BATCH REKEYING IN MYKIL KEY MANAGEMENT SYSTEM

Wesley Willett, Jyh-How Huang and Shivakant Mishra
Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430, USA.

ABSTRACT
This paper describes support for batch rekeying in Mykil, a key management system for supporting secure group multicast. Batch rekeying in Mykil allows for substantial reductions in the cost and complexity of rekeying operations, arguably the expensive portion of the system. By accumulating join and leave operations and utilizing an algorithmic approach to minimize the number of necessary rekeying messages, Mykil can achieve a marked reduction in the total number and size of messages and can significantly streamline the system's operation.

KEY WORDS
Key management, Performance, Batching.

1 Introduction

Today's increasingly complex, pervasive, and media-rich Internet applications frequently rely on secure group multicast to provide an efficient and protected data transmission to large number of users. Several secure multicast systems for large groups on top of IP multicast have been proposed [8, 6, 7, 9, 1, 2, 11, 4]. A key management server manages a set of cryptographic keys used for encrypting and decrypting the multicast data. The process of updating the cryptographic keys, and distributing them to the group members is called a *rekeying* operation. Rekeying is required in secure multicast to ensure that only the *current* group members can send encrypted multicast data, and decrypt the received multicast data.

Batching has been proposed [2, 10, 5, 8] to reduce the frequency of rekeying operations. The main idea of batching is to perform a rekeying operation only after a minimum number of member join/leave requests have been received or after a certain time interval has elapsed. This paper describes support for batching in the Mykil key management system [4, 3]. The paper presents a new algorithm for batch rekeying, incorporates it in Mykil, and demonstrates substantial reductions in cost and complexity of rekeying operations.

2 Overview of Mykil

The Mykil protocol for key management combines the best features of group-based and key-based key management hierarchies, while mitigating their weaknesses. Specifically, Mykil implements a group structure similar to that of Iolus[6], while using a key structure based on LKH [9] to control the distribution of keys inside each area in the group structure. A multicast group is divided into several areas, each of which is managed by an area controller (AC). Each AC is a member of both its area and another parent area from which it forwards multicast data.

An AC manages keys using a tree structure where each node in the tree corresponds to an auxiliary key. Each client in the AC's area is represented by a leaf node. At join, each client receives all of the keys from nodes on the path between its leaf and the root of the tree. If that member then leaves the group, only the keys it possessed must be updated. To do this, the AC simply creates new random keys along the path to the leaving client and multicasts them to the group, encrypting each key by its children before they are sent. Mykil implements this system using a binary tree of fixed depth. This means that rather than transmit n unicast messages (where n is the number of clients in the area), the AC only needs to transmit a single multicast message consisting of $O(logn)$ keys.

3 Batch Rekeying

Even though the rekeying algorithm of Mykil is considerably more efficient than any naive approach, e.g. a key star approach, it still requires a large volume of message traffic between an area controller and the area clients, particularly if there are frequent membership changes. Imagine, for instance, an area with a capacity of 4096 members, in which one quarter (1024) of the members either join or leave the group within a one-minute interval. Such a scenario is possible at the end/beginning of a pay-per-view program. Using a key star approach, 1024 joins would require 1024 separate multicast updates of the group key, each encrypted with the old group key and signed for a total of 2048 encryptions. On the other hand, 1024 leave events would require 1024 rekey events each of which would unicast a key update to every remaining member of the group for a total of more than three million updates, each of which would have to be encrypted and signed by the server, for a total of more than seven million encryptions. An auxiliary key tree hierarchy that is used with in each area in Mykil would require a similar number of join updates, but significantly reduces the number of leave updates. It requires 1024 signed multicast messages to handle 1024 leave events, each con-

taining a set of 12 keys ($tree\ depth - 1$) encrypted by the keys from their children nodes. This results in a much lower number of encryptions, only 25,600.

Fortunately, we can reduce this number even more by realizing that most of the messages in both of these examples are redundant. In the joins, for example, the same group key is being replaced 1024 different times over the course of a minute, an unnecessary overhead in most multicast applications. Also, since the clients leaving the group inevitably share a significant number of auxiliary keys, many of these keys are being updated hundreds of times in the course of a minute. By combining, or batching, multiple rekey events into a single event we can eliminate this redundancy and greatly improve rekeying performance.

The main idea of batching is to perform a rekeying operation only after a minimum number of member join/leave requests have been received, and/or a certain time interval has elapsed. In a batched system, the 1024 join events in the example above could effectively be replaced by a single multicast of the group key update at the end of the one minute interval. This single group key could be encrypted and signed for a total of only 3 encryptions. Leave events could also be combined, eliminating the need to encrypt and transmit redundant keys, providing significant reduction. Also, since a single multicast message, rather than 1024, would need to be signed and transmitted, an immediate savings is achieved there as well.

These numbers are somewhat incomplete in that they do not take into account the actual size of the individual messages, which will be larger in batched rekeying than in the non-batched rekeying implementations. However, since encryptions are the most compute-intensive part of rekeying, they do provide a compelling case for batching rekey events. We will show later that batching also provides meaningful improvements in total message size.

3.1 Batching of Join Events

While the batching of join events is of interest, and is implemented in Mykil, it is a fairly simple process, and we will not describe it in as much great detail as the batching of leave events. Essentially, the process of batching joins entails simply waiting for multiple join events to occur before transmitting an updated group key to the clients already in the area. Batching of join events occurs in conjunction with the batching of leave events and uses the same mechanism, but requires only the group key to be transmitted. In fact, because joins update only the group key and leave events also update the group key, once any join or leave event occurs, no additional processing needs to be done to add additional joins to the batch.

The unfortunate consequence of batching join events is that it delays the ability of a joining member to access group multicast data until a batched rekey event occurs. However, in many cases, doing so can result in a significant savings in terms of reduced number of messages exchanged. While each non-batched join only necessitates the distribution of a single group key, rather than an entire string of keys as in a non-batched leave event, each individual message must still be signed by the area controller in order to insure its validity, significantly increasing the processing time and total bytes transmitted.

3.2 Batching of Leave Events

Mykil batches leave events by algorithmically combining the keysets of multiple leave events. This batching functionality is broken into three distinct steps, two of which take place at the area controller, and a third is performed by the client. First, rather than generating and distributing new keys when a member leaves the group, the AC must instead track all deleted keys and combine redundant paths. At some later point, generally at a specified time interval or before a multicast event, the second step occurs. At this point all of the nodes on the accumulated leave paths must be encrypted and multicast to clients in the area. Area clients must then update the appropriate keys in their keysets in order to decrypt new multicast data.

3.3 The Update Tree

In order to provide this functionality, Mykil builds a second tree known as the update tree. This tree mirrors the original key hierarchy tree in that it is a binary tree with every node in the tree representing a key in the hierarchy and every leaf representing a client. However, in order to conserve space, the update tree is created as an unbalanced tree containing spaces only for those nodes that have been added to it. Before any client leave events have occurred, or after a batch leave event, this tree is empty. When a client leaves the area, either voluntarily or because of a subscription expiration or connection loss, the initial delete process removes the client from the AC's list of clients. The path in the key tree corresponding to the client's id is then parsed and new keys are generated. These keys are then added to the update tree along a path identical to the one in the key tree. This is made convenient by the fact that, like in the key tree, the path to a client's leaf node corresponds directly to the last d bits(where d is the depth of the fixed tree) of the binary interpretation of the client's id. In a tree of fixed depth 4, like the one given in Figure 1, a client with id 1 (001) would have a tree path of left-left-right, while a client with id 3 (011) would have a tree path left-right-right. Additional deletes follow in a similar fashion, however, because the paths of the deleted nodes will inevitably overlap one another, the construction of the update tree will become progressively easier. In the aforementioned tree of depth 4, for example, deleting node 1 immediately after deleting node 0 would result in the addition of only one new node to the update tree since both of node 1's (and hence node 0's) parents would already be in the update tree. The update tree corresponding to the deletion of nodes 0, 1, and 3 is shown in Figure 2.

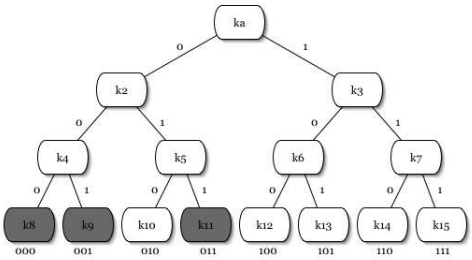

Figure 1. Client Ids.

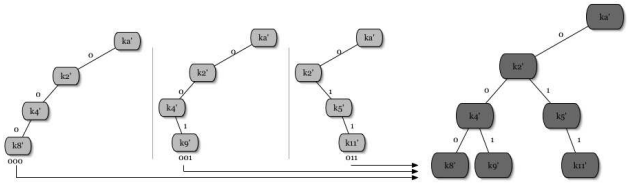

Figure 2. Update Tree after deletion of 0, 1, and 3.

3.4 Encryption Methodology

Eventually, either after a predetermined interval of time, aggregation of a predetermined number of individual leave events, or before a multicast event, the actual batch leave message is sent. A time interval, or rekey interval, may be desirable, for example, in a subscription-based system where expired clients need to be regularly removed to prevent them from receiving service for longer than they paid for. Regular processing of batches may also be desired in order to help preserve key freshness, particularly freshness of the group key used to encrypt multicast data. Maintaining fresh keys can help prevent against both replay attacks and computational attacks, although Mykil already provides significant mechanisms to protect against them. Also, processing at regular intervals allows more flexibility in processing expiring members in groups that use time-based subscriptions, since members' expiration times can be checked against the next rekey interval in advance and the update tree can be built in background in the time between rekey events rather than at moment of expiration. Processing after the receipt of a fixed number of leave events (the rekey threshold) may be desirable in order to maintain enough free space in the key tree for joining members. Remember that even while leaving members have been processed and added to the update tree, they will not be removed from the group until the batch leave message is sent. Generally, as we will see later, the amalgamation of more leave events results in a greater overall bandwidth savings, but requires a larger single message, more processing time and thus a greater delay at both the area controller and the client when the leave is processed.

In some scenarios it may be desirable to process a batch leave only before or after certain multicast events in the system. In a pay-per-view video system, for example, members generally pay for the entire film, and it may be desirable to only process leave events after the multicast of the current video session is completed, or before the start of a new one. A multicast event could also be something much less significant, even some thing as frequent as the receipt of a multicast data packet by the area controller. This could be desirable in systems where join and leave events occur rapidly but data transmission does not occur in significant amounts or at regular intervals. Batching will result in decreased bandwidth usage and fewer messages in nearly all cases, since even a batched message containing only a single leave event will be only slightly larger than a leave event in a non-batched system. This slight increase is due to the fact that a batched message must contain additional information about the tree structure, while a non-batched message only needs to contain a list of keys.

To ensure that only update messages originating at the real area controller are accepted by clients, the rekey message should be signed using the AC's private key. This step is fairly expensive, but the use of batching helps to mitigate it by dramatically reducing the number of messages that must be signed and verified.

Once a batch leave event is triggered, the entire update tree, containing new keys which replace all of those possessed by the deleted clients, is processed and multicast as a single message to the entire area. As in a standard, non-batched leave, all new keys are included, encrypted by the keys of each respective node's children. This prevents the deleted clients from decrypting any of the new keys, while allowing all other clients to decrypt keys on their path. Once the leave event occurs, the AC begins multicasting data using the updated group key, excluding those clients who have been removed.

3.5 Processing of Batch Message

Once a client receives a batched leave message, it can find its appropriate keys by simply parsing the update tree, which was transmitted in its entirety, along the path corresponding to its client id. This path can potentially be very short and require very few updates if the client is isolated from the deleted members, or may be nearly as long as the tree depth if one of the client's neighbors was removed.

3.6 Compromises

In general, batch leaving should provide a significant decrease both in the number and overall size of rekeying messages. In order to do this, it is important to note that we make a few compromises. For example, while batching reduces the overall rekey frequency and total size of the rekey messages, individual rekeying messages are significantly larger. Another consequence of batching leave events is that clients do not immediately relinquish access once they

are deleted, and can continue to receive multicast information up until the moment the batch leave is sent. [5] refers to this period of time as the vulnerability window. While in many multicast applications this may not be an issue, it may be a security concern in others. In systems where this is the case, the length of the rekeying interval can be reduced in order to strike an appropriate balance between rekeying efficiency and the security of the multicast data.

4 Implementation and Performance

We have developed a prototype of the Mykil multicast system which implements batch leaving on a network of Linux workstations. We performed our analysis using a fixed tree of depth 10 and a corresponding group size of 512. Periodic batch rekey events occur at set intervals.

4.1 Storage Requirements

Batch rekeying has little impact on the long term storage requirements of Mykil clients. As in previous versions of the Mykil protocol, clients need to store public keys for the area controller registration server and for other ACs in the group, its own public/private key set, and 12 auxiliary symmetric keys (assuming areas are capped at 4096 members), for a total of roughly 5KB.

However, since the batching of rekey events results in larger message sizes, clients' memory requirements are increased. While they do not need to store the contents of these update messages, clients must have enough available memory to load the entire update tree and parse it to find the appropriate keys. In the most expensive case, a batched leave could encompass the removal of every possible group member and could therefore be as large or larger than the key tree maintained by the area controller. Moreover, once transmitted, the update tree must contain two versions of each transmitted node, one encrypted by each of its children. For a group of 4096 members, this could mean an additional space requirement of almost 254KB. Because this requirement could prove prohibitive on devices like PDAs, rekey events would most likely be triggered at after a smaller number of leave events on such a system. If, for example, rekey events are mandated after 1024 leave events, the size(worst case) of the transmitted update tree drops to only about 100KB and even smaller rekey thresholds can be used to decrease this size further.

Storage requirements for area controllers are increased by a similar amount since the area controller must constantly maintain the update tree. As a result, the area controller likely needs an additional 50-127KB of memory (depending on the rekey threshold) in addition to the roughly 133KB required to maintain the key tree and to store public keys for the rs and other area controllers. However, area controllers are expected to be machines with significant resources, and this additional space requirement is fairly trivial.

4.2 CPU Requirements

The addition of batching to the Mykil protocol dramatically decreases the number of encryptions required by the area controller in rekey events, significantly reducing the amount of processor use. However, this is somewhat balanced by the additional operations necessary to maintain and transmit the update tree.

Clients are required to do significantly more processing before as they must now decode and parse the update tree at each rekey event, rather than examining a single key set. However, because rekey operations occur with less frequency in the batched system, the client benefits from only having to perform rekey processing at intervals.

4.3 Bandwidth Requirements

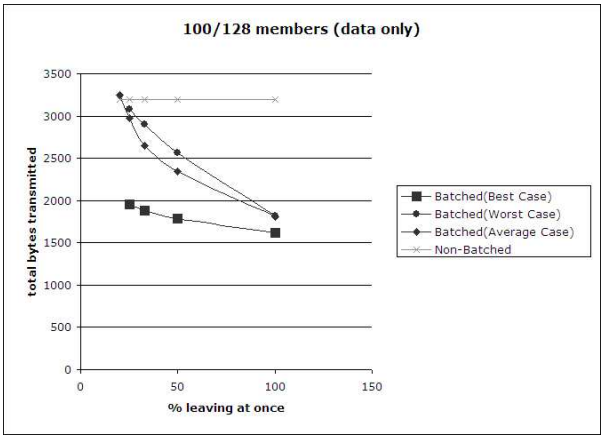

Figure 3. Number of bytes transmitted (No signatures).

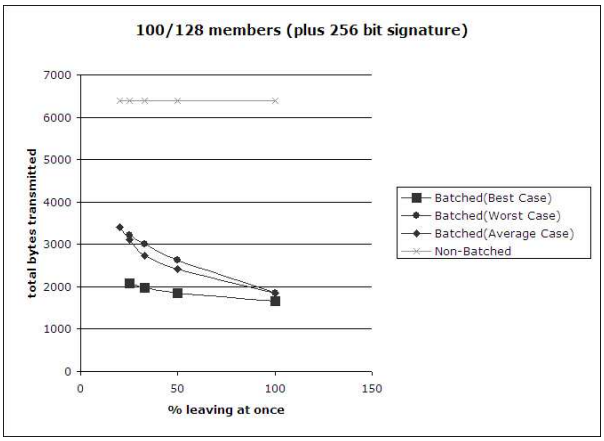

Figure 4. Number of bytes transmitted (with signatures).

Similarly, while bandwidth requirements overall will tend to decrease in the batched system, individual rekey

messages will be larger. This means that rather than transmitting a single key set (roughly 1.5KB in size) at every rekey event, the system will transmit a smaller number of larger rekey messages (ranging anywhere from 2-150KB depending on the number of leaves occurring and the leave threshold chosen). The batched system actually saves a significant amount of total bandwidth here because the area controller only needs to sign a relatively small number of messages using its public key. Figures 3 and 4 demonstrate this savings as seen in a small Mykil test area with a maximum of 128 members, with symmetric keys 32 bits long. In 3, where signature costs are not considered, the message size of small batch leaves begins to approach, and can even overtake that of the non-batched system. This is due to the fact that the update tree must contain slightly more information than a key set in order to represent the same keys, namely directional data which is required to store the unbalanced tree. It is important to note that, because of this, very small batches of leave events will always be more costly than their non-batched counterparts, even when the cost of signatures is included. Once the cost of message signing is factored in, however, the bandwidth savings of the batched case begins to ramp up dramatically, as seen in 4.

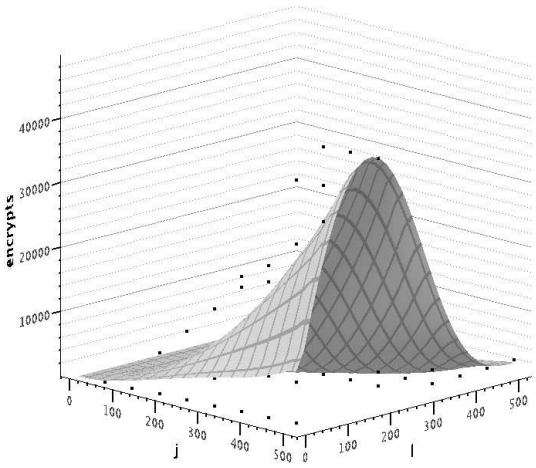

Figure 5. Number of encryptions.

This presents an interesting dilemma since, as 4 shows, the total bandwidth savings increases as greater numbers of leave events are accumulated. This bandwidth savings comes at the cost of increased memory and processor requirements as well as the potential for bottlenecks given the increasing size of individual messages. Additionally, longer rekey intervals increase the size of the vulnerability window. All of these factors must be taken into account in order to select a proper rekey interval and/or rekey threshold in a given implementation.

4.4 Batch Performance

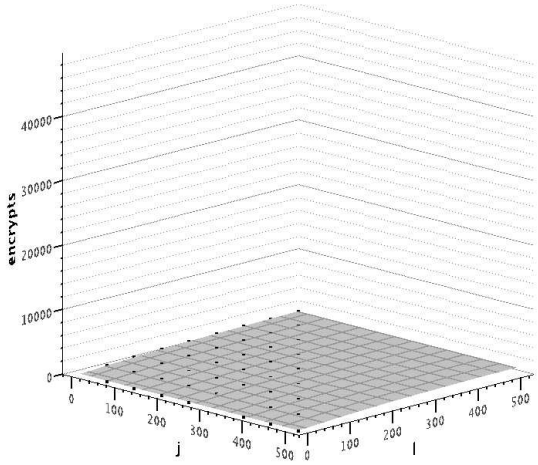

Figure 6. Number of encryptions.

When compared against the performance of a non-batched version of Mykil or a key star hierarchy, the batched Mykil system soundly outperforms the other two in terms of encryptions and the total number of messages transmitted in almost all circumstances. We continue to use the number of encryptions as a comparison scale. In this section, all comparisons are based on a worst-case scenario for batching. We assume that clients are distributed in the tree such that a batch operation produces the largest possible update tree for that number of clients. For example, in a group of 1024 clients a worst case group would include the clients 0, 1023, 511, 512, 255, 256, 767, 768 . . . etc. where each successive node added is the most isolated node remaining in the key tree. In practice we have found that the worst case scenario provides a close approximation of the average case in areas with very low or very high populations, but that the average case improves in moderately populated scenarios. The best case, in which all client nodes are located consecutively in the tree, offers even more substantial gains in moderately populated trees, but also begins to asymptotically approach the worst case as client numbers near the extremes of the area capacity. Mykil is designed to reuse vacated client positions in order to maintain a more concise tree, however, in this scenario we use a worst case analysis, since statistically such a distribution could conceivably occur.

As mentioned previously, Mykil generally exhibits significant performance gains when compared to key star implementations. As seen in Figures 5 and 6, there exist some peripheral cases, including either very small groups or situations in which nearly the entire group departs at once, where key star provides superior numbers. In large

groups with frequent membership changes, like the ones we are dealing with, these cases are unlikely to occur with any sort of regularity. Comparisons against non-batched variations (Figure 7) of Mykil also show large improvements, especially when large numbers of joins and leaves take place during the interval.

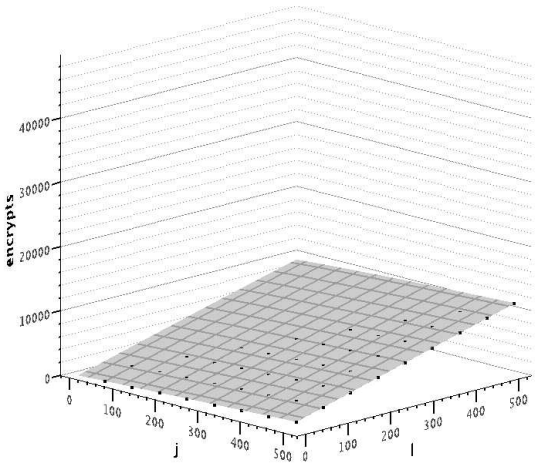

Figure 7. Comparison with non-batched version.

For reference, we also provide data (Figure 8) for a Mykil area batching only leave) events. Such a system may still be desirable in situations where new clients need to immediately begin accessing data. Because leave events are much more complex operations than join events, a leave-only system still produces far better results than one with no batching. But because each join now results in a new, signed message, there is a noticeable increase in encryptions over implementations that batch both joins and leaves.

5 Conclusion

This paper addressed the basis for and implementation of batch rekeying in the Mykil key distribution protocol. Through a prototype implementation, we quantified the performance gains provided by the batching of rekey events and compared them with the performance of existing hierarchies such as key star, as well as non-batched versions of the Mykil. Future work includes using our Mykil prototype to build a large-scale system for disseminating real multicast data. Such a system should take advantage both of Mykil's support for mobility and fault tolerance and its efficient key distribution structure.

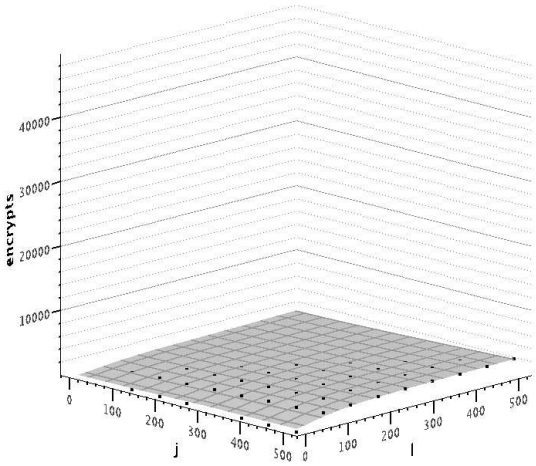

Figure 8. Comparison with non-batched version.

References

[1] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *INFOCOMM'99*.

[2] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, and D. Daha. Key management for secure Internet multicast using boolean function minimization technique. In *SIGCOMM'99*.

[3] J.-H. Huang and S. Mishra. Mykil: A Highly Scalable and Efficient Key Distribution Protocol for Large Group Multicast. In *GlobeCom 2003*.

[4] J.-H. Huang and S. Mishra. Support for Mobility and Fault Tolerance in Mykil. In *DSN 2004*.

[5] X. Li, Y. Yang, M. Gouda, and S. Lam. Batch rekeying for secure group communications. In *WWW'01*.

[6] S. Mittra. Iolus: A framework for scalable secure multicasting. In *SIGCOMM'97*.

[7] R. Molva and A. Pannetrat. Scalable multicast security in dynamic groups. In *CCS'99*.

[8] S. Setia, S. Koussih, and S. Jajodia. Kronos: A scalable group re-keying approach for secure multicast. In *IEEE Sym. on Res. in Sec. and Pri.*, 2000.

[9] C. Wong, M. Gouda, and S. Lam. Secure group communication using key graphs. In *SIGCOMM'98*.

[10] R. Yang, S. Li, B. Zhang, and S. Lam. Reliable group rekeying: A performance analysis. In *SIGCOMM'01*.

[11] S. Zhu, S. Setia, and S. Jajodia. Performance optimizations for group key management schemes. In *ICDCS'03*.