total number of users may vary drastically from time to time; a lot of users may temporarily leave the service due to network congestion and then come back later [8]–[10]. All these cause frequent membership change, and may introduce significant communication and computation overhead to update the encryptions keys.

In this paper, we address the challenging issue of frequent membership update, and investigate efficient key management for live broadcast streaming applications. We consider those applications that do not require immediate key update after every user join/leave and can tolerate a small amount of content leak. For those applications, we explore the membership dynamics and the specific user join/leave pattern, design efficient mechanisms to reduce the rekey overhead, and analyze their performance.

The rest of the paper is organized as follows. Section II introduces the system model, including the membership dynamics in live broadcast streaming and the ALX logic key tree. In Section III, we propose efficient key management schemes that uses three rekeying algorithms, the reconnection scheme, batch rekeying and grouped user placement, and analyze the performance. Simulation results are shown in Section IV, and conclusions are drawn in Section V.

II. System Model

A. Membership Dynamics in Live Streaming Applications

Prior work showed that live streaming applications share a few common characteristics in their membership dynamics.

1) *Large variation in the total number of users* is observed in many live streaming applications.

2) *Flash crowd* is the phenomenon that the peak arrival rate is much higher than the average arrival rate [9], [10]. It is observed in all short streams (which have pre-determined broadcast time and duration, such as NBA games or talk shows) and in 50% of the popular streams (whose largest number of concurrent users exceeds 1000). For example, a recent study in [9] showed that the peak arrival rate can be twice of the usual arrival rate.

3) *High reconnection rate* is observed in those programs that are both short and popular. In these programs, users may leave the service due to excessively long waiting time or poor network conditions, and reconnect to the service within a short duration. For example, in a recent live baseball game that was broadcasted by Yahoo! Japan, at the beginning of the game, in each unit time, a maximum of 200 users rejoined the program within 2 minutes after they left [8].

4) *A large number of short sessions* are observed, especially at the beginning of the streams. A recent work showed that for popular programs, over 55% of users stayed in the program for less than 5 minutes, and 30% of them stayed less than 1 minute in the program [10].

To summarize, the group membership may change drastically and frequently in live broadcast streaming applications, which poses challenges to efficient key management.

B. The ALX Logic Key Tree

Following the work in [3], we use the ALX logic key tree to achieve efficient key management. As shown in Figure 1, in an ALX tree with a total of $l+1$ levels, nodes in the upper l levels

978-1-4244-4148-8/09/$25.00 ©2009
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2009 proceedings.

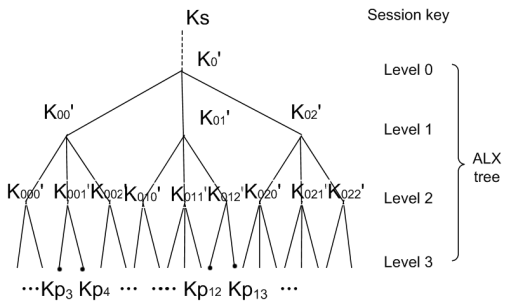

Fig. 1. An example of the ALX tree with $a = 3$ and $l = 2$.

(level 0 to level $l-1$) have a fixed degree a, while there is no constraint on the degree of nodes at level l [3]. Each leaf in the tree corresponds to a user's private key $\mathbf{K}_p$, which is shared by the corresponding user and key distribution center (KDC) only. The nodes at level 0 to level l correspond to the key encryption keys (KEKs), each of which is shared by users who are the descendants of the corresponding inner node. The node that is above the root of the tree corresponds to the session key $\mathbf{K}_s$, which is used to encrypt multimedia content. Each user has the session key $\mathbf{K}_s$ and all the keys that are on the path from his/her corresponding leaf node to the root. It was shown in [3] that the ALX tree avoids the unbalanced tree structure that may often happen in the fixed-degree logic key trees due to frequent membership update, which helps significantly reduce the rekeying overhead. This makes the ALX key tree an ideal candidate for key management in live broadcast streaming applications, where we observe large variations in the total number of users.

1) Key Update: In many secure multicast applications, it is often required to have forward and backward security to ensures that the joining/leaving user cannot access the previous/future content. To achieve forward and backward security, KDC needs to update the keys in the key tree when there is a membership update (a user join or a user leave), and securely sends the new keys to current users.

In the example in Figure 1, when user 4 joins the service, KDC puts him/her under the node labeled $\mathbf{K}'_{001}$ (besides user 3) in the key tree, and four keys, $\mathbf{K}_s$, $\mathbf{K}'_0$, $\mathbf{K}'_{00}$ and $\mathbf{K}'_{001}$, need to be updated. When there is a joining user, an efficient way to update keys is to use a secure one-way function $f(\cdot)$ [11] that is previously agreed upon by KDC and all users. For example, to update key $\mathbf{K}'_0$, KDC and all users calculate $\mathbf{K}'^{new}_0 = f(\mathbf{K}'^{old}_0)$, where $\mathbf{K}'^{old}_0$ and $\mathbf{K}'^{new}_0$ are the old and the new versions of the key $\mathbf{K}'_0$, respectively.

When user 13 leaves the service, 4 keys, $\mathbf{K}_s$, $\mathbf{K}'_0$, $\mathbf{K}'_{01}$ and $\mathbf{K}'_{012}$, need to be updated. To send the new key $\mathbf{K}'_{012}$ to user 12, KDC multicasts the encrypted message $\{\mathbf{K}'^{new}_{012}\}_{\mathbf{K}_{p12}}$. Here, $\{\mathbf{K}_1\}_{\mathbf{K}_0}$ means that the new key $\mathbf{K}_1$ is encrypted using key $\mathbf{K}_0$. To update key $\mathbf{K}'_{01}$, KDC sends three rekey messages $\{\mathbf{K}'^{new}_{01}\}_{\mathbf{K}'^{old}_{010}}$, $\{\mathbf{K}'^{new}_{01}\}_{\mathbf{K}'^{old}_{011}}$, and $\{\mathbf{K}'^{new}_{01}\}_{\mathbf{K}'^{new}_{012}}$, which only enable user 7 to 12 to find the new key $\mathbf{K}'^{new}_{01}$. Similarly, to securely update key $\mathbf{K}'_0$ and $\mathbf{K}'_s$, KDC multicasts 4 encrypted rekey messages, $\{\mathbf{K}'^{new}_0\}_{\mathbf{K}'^{old}_{00}}$, $\{\mathbf{K}'^{new}_0\}_{\mathbf{K}'^{new}_{01}}$, $\{\mathbf{K}'^{new}_0\}_{\mathbf{K}'^{old}_{02}}$ and $\{\mathbf{K}^{new}_s\}_{\mathbf{K}'^{new}_0}$. Thus, when user 13 leaves the service, a total of 8 rekey messages are needed to securely update the 4 keys that user 13 knows.

TABLE I
THE OPTIMAL VALUE OF l FOR DIFFERENT USER NUMBERS (k).

k	100-121	122-364	365-1093	1094-3280	3281-9000
l	3	4	5	6	7

From the above analysis, when a new user joins the service, KDC only needs to broadcast a message with the indices of all keys that need to be updated, and the one-way function is used to ensure all users have the updated keys. When a user leaves the service, to ensure forward security, KDC needs to send the updated keys securely to the remaining users using encryptions. Thus, the communication overhead and the computation cost to update keys with a leaving user are much higher than that when a user joins the service, and thus this paper focuses on the rekey overhead for leaving users only.

2) ALX Tree optimization: An important issue in the ALX logic key tree design is to select the optimal parameters a and l that minimize the number of rekey messages when a user leaves. Given a total of k users, assume that they are uniformly distributed in an ALX tree of degree a and level $l+1$, that is, all nodes at level l have approximately the same number of children. The analysis in [3] showed that the average number of rekey messages for one user's departure can be approximated by $C(k) = k/a^l + al$. For a given k, we enumerate all possible pairs of (a, l) and find the one that minimizes $C(k)$. We observe that $a = 3$ is the optimal tree degree when $100 \leq k \leq 9000$, and the optimal l for different ks are shown in Table I. From Table I, the optimal l remains the same for a large range of k and, therefore, the optimal ALX tree structure remains unchanged even when the total number of users changes significantly. This is a desired feature for live broadcast streaming applications, where the membership may change frequently and drastically.

C. Performance Criteria

In this paper, we explore the specific user join/leave pattern in live streaming applications and design efficient key management schemes. To evaluate the performance of the proposed schemes, we use the following criteria: C_{msg}, the average number of rekey messages sent by KDC per second, and C_{de}, the average number of rekey messages that a user needs to decrypt per second. Note that KDC's computation cost, which is the number of rekey messages that KDC needs to encrypt per second, is the same as C_{msg} (the average number of rekey messages sent by KDC per second). Thus, it is omitted here.

III. EFFICIENT KEY MANAGEMENT FOR LIVE BROADCAST STREAMING

In this work, we focus on key management of single-source live broadcast streaming networks. The source is assumed to be continuously available and has huge storage and communication resources. Following prior work [3], [9], we model users' arrival process as Poisson with mean λ, and we let each user's service duration follow an exponential distribution with mean $1/\mu$. We define α as the probability that a leaving user rejoins the same service later. For a rejoining user, we assume that the interval between his/her leave and rejoin follows an exponential distribution with mean $1/\upsilon$.

978-1-4244-4148-8/09/$25.00 ©2009
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2009 proceedings.

A. The Sequential Rekeying Scheme

We define sequential rekeying as the key management scheme that updates the keys immediately after a user join/leave and treats each reconnection as a user leave followed by an independent user join. It achieves strict forward and backward security, and provides an upper bound of the rekey overhead of key management schemes.

Assume that there are a total of $k \gg 1$ users in the service. Following the user join and service model that we consider, the average number of leaving users in a second approximately equals to $k\mu$, and the average number of arrivals per second is λ. We consider the scenario where $k\mu \ll k$, $\lambda \ll k$, and the change on the total number of users within a second can be neglected. In such a scenario, the average numbers of rekey messages for all user leaves in a second are roughly the same and approximately equal to $k/a^l + al$ [3]. Thus, we have $\mathbf{C}_{msg}^{seq} \approx k\mu\left(k/a^l + al\right)$.

To calculate C_{de}, we assume that k users are uniformly distributed in an ALX tree of degree a and level $l+1$. When user j leaves the service, all the $l+2$ keys that he/she knows (excluding j's private key) has to be updated. Let $\mathbf{K}'_{i,j}$ be the key on the ith level of the ALX tree that user j knows. Under the assumption of uniform user distribution in the key tree, averagely speaking, a total of k/a^i users, including j, share $\mathbf{K}'_{i,j}$. Thus, when user j leaves, the remaining $k/a^i - 1$ users need to decrypt the new key $\mathbf{K}'_{i,j}$. Considering all the $l+2$ keys that need to be updated when user j leaves, the average number of rekey messages that a user needs to decrypt per user leave can be approximated by $\left[\sum_{i=0}^{l}(k/a^i - 1) + (k-1)\right]/(k-1)$.

Given an average of $k\mu$ leaving users per second,

$$\begin{aligned}\mathbf{C}_{de}^{seq} &\approx \frac{k\mu}{k-1}\left[\sum_{i=0}^{l}(k/a^i - 1) + (k-1)\right] \\ &\approx \mu\left[\sum_{i=0}^{l}(k/a^i - 1) + (k-1)\right]. \qquad (1)\end{aligned}$$

B. The Reconnection Scheme

As discussed in Section II-A, high reconnection rate and large number of short sessions are observed in live streaming applications [8]. Sequential rekeying treats each reconnection as one leave and one independent user join to achieve strict forward security, which results in large rekeying overhead.

In reality, many live broadcast streaming programs do not require strict forward and backward security, and can allow content leak up to T_l seconds. (The value of T_l is determined by the content owner and is application dependent.) With the relaxed requirement on forward security, KDC allows a leaving user to keep his/her keys (and thus still be able to access the content) for up to T_l seconds after his/her leaving. In this scenario, if a user rejoins the service in less than T_l seconds after his/her leaving, KDC does not need to relocate the rejoining user to another leaf and to update the keys he/she previously had. Consequently, there is no cost associated with the rejoining user if he/she comes back within T_l seconds.

Similar to the work in [3], we use a wait-for-leaving (WFL) list to address the high reconnection rate. The WFL list records information of each leaving user, including the user's identity and his/her leaving time. We define a rejoining user's absent time as the duration between his/her leaving and his/her rejoining. A user is removed from the WFL list when his/her absent time is larger than T_l or when he/she rejoins the service within T_l seconds after his/her leaving. In this work, we assume that KDC has sufficient computation capability and storage space to maintain the WFL list.

Our proposed reconnection scheme is as follows:

1) When user j_1 leaves the service, KDC records j_1's identity and leaving time t_{j_1} in the WFL list while not removing j_1 from the ALX tree. KDC still allows j_1 to continue receiving the service in the interval from t_{j_1} to $t_{j_1} + T_l$. Then, KDC checks all other users in the WFL list, and removes from the list those whose absent times exceed T_l. For example, assume that user j_2 is in the WFL list at the current time t_{j_1}. If $t_{j_1} - t_{j_2} \geq T_l$, j_2 is removed from both the WFL list and the ALX tree, and KDC updates all the keys that j_2 knows. If $t_{j_1} - t_{j_2} < T_l$, KDC keeps j_2 in the ALX tree and the WFL list.

2) If no user leaves the service in the past second, KDC updates the WFL list and the ALX tree periodically (every second) in the same way as described above. This is to ensure that for each leaving user, the maximum content leak does not exceed T_l seconds.

3) When user j_3 joins the service, KDC first checks the WFL list. If j_3 is already in the WFL list, then j_3 is a rejoining user. KDC simply removes j_3's record from WFL list and does not update the ALX tree. Otherwise, KDC treats j_3 as a new user, and updates the keys using the one-way function.

For a given T_l, we call those rejoining users whose absent time is smaller than T_l "retry users". We use the term "departure users" to denote all other leaving users, including those who never come back and those rejoining users whose absent time is larger than T_l. In our model, each leaving user has a probability of α to rejoin the service later, and the absent time of a rejoining user follows an exponential distribution with mean $1/\upsilon$. So P^{re}, the probability that a leaving user rejoins the service within T_l seconds after he/she leaves, equals to $P^{re} = \alpha\int_0^{T_l} \upsilon e^{-t\upsilon}dt = \alpha(1 - e^{-T_l\upsilon})$.

Given a total of k users in the service, we consider the scenario where the total number of users is approximately the same within T_l seconds, and the change on k can be ignored. As such, the average number of departure users per second is $(1 - P^{re})k\mu$. Note that the proposed algorithm only removes the rekey redundancies associated with the retry users, while KDC updates all the keys that a departure user has T_l seconds after his/her leaving. Therefore,

$$\begin{aligned}\mathbf{C}_{msg}^{retry} &\approx (1 - P^{re})\, k\mu(k/a^l + al) = (1 - P^{re})\, \mathbf{C}_{msg}^{seq}, \\ \text{and } \mathbf{C}_{de}^{retry} &\approx (1 - P^{re})\frac{k\mu}{k-1}\left[\sum_{i=0}^{l}(k/a^i - 1) + (k-1)\right] \\ &= (1 - P^{re})\mathbf{C}_{de}^{seq}. \qquad (2)\end{aligned}$$

Compared with sequential rekeying, the proposed reconnection algorithm reduces the rekey overhead by a factor of P^{re}. In the extreme case of $P^{re} = 1$, that is, all leaving users rejoin the service within T_l seconds after they leave, the rekey overhead is zero and no keys need to be updated. Also, from

978-1-4244-4148-8/09/$25.00 ©2009

(2), when α and T_l increase, P^{re} is larger (that is, the average number of departure users per second is smaller), and the proposed reconnection scheme is more efficient.

C. The Batch Rekeying Scheme

A lot of live streaming applications have the flash crowd behavior and experience high membership update rate [8]–[10]. For such applications, immediate key update after each membership change may cause consecutive update of the same key within a short time interval, and sequential rekeying may introduce significant rekey overhead, as shown in [12].

For those applications that do not require strict forward security, we use batch rekeying proposed in [12] to address redundant update of the same keys. We still use sequential key update for user join such that a user can access the content immediately after he/she joins the service.[1] In this work, we apply batch rekeying to leaving users only, and KDC periodically updates keys known to leaving users at predefined time called "batch moments". We let the batch interval, which is the period between two consecutive batch moments, equal to T_l. It guarantees that the content that is leaked to a leaving user does not exceed the maximum allowed T_l seconds. Our proposed batch rekeying scheme is as follows:

- When user j_1 leaves the service, KDC puts him/her in the WFL list, records his/her identity, and keeps him/her in the ALX tree.
- When user j_2 joins the service, KDC first checks the WFL list. If j_2 is already in the WFL list, then j_2 is a rejoining user. KDC removes j_2 from WFL list and does not update the ALX tree. Otherwise, KDC treats j_2 as a new user, and updates the keys using the one-way function.
- At each batch moment, KDC checks the identities of all leaving users in the WFL list, identifies all the keys that they know, and updates all these keys together. Then, KDC clears the WFL list.

To analyze the performance of batch rekeying, we first consider one batch interval and analyze the number of leaving users who do not rejoin the service before the next batch moment. We consider the scenario where the total number of users k remains approximately the same within one batch interval. Following our model of the user join/leave pattern, in one batch interval of length T_l, the average number of joining users is approximately λT_l, and the average number of leaving users can be approximated by $k\mu T_l$.

For a rejoining user j, let t_j be the time interval between the previous batch moment and j's leaving time, and let τ_j be j's absent time (the time interval between j's leaving and j's rejoining). t_j and τ_j follow exponential distributions with means $1/\mu$ and $1/\upsilon$, respectively. For a leaving user j, the probability that he/she rejoins the service before the next batch moment is

$$\begin{aligned} P' &\approx \alpha P[t_j + \tau_j \le T_l] \\ &= \alpha\left(1 - \frac{\upsilon}{\upsilon-\mu}e^{-T_l\mu} + \frac{\mu}{\upsilon-\mu}e^{-T_l\upsilon}\right). \end{aligned} \quad (3)$$

Note that if a leaving user rejoins the service before the next batch moment, his/her leaving incurs zero rekey cost. Therefore, in each batch interval, the average number of departure users whose keys need to be updated at the next batch moment (including those who never come back and those who rejoin the service after the next batch moment) approximately equals to $k' \approx (1-P')T_l k\mu$.

1) Analysis of C_{msg}: Based on the above analysis of the number of departure users per batch interval, we first analyze the average number of rekey messages that KDC needs to send per second using batch rekeying.

Here, we consider the scenario, where k' departure users are uniformly distributed in the ALX tree of degree a and level $l+1$. This gives the upper bound of the rekey overhead of batch rekeying. In such a scenario, for a node at level $i < l+1$, an average of k'/a^i departure users are descendants of that node.

i) When $k'/a^i > 1$, that is, $0 \le i \le n$ where

$$n \approx \left\lfloor \log_a k' \right\rfloor \approx \left\lfloor \log_a \left[(1-P')T_l k\mu\right] \right\rfloor, \quad (4)$$

each node at level i has at least one descendant who is a departure user, and all the corresponding KEKs at level i have to be updated at the next batch moment. To update a KEK at level $0 \le i \le n$, KDC needs a encrypted rekey messages, and there are a total of $\sum_{i=0}^{n} a^i$ KEKs in the upper $n+1$ levels of the ALX tree that need to be updated. Consequently, the total rekey messages that are needed to update all KEKs in the upper $n+1$ levels are $C_m^{0,n} \approx a\sum_{i=0}^{n} a^i = \frac{a(1-a^{n+1})}{1-a}$.

ii) When $k'/a^i < 1$, that is, $n < i \le l$, some of the nodes at level i do not have any departure users in their descendants. Under the assumption of uniform distribution of the k' departure users in the tree, among all a^i nodes at level i, $k' < a^i$ of them have descendants that are departure users. Thus, at level $i > n$, k' KEKs need to be updated at the next batch moment. To update each KEK corresponding to a node at level $n+1 \le i \le l-1$, a rekeying messages are needed. To update the k' KEKs at level l, we use the users' private keys to encrypt the new keys. Therefore, $k/a^l - 1$ rekey messages are needed to update one KEK at level l, since each node at level l has approximately k/a^l children, including the departure user. Consequently, to update the keys at level $n+1$ to l, KDC needs to send $C_m^{n+1,l} \approx (1-P')T_l k\mu[a(l-n-1)+k/a^l-1]$ rekey messages.

Furthermore, one more rekey message is needed to update the session key $\mathbf{K}_s$. Therefore, $\overline{\mathbf{C}_{msg}^{batch}}$, the upper bound of the average number of rekey messages that KDC needs to send per second, can be approximated by

$$\begin{aligned} \overline{\mathbf{C}_{msg}^{batch}} &\approx \frac{1}{T_l}\left(C_m^{0,n} + C_m^{n+1,l} + 1\right) \quad (5) \\ &= (1-P')k\mu\left[a(l-n-1)+k/a^l-1\right] + \frac{1-a^{n+2}}{(1-a)T_l}, \end{aligned}$$

where n is in (4). From (5), $\overline{C_{msg}^{batch}}$ is a decreasing function of the batch interval T_l, and the average number of rekey messages per second is smaller when T_l is larger.

[1] Batch rekeying for joining users can also be applied to address the out-of-synchronization problem [12], and it will not affect our analysis. This is because a joining user incurs little rekey overhead that can be ignored.

978-1-4244-4148-8/09/$25.00 ©2009

2) Analysis of C_{de}: We next analyze the upper bound of the average number of rekey messages that each user needs to decrypt per second with batch rekeying.

Since all keys in the upper $n+1$ levels need to be updated, all remaining users need to decrypt and update the $n+1$ keys in the upper $n+1$ levels that he/she knows. Hence, in each batch interval, every user needs to decrypt $C_u^{0,n} = n+1$ rekey messages to update the KEKs in the upper $n+1$ levels. From the above analysis, for a KEK corresponding to a node at level $n+1 \le i \le l$ that needs to be updated, an average of k/a^i users, including the departure user, share this key. Thus, to update one KEK at level i, at the next batch moment, the remaining $k/a^i - 1$ users need to decrypt the corresponding decryption messages. Consequently, to update KEKs that correspond to nodes at level $n+1$ to l in the ALX tree, the average number of rekey messages that a user needs to decrypt per batch interval is

$$\begin{aligned} C_u^{n+1,l} &\approx \frac{(1-P')T_l k\mu \left[\sum_{i=n+1}^{l}(k/a^i - 1)\right]}{k} \\ &= (1-P')T_l\mu \left[\sum_{i=n+1}^{l}(k/a^i - 1)\right]. \end{aligned} \quad (6)$$

Furthermore, one more decryption is required for each remaining user to update the session key. Consequently, with batch rekeying, $\overline{\mathbf{C}_{de}^{batch}}$, the upper bound of the average number of rekey messages that one user needs to decrypt per second, can be approximated by

$$\begin{aligned} \overline{\mathbf{C}_{de}^{batch}} &\approx \frac{1}{T_l}\left(C_u^{0,n} + C_u^{n+1,l} + 1\right) \\ &= \frac{n+2}{T_l} + (1-P')\mu \left[\sum_{i=n+1}^{l}(k/a^i - 1)\right]. \end{aligned} \quad (7)$$

$\overline{C_{de}^{batch}}$ is a decreasing function of the batch interval T_l, and it takes a smaller value when T_l is larger.

D. Grouped User Placement in the ALX Tree

From the analysis in the previous section, the locations of leaving users in the ALX tree affect the performance of the key management scheme. In the extreme case where there is a leaving user in every branch of the ALX tree, all keys in the ALX tree have to be updated at the next batch moment, and it introduces the largest possible rekey overhead. On the contrary, if all leaving users are in the same or neighboring subtrees in the ALX tree, KDC only needs to update a few keys at the next batch moment, and the rekey cost is much smaller. Thus, to further reduce the rekey overhead, a possible solution is to strategically place joining users in the ALX tree such that leaving users in one batch interval are close to each other in the key tree.

Prior work has shown that there are many short sessions in live broadcast streaming applications, especially at the beginning of the programs [8], [10]. For such *short-stay users* who stay in the service for only a few minutes, if they come at approximately the same time, they also tend to leave at approximately the same or adjacent batch intervals. To reduce the rekey cost associated with short-stay users when they leave the service, we propose a grouped user placement scheme. For an ALX tree with level $l+1$, for two consecutive user joins, instead of randomly placing these two users in the key tree, the proposed scheme puts them under the same or adjacent nodes at level l. For those short-stay users who join and leave the service at approximately the same time, grouping them in the same or neighboring subtrees helps increase the number keys that they share and thus helps reduce the rekey overhead.

TABLE II
PARAMETERS OF THE USER JOIN/LEAVE MODEL

duration total 7200s	$\le$ 900s	901-6600s	>6600s
long-stay $(\lambda_i, 1/\mu_i)$	(3, 3500s)	(1, 1200s)	(0.25, 240s)
short-stay $(\lambda_i, 1/\mu_i)$	(3, 240s)	(1, 240s)	(0.25, 240s)

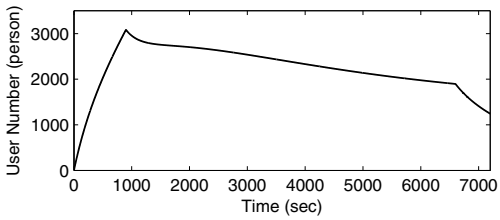

Fig. 2. An example of the user number in a short-duration live broadcast streaming application.

IV. SIMULATION RESULTS

In this section, we simulate the proposed key management schemes, and compare their performance with that of sequential rekeying. To quantify the amount of rekey redundancy reduced by the proposed algorithms, we define η as the ratio of the rekey overhead of the proposed scheme divided by that of sequential rekeying. For example, with the reconnection scheme, we let $\eta_{msg}^{retry} = \mathbf{C}_{msg}^{retry}/\mathbf{C}_{msg}^{seq}$ and $\eta_{de}^{retry} = \mathbf{C}_{de}^{retry}/\mathbf{C}_{de}^{seq}$. Same for batch rekeying and the grouped user placement scheme. When η takes a smaller value, the proposed scheme is more efficient.

In our simulations, we consider a popular short-duration live broadcast streaming of length 7200 seconds. Time is divided into three non-overlapping periods to address different user behavior at the beginning, in the middle, and at the end of the live streaming program. There are two types of users in our simulations. Half of the users are *short-stay* users who stay in the service for only a few minutes, and the other half are *long-stay* users who stay in the service until the end of the program. The parameters that we choose for different types of users and for different periods of the program are in Table II, and Figure 2 shows an example of the total number of users simulated based on the parameters in Table II.

From Figure 2, the total number of users changes from a few dozen to a maximum of 3000, and in the interval between 900s and 6600s, the total number of users is around 2000 to 3000. Following the discussion in Section II-B and Table I, we use an ALX key tree of degree $a = 3$ and $l = 6$ to maximize the performance.

Figure 3 shows the the analytical and the simulation results of the reconnection scheme. The maximum allowed content leak is fixed as $T_l = 120$ seconds in Figure 3 (a), and The reconnection scheme is more efficient when α is larger. η_{msg}^{retry} and η_{de}^{retry} drop from 95% to 82% when α increases from 0.1

978-1-4244-4148-8/09/$25.00 ©2009

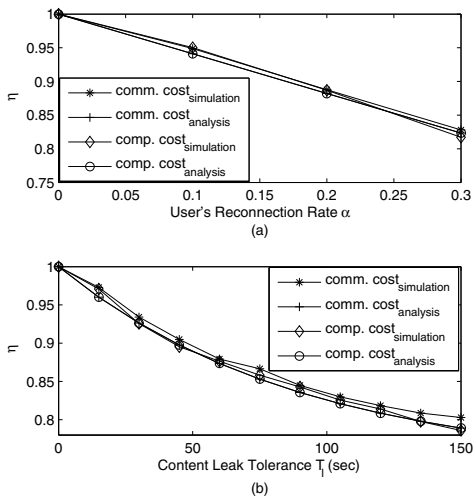

Fig. 3. Performance of the reconnection scheme. $1/\upsilon = 90$ seconds. (a): $T_l = 120$ sec, and (b): $\alpha = 0.25$.

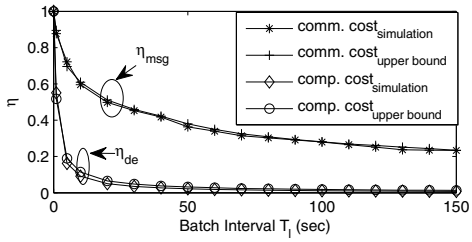

Fig. 4. Performance of the batch rekeying scheme. $1/\upsilon = 90$ seconds and $\alpha = 0.25$.

to 0.3. In Figure 3 (b), $\alpha = 0.25$ is fixed, and the reconnection scheme is more efficient when the maximum allowed content leak is larger. When T_l is increased from 40 seconds to 150 seconds, η_{msg}^{retry} and η_{de}^{retry} drop from 90% to 80%. This agrees with our analysis in Section III-B. Also, our analytical results match the simulation results in Figure 3.

Figure 4 shows the analytical and the simulation results of the batch rekeying scheme. In Figure 4, we fix $\alpha = 0.25$ and $1/\upsilon = 90$ seconds, and plot η_{msg} and η_{de} for different values of T_l. From Figure 4, batch rekeying can significantly reduce the rekey communication cost, and it is more efficient when the batch interval is larger. Batch rekeying can help reduce more than 50% of the rekey messages when $T_l \geq 20$ seconds, and η_{msg}^{batch} drops to 20% when $T_l = 150$ seconds. Also, when $T_l \geq 20$ seconds, user's decryption cost can be reduced by more than 90%, and $\eta_{de}^{batch} \approx 1\%$ when $T_l = 150$ seconds. From Fig. 4, the upper bounds that we derived in (5) and (7) are tight.

Figure 5 shows the performance of the grouped user placement scheme. Compared with random placement of joining users, grouped user placement can further help reduce the rekey cost, especially when the maximum allowed content leak T_l is small. For example, with $T_l = 20$ seconds, grouped user placement can help further reduce η_{msg} by another 5%, and η_{de} drops from 7% to 6%.

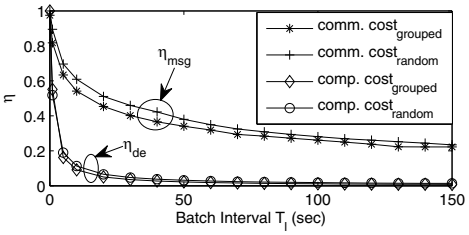

Fig. 5. Performance of the grouped user placement scheme. $1/\upsilon = 90$ seconds and $\alpha = 0.25$.

V. Conclusions

This paper explores the unique characteristics of live broadcast streaming in membership dynamics, investigates efficient key management schemes to reduce the rekey cost, and analyzes their performance. We consider applications that do not require strict forward security and can tolerate content leak up to a few minutes. For such applications, we use the reconnection scheme, batch rekeying and grouped user placement to address the high reconnection rate, the flash crowd phenomenon, and the large number of short sessions in live broadcast streaming. Both our analytical and simulation results show that the proposed schemes can significantly reduce the rekey cost. They can help reduce more than 50% of the rekey messages, and the computation cost for a user to decrypt the rekey messages is reduced by more than 90%.

References

[1] C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Tran. on Networking*, vol. 8, no. 1, pp. 16–30, Feb. 2000.

[2] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic group: One way function trees and amortized initialization," *Internet Draft,draft-irtf-smug-groupkeymgmt-oft-00.txt*, 2000.

[3] Y. Sun, W. Trappe, and K. J. R. Liu, "A scalable multicast key management scheme for heterogeneous wireless networks," *IEEE/ACM Tran. on Networking*, vol. 12, no. 4, pp. 653–666, Aug. 2004.

[4] F. Zhou, J. Xu, L. Lin, and H. Xu, "Multicast key management scheme based on toft," *IEEE 10th Int. Conf. on High Performance Computing and Comm.*, pp. 1030–1035, Sept. 2008.

[5] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to group communication," *Proc. 3rd ACM Conf. Computer and Communications Security*, pp. 31–37, 1996.

[6] Y. Mao, Y. Sun, M. Wu, and K. J. R. Liu, "Jet:dynamic join-exit-tree amortization and schedualing for contributory key management," *IEEE/ACM Tran. on Networking*, vol. 14, no. 5, pp. 1128–1540, Oct. 2006.

[7] K. Lu, Y. Qian, M. Guizani, and H. Chen, "A framework for a distributed key management scheme in heterogeneous wireless sensor networks," *IEEE Tran. on Wireless Comm.*, vol. 7, no. 2, pp. 639–647, Feb. 2008.

[8] B. Li, G. Y. Keung, S. Xie, F. Liu, Y. Sun, and H. Yin, "An empirical study of flash crowd dynamics in a p2p-based live video streaming system," *IEEE "GLOBECOM" Proc.2008*, Nov.-Dec. 2008.

[9] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analisis of live streaming workload on the internet," *Internet Measurement Conference 2004*, pp. 41–45, Oct. 2004.

[10] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supperting large-scale live streaming applications with dynamic application end-points," *ACM Special Interest Group on Data Communication 2004*, Aug.-Sept. 2004.

[11] W. Trappe and L. C. Washington, *Introduction to Cryptography with Coding Theory*, 2nd ed. Prentice Hill, 2005.

[12] X. S. Li, Y. R. Yang, M. G. Gouda, and S. S. Lam, "Batch rekeying for secure group communications," *ACM 10th International World Wide Web Conference*, pp. 525–534, May 2001.

978-1-4244-4148-8/09/$25.00 ©2009