SCIENCE CHINA
Information Sciences

• HIGHLIGHT •

October 2017, Vol. 60 108301:1–108301:3
doi: 10.1007/s11432-015-1045-1

A novel batch-based LKH tree balanced algorithm for group key management

Jie XU*, Linke LI, Sibian LU & Huayun YIN

Center for Cyber Security, School of Communication and Information Engineering, University of Electronic Science and Technology of China, Chengdu 611731*, China*

Received November 25, 2016; accepted December 29, 2016; published online April 27, 2017

Citation Xu J, Li L K, Lu S B, et al. A novel batch-based LKH tree balanced algorithm for group key management. Sci China Inf Sci, 2017, 60(10): 108301, doi: 10.1007/s11432-015-1045-1

Logical key hierarchy (LKH) is the most suitable key management scheme for a large dynamic group [1, 2]. Ref. [3] proposed LKH key management scheme, but it did not discuss how to keep tree balanced when membership changes. Ref. [4] proposed LTM (LKH Tree Manager) algorithm, but it has high relocating cost. Ref. [5] proposed a tree balanced algorithm, but had not given out detailed steps. Ref. [6] proposed batch balanced algorithm (BBA), but it causes high cost for reconstructing the tree. In this paper, we propose three merging algorithms and a novel tree balanced algorithm.

Definitions and notations. Node deletion: in a binary key tree, when a node lacks of left subtree or right subtree, it will be replaced by its remaining subtree. Height of node: we define $H_{\text{Max\_nodex}}$ and $H_{\text{Min\_nodex}}$ as the maximum height and the minimum height of a node x. Their values are equal to the maximum length and the minimum length from the node x to its child nodes, respectively. If the node has no child node, its height is 1. Height of tree: we define $H_{\text{Max\_tree}}$ and $H_{\text{Min\_tree}}$ as its maximum height and its minimum height. Their values are equal to the maximum height and the minimum height of its root node, respectively. Balanced node: if $H_{\text{Max\_nodex}} - H_{\text{Min\_nodex}} \leqslant 1$, the node x can be called a balanced node. Balanced key tree: in the key tree, if the root node is a balanced node, it can be called a balanced key tree. Min height subtree: in a balanced key tree, if a node x has $H_{\text{Max\_nodex}} = H_{\text{Min\_nodex}}$, and its parent node p has $H_{\text{Max\_nodep}} \neq H_{\text{Min\_nodep}}$, then this node x and all its child nodes make up a minimum height subtree.

Trees merging algorithms. Assume that there are two balanced binary key trees, named them as tree1 and tree2. In this paper, we suppose $H_{\text{Max\_tree1}} \geqslant H_{\text{Max\_tree2}}$.

Trees merging algorithm-1. When $H_{\text{Max\_tree1}} - H_{\text{Max\_tree2}} \leqslant 1$, we only need to create a new node and make the tree1 and the tree2 as the new node's left subtree and right subtree.

Trees merging algorithm-2. When $H_{\text{Max\_tree1}}$ and $H_{\text{Max\_tree2}}$ are more than 1, the criteria for choosing trees merging algorithm-2 is that the maximum and minimum height of higher merging tree are equal, i.e., $H_{\text{Max\_tree1}} = H_{\text{Min\_tree1}}$. In this case, we merge tree1 and tree2 as the following steps.

Step 1. We define $h = H_{\text{Max\_tree1}} - H_{\text{Max\_tree2}}$.

Step 2. Search for a node at the level h of tree1. If there is a key to be updated, mark this node; else from left to right, mark the last node at the level h.

Step 3. Create a new key node to replace the marked node. Then it makes the tree2 and the subtree composed by the mark node and its entire children nodes as the new node's left subtree and

* Corresponding author (email: xuj@uestc.edu.cn)

The authors declare that they have no conflict of interest.

right subtree.

Trees merging algorithm-3. When $H_{\text{Max\_tree1}} - H_{\text{Max\_tree2}} \succ 1$ is more than 1, the criteria for choosing trees merging algorithm-3 is that the maximum and minimum height of higher merging tree are not equal, i.e., $H_{\text{Max\_tree1}} \neq H_{\text{Min\_tree1}}$.

Set_M denotes a set that contains all the min height subtrees of tree1. Set_T denotes a set that contains the trees that will be merged into tree1. Initially, there is only tree2. Num_M denotes the total number of min height subtrees in Set_M. H_i denotes the height of the min height subtree in Set_M. MinTree_H_i denotes min height subtree whose height is H_i. H_MergeTree denotes the highest tree in the Set_T.

In this algorithm, merge tree1 and tree2 as the following steps.

Step 1. The algorithm finds out all the min height subtrees of the tree1 and records these min height subtrees into Set_M. At the same time, the algorithm classifies these min height subtrees by their height, there is $H_1 \geqslant H_2 \geqslant \cdots \geqslant H_{\text{Num\_}M}$.

Step 2. The algorithm finds out the H_MergeTree in Set_T. Then it compares $H_{\text{Max\_}H\text{\_MergeTree}}$ with the height of these subtrees in Set_M, there are three cases.

Case I. When $H_{\text{Max\_}H\text{\_MergeTree}} = H_i$ $(1 \leqslant i \leqslant \text{Num\_}M)$, firstly, the algorithm merges H_MergeTree and MinTree_H_i with trees merging algorithm-1. Then the MinTree_H_i in the tree1 is replaced by the new merged tree. Finally, H_MergeTree is deleted from Set_T.

Case II. When $H_{\text{Max\_}H\text{\_MergeTree}} \prec H_i$ $(1 \leqslant i \leqslant \text{Num\_}M)$ and $H_{\text{Max\_}H\text{\_MergeTree}} \succ H_{i+1}$ (if $i \neq \text{Num\_}M$), if $H_i - H_{\text{Max\_}H\text{\_MergeTree}} \succ 1$, the algorithm merges H_MergeTree and MinTree_H_i with the tree merging algorithm-2; else it merges them with the trees merging algorithm-1. Then the MinTree_H_i in the tree1 is replaced by the new merged tree. Finally, H_MergeTree is deleted from Set_T.

Case III. When $H_{\text{Max\_}H\text{\_MergeTree}} \succ H_1$, in this case, H_MergeTree cannot be merged into tree1 directly, which needs to be split into smaller subtrees. Firstly, the algorithm searches for a balanced subtree in H_MergeTree, and this subtree must satisfy that its maximum height is equal to H_1. Here it is named as H_MergeTree_H_1. Then, the algorithm marks all the nodes on the path from the parent node of H_MergeTree_H_1 to the root of H_MergeTree. Then it removes all the marked nodes. After removal, there will be some remaining subtrees except H_MergeTree_H_1, so these remaining subtrees will be recorded into Set_T. Finally, the algorithm merges H_MergeTree_H_1 and MinTree_H_1 with tree merging algorithm-1. Then the MinTree_H_1 in the tree1 is replaced by the new merged tree. Besides, H_MergeTree is deleted from Set_T.

Step 3. The algorithm goes to the step 1 until Set_T is null.

Trees merging algorithms. We define N_{J} and N_{D} as the number of joining and departing members respectively.

Case I, $N_{\text{J}} \succ N_{\text{D}}$. Firstly, the algorithm selects N_{D} members from the joining members to replace the departing members. Then the algorithm groups all the remaining joining members and the initial key tree into one balanced key tree. Here, the remaining joining members are treated as single-node subtrees. Every time, the algorithm selects two shortest subtrees and merges them into a new balanced key tree, until all the remaining joining members and the initial key tree are merged into one key tree. Finally, the group center (GC) generates the rekeying and relocating messages and broadcasts them to the members.

Case II, $N_{\text{J}} = N_{\text{D}}$. The algorithm replaces the departing members with the joining members directly. Then, the GC generates the rekeying messages and broadcasts them to the members.

Case III, $N_{\text{J}} \prec N_{\text{D}}$. Step 1, the algorithm selects N_{J} departing members and replaces them with the joining members. Step 2, the algorithm deletes all the departing members which are not replaced in the last step 1. Step 3, node deletion in the step 2 may cause the key tree to be unbalanced, so the algorithm needs to visit the key tree from down to top and from right and left to check whether every node is balanced. When there is an unbalanced node, the algorithm will utilize the trees merging algorithms to merge its left subtree and right subtree into a new balanced subtree, and then the unbalanced node will be replaced by the new merged balanced subtree. Step 4, The GC generates rekeying messages and relocating messages, and broadcasts them to the members.

Simulation results. Rekeying cost denotes the total number of rekeying messages that needs to be sent to all authorized group members in order to inform them the new group key. From Figure 1(a) and (b), the results show that LTM's rekeying cost is much higher than other two algorithms', the BBA and our balanced algorithm have approximate rekeying cost.

Relocating cost denotes the total number of the relocating messages that need to be sent to all the moved members in order to inform them their new location. From Figure 1(c), the results show that our algorithm has the lowest relocating cost.

Reconstructing cost denotes the total number of nodes that needs to be altered when member-

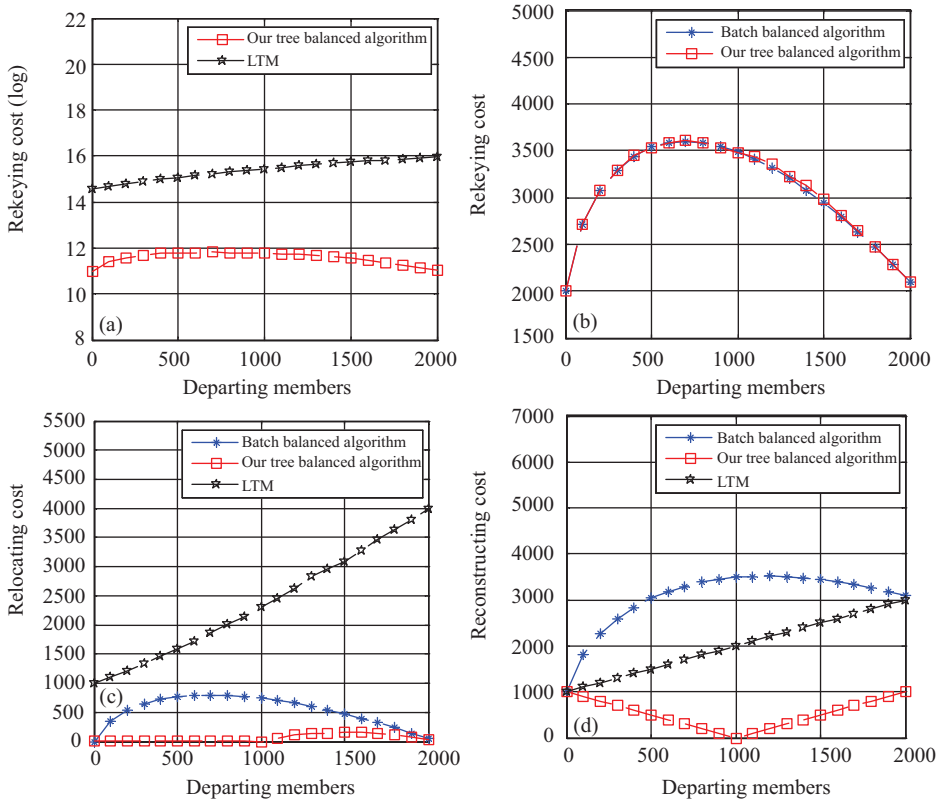

Figure 1 (Color online) Rekeying cost, relocating cost and reconstructing cost of our algorithm, LTM and BBA. (a) Rekeying cost (log); (b) rekeying cost; (c) relocating cost; (d) reconstructing cost.

ship changes. The lower reconstructing cost means the GC needs to alter fewer nodes to reconstruct a key tree based on the initial key tree. From Figure 1(d), the results show that the BBA has the highest reconstructing cost, and our tree balanced algorithm has the lowest reconstructing cost.

Conclusion. In this paper, in order to ensure the balance of the key tree when batch members join or depart, we propose three merging algorithms for merging two balanced key trees into a new one. Additionally, based on these three merging algorithms, we propose a tree balanced algorithm. This tree balanced algorithm can maintain a key tree balanced in major case. The simulation results show that its rekeying cost, relocating cost and reconstructing cost are lower than typical algorithms.

Acknowledgements This work was supported by National Key Research and Development Program (973 Program) (Grant No. 2016YFB0800100), National High Technology Research and Development Program of China (863 Program) (Grant No. 2015AA01 6102), Sichuan Province Scientific and Technological Support Project (Grant Nos. 2014GZ0017, 2016GZ0093), National Natural Science Foundation of China (Grant No. 61201128), and Fundamental Research Funds for the Central Universities (Grant No. ZYGX2015J009).

References

1 Sakamoto N. An efficient structure for LKH key tree on secure multicast communications. In: Proceedings of IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Las Vegas, 2014. 1–7
2 Teng J K, Wu C K, Tang C M, et al. A strongly secure identity-based authenticated group key exchange protocol. Sci China Inf Sci, 2015, 58: 092108
3 Wong C K, Gouda M, Lam S S. Secure group communications using key graphs. IEEE/ACM Trans Netw, 2000, 8: 16–30
4 Kwak D W, Lee S J, Kim J W, et al. An efficient LKH tree balancing algorithm for group key management. IEEE Commun Lett, 2006, 10: 222–224
5 Liu Z H, Lai Y X, Ren X B. An efficient LKH tree balancing algorithm for group key management. In: Proceedings of International Conference on Control Engineering and Communication Technology, Shenyang, 2012. 1003–1005
6 Ng W H D, Howarth M, Sun Z, et al. Dynamic balanced key tree management for secure multicast communications. IEEE Trans Comput, 2007, 56: 590–605