Theoretical Computer Science 410 (2009) 1013–1021

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

# Approximately optimal trees for group key management with batch updates[☆]

Minming Li[a,*], Ze Feng[a], Nan Zang[b], Ronald L. Graham[b], Frances F. Yao[a]

[a] *Department of Computer Science, City University of Hong Kong, Hong Kong*

[b] *Department of Computer Science and Engineering, University of California at San Diego, United States*

ARTICLE INFO

*Keywords:*
Key management
Key trees
Batch update
Approximation algorithms
Optimality

ABSTRACT

We investigate the group key management problem for broadcasting applications. Previous work showed that, in handling key updates, batch rekeying can be more cost effective than individual rekeying. One model for batch rekeying is to assume that every user has probability $p$ of being replaced by a new user during a batch period with the total number of users unchanged. Under this model, it was recently shown that an optimal key tree can be constructed in linear time when $p$ is a constant and in $O(n^4)$ time when $p \to 0$. In this paper, we investigate more efficient algorithms for the case $p \to 0$, i.e., when membership changes are sparse. We design an $O(n)$ heuristic algorithm for the sparse case and show that it produces a nearly 2-approximation to the optimal key tree. Simulation results show that its performance is even better in practice. We also design a refined heuristic algorithm and show that it achieves an approximation ratio of $1 + \epsilon$ for any fixed $\epsilon > 0$ and $n$, as $p \to 0$. Finally, we give another approximation algorithm for any $p \in (0, 0.693)$ which is shown to be quite good by our simulations.

## 1. Introduction

With the increase of subscription-based network services, strategies for achieving secure multicast in networks are becoming more important. For example, to limit the service access to the authorized subscribers only, mechanisms such as content encryption and selective distribution of decryption keys have been found useful. One can regard the secure multicast problem as a group broadcast problem, where we have $n$ subscribers and a group controller (GC) that periodically broadcasts messages (e.g., a video clip) to all subscribers over an insecure channel. To guarantee that only the authorized users can decode the contents of the messages, the GC will dynamically maintain a key structure for the whole group. Whenever a user leaves or joins, the GC will generate some new keys as necessary and notify the remaining users of the group in some secure way. Surveys of key management for secure group communications can be found in [1,2].

In this paper, we consider the key tree model [3] for the key management problem. We describe this model briefly as follows (a precise formulation is given in Section 2). Every leaf node of the key tree represents a user and stores his individual key. Every internal node stores a key shared by all leaf descendants of that internal node. Every user possesses all the keys along the path from the leaf node (representing the user) to the root. To prevent revoked users from knowing future message

☆ This work was supported in part by the National Basic Research Program of China Grant 2007CB807900, 2007CB807901, the US National Science Foundation Grant CCR-0310991, grants from the Research Grants Council of Hong Kong under Project Number CityU 122105, 1165/04E and a grant from City University of Hong Kong (Project no. 9610063).

* Corresponding author.
*E-mail addresses:* minmli@cs.cityu.edu.hk (M. Li), fengze@cs.cityu.edu.hk (Z. Feng), nzang@cs.ucsd.edu (N. Zang), graham@ucsd.edu (R.L. Graham), csfyao@cityu.edu.hk (F.F. Yao).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.10.034

contents and also to prevent new users from knowing past message contents, the GC updates a subset of keys, whenever a new user joins or a current user leaves, as follows. As long as there is a user change among the leaf descendants of an internal node $v$, the GC will (1) replace the old key stored at $v$ with a new key, and (2) broadcast (to all users) the new key encrypted with the key stored at each child node of $v$. Note that only users corresponding to the leaf descendants of $v$ can decipher useful information from the broadcast. Furthermore, this procedure must be done in a bottom-up fashion (i.e., starting with the lowest $v$ whose key must be updated—see Section 2 for details) to guarantee that a revoked user will not know the new keys. The cost of the above procedure counts the number of encryptions used in step (2) above (or equivalently, the number of broadcasts made by the GC).

When users change frequently, the method for updating the group keys whenever a user leaves or joins may be too costly. Thus, a batch rekeying strategy was proposed by Li et al. in [4], whereby rekeying is done only periodically instead of immediately after each membership change. It was shown by simulation that among the totally balanced key trees (where all internal nodes of the tree have branching degree $2^i$), degree 4 is the best when the number of requests (leave/join) within a batch is not large. For a large number of requests, a star (a tree of depth 1) outperforms all such balanced key trees. Further work on the batch rekeying model was done by Zhu et al. in [5]. They introduced a new model where the number of joins is assumed to be equal to the number of leaves during a batch updating period and every user has probability $p$ of being replaced by a new user for some $p$. They studied the optimal tree structure subject to two restrictions: (A) the tree is totally balanced, and (B) every node on level $i$ has $2^{k_i}$ children for some parameter $k_i$ depending on $i$. Under these restrictions, characterizations of the optimal key tree were given together with a construction algorithm.

Recently, Graham, Li, and Yao [6] studied the structure of the true optimal key tree when restrictions (A) and (B) are both removed. They proved that, when $p > 1 - 3^{-1/3} \approx 0.307$, the optimal tree is an $n$-star. When $p \leq 1 - 3^{-1/3}$, they proved a constant upper bound 4 for the branching degree of any internal node $v$ other than the root, and also an upper bound of $-4/\log q$, where $q = 1 - p$, for the size of the subtree rooted at $v$. By using these characterizations, they designed an $O(n^4)$ algorithm for computing the optimal key tree for $n$ users. The running time of their algorithm is in fact linear when $p$ is a fixed constant, but becomes $O(n^4)$ when $p$ approaches 0. Although polynomial, the $O(n^4)$ complexity is still too costly for large scale applications. Indeed, the case $p \to 0$ (when user changes are sparse) is a realistic scenario in many applications. In this paper, we investigate more efficient heuristics for the sparse case. As shown in [6], degree 3 is quite favored in the optimal tree as $p \to 0$. In fact, their results implied that, for $n = 3^t$, the optimal key tree is a balanced ternary tree, and for many other values of $n$, the optimal tree is as close to a balanced ternary tree with $n$ leaves as possible, subject to some number-theoretical properties of $n$.

In this paper, we investigate how closely a "simple" ternary tree can approximate the optimal tree for arbitrary $n$. We propose a heuristic *LR* which constructs a ternary tree in a left to right manner and prove that it gives a nearly 2-approximation to the optimal tree. Simulation results show that the heuristic performs much better than the theoretical bound we obtain. We then design a refined heuristic *LB* whose approximation ratio is shown to be $1 + \epsilon$ as $p \to 0$ where $\epsilon$ can be made arbitrarily small. We also generalize the heuristic *LR* to a heuristic GLR for the case when $p \in (0, 1 - 3^{-1/3})$.

The rest of the paper is organized as follows. In Section 2, we describe the batch update model in detail. In Section 3, we establish a lower bound for the optimal tree cost as $p \to 0$. The lower bound is useful for obtaining performance ratios for our approximation algorithms. In Section 4, we describe the heuristic LR and analyze its performance against the optimal key tree; some simulation results are also given. Then we design a refined heuristic LB and analyze its performance in Section 5. In Section 6, we generalize the heuristic LR for $p$ in a more general range, and show that the generalized LR (GLR) performs well by simulation results. Finally, we summarize our results and mention some open problems in Section 7.

## 2. Preliminaries

Before giving a precise formulation of the key tree optimization problem to be considered, we briefly discuss its motivation and review the basic key tree model for group key management. This model is referred to in the literature either as a key tree [3] or LKH (logical key hierarchy) [7].

In the key tree model, there are a Group Controller (GC), represented by the root, and $n$ subscribers (or users) represented by the $n$ leaves of the tree. The tree structure is used by the GC for key management purposes. Associated with every node of the tree (whether internal node or leaf) is an encryption key. The key associated with the root is called the Traffic Encryption Key (TEK), which is used by the subscribers for accessing encrypted service contents. The key $k_v$ associated with each non-root node $v$ is called a Key Encryption Key (KEK), which is used for updating the TEK when necessary. Each subscriber possesses all the keys along the path from the leaf representing the subscriber to the root.

In the batch update model to be considered, only simultaneous join/leave is allowed; that is, whenever there is a revoked user, a new user will be assigned to that vacant position. This assumption is justified since, in a steady state, the number of joins and departures would be roughly equal during a batch processing period. To guarantee forward and backward security, a new user assigned to a leaf position will be given a new key by the GC and, furthermore, all the keys associated with the ancestors of the leaf must be updated by the GC. The updates are performed from the lowest ancestor upward for security reasons. We then explain the updating procedure together with the updating cost in the following.

The GC first communicates with each new subscriber separately to assign a new key to the corresponding leaf. After that, the GC will broadcast certain encrypted messages to all subscribers in such a way that each valid subscriber will know all the new keys associated with its leaf-to-root path while the revoked subscribers will not know any of the new keys. The

GC accomplishes this task by broadcasting the new keys, in encrypted form, from the lowest level upward as follows. Let $v$ be an internal node at the lowest level whose key needs to be (but has not yet been) updated. For each child $u$ of $v$, the GC broadcasts a message containing $E_{k_u^{new}}(k_v^{new})$, which means the encryption of $k_v^{new}$ with the key $k_u^{new}$. Thus the GC sends out $d_v$ broadcast messages for updating $k_v$ if $v$ has $d_v$ children. Updating this way ensures that the revoked subscribers will not know any information about the new keys (as long as they do not know the new key $k_u^{new}$ in a lower level, they cannot get the information of the new key $k_v^{new}$ in a higher level) while current subscribers can use one of their KEKs to decrypt the useful $E_{k_u^{new}}(k_v^{new})$ sequentially until they get the new TEK.

We adopt the probabilistic model introduced in [5] that each of the $n$ positions has the same probability $p$ to independently experience subscriber change during a batch rekeying period. Under this model, an internal node $v$ with $N_v$ leaf descendants will have probability $1 - q^{N_v}$ that its associated key $k_v$ requires updating, where $q = 1 - p$. The updating incurs $d_v \cdot (1 - q^{N_v})$ expected broadcast messages by the procedure described above. We thus define the expected updating cost $C(T)$ of a key tree $T$ by $C(T) = \sum_v d_v \cdot (1 - q^{N_v})$, where the sum is taken over all the internal nodes $v$ of $T$. It is more convenient to remove the factor $d_v$ from the formula by associating the weight $1 - q^{N_v}$ with each of $v$'s children. This way we express $C(T)$ as a node weight summation: for each non-root tree node $u$, its node weight is defined to be $1 - q^{N_v}$, where $v$ is $u$'s parent. The optimization problem we are interested in can now be formulated as follows.

**Optimal key tree for batch updates:** We are given two parameters, $0 \leq p \leq 1$ and $n > 0$. Let $q = 1 - p$. For a rooted tree $T$ with $n$ leaves and node set $V$ (including internal nodes and leaves), define a weight function $w(u)$ on $V$ as follows. Let $w(r) = 0$ for root $r$. For every non-root node $u$, let $w(u) = 1 - q^{N_v}$, where $v$ is $u$'s parent. Define the cost of $T$ as $C(T) = \sum_{u \in V} w(u)$. Find a $T$ for which $C(T)$ is minimized. We say that such a tree is $(p, n)$-*optimal*, and denote its cost by $\mathrm{OPT}(p, n)$.

## 3. Lower bound for optimal tree cost as $p \to 0$

In a tree $T$ with $n$ leaves, denote the set of leaf nodes as $L(T)$, and for each leaf $u$, let the set of ancestor nodes of $u$ (including $u$ itself) be denoted by $\mathrm{Anc}(u)$. To obtain a lower bound for the optimal tree cost, we first rewrite $C(T)$ as

$$C(T) = \sum_{u \in V} w(u) = \sum_{u \in V} N_u \cdot \frac{w(u)}{N_u} = \sum_{u \in L(T)} \sum_{x \in \mathrm{Anc}(u)} \frac{w(x)}{N_x} = \sum_{u \in L(T)} c(u),$$

where we define $c(u) = \sum_{x \in \mathrm{Anc}(u)} \frac{w(x)}{N_x}$. In other words, we distribute the weight $w(u)$ associated with every node $u \in V$ evenly among its leaf descendants, and then sum the cost over all the leaves of $T$.

Let the path from a leaf $u$ to the root $r$ be $p_0 p_1 \ldots p_{k-1} p_k$, where $p_0 = u$ and $p_k = r$. Note that $c(u) = \sum_{i=0}^{k-1} \frac{1 - q^{N_{p_{i+1}}}}{N_{p_i}}$ is uniquely determined by the sequence of numbers $\{N_{p_0}, N_{p_1}, \ldots, N_{p_k}\}$, where $N_{p_0} = 1$ and $N_{p_k} = n$. We will thus extend the definition of $c$ to all such sequences $\{a_0, a_1, \ldots, a_k\}$, and analyze the minimum value of $c$.

**Definition 1.** Let $S_n$ denote any sequence of integers $\{a_0, a_1, \ldots, a_k\}$ satisfying $1 = a_1 < a_2 < \cdots < a_k = n$. We call $S_n$ an $n$-progression. Define $c(p, S_n)$ to be $c(p, S_n) = \sum_{i=1}^{k} \frac{1 - q^{a_i}}{a_{i-1}}$, and let $F(p, n)$ be the minimum of $c(p, S_n)$ over all $n$-progressions $S_n$. For $n = 3^t$, the special $n$-progression $\{1, 3, 9, \ldots, 3^{t-1}, 3^t\}$ will be denoted by $S_n^*$.

Thus, we have $C(T) \geq n \cdot F(p, n)$ for any tree $T$ with $n$ leaves, and hence

$$\mathrm{OPT}(p, n) \geq n \cdot F(p, n). \tag{1}$$

Next we focus on properties of $c(p, S_n)$ and $F(p, n)$. First, we derive the following monotone property for $F(p, n)$.

**Lemma 2.** $F(p, n) < F(p, n + 1)$.

**Proof.** Suppose $S_{n+1} = \{1, a_1, a_2, \ldots, a_{k-1}, n + 1\}$ is the optimal $(n + 1)$-progression that achieves the value $F(p, n + 1)$. Let $S_n = \{1, a_1, a_2, \ldots, a_{k-1}, n\}$. Because $\frac{1 - q^{n+1}}{a_{k-1}} > \frac{1 - q^n}{a_{k-1}}$, we know that $c(p, S_{n+1}) > c(p, S_n)$. By definition, we have $F(p, n) \leq c(p, S_n)$. Combining these two facts, we have $F(p, n) < F(p, n + 1)$. □

For a given $n$-progression $S_n = \{1, a_1, a_2, \ldots, a_{k-1}, n\}$, the slope of $c(p, S_n)$ at $p = 0$ is denoted by $\lambda_{S_n}$ and can be expressed as $\lambda_{S_n} = \sum_{i=0}^{k-1} \frac{a_{i+1}}{a_i}$, where $a_0 = 1$ and $a_k = n$. The minimum $c(p, S_n)$ as $p \to 0$ will be achieved by those $S_n$ with $c(p, S_n)$ having the smallest slope at $p = 0$. We next prove the following lemma.

**Lemma 3.** *When $n = 3^t$, the $n$-progression $S_n^* = \{1, 3, 9, \ldots, 3^{t-1}, 3^t\}$ satisfies the following relation: $\lambda_{S_n} \geq 0.995 \cdot \lambda_{S_n^*}$ for any $S_n$.*

**Proof.** For positive numbers $b_1, b_2, \ldots, b_k$, we have $\sum_{i=1}^{k} b_i \geq k(\prod_{i=1}^{k} b_i)^{\frac{1}{k}}$. Therefore, $\lambda_{S_n} \geq k n^{\frac{1}{k}}$ if $S_n$ consists of $k$ numbers. We now estimate a lower bound for $f(k) = k n^{\frac{1}{k}}$ when $n = 3^t$. Consider $g(k) = \log_3 f(k) = \frac{\ln k}{\ln 3} + \frac{t}{k}$. Notice that $g'(k) = \frac{1}{k \ln 3} - \frac{t}{k^2}$. Therefore, we have $g'(k) < 0$ when $k < t \ln 3$ and $g'(k) > 0$ when $k > t \ln 3$. This implies

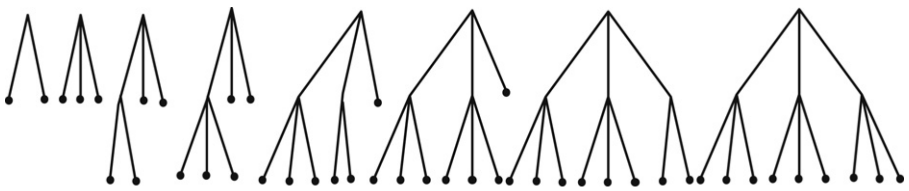


**Fig. 1.** Trees generated by LR for $n = 2$–$9$.

that $g(k)$, and hence $f(k)$, is minimized when $k = t\ln 3$. Therefore, we have $f(k) \geq f(t\ln 3) = t3^{\frac{1}{\ln 3}}\ln 3$, which implies $\lambda_{S_n} \geq t3^{\frac{1}{\ln 3}}\ln 3$. On the other hand, we know that $\lambda_{S_n^*} = f(t) = 3t$. Hence we have

$$\frac{\lambda_{S_n}}{\lambda_{S_n^*}} \geq \frac{f(t\ln 3)}{f(t)} = 3^{\frac{1+\ln\ln 3}{\ln 3}-1} \approx 0.995025 \geq 0.995. \quad \square$$

We obtain the following theorem from the above analysis.

**Theorem 4.** *For* $3^t \leq n < 3^{t+1}$, *we have* $\mathrm{OPT}(p, n) \geq 0.995 \cdot n \cdot c(p, S_{3^t}^*)$ *when* $p \to 0$.

**Proof.** This is a direct consequence of Lemmas 2 and 3 and inequality (1). $\square$

## 4. Heuristic LR and its approximation ratio

We design the heuristic LR as follows. LR maintains an almost balanced ternary tree (i.e., the depth of any two leaves differ by at most 1) in which at most one internal node has degree less than 3. Moreover, LR adds new leaves incrementally in a left to right order. Fig. 1 shows the tree we get by using LR for $n = 2, \ldots, 9$. We can also recursively build a key tree using LR in the following way. For a tree with $n \geq 3$ leaves, the number of leaves in the root's three subtrees is decided by the table below, while for a tree with 2 leaves, the tree structure is a root with two children (a star).

| | No. of leaves (Left) | (Middle) | (Right) |
|---|---|---|---|
| $3^t \leq n < 5\cdot 3^{t-1}$ | $n - 2\cdot 3^{t-1}$ | $3^{t-1}$ | $3^{t-1}$ |
| $5\cdot 3^{t-1} \leq n < 7\cdot 3^{t-1}$ | $3^t$ | $n - 4\cdot 3^{t-1}$ | $3^{t-1}$ |
| $7\cdot 3^{t-1} \leq n < 3^{t+1}$ | $3^t$ | $3^t$ | $n - 2\cdot 3^t$ |

We denote the tree with $n$ leaves constructed by LR as $T_n$. Note that this heuristic only needs linear time to construct a ternary tree. Furthermore, the structure of the ternary tree can be decided in $\log n$ time, because, every time we go down the tree, there is at most one subtree whose number of leaves is not a power of 3 and needs further calculation.

Let $\mathrm{LR}(p, n)$ denote the cost of the ternary tree constructed by LR for given $n$ and $p$. To obtain an upper bound for $\mathrm{LR}(p, n)$, we first prove the following lemmas.

**Lemma 5.** *The inequality* $\mathrm{LR}(p, n) < \mathrm{LR}(p, n+1)$ *holds for all* $n > 0$ *and* $0 < p < 1$.

**Proof.** We view $C(T_n)$ as the node weight summation given in Section 2 and compare the cost of the corresponding nodes $w(u)$ and $w(u')$ in $T_n$ and $T_{n+1}$, respectively. Due to the addition of one leaf node, if $w(u) = 1 - q^k$, then $w(u') = w(u)$ or $w(u') = 1 - q^{k+1}$. Therefore we have $w(u) \leq w(u')$. There are also additional weights associated with nodes that appear in $T_{n+1}$ but not in $T_n$. This proves the lemma. $\square$

**Lemma 6.** *For any integer* $t > 0$ *and* $0 < q < 1$, *we have* $\frac{1-q^{3^t}}{3^{t-1}} > \frac{1-q^{3^{t+1}}}{3^t}$.

**Proof.** Note that $3\sum_{i=1}^{3^t} q^{i-1} > (1 + q^{3^t} + q^{2\cdot 3^t})\sum_{i=1}^{3^t} q^{i-1} = \sum_{i=1}^{3^{t+1}} q^{i-1}$. The lemma is proved by multiplying $\frac{1-q}{3^t}$ on both sides. $\square$

**Lemma 7.** *For any integer* $t > 0$ *and* $0 < q < 1$, *we have*

$$\mathrm{LR}(p, 3^{t+1}) < 3\left(1 + \frac{1}{t}\right)\mathrm{LR}(p, 3^t).$$

**Proof.** By Lemma 6 and the definitions, we have $c(p, S_{3^t}^*)/t > c(p, S_{3^{t+1}}^*)/(t+1)$. Therefore, we have

$$\mathrm{LR}(p, 3^{t+1}) = 3^{t+1}\cdot c(p, S_{3^{t+1}}^*) < 3\left(1 + \frac{1}{t}\right)\cdot 3^t \cdot c(p, S_{3^t}^*) = 3\left(1 + \frac{1}{t}\right)\mathrm{LR}(p, 3^t). \quad \square$$

Now we are ready to prove the first approximation ratio.

**Theorem 8.** *When* $p \to 0$, *we have* $\mathrm{LR}(p, n) < 3.015(1 + \frac{1}{\lfloor \log_3 n\rfloor})\mathrm{OPT}(p, n)$.

**Proof.** Suppose $3^t \leq n < 3^{t+1}$. We claim the following:

$$\begin{aligned}
\mathrm{LR}(p,n) &< \mathrm{LR}(p,3^{t+1}) \\
&< 3\left(1+\frac{1}{t}\right)\mathrm{LR}(p,3^t) \\
&= 3\left(1+\frac{1}{t}\right)3^t \cdot c(p,S^*_{3^t}) \\
&\leq 3.015\left(1+\frac{1}{t}\right)\mathrm{OPT}(p,n).
\end{aligned}$$

The first inequality is implied by Lemma 5 and the second one by Lemma 7. The last inequality holds due to Theorem 4. □

In the above discussion, we use the smallest balanced ternary tree with no fewer than $n$ leaves as an upper bound for $\mathrm{LR}(p,n)$. By adding a small number of leaves instead of filling the whole level, we can obtain a better approximation ratio, which is shown below.

We divide the integers in the range $(3^t, 3^{t+1}]$ into three consecutive subsets of equal size $H = \frac{3^{t+1}-3^t}{3}$ as follows:

$$P_1 = (3^t, 3^t + H], \qquad P_2 = (3^t + H, 3^t + 2H], \qquad P_3 = (3^t + 2H, 3^{t+1}].$$

For any $n \in P_i$, we can use $\mathrm{LR}(p,n')$, where $n' = \max P_i$ to upper bound the value of $\mathrm{LR}(p,n)$ by Lemma 5. Let $\Delta_t = \mathrm{LR}(p,3^t) - \mathrm{LR}(p,3^{t-1})$ and define $a = 1 - q^{3^{t+1}}$. Notice that

$$\mathrm{LR}(p,3^{t+1}) = 3a + 3\cdot\mathrm{LR}(p,3^t) = 3a + 3\Delta_t + 3\cdot\mathrm{LR}(p,3^{t-1}).$$

It is not hard to verify the following inequalities based on the definition of the tree cost:

$$\begin{aligned}
&\mathrm{LR}(p, 7\cdot 3^{t-1}) < \mathrm{LR}(p,3^{t+1}) - \Delta_t, \\
&\mathrm{LR}(p, 5\cdot 3^{t-1}) < \mathrm{LR}(p,3^{t+1}) - 2\Delta_t.
\end{aligned}$$

We now derive a lower bound for the value of $\Delta_t$.

**Lemma 9.** *For* $0 < p < 1$, *we have* $\Delta_t \geq \frac{1}{6}\cdot\mathrm{LR}(p,3^{t+1})$.

**Proof.** We only need to prove $\Delta_t > a + \mathrm{LR}(p,3^{t-1})$. By the definition of $\Delta_t$, we know that $\Delta_t = 2\cdot\mathrm{LR}(p,3^{t-1}) + 3(1-q^{3^t})$. Then by using Lemma 6, we have $3(1-q^{3^t}) \geq (1-q^{3^{t+1}})$, which implies $\mathrm{LR}(p,3^{t-1}) + 3(1-q^{3^t}) \geq (1-q^{3^{t+1}})$. Therefore, we have $\Delta = 2\cdot\mathrm{LR}(p,3^{t-1}) + 3(1-q^{3^t}) > \mathrm{LR}(p,3^{t-1}) + a$. □

By making use of Lemma 9, we can obtain the following theorem on the performance of LR.

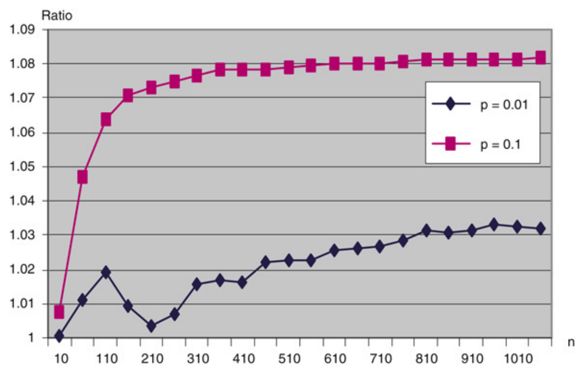
**Theorem 10.** *When* $p \to 0$, *we have* $\mathrm{LR}(p,n) < 2.01(1+\frac{1}{\lfloor \log_3 n\rfloor})\mathrm{OPT}(p,n)$.

**Proof.** We prove the theorem using Lemma 9 and similar arguments used in Theorem 8. The discussion below is divided into three cases according to the value of $n$.
Case (A). $3^t < n \leq 5\cdot 3^{t-1}$.

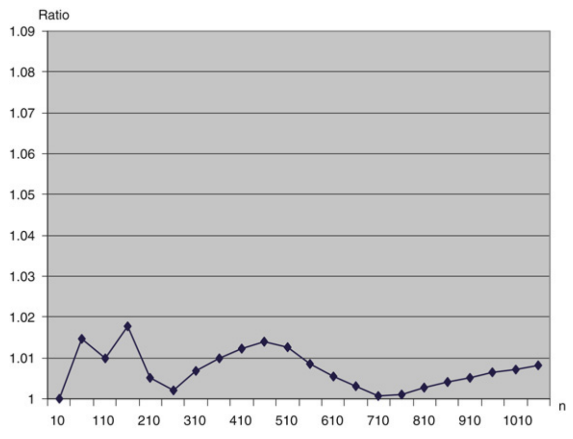
$$\begin{aligned}
\mathrm{LR}(p,n) &< \mathrm{LR}(p, 5\cdot 3^{t-1}) \\
&< \frac{2}{3}\mathrm{LR}(p,3^{t+1}) \\
&< \frac{2}{3}\cdot 3.015\left(1+\frac{1}{t}\right)\cdot\frac{3^t}{n}\cdot\mathrm{OPT}(p,n) \\
&\leq 2.01\left(1+\frac{1}{\lfloor \log_3 n\rfloor}\right)\cdot\mathrm{OPT}(p,n).
\end{aligned}$$

Case (B). $5\cdot 3^{t-1} < n \leq 7\cdot 3^{t-1}$.

$$\begin{aligned}
\mathrm{LR}(p,n) &< \mathrm{LR}(p, 7\cdot 3^{t-1}) \\
&< \frac{5}{6}\mathrm{LR}(p,3^{t+1}) \\
&< \frac{5}{6}\cdot 3.015\left(1+\frac{1}{t}\right)\cdot\frac{3^t}{n}\cdot\mathrm{OPT}(p,n) \\
&< \frac{5}{6}\cdot 3.015\left(1+\frac{1}{t}\right)\cdot\frac{3}{5}\cdot\mathrm{OPT}(p,n) \\
&< 2.01\left(1+\frac{1}{\lfloor \log_3 n\rfloor}\right)\cdot\mathrm{OPT}(p,n).
\end{aligned}$$



(a) $p = 0.1$ and $p = 0.01$.



(b) $p = 0.01/n^3$.

**Fig. 2.** Simulation results on $\text{Ratio}(p, n)$ for $p = 0.1$, $p = 0.01$ and $p = 0.01/n^3$.

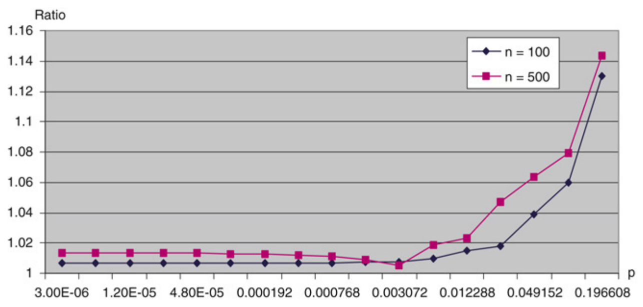


**Fig. 3.** Simulation results for fixed $n$.

Case (C). $7 \cdot 3^{t-1} < n \leq 3^{t+1}$.

$$\begin{aligned}
\text{LR}(p, n) &< \text{LR}(p, 3^{t+1}) \\
&< 3.015\left(1 + \frac{1}{t}\right) \cdot \frac{3^t}{n} \cdot \text{OPT}(p, n) \\
&< 3.015\left(1 + \frac{1}{t}\right) \cdot \frac{3}{7} \cdot \text{OPT}(p, n) \\
&< 2.01\left(1 + \frac{1}{\lfloor \log_3 n \rfloor}\right) \cdot \text{OPT}(p, n). \quad \square
\end{aligned}$$

We run simulations on the performance of LR for various values of $p$ and $n$. Define $\text{Ratio}(p, n) = \text{LR}(p, n)/\text{OPT}(p, n)$. Fig. 2(a) shows $\text{Ratio}(p, n)$ as a function of $n$ for $p = 0.1$ and $p = 0.01$ respectively. Within the simulation range, we see that, for the same $n$, $\text{Ratio}(p, n)$ is larger when $p$ is larger. For $p = 0.1$ the maximum ratio within the range is below 1.09. We then simulated the performance of LR when $p \to 0$ as a function of $n$. Fig. 2(b) shows $\text{Ratio}(p, n)$ when we set $p = 0.01/n^3$. We found the maximum ratio reached within the simulation range to be less than 1.018. Fig. 3 plots $\text{Ratio}(p, n)$ as a function of $p$ while $n$ is chosen to be $n = 100$ and $n = 500$, respectively. Notice that each curve has some dips and is not monotonically increasing with $p$.

## 5. Heuristic LB and its approximation ratio

In this section, we consider a more refined heuristic LB (Level Balance) and show that it has an approximation ratio of $1 + \epsilon$ as $p \to 0$ (for fixed $n$).

The heuristic LB adopts two different strategies for adding a new leaf to a tree when $n$ grows from $3^t$ to $3^{t+1}$. Let $T'_n$ denote the ternary tree with $n$ leaves constructed by LB. When $3^t < n \leq 2 \cdot 3^t$, the heuristic LB changes the leftmost leaf on level $t$ in $T'_{n-1}$ into a 2-star (i.e., an internal node with two leaf children); when $2 \cdot 3^t < n \leq 3^{t+1}$, the heuristic LB changes the leftmost 2-star on level $t$ in $T'_{n-1}$ into a 3-star (an internal node with three leaf children).

In the following discussion, we refer to the initial formulation of $C(T)$ as a node weight summation over all nodes: $C(T) = \sum_{u \in V} w(u)$. Define the slope of $C(T)$ at $p = 0$ as $\beta_T$ and let $t = \lfloor \log_3 n \rfloor$. When $T'_{3^t}$ changes incrementally to

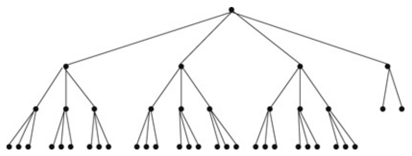


**Fig. 4.** Approximate tree structure generated by GLR when $n = 30$ $p = 0.1$.

$T'_n$, where $3^t < n \le 2 \cdot 3^t$, every time the size is increased by 1, we change a single leaf node to a 2-star. Therefore, in all intermediate levels (except for the root and the bottom level), exact three nodes will undergo a weight change: it changes from $1 - q^{N_v}$ to $1 - q^{N_v+1}$. For the two newly added nodes, the weight of each node is $1 - q^2$. Altogether, these changes contribute an increase of $3t + 2 \cdot 2$ to the slope $\beta_{T'_n}$. When $T'_{2\cdot 3^t}$ is changed incrementally to $T'_n$, where $2 \cdot 3^t < n \le 3^{t+1}$, by using a similar argument, we conclude that every time the size is increased by 1, all weight changes together contribute an increase of $3t + 9 - 4 = 3t + 5$ to the slope $\beta_{T'_n}$, where $9 - 4$ means the slope increase due to $3(1 - q^3) - 2(1 - q^2)$ (because we change a 2-star to a 3-star). To summarize, our new heuristic has the following property.

$$\beta_{T'_n} = \begin{cases} \beta_{T'_{n-1}} + 3t + 4 & \text{if } 3^t < n \le 2 \cdot 3^t, \\ \beta_{T'_{n-1}} + 3t + 5 & \text{if } 2 \cdot 3^t < n \le 3^{t+1}. \end{cases} \tag{2}$$

Using the base value of $\beta_{T'_n}$ and the recurrence relation (2), we have the following lemma.

**Lemma 11.**

$$\beta_{T'_n} = \begin{cases} (3t + 4)n - 4 \cdot 3^t & \textit{if } 3^t < n \le 2 \cdot 3^t, \\ (3t + 5)n - 6 \cdot 3^t & \textit{if } 2 \cdot 3^t < n \le 3^{t+1}. \end{cases}$$

By Lemma 2 and Theorem 4, we can prove the following theorem.

**Theorem 12.** *When $p \to 0$, we have $C(T'_n) < 1.005(1 + \frac{1}{\lfloor \log_3 n \rfloor})\mathrm{OPT}(p, n)$.*

**Proof.** To compare the cost of two trees as $p \to 0$ is in fact comparing slopes of the corresponding $C(T)$ at $p = 0$. Note that, for a full ternary tree $T$ with height $t$, the slope of $C(T)$ at $p = 0$ equals $3t \cdot 3^t$. Always let $t = \lfloor \log_3 n \rfloor$ in the following. Using Lemma 4, we can prove the theorem by proving $\beta_{T'_n} < (1 + \frac{1}{t}) \cdot 3t \cdot n$ as follows.
Case (A). $n = 3^t + r$, where $0 < r \le 3^t$. In this case, we have

$$\begin{aligned} \beta_{T'_n} &= 3t \cdot 3^t + r(3t + 4) \\ &< 3t \cdot 3^t + r(3t + 3) + 3 \cdot 3^t \\ &= (3t + 3)(3^t + r) \\ &= \left(1 + \frac{1}{t}\right) \cdot 3t \cdot n. \end{aligned}$$

Case (B). $n = 2 \cdot 3^t + r$, where $0 < r \le 3^t$. In this case, we have

$$\begin{aligned} \beta_{T'_n} &= 3t \cdot 3^t + 3^t \cdot (3t + 4) + r(3t + 5) \\ &= 2 \cdot 3t \cdot 3^t + 4 \cdot 3^t + 3t \cdot r + 5r \\ &\le 2 \cdot 3t \cdot 3^t + 6 \cdot 3^t + 3t \cdot r + 3r \\ &= (3t + 3)(2 \cdot 3^t + r) \\ &= \left(1 + \frac{1}{t}\right) \cdot 3t \cdot n. \quad \square \end{aligned}$$

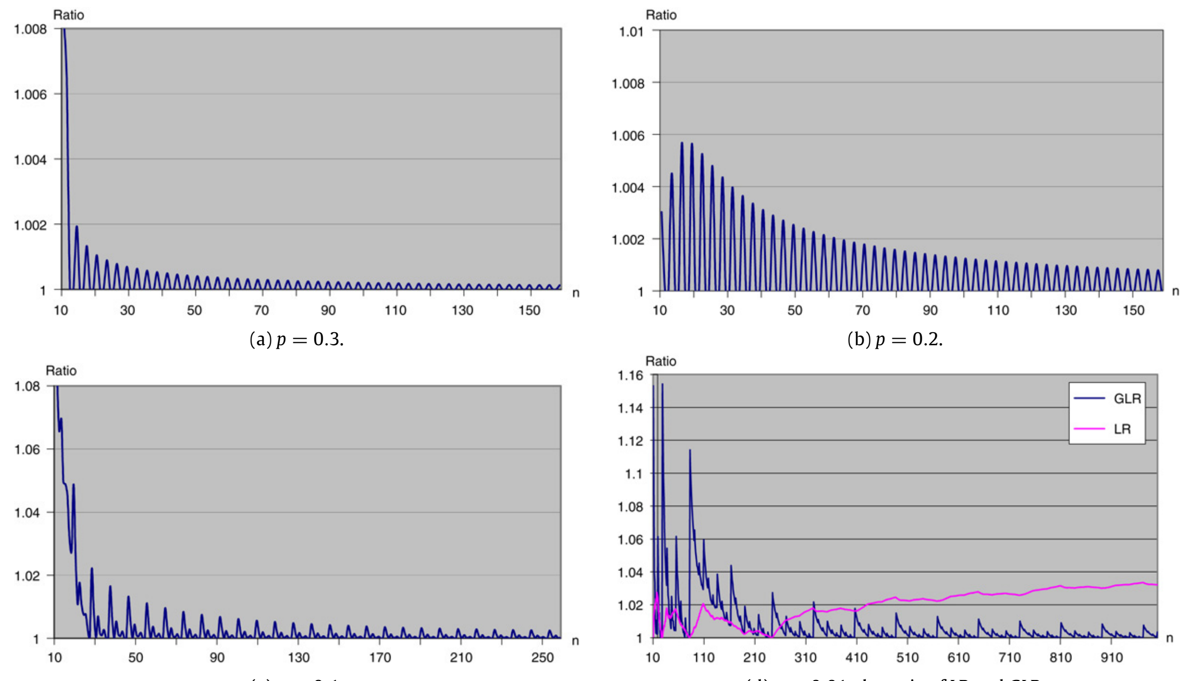
The upper bound in Theorem 12 can be further improved to $1+\epsilon$, where $\epsilon$ can be made arbitrarily small when $p \to 0$. To accomplish this, we make use of the following technical lemma, which was originally proved in [6] for a different purpose.

**Lemma 13.** *For any tree $T$ with $n$ leaves, we have*

$$\beta_T \ge \begin{cases} (3t + 4)n - 4 \cdot 3^t & \textit{if } 3^t < n \le 2 \cdot 3^t, \\ (3t + 5)n - 6 \cdot 3^t & \textit{if } 2 \cdot 3^t < n \le 3^{t+1}. \end{cases}$$

By comparing Lemmas 11 and 13 we can deduce that, for the tree $T'_n$ constructed by the heuristic LB, the slope of $C(T'_n)$ is equal to the lower bound of the slope of the optimal key tree with $n$ leaves. Therefore, as $p \to 0$, the approximation ratio of LB can be arbitrarily close to 1. This leads to the following theorem.

**Theorem 14.** *For any fixed $n$, we have $C(T'_n) < (1 + \epsilon)\mathrm{OPT}(p, n)$ when $p \to 0$.*



(a) $p = 0.3$.

(b) $p = 0.2$.

(c) $p = 0.1$.

(d) $p = 0.01$, the ratio of LR and GLR.

**Fig. 5.** Simulation results on Ratio$(p, n)$ for $p = 0.3$, $p = 0.2$, $p = 0.1$, and $p = 0.01$.

## 6. Generalized heuristic LR

The construction method of the heuristic LR can be extended to the case when $p$ is in a more general range. In [6], it was shown that a star is the optimal tree when $p \in (1 - (1/3)^{1/3}, 1)$. Here, for a given value $p \in (0, 1 - (1/3)^{1/3})$ and a given number of leaves $n$, we show how to construct an approximate tree GLR$(q, n)$ (where $q = 1 - p$). We found by simulations that the approximate tree performs very well.

The idea of the construction is as follows. First, we find a "good" complete ternary tree; then the root of the GLR$(q, n)$ is forced to contain as many such "good" complete ternary trees as possible, with one "leftover" tree. We will give some definitions before formally introducing the heuristic.

**Definition 15.** When $n = 3^t$, we define a special group of $n$-progressions, $SG(n) = \{T_n^d = \{1, 3, 9, \ldots, 3^d, 3^t\} | d = 0, 1, \ldots, t - 1\}$. For $n = 3^t$, we denote $c_n^*(q) = \min\{c(q, T_n^d) | d = 0, \ldots, t - 1\}$ and let $d^*(q)$ be the value of $d$ when $c_n^*(q)$ is achieved.

For example, when $n = 27$, $SG(27) = \{\{1, 27\}, \{1, 3, 27\}, \{1, 3, 9, 27\}\}$. $c_{27}^*(q) = \min\{1 - q^{27}, 1 - q^3 + \frac{1-q^{27}}{3}, 1 - q^3 + \frac{1-q^9}{3} + \frac{1-q^{27}}{9}\}$.

When $q \in [(1/3)^{1/3}, 1)$, $c(q, T_{3^t}^0)$ is $1 - q^{3^t}$, which is always larger than $c(q, T_{3^t}^1) = 1 - q^3 + \frac{1}{3}(1 - q^{3^t})$. We observe that $c_{3^t}^*(q)$ can be calculated recursively, $c_{3^t}^*(q) = 1 - q^3 + \frac{1}{3}c_{3^{t-1}}^*(q^3)$, when $t > 1$ and $q \in [0.693, 1)$. There are two base cases: one is $c_3^*(q) = 1 - q^3$; the other is $c_{3^t}^*(q) = 1 - q^{3^t}$ when $q \in (0, (1/3)^{1/3})$, because the optimal tree is a star when $q \in (0, 0.693)$ [6]. From the recursive formula, we can get the value of $d^*(q) = \min\{\lfloor \log_3(\log_q(\frac{1}{3}))\rfloor, t - 1\}$, where $t = \lceil \log_3 n\rceil$.

For a given $q$ and $n$, we construct an approximate tree GLR$(q, n)$ as follows: first we calculate the value $d^*(q) = \min\{\lfloor \log_3(\log_q(\frac{1}{3}))\rfloor, t - 1\}$, where $t = \lceil \log_3 n\rceil$, and construct a complete ternary tree $T^*(q)$ of depth $d^*(q)$; the root of GLR$(q, n)$ contains $n_1 + 1$ subtrees: $n_1$ of them are $T^*(q)$ and one is GLR$(q, L_1)$, where $n_1 = \lfloor \frac{n}{3^{d^*(q)}} \rfloor$ and $L_1 = n \mod 3^{d^*(q)}$.

Take $n = 30$ and $q = 0.9$ as an example: $d^*(0.9) = \min\{\lfloor \log_3(\log_{0.9}(\frac{1}{3}))\rfloor, \lceil \log_3 n\rceil - 1\} = 2$. The "good" complete ternary tree has $3^{d^*} = 9$ leaves; the root has four subtrees, three of which are complete ternary trees with 9 leaves and one of which is a "leftover" subtree GLR$(0.9, 2)$, which can be constructed recursively. The structure of GLR$(0.9, 30)$ is shown in Fig. 4.

We ran simulations on the performance of GLR for various values of $p$ and $n$. We redefine Ratio$(p, n)$ = GLR$(p, n)$/OPT$(p, n)$. Fig. 5 shows Ratio$(p, n)$ as a function of $n$ for $p = 0.3, 0.2, 0.1$, and $0.01$. Within the simulation range, we see that, for a fixed value $p$, the Ratio curve oscillates, but tends to 0 as $n$ gets larger. For $p = 0.1$, the maximum ratio is below 1.02, when $n$ is in the range $[50, 250]$. As $p \to 0$, both the heuristic *LR* and the heuristic GLR maintain almost balanced ternary trees, but they deal with the "leftover" leaves in two different ways: *LR* puts all the "leftover" leaves at the bottom level of the complete ternary tree; GLR bundles all the "leftover" leaves as a "leftover" tree and puts it as a sibling of the complete ternary trees. In Fig. 5(d), we compare the approximation ratios of *LR* and GLR when $p = 0.01$ and $n = 1..1000$. From the simulation results, we see that when $n$ is large, GLR appears to perform better.

## 7. Conclusions

In this paper, we consider the group key management problem for broadcasting applications. In particular, we focus on the case when membership changes are sparse. Under the assumption that every user has probability $p$ of being replaced by a new user during a batch rekeying period, the previously available algorithm requires $O(n^4)$ time to build the optimal key tree as $p \to 0$. We design a linear-time heuristic LR to construct an approximately good key tree and analyze its performance as $p \to 0$. We prove that LR produces a nearly 2-approximation to the optimal key tree. Simulation results show that LR performs much better than the theoretical bound we obtain. We also design a refined heuristic LB whose approximation ratio is shown to be $1 + \epsilon$ as $p \to 0$. At the end of this paper, we extended the heuristic LR to the algorithm GLR for the case when $p$ is in a more general range $p \in (0, 0.693)$ (In [6], it was shown that a star is the optimal tree when $p \in (0.307, 1)$.) Simulation results show that GLR performs very well as $n$ increases.

Some interesting problems remain open. Although simulation results show GLR performs very well for $p \in (0, 0.693)$, we are not able to prove a constant approximation ratio for this range. In order to prove a constant bound, one needs to understand better the mathematical structure of the optimal $n$-progression $S_n$ that achieves the value $F(p, n)$ for arbitrary $p$ and $n$.

## References

[1] S. Rafaeli, D. Hutchison, A survey of key management for secure group communication, ACM Computing Surveys 35 (3) (2003) 309–329.
[2] M.T. Goodrich, J.Z. Sun, R. Tamassia, Efficient tree-based revocation in groups of low-state devices, in: Proceedings of CRYPTO, 2004.
[3] C.K. Wong, M.G. Gouda, S.S. Lam, Secure group communications using key graphs, IEEE/ACM Transactions on Networking 8 (1) (2003) 6–30.
[4] X.S. Li, Y.R. Yang, M.G. Gouda, S.S. Lam, Batch re-keying for secure group communications, WWW10, Hong Kong, May 2–5, 2001.
[5] F. Zhu, A. Chan, G. Noubir, Optimal tree structure for key management of simultaneous join/leave in secure multicast, in: Proceedings of MILCOM, 2003.
[6] R.L. Graham, M. Li, F.F. Yao, Optimal tree structures for group key management with batch updates, SIAM Journal on Discrete Mathematics 21 (2) (2007) 532–547.
[7] D. Wallner, E. Harder, R.C. Agee, Key management for multicast: Issues and architectures, RFC 2627, June 1999.