Design of a High Performance Implementation of a Tree-based Multicast Key Distribution Protocol

Heba Kamal Aslan[1] and Ghada Farouk Elkabbany[2]
(Corresponding author: Heba Kamal Aslan)

Informatics Dept., Electronics Research Institute, 11-a Helmi Hussein St., Manial, Cairo, Egypt[1]
Informatics Dept., Electronics Research Institute, Cairo, Egypt[2]
(E-mail: {hebaaslan, ghada_kabbany}@yahoo.com)

(Received Nov. 16, 2011; revised and accepted Feb. 1, 2012)

Abstract

The key distribution problem is considered one of the most important issues for providing secure multicast communication. Logical Key Hierarchy (LKH) protocols are considered one of the best solutions proposed for solving the scalability of multicast key distribution protocols. The use of LKH protocols reduces the computation complexity cost from *O(n)*, where *n* is the number of the whole group members to *O(log(n))*. In the present paper, a design of a high performance protocol for securing multicast communication is proposed. The proposed protocol is based on the idea of organizing the keys in a tree as in LKH protocols. In order to achieve lower computation overhead, the proposed protocol uses a multi-processor system. It has to be noted that LKH protocol relies heavily on one central point, therefore, it represents a single point of failure and for a large tree; the server's throughput can represent a bottleneck. The use of multiple processors could solve this problem and enhance the server's throughput. The proposed protocol is analyzed according to: the number of processors, the tree height and the tree degree. The analysis shows that the use of multiprocessor system will enhance the system performance which is considered an important factor for both real time and wireless applications.

Keywords: Centralized approaches and multiprocessor System, group communication, logical key hierarchy, multicast key distribution

1 Introduction

The key distribution problem is considered one of the most important issues for providing secure communication. Many protocols that address the key distribution problem for unicast or point-to-point communication [7, 9] have been studied. Extending these protocols for applications (such as teleconference, pay per view, collaborative work, …etc.) which are based on group communication is not a practical solution. In group communication, all group members share one key called group key. This key must be securely delivered to the group members; one solution to this problem is to use a central entity that shares a symmetric key with the entire group members. This central entity will have the role of generating the group key and distributing it to each group member encrypted by the shared symmetric key between this entity and the group member. After establishing the group key, in case of any change in the group (join, leave, merge, and division), the central entity will again generate another group key and encrypt it using each symmetric key shared between it and each group member. Therefore, the cost of computation and communication will be linearly dependent on the number of group members, i.e. the complexity cost will be of *O(n)*, where *n* is the number of group members. For large groups or groups characterized by frequent change, this huge amount of computation and communication can decrease the group's performance.

In recent years, many approaches for solving the problem of group key distribution were proposed. These approaches can be classified as follows:

Centralized approaches use one central entity to maintain the security of the whole group. For large groups, those protocols are not scalable. In addition, the central entity represents a single point of failure.

Distributed subgroup approaches, where the whole group is divided into several subgroups. One subgroup controller maintains each subgroup. These protocols solve the problem of scalability. Another advantage of these protocols is that in case of failure of one subgroup controller, this does not lead to the failure of the whole group.

Decentralized approaches, where the whole group members contribute in the group key generation. As for the centralized approaches, these protocols are not scalable for large groups since it requires large computations among the group members.

In the present paper, a design of a high performance protocol for securing multicast communication is proposed. The proposed protocol is based on the idea of organizing the keys in a tree as in LKH protocols. In order to achieve lower computation overhead, the proposed protocol uses a multi-processor system. It has to be noted that LKH protocol relies heavily on one central point, therefore, it represents single point of failure and for a large tree; the server's throughput can represent a bottleneck. The use of multiple processors could solve this problem and enhance the server's throughput. The proposed protocol is analyzed according to: the number of processors, the tree height and the tree degree. The analysis shows that the use of multiprocessor system will enhance the system performance. Increasing the number of processors reduces the total execution time until reaching the system's saturation. Therefore, the use of multiprocessor system will significantly reduce the computation overhead which is considered an important factor for both real time and wireless applications. In addition, the proposed design is scalable. This is an important factor in real applications where the number of users changes repeatedly. The paper is organized as follows: in Section 2, a survey of group key distribution protocols is detailed. In Section 3, a background of multiprocessor systems is given. Then, the proposed protocol is detailed in Section 4. In Section 5, a performance evaluation of the proposed protocol is given. Finally, the paper concludes in Section 6.

2 Related Work

As a result of the spread use of Internet applications characterized by multicast communication, the need to establish a group key becomes a vital requirement. Group key distribution protocols play an important role to deliver security. They are considered the main part to obtain a secure system. A good key distribution protocol must satisfy the following requirements [19]:

- Providing confidentiality, which means that only authorized group members have access to key update messages.
- Providing backward secrecy by preventing a new group member to have access to previous messages exchanged before it joins the group.
- Providing forward secrecy by preventing a leaving group member to have access to messages exchanged after it leaves the group.
- Key independence, which means that the disclosure of any key does not lead to the disclosure of other keys.
- Minimizing collusion attacks, which means that evicted members must not be able to work together and share their individual information to regain access to the group key.
- Minimizing the key update message length in order to enhance system performance.
- Minimizing the operations needed to perform re-key in order to enhance the key distribution center performance.
- Minimizing storage requirements at both the key distribution center and each group member.

In recent years, many approaches for solving the problem of group key distribution were proposed. These approaches can be classified as follows: centralized group key distribution protocols, distributed subgroup key distribution protocols and decentralized group key distribution protocols. In centralized group key distribution protocols, one central entity is incorporated to play the role of group manager. This controller shares a symmetric key with all group members. In case of any member change (leave, join, merge, or division), the manager has to perform many encryptions and broadcast several messages for the changed keys. For a large group and frequent member change, the manager's throughput can represent a bottleneck for the group's performance. Therefore, for large groups, these protocols are not scalable. In addition, the central entity represents a single point of failure. Examples of this technique can be found in [2, 5, 15, 17, 18, 20, 22, 24, 26, 27]. The centralized approaches are generally based on the idea of Logical Key Hierarchy (LKH), which is introduced in [26]. In LKH, a key distribution center maintains a key tree as shown in Figure 1. Each node in the tree represents a symmetric key. The leaves represent the group members. Each member knows all the symmetric keys from its leaf to the root. For example, U_1 knows the set of keys: $\{k_4, k_2, k_1\}$. It has to be noted that k_1 represents the group key. If U_5 joins the group, it will be assigned a symmetric key k_8 as shown in Figure 2. To maintain backward secrecy, both k_3 and k_1 must be changed to k_{3new} and k_{1new}. The key server generates k_{3new} and k_{1new} and broadcasts the following message: $\{k_{1new}\}k_1$, $\{k_{3new}\}k_3$, $\{k_{3new}, k_{1new}\}k_8$. U_1, U_2, U_3 and U_4 obtain k_{1new} by decrypting the first part of the message using the old value k_1, U_3 and U_4 obtain k_{3new} by decrypting the second part of the message using the old value k_3, and U_5 obtains k_{3new} and k_{1new} by decrypting the third part. As stated in [26], the manager needs to perform *2h* encryptions in case of a member join, where *h* represents the height of the tree and equals to *log(n)*, *n* represents the total number of group members. In addition, the length of the transmitted message equals to *2h* keys. If U_4 leaves the group, to maintain forward secrecy, both k_3 and k_1 must be changed to k_{3new} and k_{1new} as shown in Figure 3. The key server generates k_{3new} and k_{1new} and broadcasts the following message: $\{k_{3new}, k_{1new}\}k_6$, $\{k_{1new}\}k_2$. U_3 obtains the new values k_{3new} and k_{1new} by decrypting the first part of the message, and U_1 and U_2 get the new value k_{1new} by decrypting the second part of the message. As derived in [26], the encryption operations needed by the manager for the case of a member leave equals to *2h-1* and the length of the transmitted message equals to *2h-1* keys. As mentioned in [26], the use of key tree will reduce the complexity cost from *O(n)* to *O(log(n))*. Further, the storage at each group member equals *h+1* keys and the storage at the server equals *2n-1* keys. It has to be noted that the centralized approaches rely heavily on one central point, which can represent a single point of failure. In addition for a large tree, the server's throughput can represent a bottleneck.

In distributed subgroup approaches, the whole group is divided into several subgroups. One Subgroup Controller (SC) that shares a symmetric key with the Group Controller (GC) maintains each subgroup. The role of SC is to establish a subgroup key to be used within the subgroup and to translate messages sent from GC to the subgroup members. This approach solves the problem of scalability; any member change will only affect the subgroup where this member belongs. Another advantage is that the failure of one SC will not lead to the failure of the whole group. The disadvantage of this approach is the need of decrypting group messages at the SC and re-encrypting using the subgroup key. This solution reduces the communication and computation complexity to *O(n/m)*, where *m* represents the number of subgroups. Examples of this approach can be found in [3, 6, 16, 21].

In decentralized group approaches, also called contributory protocols, each group member contributes to generate the group key. Decentralized approaches are characterized by having no group controller. The decentralized approach is generally based on large exponentiations using modulo operations. For information about modulo exponentiations, the reader can refer to [9]. Although this approach does not depend on a single entity to establish the group key, it suffers from the need of several modulo exponentiations, which makes it infeasible for large groups. Examples of this approach can be found in [4, 13, 25]. In the next section, a background of multiprocessor systems is given.

3 Background of Multiprocessor Systems

A multiprocessor system can be defined as a collection of autonomous processors, which can communicate with each other through a communication medium. The communication between processors can be done either through a shared memory medium or through links which interconnect processes directly with each other (message passing systems). In the shared memory model, processors can access in parallel memory locations which they share with all other processors. The communication between processors is achieved by writing information to common memory locations. In the message passing model (distributed memory system), each processor can read and write information only to a local memory, thus it must exchange information by sending messages via links to other processors. It is assumed that all processors of the message passing system execute the same algorithm and work correctly for any possible interconnection of processors [8]. Distributed-memory systems (DM) have some advantages over the shared-memory (SM) systems. *First*, DM systems require relatively design effort less than SM systems. *Second*, it is easily to expand (scalable) that is to say, as the number of processors increase, the memory size increases, the total memory bandwidth increases, processing capability of the system increases. *Third* nodes with different type of processors can be used to adapt the specialized problems (flexible). For the above reasons, we will focus on the parallel systems based on distributed memory.

A Message Passing (MP) system involves connecting multiple independent nodes each contains a processor and its local memory. There is no sharing of primary memory, but each processor has its own memory. The contents of each memory can only be accessed by its processor. When a processor needs information owned by another processor, the information is sent as a message from one processor to the other. Messages can carry information between nodes, also synchronization node activities. There are no restrictions on the number of available processors. The processors of this type operate independently of one another [12]. A distributed memory system configuration is shown in Figure 4. In distributed memory systems,

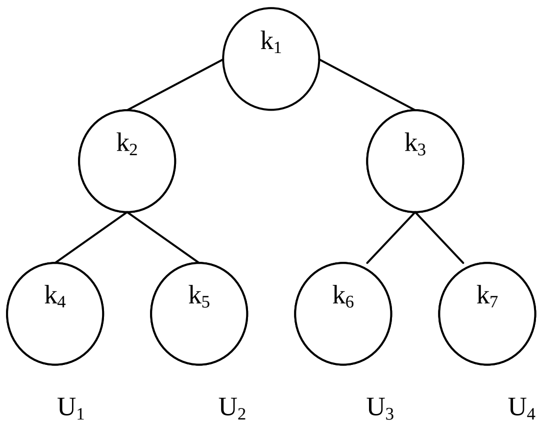


Figure 1: Hierarchical key tree

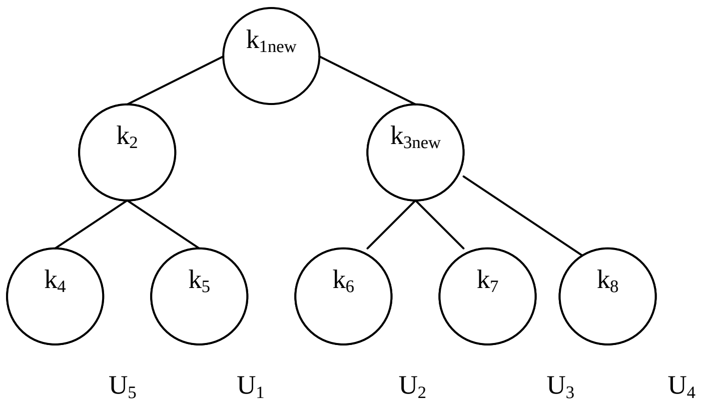


Figure 2: Key tree update (case of join)

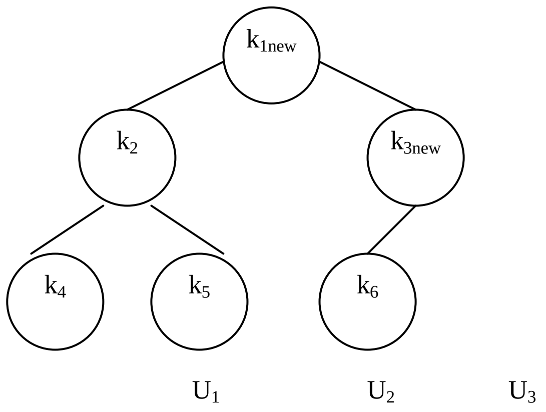


Figure 3: Key tree update (case of leave)

communication mechanisms can be divided into two major classes: synchronous and asynchronous. In *synchronous mechanism*, when the sender transmits a message, it waits until the receiver responds to acknowledge that the message has been received. *In asynchronous communication mechanism*, no acknowledgment, which means that, after sending a message, the sender does not wait for acknowledgment and immediately continue its execution. In the next section, a design of a high performance implementation of a tree-based multicast key distribution protocol is illustrated.

4 Design of High Performance Implementation of a Tree-Based Multicast Key Distribution Protocol

The proposed protocol is based on the idea of organizing the keys in a tree as in LKH protocols as shown in Figure 5. The tree is maintained and organized by a Group Manager (GM). In case of any group change, GM has to perform several calculations. In order to achieve lower computation overhead, the proposed protocol uses a multi-processor system. It has to be noted that the centralized approaches rely heavily on one central point, therefore, it represents a single point of failure and for a large tree; the server's throughput can represent a bottleneck. The use of multiple processors could solve this problem and enhance the server's throughput. In order to describe the system, the following parameters are used:

M	: number of processors
Speed	: speed of any processor (Mhz -GhZ).
b_g	: global communication bandwidth = the maximum rate at which the interconnection network can propagate information once the message enters the network (bits/s)
S_{mess}	: message size (bits)
P_{mem}	: processor memory size (Gbits)
B	: block size (bits)
Th	: symmetric key encryption algorithm throughput (bits/s)
h	: tree height
d	: tree degree
K	: total keys number
T_s	: time required to execute the sequential version of the program (sec.)
T_{comp}	: computation time (sec.)
T_{comm}	: total communication time (sec.)
T_{par}	: total parallel time (sec.)

In the next subsections, description of the proposed protocol will be detailed. The focus of this paper will be on the following cases: member join or member leave.

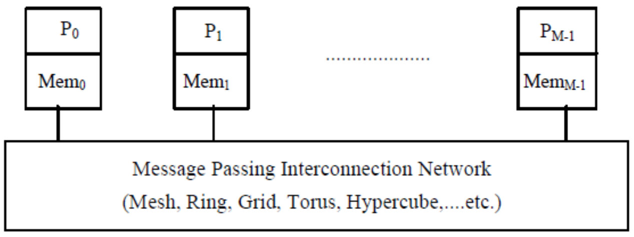

P_0
Mem_0
P_1
Mem_1
P_{M-1}
Mem_{M-1}
Message Passing Interconnection Network
(Mesh, Ring, Grid, Torus, Hypercube,....etc.)

Figure 4: A distributed-memory system configuration

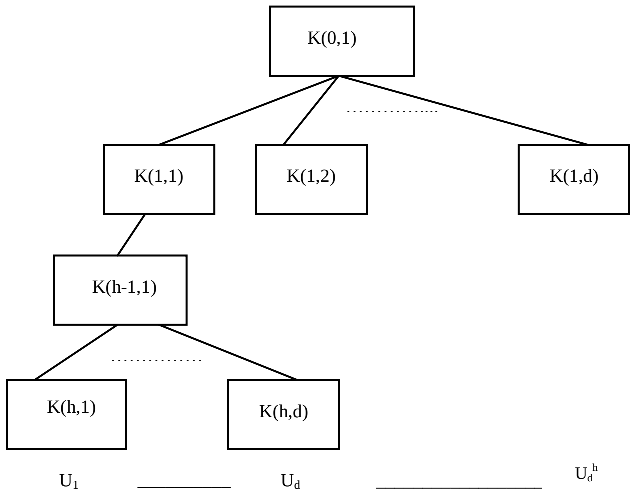


Figure 5: Example of 1 key hierarchy

4.1 Member Join

Assume a new member U_d joins the group, to preserve backward secrecy; all the keys along its path must be changed as shown in Figure 6. Therefore, the keys K(i, 1), where *i* ranges from *0 to h-1*, must be changed to $K_{new}(i, 1)$. It has to be noted that K(0, 1) is the group key shared by all members. Consequently, GM generates the new keys and calculates the following:

$\{K_{new}(0, 1)\}K(0, 1)$, $\{K_{new}(1, 1)\}K(1, 1)$, …, $\{K_{new}(h-1, 1)\}K(h-1, 1)$,

$\{K_{new}(0, 1), K_{new}(1, 1), …, K_{new}(h-1, 1)\}K(h, d)$

Therefore, GM needs to perform *2h* keys encryptions. The encryption is done in Electronic Code Book (ECB) mode. There are two important aspects of any algorithm: (i) the amount of time required to execute the algorithm and (ii) the amount of memory space needed during run-time. Execution time, which refers to the total running time of the program, is the most obvious way of describing the performance of parallel programs. The aim of using parallel processing is to decrease the execution time of the problem implementation. In parallel systems, the total execution time (parallel time) T_{par} is the sum of the computation time T_{comp}, and the overhead time T_{ov}. The sources of overhead are local communication overhead (access memory, memory contention, and synchronization), interprocessor network latency (message latency) and application overhead [1, 11]. For simplicity, we will only concern with message latency time. Thus, T_{par} is given as follows:

$$T_{par} = T_{comp} + T_{comm} \tag{1}$$

Where

$$T_{comp} = \max\{T_{comp}(P_i)\} \quad 0 \le i < M-1 \tag{2}$$

$$T_{comm} = \left(\sum_{i=0}^{M-1} \sum_{j=0}^{M-1} T_{mess}(P_i, P_j) + \alpha + \alpha_1 \right) \tag{3}$$

$$\begin{aligned} T_{mess}(P_i,P_j) = {} & \text{Sender overhead} \\ & + [(S_{mess}(P_i,P_j)/b_g) * hop(P_i,P_j)] \\ & + \text{Receiver overhead} \end{aligned} \tag{4}$$

where, *$Tcomp(P_i)$* is the computation time of processor P_i (*i* ranges from *0 to M-1*), T_{comm} is the total communication time, *$T_{mess}(P_i,P_j)$* is the time needed to send messages between P_i and P_j, α is the start up cost of initiating a message on any processor, α_1 is the start up cost of initiating a bus by any processor, *$S_{mess}(P_i,P_j)$* is the size of the message exchanged between P_i and P_j, *$hop(P_i,P_j)$* is the number of hops between P_i and P_j, sender overhead is the time for the processor to inject the message into the network (including both hardware and software components), and the receiver overhead is the time for the processor to pull the message from the network. It has to be noted that only one of the processors (for example P_0) generates the new keys, distributes the tasks (encryption operations) to the other processors and collects encryption information from them. For the sake of fault tolerance, in case of P_0 failure, any of the other processors (P_i and *i* ranges from *1 to M-1*) can take the role of P_0. Therefore, each processor has to store all symmetric keys (K), where $K = \frac{d^{h+1}-1}{d-1}$. To calculate the communication time T_{comm}, the following assumptions are made:

- The sender overhead, receiver overhead, α, and α_1 are very small with respect to the message latency. Therefore, it will be neglected in our calculations.
- Asynchronous communication mechanism
- The number of hops between P_i and P_j *$hop(P_i,P_j)$* equals one for all *i,j* (fully connection).
- Time of key generation is very small with respect to the encryption time. So, it will be neglected in our calculations.
- Only P_0 and P_i exchange messages with each other, i.e. no communication between P_i and P_j, where *i* and *j* range from *1 to M-1*.

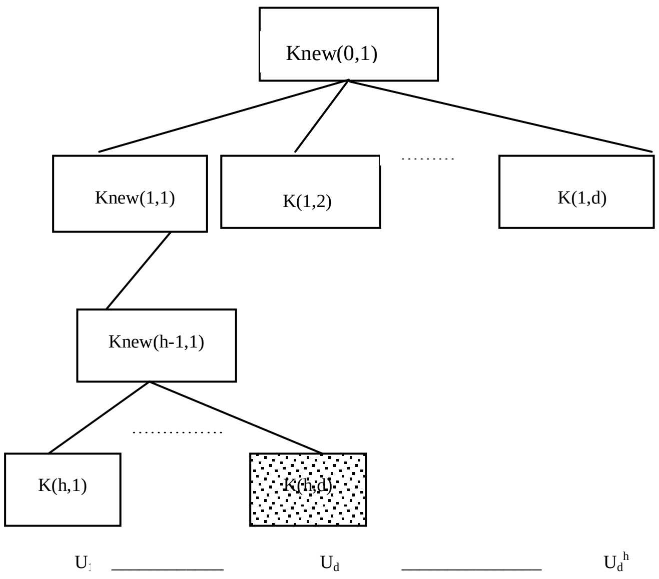


Figure 6: Example of a member joins

Therefore, the communication time is given as follows:

$$T_{comm} = \frac{\sum_{i=1}^{M-1} S_{mess}(P_0, P_i)}{b_g} \tag{5}$$

In the following paragraphs, the sequential time T_S, the computation time T_{comp}, the communication time T_{comm}, and the total parallel time T_{par} are calculated for different values of *M*. First, for *M =1*, only one processor is used, there is no communication time and the number of tasks to be performed is *2h* encryptions. The sequential time T_S required to calculate these tasks is given by Equation (6).

$$T_s = T_{comp} = \frac{2h * B}{Th} \tag{6}$$

For the case of *M=2*, using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (7) and Equation (8).

$$T_{comp} = \frac{h * B}{Th} \tag{7}$$

$$T_{comm} = \frac{h * B}{b_g} \tag{8}$$

Therefore, using Equations (1), (7) and (8), T_{par} equals:

$$T_{par} = \left\lceil \frac{h * B}{Th} \right\rceil + \left\lceil \frac{h * B}{b_g} \right\rceil \tag{9}$$

Finally for $2 < M \le h$, as mentioned above the total number of tasks is *2h*. While distributing tasks among processors, two cases arise: the first is the case where (*h mod M = 0*), each processor has to perform $\left(\frac{h}{M}\right) * 2$ tasks. The latter is the case where (h mod $M \neq 0$), each processor has to perform $\left(\left|\frac{h}{M}\right| * 2\right)$ tasks and the remaining tasks are assigned to the first $\left(h - M * \left|\frac{h}{M}\right|\right)$ processors. Therefore, the first $\left(h - M * \left|\frac{h}{M}\right|\right)$ processors are assigned one more row (2 tasks); thus the number of tasks to be performed is $\left(\left|\frac{h}{M}\right| + 1\right) * 2$ tasks. For the case where *(h mod M) = 0*, using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (10) and Equation (11).

$$T_{comp} = \frac{2 * \left(\frac{h}{M}\right) * B}{Th} \tag{10}$$

$$T_{comm} = \frac{2 * \left[h - \left(\frac{h}{M}\right)\right] * B}{b_g} \tag{11}$$

Therefore, using Equation (1), Equation (10) and Equation (11), T_{par} equals:

$$T_{par} = \left[\frac{2 * \left(\frac{h}{M}\right) * B}{Th}\right] + \left[\frac{2 * \left[h - \left(\frac{h}{M}\right)\right] * B}{b_g}\right] \tag{12}$$

For the second case where *(h mod M) ≠ 0,* using Equation (2) and Equation (5), the computation time T_{comp}, and the communication time T_{comm} are given by Equation (13) and Equation (14).

$$T_{comp} = \frac{2 * \left[\left|\frac{h}{M}\right| + 1\right] * B}{Th} \tag{13}$$

$$T_{comm} = \frac{2 * \left[h - \left(\left|\frac{h}{M}\right| + 1\right)\right] * B}{b_g} \tag{14}$$

Therefore, using Equation (1), Equation (13) and Equation (14), T_{par} equals:

$$T_{par} = \left[\frac{2 * \left[\left|\frac{h}{M}\right| + 1\right] * B}{Th}\right] + \left[\frac{2 * \left[h - \left(\left|\frac{h}{M}\right| + 1\right)\right] * B}{b_g}\right] \tag{15}$$

In the next subsection, description of the proposed protocol for the case of a member leaves will be detailed.

4.2 Member Leave

Assume U_d leaves the group, to preserve forward secrecy; all the keys along its path must be changed as shown in Figure 7. Therefore, the keys K(i, 1), where *i* ranges from *0* to *h-1*, must be changed to K_{new}(i, 1). Consequently, GM generates the new keys and calculates the following:

$\{K_{new}(0, 1)\}K_{new}(1, 1)$, $\{K_{new}(0, 1)\}K(1, 2)$, …, $\{K_{new}(0, 1)\}K(1, d)$,

$\{K_{new}(1, 1)\}K_{new}(2, 1)$, $\{K_{new}(1, 1)\}K(2, 2)$, …, $\{K_{new}(1, 1)\}K(2, d)$,

.

.

$\{K_{new}(h-1, 1)\}K(h, 1)$, $\{K_{new}(h-1, 1)\}K(h, 2)$, …, $\{K_{new}(h-1, 1)\}K(h, d-1)$

Therefore, GM needs to perform *d*h-1* keys encryptions. The encryption is done in Electronic Code Book (ECB) mode. As in the join case, the sequential time T_S, the computation time T_{comp}, the communication time T_{comm} and the total parallel time T_{par} are calculated for different values of *M*. First, for *M=1*, only one processor is used, there is no communication time and the number of tasks to be performed is *d*h-1* encryptions. The sequential time T_S required to calculate these tasks is given by Equation (16).

$$T_s = T_{comp} = \frac{((d * h) - 1) * B}{Th} \tag{16}$$

For the case of *M=2*, two cases arise: the first for h even and the second for *h* odd. For the case where *h* is even, the master processor P_0 has to calculate the last $\left(\frac{h}{2}\right)$ rows (i.e $\left[\frac{hd}{2} - 1\right]$ key encryptions), while the other processor P_1 has to calculate the first $\left(\frac{h}{2}\right)$ rows (i.e $\left[\frac{hd}{2}\right]$ key encryptions). Therefore, P_0 sends $\left[\frac{h}{2} + 1\right]$ keys to be

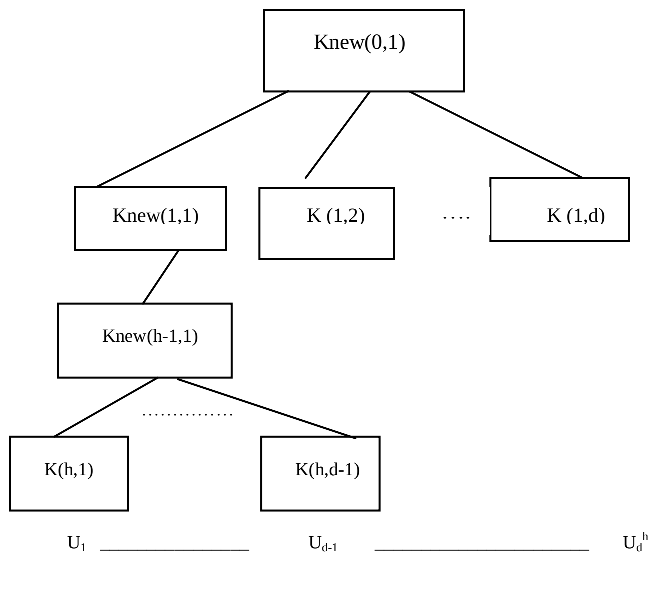


Figure 7: Example of a member leaves

encrypted. After encrypting the keys, P_1 sends $\left[\frac{hd}{2}\right]$ encrypted keys to P_0. Using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (17) and Equation (18).

$$T_{comp} = \frac{\left(\frac{d*h}{2}\right)*B}{Th} \qquad (17)$$

$$T_{comm} = \frac{\left[\left(\frac{h}{2}(d+1)\right)+1\right]*B}{b_g} \qquad (18)$$

Therefore, using Equation (1), Equation (17) and Equation (18), T_{par} equals:

$$T_{par} = \left[\frac{\left(\frac{d*h}{2}\right)*B}{Th}\right] + \left[\frac{\left[1+\left(\frac{h}{2}(d+1)\right)\right]*B}{b_g}\right] \qquad (19)$$

For the second case where h is odd, the master processor P_0 has to calculate the last $\left(\left|\frac{h}{2}\right|+1\right)$ rows (i.e $\left[\left(\left|\frac{h}{2}\right|*d\right)+(d-1)\right]$ key encryptions), while the other processor P_1 has to calculate the first $\left|\frac{h}{2}\right|$ rows (i.e $\left[\left|\frac{h}{2}\right|*d\right]$ key encryptions). Therefore, P_0 sends $\left[\left|\frac{h}{2}\right|+1\right]$ keys to be encrypted. After encrypting the keys, P_1 sends $\left[\left|\frac{h}{2}\right|*d\right]$ encrypted keys to P_0. Using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (20) and Equation (21).

$$T_{comp} = \frac{\left\{\left[d*\left(h-\left|\frac{h}{2}\right|\right)\right]-1\right\}*B}{Th} \qquad (20)$$

$$T_{comm} = \frac{\left\{\left[\left|\frac{h}{2}\right|+1\right]+\left[d*\left|\frac{h}{2}\right|\right]\right\}*B}{b_g} \qquad (21)$$

Therefore, using Equation (1), Equation (20) and Equation (21), T_{par} equals:

$$T_{par} = \left[\frac{\left\{\left[d*\left(h-\left|\frac{h}{2}\right|\right)\right]-1\right\}*B}{Th}\right] + \left[\frac{\left\{\left[\left|\frac{h}{2}\right|+1\right]+\left[d*\left|\frac{h}{2}\right|\right]\right\}*B}{b_g}\right] \qquad (22)$$

Finally for $2 < M \leq h$, as mentioned above the total number of tasks is $d*h-1$. While distributing tasks among processors two cases arise: the first is the case where $((h\text{-}1) \bmod M = 0)$ and the latter for the case where $((h\text{-}1) \bmod M \neq 0)$. For the case where $((h\text{-}1) \bmod M) = 0$, each processor from P_1 to $P_{M\text{-}1}$ has to perform $\left(d*\left|\frac{h}{M}\right|\right)$ tasks, while P_0 performs $\left\{\left[d*\left(\left|\frac{h}{M}\right|+1\right)\right]-1\right\}$ tasks. Therefore, P_0 sends $\left[\left|\frac{h}{M}\right|+1\right]$ keys to each processor (from P_1 to $P_{M\text{-}1}$). After encrypting the keys, each processor (from P_1 to $P_{M\text{-}1}$) sends $\left[\left|\frac{h}{M}\right|*d\right]$ encrypted keys to P_0. Using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (23) and Equation (24).

$$T_{comp} = \frac{\left\{\left[d*\left(\left|\frac{h}{M}\right|+1\right)\right]-1\right\}*B}{Th} \qquad (23)$$

$$T_{comm} = \frac{\left\{(M-1) * \left[\left|\frac{h}{M}\right| + d\left|\frac{h}{M}\right| + 1\right]\right\} * B}{b_g} \tag{24}$$

Therefore, using Equation (1), Equation (23) and Equation (24), T_{par} equals:

$$T_{par} = \left[\frac{\left\{\left[d * \left(\left|\frac{h}{M}\right| + 1\right)\right] - 1\right\} * B}{Th}\right] + \left[\frac{\left\{(M-1) * \left[\left|\frac{h}{M}\right| + d\left|\frac{h}{M}\right| + 1\right]\right\} * B}{b_g}\right] \tag{25}$$

For the second case where *((h-1) mod M) ≠ 0*, two cases occur, the first for (h mod M) = *0* and the latter for (h mod M) ≠ 0. For the case where (h mod M) = 0, each processor from P_1 to P_{M-1} has to perform $\left[d * \left(\frac{h}{M}\right)\right]$ tasks and P_0 has to perform $\left\{\left[d * \left(\frac{h}{M}\right)\right] - 1\right\}$ tasks. Therefore, P_0 sends $\left[\left(\frac{h}{M}\right) + 1\right]$ keys to each processor (from P_1 to P_{M-1}). After encrypting the keys, each processor (from P_1 to P_{M-1}) sends $\left[\left(\frac{h}{M}\right) * d\right]$ encrypted keys to P_0. Using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (26) and Equation (27).

$$T_{comp} = \frac{d * \left(\frac{h}{M}\right) * B}{Th} \tag{26}$$

$$T_{comm} = \frac{\left\{(M-1) * \left[\frac{dh}{M} + \frac{h}{M} + 1\right]\right\} * B}{b_g} \tag{27}$$

Therefore, using Equation (1), Equation (26) and Equation (27), T_{par} equals:

$$T_{par} = \left[\frac{d * \left(\frac{h}{M}\right) * B}{Th}\right] + \left[\frac{\left\{(M-1) * \left[\frac{dh}{M} + \frac{h}{M} + 1\right]\right\} * B}{b_g}\right] \tag{28}$$

Finally, for the case where *((h-1) mod M) ≠ 0* and *(h mod M) ≠ 0*, P_0 has to perform $\left\{\left[d * \left(\left|\frac{h}{M}\right| + 1\right)\right] - 1\right\}$ tasks, and P_j, where $\left[1 = j \le \left(h - M * \left|\frac{h}{M}\right|\right)\right]$ has to perform $\left[d * \left(\left|\frac{h}{M}\right| + 1\right)\right]$ tasks, finally the remaining processors has to perform $\left(d * \left|\frac{h}{M}\right|\right)$. Therefore, P_0 sends $\left[\left|\frac{h}{M}\right| + 2\right]$ keys to $\left(h - M * \left|\frac{h}{M}\right| - 1\right)$ processors and $\left[\left|\frac{h}{M}\right| + 1\right]$ keys to $\left\{M - \left(h - M * \left|\frac{h}{M}\right|\right)\right\}$ processors. After encrypting the keys, the processors from P_1 to P_{M-1} send $\left\{(dh-1) - \left[\left[d * \left(\left|\frac{h}{M}\right| + 1\right)\right] - 1\right]\right\}$ encrypted keys to P_0. Using Equation (2) and Equation (5), the computation time T_{comp} and the communication time T_{comm} are given by Equation (29) and Equation (30).

$$T_{comp} = \frac{d * \left[\left|\frac{h}{M}\right| + 1\right] * B}{Th} \tag{29}$$

$$T_{comm} = \frac{\left\{\left[(M-2) - \left|\frac{h}{M}\right| + h\right] + \left[d * \left(h - \left|\frac{h}{M}\right| - 1\right)\right]\right\} * B}{b_g} \tag{30}$$

Therefore, using Equation (1), Equation (29) and Equation (30), T_{par} equals:

$$T_{par} = \left[\frac{d * \left[\left|\frac{h}{M}\right| + 1\right] * B}{Th}\right] + \left[\frac{\left\{\left[(M-2) - \left|\frac{h}{M}\right| + h\right] + \left[d * \left(h - \left|\frac{h}{M}\right| - 1\right)\right]\right\} * B}{b_g}\right] \tag{31}$$

In the next section, evaluation of the proposed protocol and experimental results will be depicted.

5 Evaluation and Experimental Results

The main reason for building parallel computers is to achieve higher performance. Experimentation, analytical modeling and simulation are three well-known techniques that can be used for quantifying parallel systems performance [23]. Experimental, involves implementing the application and measuring the performance on a real machine. *Analytical models,* abstract hardware and application details in a parallel system and capture complex system's feature by simple mathematical formulae. *Simulation*s exploit computer resources to models and imitate the behavior of a real system in a controlled manner [11, 14, 23]. In this work, the analytical modeling approach will be used to evaluate the system performance. In the next subsection, metrics used for quantifying the parallel systems will be introduced.

5.1 Performance Metrics and Scalability

Many performance metrics have been proposed to quantify the parallel systems. Among of them are execution time, speedup, efficiency, communication overheads, scalability and the degree of improvement [8, 10, 11]. *Execution time* also called parallel time T_{par} is referred to the total running time of the program. It is the most obvious way of describing the performance of parallel programs. The aim of using parallel systems is to decrease the execution time of the problem implementation. *Speedup* S_p, relates the time taken to solve the problem on a single processor machine to the time taken to solve the same problem using parallel implementation. Speed-up S_p, of a parallel program running on M processors is defined as the ratio T_s/T_{par}, where T_s is the time required to execute the best sequential version of the program, and T_{par} is the time taken to execute the program on M processors. The ideal parallel system (of M processors) will solve the problem M times faster than the serial one ($S_p=M$) and it is said to be linear speedup. *Efficiency*, Ep, is defined as the ratio S_p/M. Optimum computation time, and therefore linear speed-up, equates to an efficiency of 1 (100%). To achieve this level of efficiency every processor must spent 100% of its time performing useful computation. In parallel systems, communication is the costly part; therefore high Computation To Communication Ratio (CTCR) is very beneficial. CTCR is defined by the average computation cost divided by the average communication cost on a given system. *Degree of improvement* is the percentage of improvement in system performance with respect to sequential execution and can be determined by $(T_s-T_{par})/T_s$. Finally, a parallel system is *scalable* if its performance continues to improve as the size of the system (problem sizes as well as the machine size) increase. The simplest definition of scalability is that the performance of a parallel system increases linearly with respect to the number of processors used for a given application [23]. In the next subsection, discussion of results will be detailed.

5.2 Discussion of Results

The proposed protocol is evaluated for both join and leave cases; and for different values of d and h. In our proposed design, we assume that the multiprocessor system is based on message passing system and uses the Advanced Encryption Standard (AES) protocol in ECB mode for encryption. In our implementation, we assume that each processor's speed is 2.5GhZ, P_{mem} equals 1Gbits, bg is 4.2Gbps, Th equals 1.4Gbps and B is 128bits. Figures 8-12 show the analysis of the proposed protocol according to the metrics given in Sec. 5.1. While Figure 8 illustrates the total execution time in case of a member joins, Figure 9 shows the execution time in case of a member leaves. Whilst Figure 10 illustrates the system performance (computation time, communication time, speed up, efficiency and the improvement degree) in case of a member joins, Figure 11 shows the system performance in case of a member leaves. Figure 12 illustrates the execution time for the same number of users when a member leaves the tree. From the figures, the following observations are noted:

- Figures 8-9 summarize the total execution time (parallel time) for different values of h for both join and leave cases. As the number of processors increases, the total execution time decreases irrespective of the value of h. Increasing the number of processors leads to the increase in the communication overhead which becomes an increasing factor and can exceed the total execution time. Consequently, the system's efficiency will decrease. Therefore, the number of processors must not exceed a certain number which is called system's saturation. As shown in Figure 8, the saturation occurs when the number of processors equals ($h/2$). On the other hand, the leave case reaches this saturation at $\left(M-\left|\frac{h}{M}\right|\leq 1\right)$, for the binary tree, as shown in Figure 9.
- Figure 10(a,b) and Figure 11(a,b) show that as the number of processors increases, the computation time decreases, while the communication time increases, for all values of h. The communication overhead time values of the join case are less than those values of the leave case for the same number of h. This is obvious since the distributed tasks required for re-keying in the join case is less than that in the leave case. In case of a new member joins the tree, the computation time reduces by approximately 50% when the number of processors increases from one to two; for all values of h. While the total execution time reduces by 35% due to the communication overhead. On the other hand, in case of a member leaves the tree the computation time reduces by approximately 50% when the number of processors increases from one to two, while total execution time reduces by about 22%; for all values of h.
- Figure 10(a) and Figure 11(a) show that when the problem size increases, the execution time decreases with respect to the increasing of the processor number. This leads to the conclusion that the proposed design is scalable according to the metrics given in Section 5.1.
- In both join and leave cases, upon increasing the number of processors, the speedup will increase, and consequently the efficiency will decrease as illustrated in Figure10(c,d) and Figure 11(c,d).
- Figure 10(e) shows the degree of improvement, compared to the sequential performance for join situation, when a new member joins the binary tree.

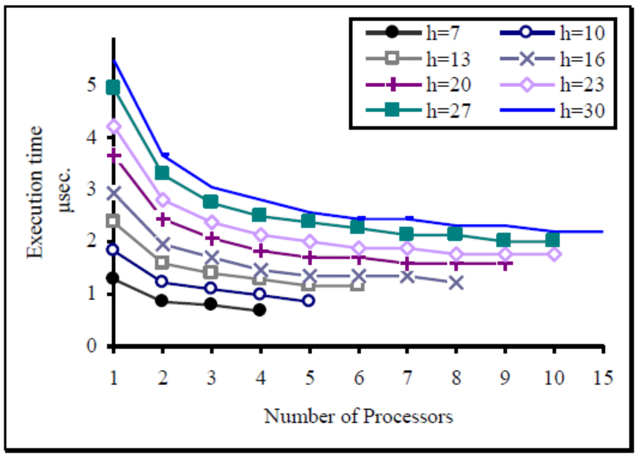


Figure 8: The total execution time (parallel time) after parallelization when a new member joins the tree (for d=2 and different values of h)

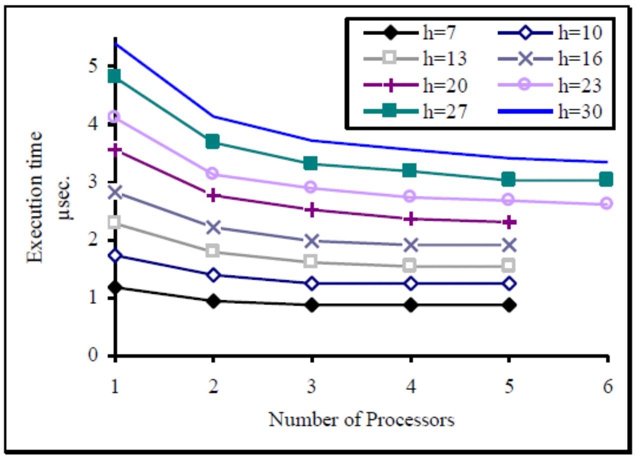
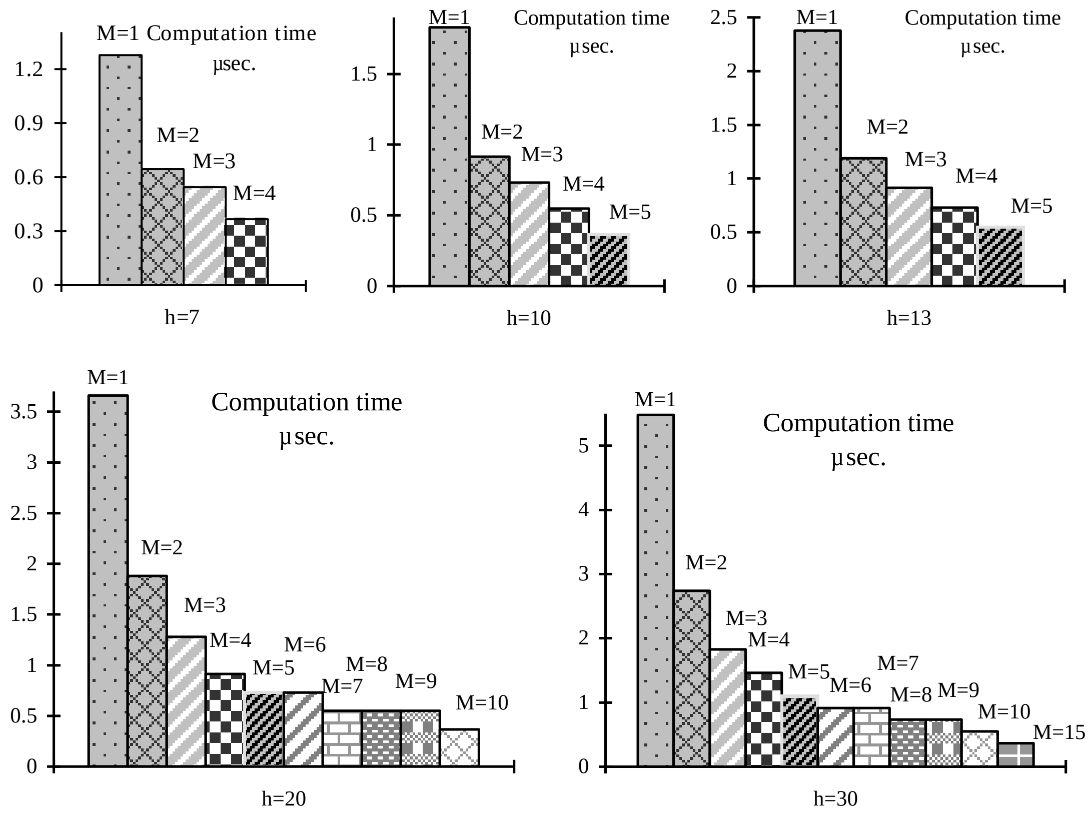
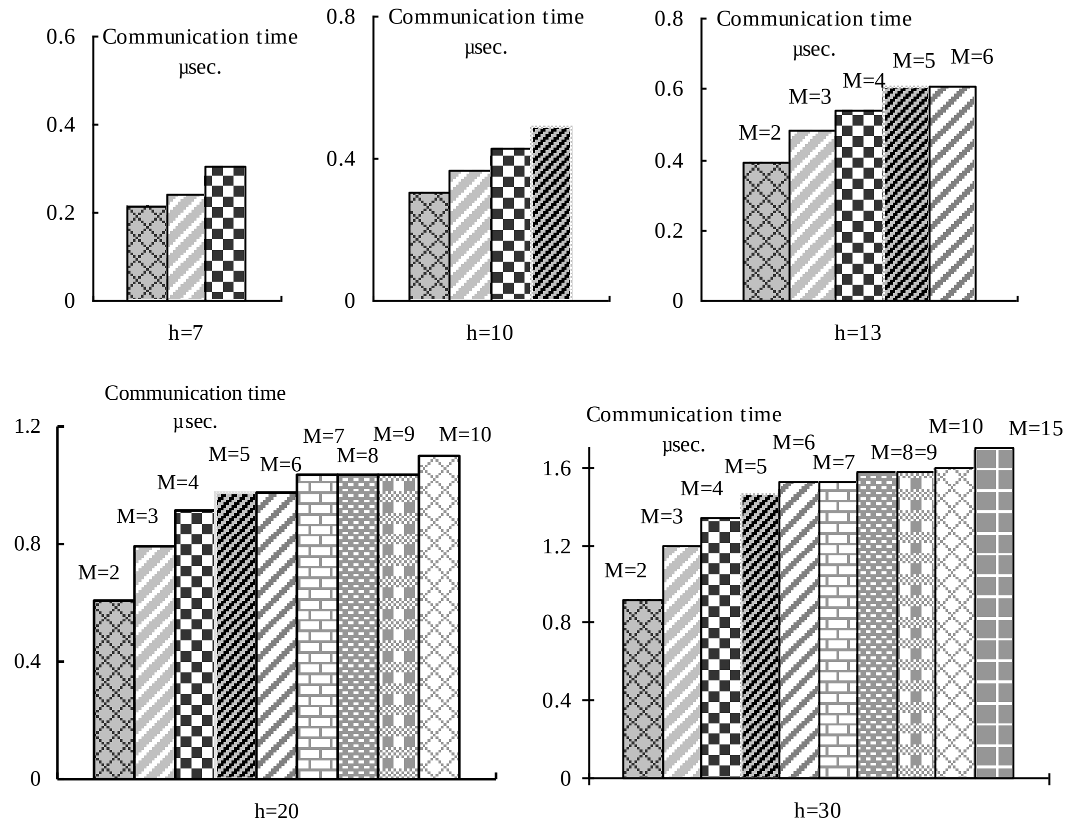


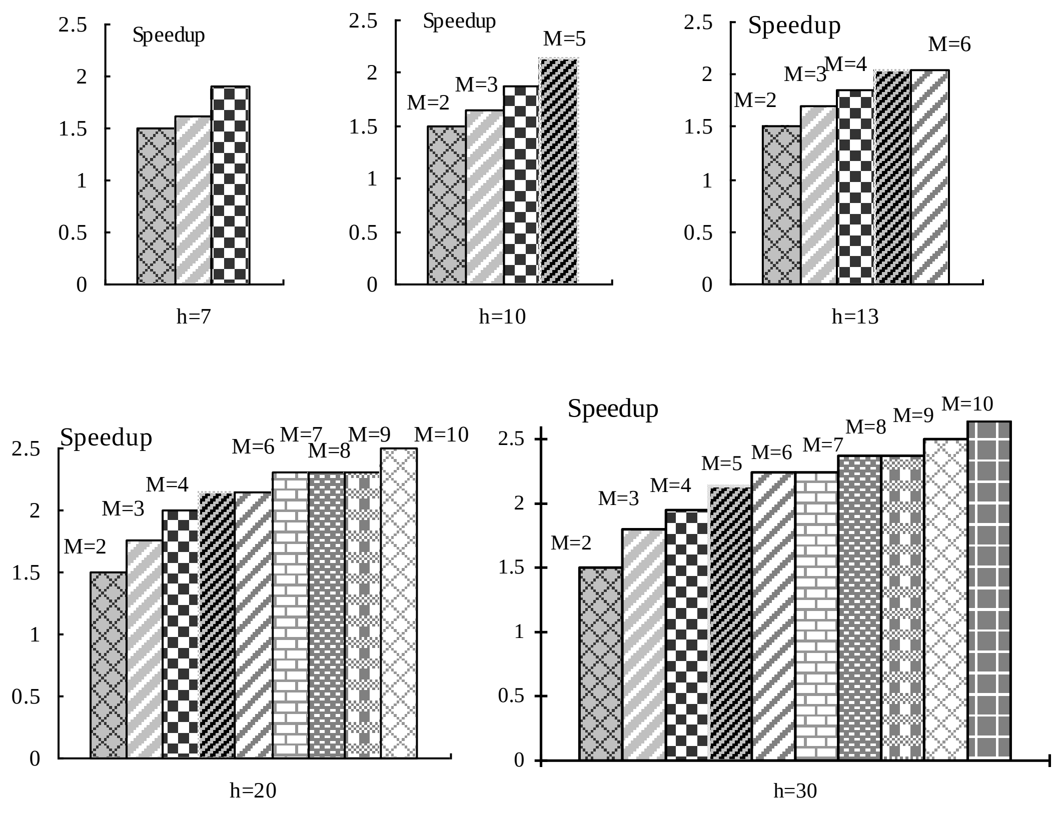
Figure 9: The total execution time (parallel time) after parallelization when a member leaves the tree (for d=2 and different values of h)



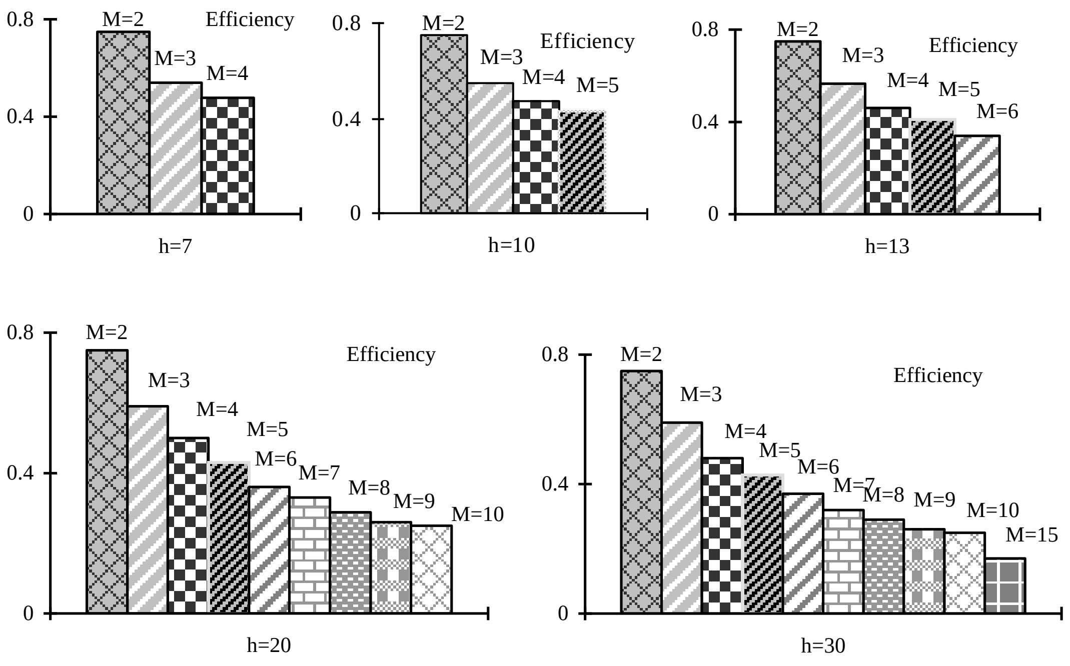
(a) Computation time



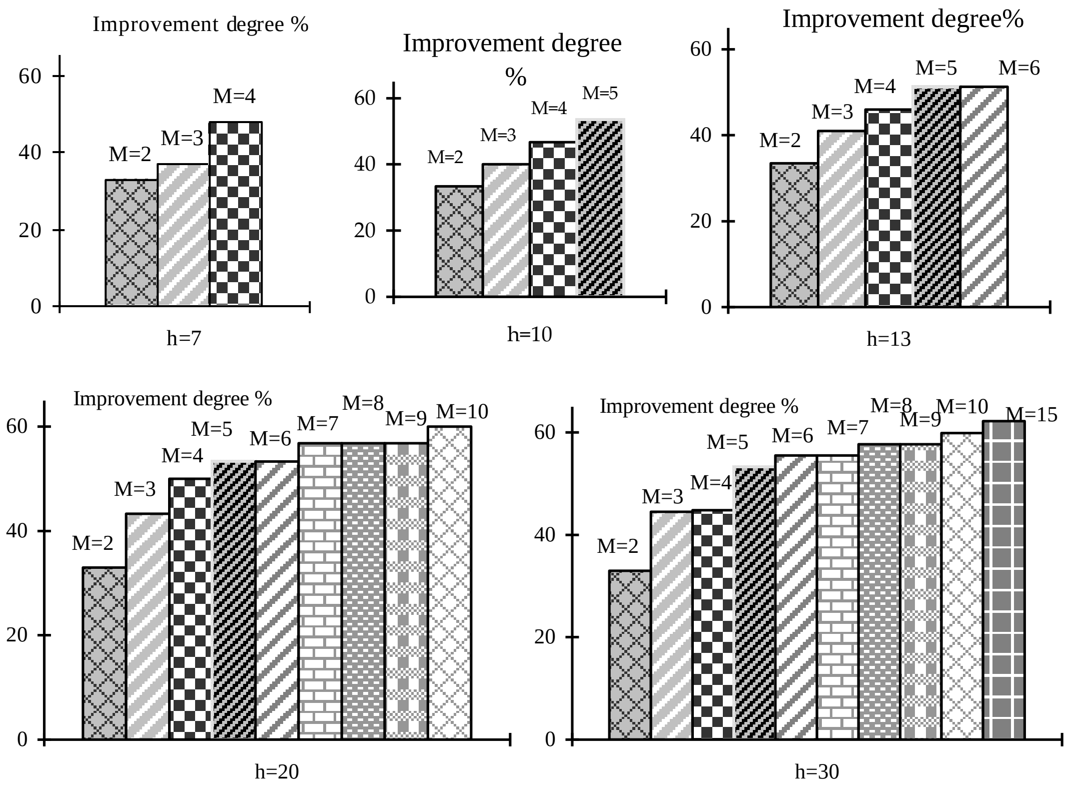
(b) Communication time



(c) Speedup

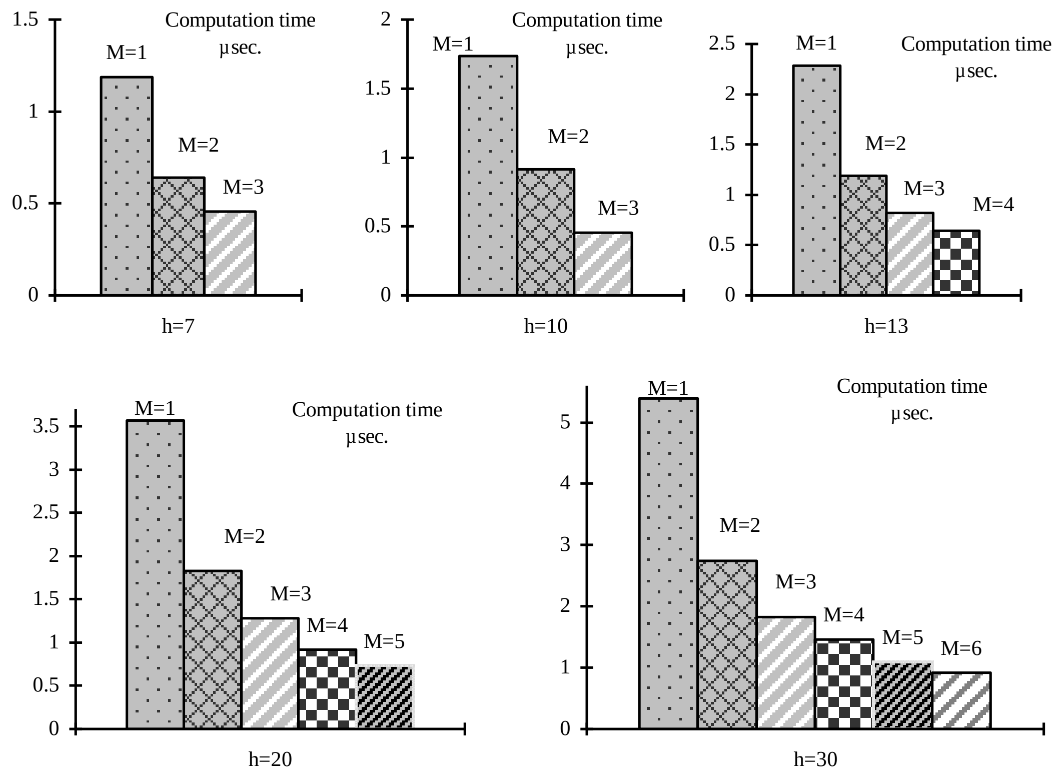


(d) Efficiency

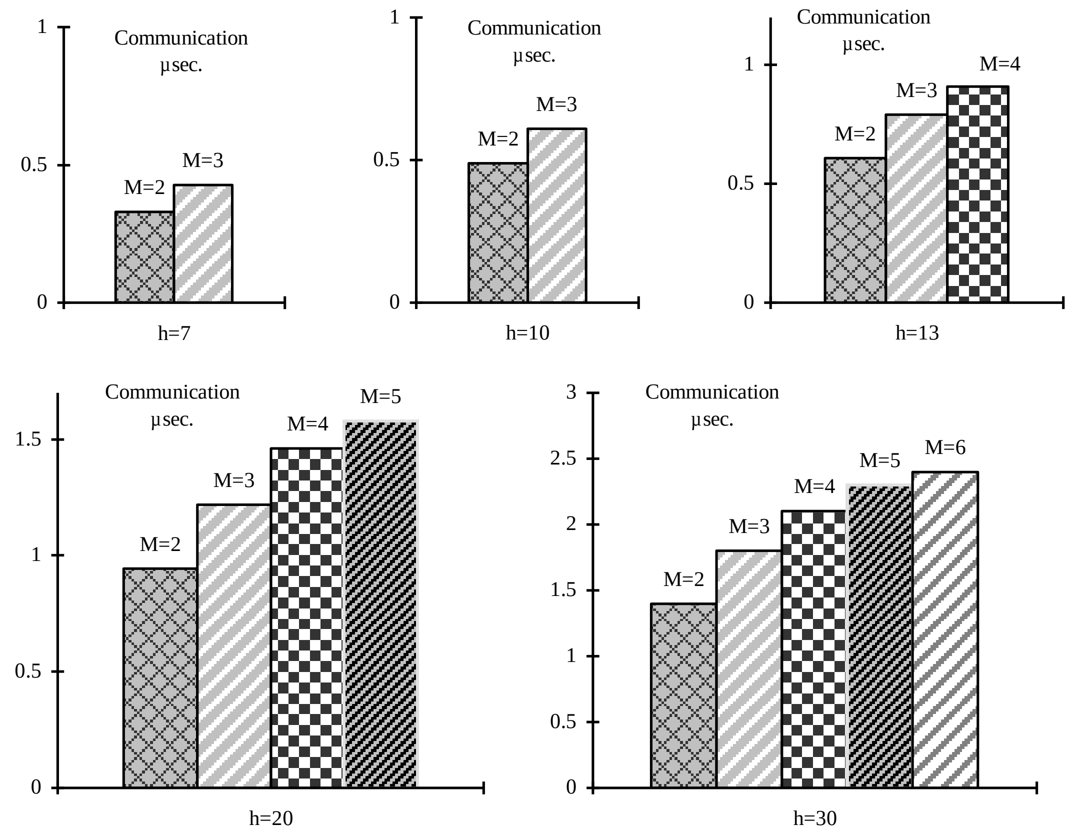


(e) Improvement degree w.r.t before parallelization

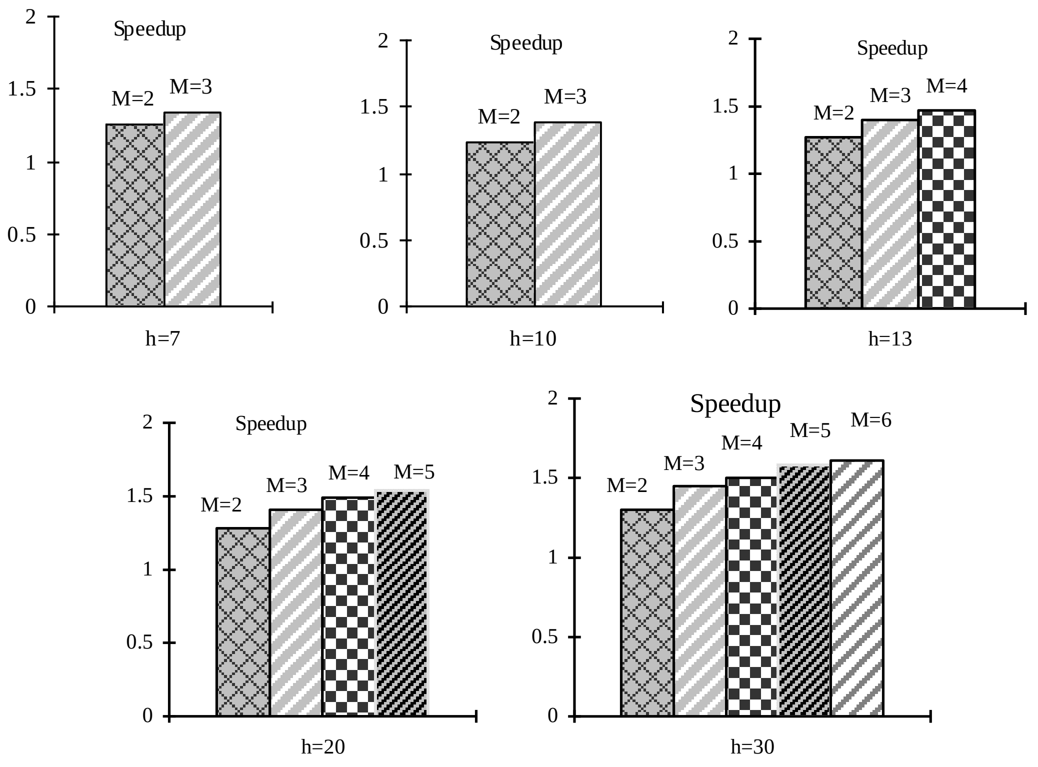
Figure 10: The system performance: computation time, communication time, speed up, efficiency and the improvement degree, when a member joins the binary tree (for d=2 and different values of h)



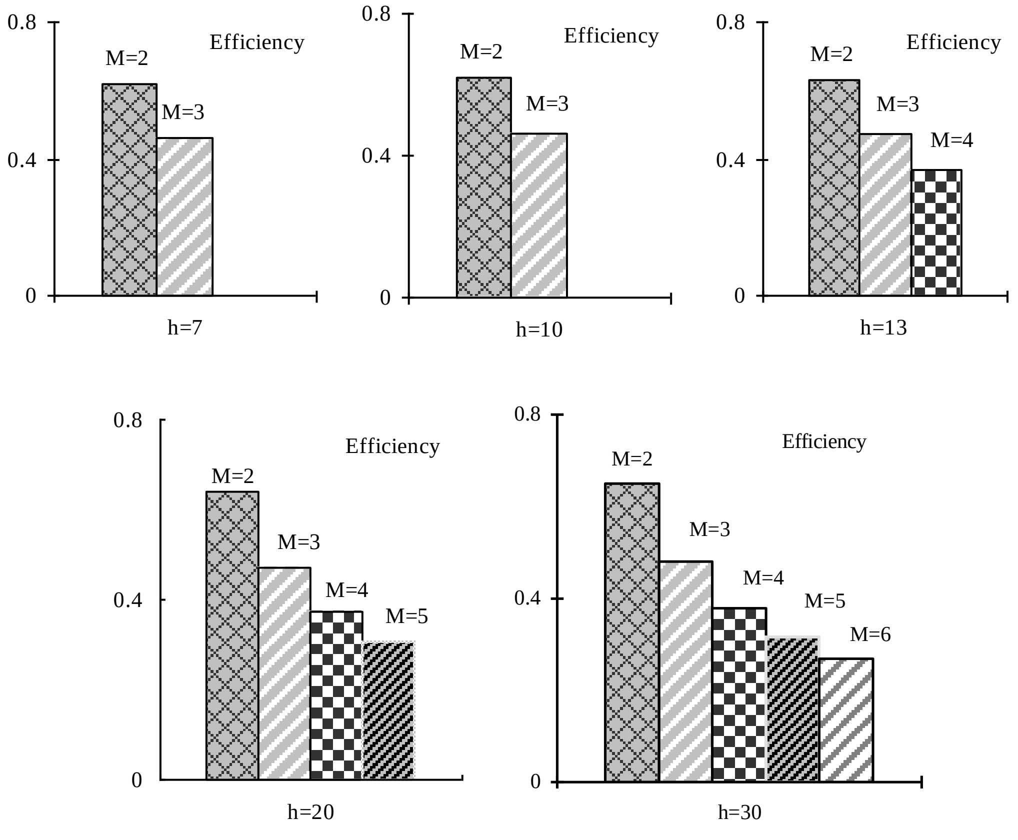
(a) Computation time



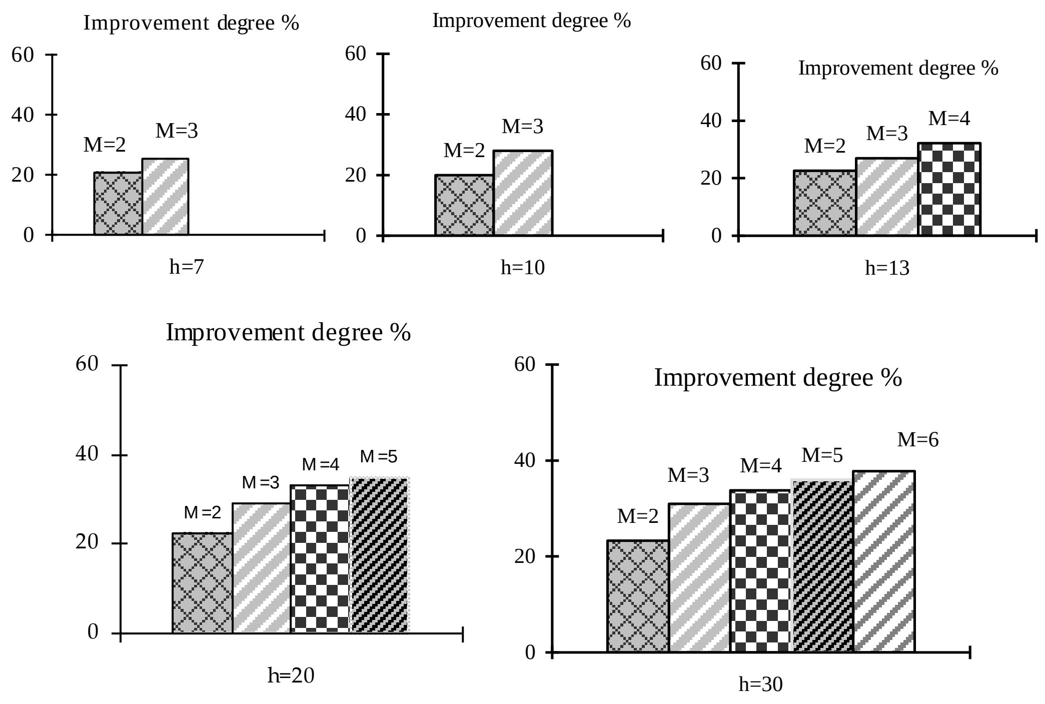
(b) Communication time



(c) Speedup

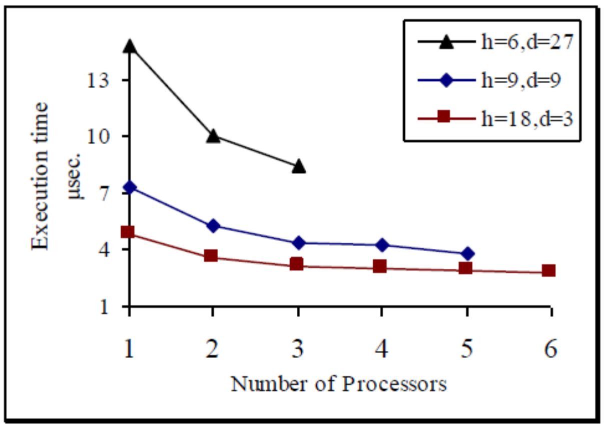


(d) Efficiency

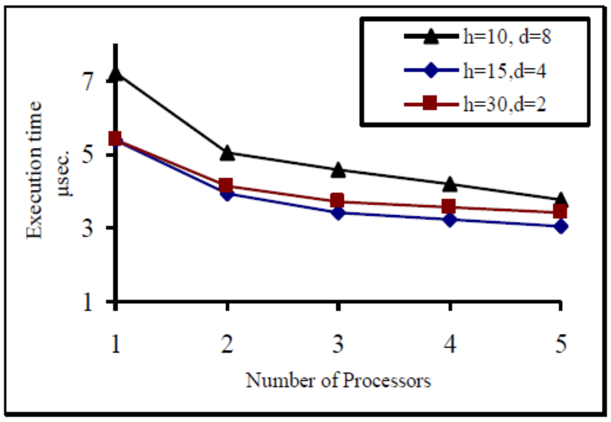


(e) Improvement degree

Figure 11: The system performance: computation time, communication time, speed up, efficiency and the improvement degree, when a member leaves the binary tree (for d=2 and different values of h)



(a) Number of users = 378 * 10^6



(b) Number of users = 1007* 10^6

Figure 12: The execution time for the same number of users when a member leaves the tree

the degree of improvement is 47.7%, 53.3%, 51.3%, 60% and 62.2% for *h*= 7, 10, 13, 20 and 30respectively. On the other hand, for the leave situation the improvement degree is 25.5%, 28%, 32.2%, 35% and 37.8% for the same values of h as shown in Figure 11(e). From these figures, we can deduce that as increasing the number of *h*, the improvement degree increases. This is due to the fact that the computation time increases as increasing the tree height**.** This shows the advantage of using a multiprocessor system in enhancing the system performance.

- Figure 12 describes the effect of increasing the value of *d* for the leave situation. It compares the system performance for the same number of users. This figure shows that, when the value of *d* increases both the computation and communication time increase. Therefore, the total execution time increases. Consequently, minimizing the value of *d* is the optimum solution, i.e. the binary tree is the best choice.

The analysis shows that the use of multiprocessor system will enhance the system performance. Increasing the number of processors reduces the total execution time until reaching the system's saturation. In addition, the proposed design is scalable. This is an important factor in real applications where the number of users changes repeatedly.

6 Conclusions

In the present paper, a design of a high performance implementation of a tree-based multicast key distribution protocol is proposed. In order to solve the problem of distributing a symmetric key between the whole group members, the group is organized in a logical key hierarchy as in LKH protocols. In order to achieve lower computation overhead, the proposed protocol uses a multi-processor system. It has to be noted that LKH protocol relies heavily on one central point, therefore, it represents a single point of failure and for a large tree; the server's throughput can represent a bottleneck. The use of multiple processors could solve this problem and enhance the server's throughput. The proposed protocol is analyzed according to: the number of processors, the tree height and the tree degree. Experimental results illustrate the improved performance of the proposed protocol compared to sequential system even with significant communication overheads. This improvement is true for all values of *h* and *d* even with highly size tree. It outperforms the sequential performance for both join and leave situations. When a new member joins the binary tree, the degree of improvement is 47.7%, 53.3%, 51.3%, 60% and 62.2% for *h*= 7, 10, 13, 20 and 30 respectively. On the other hand, for the leave situation, the improvement degree is 25.5%, 28%, 32.2%, 35% and 37.8% for the same values of *h*. The proposed protocol achieves lower communication time in case of a member joins the tree than its corresponding values of the leave situations. This is obvious since the distributed tasks required for re-keying in the join case is less than that in the leave case. In addition, the analysis shows that the proposed design is scalable according to the metrics given in Sec. 5.1. To test the scalability of the proposed protocol, it is tested on different problem sizes. The above results indicate that when the problem size *d*h* increases our protocol improves the overall system performance with respect to the increasing of the processor number. This leads to the conclusion that the proposed design is scalable. Another experiment which discusses the effect of increasing the value of *d* is done. The experiment shows that when the value of d increases both the computation and communication time increase. Therefore, the total execution time increases. Consequently, minimizing the value of *d* is the optimum solution, i.e. the binary tree is the best choice. The abovementioned analysis shows that the use of multiprocessor system will significantly reduce the computation overhead which is considered an important factor for both real time and wireless applications.

References

[1] M. Abdel-Baky, *New Routing Techniques for High Message Passing Systems Performance*, A Ph.D. Thesis, Dept. of Mathematics, Faculty of Science, Cairo University, 2000.

[2] H. K. Aslan, "Two-level controllers hierarchy for a scalable and distributed multicast security protocol," *Computers & Security*, vol. 24, no. 5, pp.399-408, 2005.

[3] M. Bouassida, I. Chrisment, and O. Festor, "Group Key Management in Manets", *International Journal of Network Security*, vol. 6, no. 1, pp. 67-79, 2008.

[4] C. Boyd, "On key agreement and conference key agreement," *Lecture Notes in Computer Science*, vol. 1270, pp. 294-302, Springer-Verlag, 1997.

[5] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Saha, "Key management for secure internet multicast using boolean function minimization techniques," in *Proceedings of the IEEE INFOCOM*, pp. 689-698, New York, USA, 1999.

[6] B. Decleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, and C. Zhang, "Secure group communications for wireless networks," in *Proceedings of the Milcon*, pp. 113-117, 2001.

[7] M. T. El-Hadidi, N. H. Hegazi, and H. K. Aslan, "Logic-based analysis of a new hybrid encryption protocol for authentication and key distribution," *IFIP SEC'98 Conference*, pp. 173-183,1998.

[8] H. El-Rewini and M. Abd-El-Barr, *Advanced Computer and Parallel Processing*, John Wiley & Sons, Inc., 2005.

[9] W. Ford, *Computer Communications Security- Principals, Standard Protocols and Techniques*, Prentice-Hall, New Jersey, 1994.

[10] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, 1995.

[11] J. Hennessy and D. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan Kaufmann Publishers, 2003.

[12] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[13] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM Conference in Computer and Communication Security*, pp. 235-244, Athens, Greece, 2000.

[14] Z. Lan, V. Taylor, and G. Bryan, "Dynamic load balancing of SAMR applications on distributed systems," in *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pp.36-36, Denver, Colorado, 2001.

[15] D. A. McGrew and A. T. Sherman, *Key Establishment in Large Dynamic Groups Using One-way Function*

Trees, Technical Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, 1998.

[16] S. Mittra, "Iolus: a framework for scalable secure multicasting," in *Proceedings of the ACM SIGCOMM' 97*, pp. 277-288, New York, USA, 1997.

[17] J. A. M. Naranjo, L. G. Casado, and J. A. Lopez-Ramos, "Group oriented renewal of secrets and its application to secure multicast," *Journal of Information Science and Engineering*, vol. 27, no. 4, pp. 1303-1313, 2011.

[18] A. Perrig, D. Song, and J. D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 247-262, Oakland, California, USA, 2001.

[19] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys*, vol. 35, no. 3, pp. 309-329, 2003.

[20] M. M. Rasslan, Y. H. Dakroury, and H. K. Aslan, "A new secure multicast key distribution protocol using combinatorial boolean approach," *International Journal of Network Security*, vol. 8, no. 1, pp. 75-89, 2011.

[21] S. Setia, S. Koussih, and S. Jajodia, "Kronos: a scalable group re-keying approach for secure multicast," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 215-228, Oakland, California, USA, 2000.

[22] S. Setia, S. Zhu, and S. R. Jajodia, *A Scalable and Reliable Key Distribution Protocol for Multicast group Rekeying*, Technical Report, George Mason University, January 2002.

[23] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran, "An application-driven study of parallel system overheads and network bandwidth requirements," *IEEE Transaction on Parallel and Distributed Systems*, vol. 10, no. 3, pp. 183-192, 1999.

[24] R. Srinivasan, V. Vaidehi, R. Rajaraman, S. Kanagaraj, R. Chidambaram Kalimuthu, and R. Dharmaraj, "Secure Group Key Management Scheme for Multicast Networks", *International Journal of Network Security*, vol.11, no.1, pp. 33-38, 2010.

[25] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," in *Proceedings of the 3rd ACM Conference on Computers and Communications Security*, pp. 31-37, New York, USA, 1996.

[26] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graph," in *Proceedings of the ACM SIGCOMM' 98*, pp. 68-79, Canada, 1998.

[27] C. K. Wong, S. S. Lam, D. Y. Lee, and Y. R. Yang, "Protocol design for scalable and reliable group rekeying," in *Proceedings of SPIE Conference on Scalability and Traffic Control in IP Networks*, pp. 908-922, Denver, CO, 2001.

Heba Kamal Aslan is an Associate Professor at Electronics Research Institute, Cairo-Egypt. She received her B. Sc. degree, M. Sc. degree an Electronics and Communications Engineering from Faculty of Engineering, Cairo University, Egypt in 1990, 1994 and 1998 respectively. Aslan has supervised several masters and Ph. D. students in the field of computer network security. Her research interests include: Key Distribution Protocols, Authentication Protocols, Logical Analysis of Protocols and Intrusion Detection Systems.

Ghada Farouk ElKabbany is an Assistant Professor at Electronics Research Institute, Cairo-Egypt. She received her B. Sc. degree, M. Sc. degree and Ph. D. degree in Electronics and Communications Engineering from Faculty of Engineering, Cairo University, Egypt. Her research interests include: High Performance Computing (HPC), Robotics and Computer Network Security.