A Flexible Privacy enhanced Location Based Services System Framework and Practice

Y. Sun[†], T. F. La Porta[†], P. Kermani[‡]

[†]Networking and Security Research Center

Department of Computer Science and Engineering

The Pennsylvania State University, University Park, PA 16802

[‡]IBM T.J. Watson Research, Hawthorne, NY 10532

Email: {yasun, tlp}@cse.psu.edu, kermani@us.ibm.com

Abstract

Location based services (LBS) are becoming increasingly important to the success and attractiveness of next generation wireless systems. However, a natural tension arises between the need for user privacy and the flexible use of location information. In this paper we present a framework to support privacy enhanced location based services. We classify the services according to several basic criteria and we propose a hierarchical key distribution method to support these services. The main idea behind the system is to hierarchically encrypt location information under different keys, and distribute the appropriate keys only to group members with the necessary permission. Four methods are proposed to deliver hierarchical location information while maintaining privacy. We propose a key tree rebalancing algorithm to maintain the re-keying performance of the group key management. Furthermore, we present a practical LBS system implementation. Hierarchical location information coding offers flexible location information access which enables a rich set of location based services. Our load tests show such a system is highly practical with good efficiency and scalability.

I. INTRODUCTION

Location based services (LBS) are one of the most desirable classes of service expected to be offered in future wireless systems. Service providers envision offering many new services based on user location as well as augmenting many existing services with location information.

A natural tension arises when attempting to protect user privacy while building a system that allows for flexible use of location information. Specifically, an LBS system must be able to provide means to protect the privacy of a user's location. Great effort has been given to protect user identity and location within basic cellular communication services. This protection must not be compromised with the addition of LBS. Flexibility is also essential so that new services may be introduced and modified quickly. However, providing flexible access to location information will weaken user privacy if care is not taken.

Our solution is based on the following philosophy: access to location information should be controlled by the user. The user defines a group of entities that are allowed to access its location information called a *location*

information group. This group may include other end users and servers in the network that use the location information to provide intelligent services. To increase flexibility, our system supports the hierarchical coding of location information so that different group members may have access to only the granularity of location information required to deliver the appropriate service.

Our approach is based on providing members of the location information group keys that allow them to decrypt location information. The architecture we propose supports loose coupling with a network thus allowing third party control. Emerging social and community networks are especially promising platforms for LBSs deployment. Most next generation LBSs are based on location or proximity sharing with groups. These services consist of groups with different sizes and have various service requirements.

Our framework supports location privacy control by the end user. The architecture supports a range of trust models that trade-off user control and scalability. The flexible location privacy policies are enforced by multi-level group key management. It is challenging to efficiently distribute the keys in the face of changing group membership while supporting hierarchical location information. There is a trade-off in complexity of key management, multicast group management, and delivery overhead depending on the group dynamics.

This paper presents an architecture, implementation and evaluation of a system for providing privacy enhanced LBS with service flexibility. The following is a summary of the contributions and results in this paper:

- We present an architecture to support flexible location based services while maintaining location privacy. The architecture allows users to share their location information at different levels of granularity and with different levels of user control. We map this architecture onto emerging location based services.
- We propose four different methods to support hierarchical location information dissemination to provide flexible location privacy control. To show how this hierarchy can be supported we propose extensions to one suitable key distribution protocol, Multimedia Internet KEYing (MIKEY)[1], to enable efficient re-keying through the application of the Logical Key Hierarchy (LKH) [2][3] protocol. We further tailor the protocol according to four different hierarchical location information dissemination methods and analyze the protocol efficiency.
- We design, implement and extensively evaluate a modified balanced binary key tree (M_AVL) rebalancing algorithm which maintains re-keying performance when used with group key management. We further propose a batch group member removal scheme to optimize the re-keying overhead. When using M_AVL, the key tree is 15% shorter on average than when no rebalancing is performed. Furthermore, if batch leaving is supported multicast traffic is reduced by more than 50%. We also show how trade-offs may be made in terms of location privacy and performance.

- We implement an LBS system extending the concepts we propose. This includes defining and implementing APIs for a variety of service functions and the detailed protocols for the MIKEY-LKH based group key management protocol. To examine the scalability and the efficiency of the system, we conduct comprehensive load tests for large group services.
- To exercise the APIs and show the utility of the system we implement two basic services, *LocateFriends* and *BuddyAlert*, which represent the location sharing and the proximity service respectively, on the application server as enhancements to instant message services. We also demonstrate the trade-off of different policy enforcements with regard to different service requirements.
- Finally we extend our system by integrating an open service architecture on which our system can be mapped.

The remainder of the paper is organized as follows. Section II discusses related research. Section III presents an overview of our system. In Section IV we describe the details of hierarchical coding and location information delivery. Section V presents our group management schemes and shows the analysis results. In Section VI we define the protocol and key tree rebalancing algorithm in details. Section VII demonstrates the system implementation and privacy protection options. Section VIII explores the performance of the system implementation. We enhance the system with the open service architecture in Section IX. We conclude in Section X.

II. RELATED WORK

Location-based services and applications have been emerging with the rapid development of indoor and outdoor positioning technologies such as GPS and RFID. Localization systems such as POLS [4] have been developed to support end device controlled location determination. Standards exist in both the 3G [5] and IETF [6] arenas. In particular, the IETF geopriv group identifies the need to securely gather and transfer location information for location services, while at the same time protecting the privacy of the individuals involved. They do not identify the requirements for LBSs in future large scale community and social networks. No group based privacy control solution is identified and no end user based solution is proposed. Parlay/OSA provides a standardized, extendable and scalable interface that enables rapid creation of telecommunication services which can be used to build integrated location-based services [5].

Distributed trust models in pervasive computing have been extensively studied [7][8][9]. In [7] the relationships among the access to general information is integrated in a certificate-based access control scheme. In [8] an omnipresent trust model based on multi-hop recommendation and a behavioral model to handle interactions

is proposed for pervasive environments. In [9] a Bayesian framework for trust formation and evolution in pervasive computing is presented. These models can be used to form or bundle the access control of location information.

There are concerns on how to protect location privacy while still providing the benefits of LBS. Previous work can be classified into two different approaches: solutions based on privacy policies [10][11][6], and those based on a location anonymizer[12][13] [14][15][16]. The latter provides fuzzy location to make the identification of a device un-linkable to its location. The main drawback of this method is that it may lead to inferior quality of service when accurate location information is desired. Also, anonymization is not effective when there is no mutual trust between the user and the central anonymity server. A more recent work [17] presents user controllable privacy policies in a People Finder Application. We presents a framework that can fully support location privacy control by the end device.

SmokeScreen [18] proposes mechanisms to share presence among trusted parties by broadcasting clique beacons, and with strangers by delegating privacy control to a broker. It uses cryptographic methods to protect the presence information. It does not specify how the common group key is setup or address re-keying when membership is revoked.

One important aspect of group communication security is access control, which can be achieved by encrypting the communication content using a secret key known to all group members. Many group key management protocols have been proposed to support these solutions. Solutions may be classified based on the existence of a Key Distribution Center, which is responsible for the key generation, and whether key management and distribution are centralized or distributed [2].

In a centralized system, there is only one entity controlling the group; protocols for these systems are often based on logical key trees, e.g. Logical Key Hierarchy (LKH) [3][2], One-way Function Tree (OFT) [19], or Efficient Large-Group Key (ELK) [20]. The best solutions appear to be those using a hierarchical tree of key-encrypting keys. They achieve good overall results without compromising any aspects of security. We extend LKH to support multiple level location information encryption. Our extensions can be modified to adapt to other logical key trees based schemes as well.

Key tree rebalance is important to maintain the re-keying performance of group key management. A method proposed in [21] is based on the AVL tree. The technique does not take into account backward security requirements. Moyer et al. [22] propose two enhancements to rebalance the key tree after removing a member. Our scheme is based on a modified balance binary tree in which the key tree balance is maintained by swapping subtrees. This approach incurs less re-keying overhead.

Hao et al. [23] propose to use a leaving tree in addition to an AVL tree sorted by departure time where a member's departure time is known in advance. Zhu et al. [24] propose two optimizations for the LKH scheme based on the temporal patterns or the loss probabilities of group members in addition to batch re-keying. We adopt the batch removal scheme to further optimize the re-keying performance.

III. SOLUTION OVERVIEW

In this section we describe the location based services architecture, give an example of location information dissemination, discuss the service scope, trust and threat model, and illustrate location sharing in social networks with simple services using this architecture.

A. Service Architecture and Trust Model

The architecture we propose is shown in Figure 1. This architecture includes the basic functions required to provide an LBS and does not imply a physical implementation or deployment.

The user device may generate, or assist in generating, its own location information, and may receive the location of other end users as part of a service. To be able to correctly receive the location information of a client, the recipient must be a member in a location information group controlled by the client. The location information group defines the members that receive location information at a particular granularity. Access to the location information by a group is controlled through the distribution of keys that decrypt the location information.

The architecture supports a range of control by the end user. In the most highly controlled case, the end user generates their own location information and encrypts it, and distributes keys directly to the other members of the location information group using a protocol like Diffie-Hellman, or even running a group key management protocol as described later in this paper. This solution is feasible for small information groups. For larger groups, the end user may form a trust relationship with a server in the network. Depending on the level of trust, the server(s) may have more or less access to the location information. At one level, the user may allow the network to store and distribute its location information in an encrypted form and still manage key distribution itself. If the end user has more trust in the network, it may inform a server of the members of the location information group and allow the server to perform key distribution and location information dissemination on its behalf.

The Location Server (LS) is a network server that stores location information. If the LS is a member of the location information group, it may also process the location information. The group server manages the location information groups that are to receive the location information including both end users and network servers.

Fig. 1. Location based services architecture.

If it is not trusted by the end user, this just entails managing the multicast delivery of the encrypted location information to the members of the group. If it is trusted, the group server also manages the distribution of the keys to the location information group. The application servers (AS) provide the end service which uses the location information. These functions may be combined in various ways in a network deployment.

One important aspect of this system is the *hierarchical coding of location information* to allow efficient and flexible location information access control. When location information is generated, for example using GPS or TDOA techniques, it will typically be in a (x,y) format. This information may then be processed to produce a set of information of differing granularity, for example $\{country, state, city, address, (x,y)\}$. This processing may be done on the end device or it may be done inside the network at the location server if the user has a trust relationship with this server. Our approach is based on providing members of the location information group keys that allow them to decrypt location information. An end user may approve sharing of only the granularity of location information that they desire, and application servers will only have access to the information required to provide their service. The access is granted through the group key management protocol.

For the remainder of this paper we assume there are five classes of location information, with *class 1* corresponding to the most general representation of user location, and *class 5* corresponding to the most specific.

B. Location Information Dissemination

Figure 1 shows a representative example of location information dissemination. In this example, location information is generated by the end device and uploaded into the LS (message 1). The application server requests the location information from the LS on-demand (message 2), and sends it to the group server from which it is disseminated to the end users (message 3). This way, the application server is only concerned with triggers to disseminate the information, and the group server is responsible for managing group membership. In an actual deployment, these functions may be combined for the most efficient operation.

It is important to note that when information is stored in the LS, it is not necessarily accessible by the LS. For example, location information may be generated at the end device, perhaps using GPS, encrypted, and then uploaded to the LS. In this case, the LS will store the information and help disseminate it, thus relieving the end device of this burden, but will not be able to add value to any service. Alternatively, the LS may be a member of a location information group, and be able to decrypt the information and process it, thus adding value to the services. In this case, the LS must have a trust relationship with the end user.

Finally, consider that certain application servers are members of the location information group. These application servers may add their own processing to further increase the intelligence of the services they provide. By supporting hierarchical coding of the location information, they may only be allowed to access the level of information they required for their service.

The flow in Figure 1 is easily modified to support scenarios in which no LS is present, or in which the location information is pushed to the ASs at a specified frequency or in response to a specified event.

C. Services Scope and Threat Model

In this section we revisit the requirements of different Location Based Services and evaluate how our architecture accommodates their needs.

The first generation LBSs are mostly “finder services” which show the users a nearby single point of interest. These services are provided by a server based on self-referenced, single-target and sporadic pull location access. Location privacy is provided by a degree of anonymity by many of these systems.

Next generation emerging LBSs include location sharing and proximity based entertainment, community and social network services. These services consist of large groups and require the delivery of location information to multiple targets with various location access granularity. They are typically push based.

The framework we propose fits the best for this promising class of service. The traditional LBS architecture is network centric, where location information is tightly coupled to specific network operators and not open to the 3rd party Location Based Service Providers. We present an architecture which is end device centric and loosely coupled where the location access is through an open service architecture based on a layer of middleware which is accessible to the 3rd party service providers and users. This architecture adds a Group Server (GS) to provide location access control. Another benefit of this architecture is that the Location Server (LS) can be loosely coupled with the core network. This is essential to support LBSs in heterogeneous networks. Our implementation in Section VII includes all the system components in the architecture.

In our system, an adversary is defined as any party who is not trusted to record a user’s location at a specific granularity and time based on these trust parameters. Our scheme is resilient to this threat since the location

information is encrypted and no information is leaked to eavesdroppers. Our scheme is resilient to collusion among adversaries, and is also resilient to collusion between adversaries and trusted users for instantaneous location tracking if we change the group keys frequently. However, we can not prevent trusted users from unauthorized sharing of secret keys or location information with adversaries.

D. Location Sharing in Social Network and Service examples

There are several emerging on-line social network services such as Friendster, MySpace and Facebook. As of December 18, 2007, over 300 million accounts are registered in MySpace [25]. Friendster currently has 50 million users [26]. Facebook has over 59 million active users worldwide, and more than 80% of Facebook members have used at least one of the 7,000 applications built on the platform [27]. Typical users of Facebook often have social networks of over one hundred users [28]. Community services are becoming more popular. Combined with location information these networks can provide new platforms for emerging mobile community services.

Other location based game and community services are targeting both large and small groups, with high location privacy requirements and high group dynamics. One promising set of applications are Massively Multiplayer Online Role Playing Games (MMORPG) [29]. Botfighters2 [30] and CitiTag [31] are examples of such games which make use of user location and collaboration among members. With more than one hundred thousand players, Botfighters2 is attractive to people who like to play on-line games in groups through their mobile cell phones. A greater degree of location accuracy and social interaction is required to attain their design goals. One of most famous MMORPG, World of Warcraft (WoW), has over 9 million subscribers worldwide [32]. Players join guilds (groups) in order to conduct raids against enemy territories and instances. The average guild size is around 20 and the largest guilds have more than 200 [33] members.

Social networks and their service models are still emerging. Many issues related to privacy are just now being addressed because such large communities on-line have not existed before. Therefore, it is difficult to assume a single model of user behavior among different applications in social networks. Real-time transient entertainment applications are becoming more common in Facebook (such as “Track Santa”) which require low latency group dynamics.

Most of these services are based on sharing the location or proximity information of a group of users. This sharing is the core building block for advanced services. In the following we describe how the basic location proximity service can be implemented and the impact of each of the different deployment options discussed in Sections III-A- III-B on the operation of the service.

The service works as follows. An alert message is sent to subscribers when a buddy is within a certain proximity. The service can be integrated with an Instant Message (IM) system or social network service system (e.g., Facebook) which requests the location information of its subscribers from the LS.

We first consider the case in which only the end users are members of the location information group, i.e., the IM server or social network server is not a member of the location information group. In this case the server plays the role of the application server in Figure 1 and requests location information from the LS periodically. The LS provides the encrypted information to the server, which sends it to the group server to be disseminated to all members of the buddy list (or guild, for example). The end devices of these subscribers decrypt the information, compare it to their own location, and generate an alert locally if a buddy is within proximity of the device.

This implementation of the service has two interesting characteristics. First, the end user has complete control over which buddies know its location information; it may decide to limit the knowledge of its location to a subset (or none) of its buddies. Second, the value added service of proximity detection is implemented primarily in the end device; the server is simply facilitating the delivery of the information.

Now consider the case in which the server is a member of the location information group. In this case, when the receives the location information from the LS, it will decrypt the information, compare it to the location of the other members of the buddy list, and execute directed alerts to only those members that are within proximity of each other. This implementation has three important characteristics. First, the end user must trust the server to only disclose its location information to approved buddies. Second, the delivery of alerts is more efficient than in the end user-based system discussed above, especially for large groups. And third, the server is able to provide increased value to the service without placing a burden on the end device.

IV. DELIVERY OF HIERARCHICAL LOCATION INFORMATION

In this section we discuss four methods of delivering hierarchical location information. Each method has trade-offs in terms of message delivery efficiency and group management complexity. We discuss the key distribution schemes and demonstrate the analysis results in Section V. We present the protocol in detail in Section VI. The interaction between hierarchical coding of location information and group key management are critical for overall system performance. Therefore in this paper, different schemes are extensively evaluated in our real LBS system in Section VIII.

Coding by Information Class In this method, each class of location information, i , is encrypted with its own key, K_i , and location information groups are assigned based on a single class of information. With this system, a user requiring all c levels of location information will join all c groups and receive each piece of

TABLE I
KEYING CHARACTERISTICS

Function	GDOI	GSAKMP	MIKEY
Registration	Y	Y	Y
Re-keying	Y	Y	N
Round-trip time	3.4-5.0	1.5-2.5	0.5-1.0
Hierarchy	N	Y	N

information independently of the others. The *class i* message has a message format $\{Loc_i\}_{<K_i>}$. This scheme is suitable when users typically desire a single class of location information. However, it requires the management of c groups, and if users require more than one class of information, they must join more than one group. Each user will receive c multicast messages; one for each group to which it belongs.

Coding by Group In this method, a group is defined by the highest class of location information accessible to that group; a group receives all location information at or below this class encrypted by a single key, K_{g_i} . The *group i* message takes a message format $\{Loc_1, Loc_2, \dots, Loc_i\}_{<K_{g_i}>}$. This method is suitable when users want more than a single class of information. It uses longer messages than coding by class, but requires users to join fewer groups.

Nested Hierarchy Coding In this method location information is encrypted using a nested hierarchy. End users belong to a location information group according to the maximum level of information they require. Each group has a group key K_{g_i} . Formally, the information is disseminated with a message format, $\{Loc_1, \dots, \{Loc_i, \dots, \{Loc_c\}_{<K_{g_c}>\dots}\}_{<K_{g_i}>\dots}\}_{<K_{g_1}>}$. With this method, only a single multicast group for location delivery must be maintained and the separation of groups is solely dependent on distributing keys.

Flat Coding This scheme assumes a flat relationship among sub-groups. Each member of a location information group receives all of the information at or below the maximum level it requires. The message format for location information delivery to a *class c* user is $\{\{Loc_1\}_{<K_{g_1}>}, \dots, \{Loc_i\}_{<K_{g_i}>}, \dots, \{Loc_c\}_{<K_{g_c}>}\}$. All the location information for a single class of user is included in each message. Each class of user receives messages that only contain up to that class of information; thus the number of multicast groups managed is equal to the number of location classes.

V. GROUP KEY MANAGEMENT

As is evident, a main element of the system is the ability to distribute keys to the appropriate group members. The key distribution protocol must accommodate the hierarchical coding methods discussed in Section IV. The efficiency of the keying protocol will vary depending on group size and group dynamics. It is natural to apply group key management protocols used for multicast communication or collaborative systems to this problem. Sections V-A and V-B introduce the key distribution schemes and analyze the efficiency. Batch re-keying and

re-keying under fine-grain privacy configurations are also explored to demonstrate the flexibility of the privacy control. We summarize the analysis evaluation at the end of Section V-B.

Within the IETF there are several standards or draft standards that address key management for groups. These include the Group Domain of Interpretation (GDOI - RFC3547) [34], GSAKMP(RFC4535) [35], and MIKEY (RFC3830) [1]. Table I compares these three protocols for a small set of important characteristics.

We choose to extend MIKEY [1] as a representative key management protocol because it is a lightweight protocol that can establish group keys with a low latency and has been adopted by the 3G MBMS (Multimedia Broadcast/Multicast Services). Note that the latest GSAKMP and GDOI protocols also recommend LKH in their re-keying schemes and may also be suitably extended.

The MIKEY protocol specifies only the basic registration protocol. Three different methods can be chosen to establish or transport group keys in order to satisfy the system requirements: with the use of a pre-shared key, public-key encryption, and Diffie-Hellman (DH) key exchange. The pre-shared case is the most efficient way to handle the key transport due to the use of symmetric cryptography only. In future integrated networks, users can benefit from the pre-shared keys with the primary network, for example using the existing architecture in 3G networks. If the services are only provided within small size group and without central sever, peer-to-peer Diffie-Hellman (DH) key exchange can be used and re-keying can reuse the basic protocol. The network components can be simplified accordingly.

In summary, there are two limitations of MIKEY. First, it does not support a hierarchy of group key servers which may be important to scale the solution. Second, it does not support re-keying which is essential because group members may leave or join, and thus new keys must be established. We address these limitations in Sections V-A after providing a brief overview of MIKEY and LKH.

A. MIKEY-LKH Applied to Key Distribution Schemes

MIKEY uses a three level key management structure to distribute group keys to the clients. The User Key (MUK) is a point-to-point key between the multicast server and each client. This key is used by the key server to authenticate each client and secure the delivery of MSK. The Service Key (MSK) is shared between the multicast server and all group members. The MSK protects the distribution of Traffic Key (MTK). The MTK is the group key used to protect the data exchanged between members of the group.

The MSK is delivered to the group members after they are authenticated. Delivery of this key is point-to-point through push, pull or push solicited pull. The MTK traffic key delivery can utilize a multicast mechanism. Because MIKEY does not support re-keying, each time a group member joins or leaves, a new MSK must

Fig. 2. LKH hierarchy applied to MIKEY.

be established with all group members, and a new MTK must be multicast to the entire group. Therefore, the overhead for re-keying is $O(N)$, and is not scalable to large groups.

We apply Logical Key Hierarchy (LKH) to MIKEY to improve scalability with re-keying. The LKH protocol uses a balanced tree to represent a logical key tree as shown in Figure 2. Each user owns the keys on the path from the leaf node to the root of the tree. The keys on the path between the leaf node and root of the tree are Key Encryption Keys (KEK); the keys at the leaves of the tree are data encryption keys owned by group members; the root key is the Group Key (GK). In the basic MIKEY protocol, MSK is the Group Key and there is no Key Encryption Key to facilitate the distribution of the Group Keys.

There are two types of security that a key distribution algorithm can provide: backward security, meaning that a new member is not able to decrypt past group communications, and forward security, meaning that a departing member is not able to decrypt future group communications. To maintain backward and forward security, when a user joins or leaves, all the keys on the path have to be changed. For a join or leave event the communication overhead is $O(\log N)$. If the forward and backward security requirement is relaxed, batch re-keying may be performed to reduce the overhead as discussed below.

The four schemes for delivering hierarchical location information and the corresponding four LKH extensions each have trade-offs in terms of message delivery efficiency and group management complexity. We discuss the efficiency of re-keying for the different methods of distributing the hierarchically coded location information below. To quantify this overhead we define the following variables:

- N : number of users in the group, h : height of the tree ($h = \log N$);
- K : encrypted key size, E : cost of encryption operation, D : cost of decryption operation;
- C_m : overhead in terms of bytes for method m , P : overhead for delivering a re-keying packet;
- r_c : ratio of users of class c in terms of percentage, c_{max} : the maximum location classes or sub-groups.

Table II summarizes the messaging, storage and computation overhead for the basic LKH scheme. The overhead for re-keying in the basic LKH scheme is given by:

$$C_{LKH}(N) = 2K \log N + P \quad (1)$$

TABLE II
SUMMARY OF LKH OVERHEAD

(a) Message overhead			(b) Storage overhead	
Join Event		Leave Event	Key Server	Member
<i>Multicast</i>	<i>Unicast</i>	(2h-1)K	(2h-1)K	(h+1)K
(2h-1)K	(h+1)K			

(c) Computation overhead			
Join Event		Leave Event	
KDC	Member	KDC	Member
3hE	(h+1)D	2hE	hD

1) *Coding by information class*: The key server needs to maintain separate groups for each class of location information. A user must subscribe to multiple groups to receive more than one granularity of location information. When re-keying due to a join or leave event, a new key must be distributed to all users that subscribe to the same classes of location information. The tree structure is shown in Figure 3-(a). The cost of re-keying may be expressed as:

$$C_{class}(N) = \sum_{i=1}^{c_{max}} r_i (2K (\sum_{j=1}^i \log \sum_{m=j}^{c_{max}} ((r_m)N)) + iP) \quad (2)$$

Each user of class i must store $\sum_{j=1}^i (h_j + 1)$ keys for each level of location information it wishes to receive, where h_j is the height of the key tree of group level j . The number of multicast groups managed is the same as the number of classes of location information.

2) *Coding by group*: Each group re-keys independently of all other groups. Therefore, if re-keying is performed on a join or leave of a user in group, only those users in group i must be re-keyed. The tree structure is shown in Figure 3-(b). The re-keying procedures follow the basic re-keying protocol as for the basic LKH scheme. The cost of re-keying may be expressed as:

$$C_{group}(N) = \sum_{i=1}^{c_{max}} r_i (2K \log(r_i N) + P) \quad (3)$$

Each user must store $(1 + h_i)$ keys, where h_i is the height of the key tree of group i . Again, the number of multicast groups is equal to the number of location classes.

3) *Nested hierarchy coding*: For nested hierarchical coding a modified LKH tree can be used. The basic assumption for LKH is that the user knows the keys from the leaf (itself) up to the root, and the root key is the group key. In our definition, this is still true, but the modified LKH tree is not a balanced tree while each sub-group is still a balanced tree. The height of the tree is decided by the biggest group and is in general $O(\log N)$. The tree structure is shown in Figure 3-(c).

Fig. 3. Key trees for four coding schemes: (a) class; (b) group; (c) nested; (d) flat.

When a user joins a group, the server adds the user to the appropriate group. The key server will change the keys on the path between the root and the new user. A similar re-keying takes place when a user leaves the tree. Note that when a user with a low priority re-keys, only one group key, K_{g1} , needs to be changed. When a highest class user, class c , re-keys, all the group keys must be changed. This will be efficient if there are many more low priority users than high priority users. In general, the re-keying overhead for the nested method is:

$$C_{nested}(N) = \sum_{i=1}^{c_{max}} r_i (2K(\log(r_i N) + i) + P) \quad (4)$$

A user in class c must store $c + h_c$ keys. This includes one for each class of location information it will access (c), plus the height of the tree of its group, h_c .

4) *Flat encryption*: To make the protocol more general the flat encryption scheme does not rely on information hierarchy and assumes a flat relationship among sub-groups. Each sub-group is still a balanced tree and the key server maintains the information subscription mapping among sub-groups. The tree structure is shown in Figure 3-(d).

When a user joins a group, the key server decides in which sub-group to put the logical node for the user. The re-keying message will be handled by all the relative sub-groups. For example, if a user joins sub-group c , all the sub-groups keys need be changed. The re-keying overhead for this scheme is:

$$C_{flat}(N) = \sum_{i=1}^{c_{max}} r_i (K(2 \log(r_i N) + \sum_{j=1}^i (c_{max} - j + 1)) + c_{max} * P) \quad (5)$$

Each user must store $(c + h_c + 1)$ keys where c is the class of the user, and h_c is the height of the c -class tree.

5) *Batch Re-keying*: Sections V-A.1- V-A.4 derive equations to quantify the average re-keying overhead for the four schemes. If the policy is to re-key on each join or leave event, the overhead calculated above occurs at the rate of the join and leaves. Batch re-keying can be used to reduce the re-keying overhead. In this case the re-keying only happens when a time threshold or count threshold is met. The tradeoff for batch removal is that we can only support weaker forward security because the keys are not updated immediately.

Note that different policies may be applied to different classes of information. For example, batch re-keying may be performed on the low granularity classes, and join/leave re-keying may be performed on the higher levels of granularity information.

B. Key Distribution Schemes Analysis and Discussion

Based on the four hierarchical coding schemes, we can deduce the average height of the tree with the different location precision levels. In this section we first analyze the performance of each keying scheme related to group dynamics assuming re-keying based on join/leave events. Then we show the batch re-keying results for two typical group sizes. Finally we discuss the trade-offs of the various methods described above. We assume the ratio among the 5 different sub-groups is $r_1 : r_2 : r_3 : r_4 : r_5$. We set, the join/leave rate to be 100/hour. The size of an encrypted key is $K = 20bytes$, and the protocol overhead is $P = 50bytes$. The overall overhead for re-keying is then simply $C_{method}(N)$, where C_{method} is the byte overhead of re-keying for each method as defined in equations 2-5.

In Figure 4 we show the overhead for $r_1 : r_2 : r_3 : r_4 : r_5 = 30 : 10 : 8 : 5 : 2$. This represents a system in which many users have access to low granularity location information (e.g., country or state), and few have access to high precision location information. From the figure, we can see that the three schemes (group, nested, and flat) that accommodate hierarchical coding of location information have better performance than the direct application of LKH (coding by class) in terms of re-keying cost. The most efficient scheme in this respect is coding by group, i.e., each user group receives all data under a single key. This is intuitively most efficient as it requires re-keying only for users directly affected by the group membership change. With this ratio nested coding is more efficient than flat coding.

If we reverse the ratio $r_1 : r_2 : r_3 : r_4 : r_5 = 2 : 5 : 8 : 10 : 30$, we see increasing overhead for the coding by class scheme because more users subscribe to multiple groups (not shown). This represents a system in which many users have access to high precision location information. In this case the three schemes that accommodate hierarchical encoding perform similarly.

Figure 5 shows how batch re-keying can further reduce the re-keying overhead in the worst case. We assume inter-arrival time of re-keying members conforms to a normal distribution and are equally distributed among

Fig. 4. Hierarchical LKH re-keying overhead ($r_1 : r_2 : r_3 : r_4 : r_5 = 30 : 10 : 8 : 5 : 2$).

Fig. 5. Batch re-keying overhead ($N = 100$).

the sub-trees. With the smallest batch interval of 5 minutes, the re-keying overhead can be reduced by half. Batch time or batch count should be chosen based on N and the join/leave rate. In Figure 5 with typically a small group ($N = 100$), a 5 minute batch interval is enough to reduce the re-keying overhead to around $10KB/hour$. In contrast, for a large group ($N = 800$, not shown), to accommodate the similar bandwidth, the batch interval has to be set to 25 minutes. The overhead for coding by group scheme are the same for both ratios.

Figure 6 and Figure 7 show how flexible batch policies can be applied on different location levels. For example, a user may not care about people knowing a short history of their country and state information, and thus apply batch re-keying to these low granularity information classes, perhaps performing re-keying once in the morning and once at night. For more precise location information, they may mandate re-keying upon join/leave events to ensure forward and backward security. We assume the inter-arrival time of re-keying members conforms to a normal distribution and are equally distributed among the sub-trees. Both figures show that the re-keying overhead optimization depends on the size of the non-batch re-keying groups. When there are more people in higher level location groups, the overhead does not decrease dramatically below $10KB/hour$ even when 12 hour batch re-keying is configured for 4 levels of location information. When the precise location information is shared with only small groups of people, 12 hour batch re-keying for the country and state information reduces the overhead to within $10KB/hour$. This holds true for both the small group and the large groups in Figure 6 and 7.

The trade-offs of the various methods described above are summarized in Table III.

The coding by class scheme is the direct application of the basic LKH scheme. The coding by group solution is attractive for four reasons. First, it results in low re-keying overhead as shown in Figures 4 and ???. Second, the delivery of messages is efficient because users only receive the data they require. Third, each user receives the data at and below their maximum class level. This potentially simplifies service logic because the recipients of the high granularity data do not have to process it to derive the lower granularity information. Finally, the

Fig. 6. Batch re-keying overhead with different policy per level ($N = 100$).

Fig. 7. Batch re-keying overhead with different policy per level ($N = 800$).

TABLE III
SUMMARY OF MIKEY-LKH OVERHEAD FOR FOUR SCHEMES

Method	Class	Group	Nested	Flat
User storage	$\sum_{i=1}^c (h_i + 1) * K$	$(h_c + 1) * K$	$(h_c + c) * K$	$(h_c + c + 1) * K$
Server storage	$\sum_{i=1}^{c_{max}} (r_i/r_1) * (2N - 1) * K$	$(2N - 1) * K$	$(2N - 1 + C_{max}) * K$	$(2N - 1 + C_{max}) * K$
Multicast groups	$N * c_{max}$	$N * c_{max}$	N	$N * c_{max}$
Re-key	Equation-2	Equation-3	Equation-4	Equation-5
Message delivery	$\{Loc_i\}_{<Kg_i>}, 1 \leq i \leq c$	$\{Loc_1, \dots, Loc_i\}_{<Kg_i>}, 1 \leq i \leq c$	$\{Loc_1, \dots, \{Loc_i, \dots\}_{<Kg_i> \dots\}_{<Kg_1>}$	$\{\{Loc_1\}_{<Kg_1>}, \dots, \{Loc_c\}_{<Kg_c>}\}$
Compu. at end device	$c * D$	$1 * D$	$c * D$	$c * D$
Flexibility	no	neutral	neutral	attractive

key storage requirements on each user device are low.

The main drawback of the coding by group solution is that there are c groups to be managed from both a re-keying and data dissemination perspective. From a re-keying perspective, a tree for each group is maintained. From a data dissemination perspective, each group is delivered a message containing only the data they request.

Like the coding by group solution, with the nested coding solution a user receives all the data at and below its maximum level. It has an additional benefit in that only a single multicast group must be maintained for data dissemination and re-keying.

The flat coding scheme is the most flexible from the user perspective. In this scheme a user may only join the groups they desire. Likewise this method allows users more flexibility in deciding which data to disseminate. It provides the possibility that a user may receive all data at or below their maximum level if so desired.

VI. MIKEY-LKH PROTOCOL AND REBALANCE ALGORITHM DESIGN

In this section we define the key management protocol in detail. It is intuitive to use a binary key tree-based solution for re-keying in which the basic insert or delete operations happen only on the leaf nodes. However, this

may lead to an imbalanced tree which results in poor performance. We address this problem in Sections VI-B.

A. MIKEY-LKH Based Group Key Management Protocol

Below we describe the details of our MIKEY-LKH based group key management protocol. For the sake of clarity, assume the group server is running this protocol. As previously discussed, it can be run on the end device.

When a user joins a location information group, the initial key distribution is made by encapsulating the key information message over a SSL/TLS channel.

The re-keying protocol is the focus of our group key management. The Key Encryption Keys (KEKs) along the key tree path to the tree root are used to encode/decode the updated keys; the group key information is then multicast to group members. The re-key message uses the following format:

$$S \rightarrow * : Hdr, TS, ID_S, \{KData\}_m, ID_{GK'}, \\ MAC(K_{GK'}, Hdr|TS|ID_S|\{KData\}_m|ID_{GK'})$$

$KData$ is defined as follows,

$$KData : \{ID_{KEK_i^{child}}, \{ID_{KEK_i}, KEK_i\}_{<KEK_i^{child}>}\}$$

Notation:

- Hdr is the message header; TS is the time stamp; ID_S is the sender ID of the server.
- KEK is the Key Encryption Key; GK is the Group Key; GK' is the old Group Key; ID_K is the ID of the key K .
- m is the number of the updated keys.
- KEK_i^{child} is the KEK of KEK_i 's child node, where $0 < i < m$.
- $MAC(K_A, H|M)$ is a Message Authentication Code of values H and M , using key K_A .
- $\{M\}_{<K>}$ is a value M encrypted in key K .
- $\{KData\}_m$ is the encrypted keys payload carried in the message.

The key payload in the MIKEY message is extended to support the Key ID list information and KEKs for the LKH scheme. The four LKH extensions introduced in section V-A have different key data payloads corresponding to their key tree structures. Only one multicast message is sent for the nested and the coding by group schemes when a member joins or leaves. More re-keying messages are sent out for the coding by information and the flat schemes depending on the desired location information granularity.

The Time Stamp (TS) in the re-keying message is used for replay protection. AES with a 128-bit key is used for encryption and decryption. HMAC-SHA1 is used for the Message Authentication Code which covers entire message.

Fig. 8. M-AVL key tree structure.

B. Key Tree Rebalancing

The size of key data payload carried in the MIKEY message introduced in section VI-A has direct impact on the re-keying protocol performance. In general, for a balanced binary tree with N leaves, the distance from the root to any leaf is $\log N$. A tree is balanced if the distances from the root node to any two leaf nodes differ by no more than one [21][36]. But if the tree becomes unbalanced, the distance from the root to a leaf can become as high as N . This may degrade the re-keying overhead from $O(\log N)$ to $O(N)$. It is desirable to keep a key management tree as balanced as possible.

Moyer et al. [22] propose two enhancements to rebalance the key tree after removing a member. The first method is to find a member at the deepest tree level to replace the deleted member. This method has up to $4h$ cost, where h is the height of the tree. The second method only invokes the rebalance operation periodically. This solution may incur high overhead since it only replaces one node at a time. This motivates our modified AVL (M-AVL) algorithm which utilizes batch swapping among subtrees instead of separate node replacement.

We define algorithms for *Insert* and *Remove* operations and implement them in our group server. We further propose a batch group member removal algorithm to optimize the re-keying performance. The performance of our key tree algorithms is evaluated in Section VIII. We consider a general Balanced Binary Tree [37] in which not all nodes conform to the balance condition. We allow the tree to have any shape as long as the tree is general balanced, e.g., keeps the logarithmic tree height. Such a balance definition results in a super class of all other classes of balanced trees which is especially useful to design an optimized key tree algorithm.

The general balance rule is defined as following:

$$h \leq c * \log N + b$$

where c, b is constant, $c \geq 1$, h is the height of the tree.

We define the M-AVL algorithm to maintain the key tree balance when members join or leave the group. In addition to the general balance rule, we have the following assumptions.

Algorithm 1 Insert

Input: A new *TreeNode* n , a *M_AVLTree* T the new node to be inserted;
Output: A list L of tree nodes with keys to be changed;
Procedure:
1: **if** T is empty **then**
2: insert n into the empty tree T
3: **else if** T is full **then**
4: create a new root node P
5: current root becomes the left child of P
6: n becomes the right child of P
7: **else**
8: find a shallowest node S
9: create a new internal node B
10: S becomes the left child of B and n becomes the right child of B
11: B becomes the child of S 's parent
12: **end if**
13: update node information along the insertion path and add the nodes into L
14: return L

Algorithm 2 RemoveOneNode

Input: A *TreeNode* n , a *M_AVLTree* T the node to be removed;
Output: A list L of tree nodes with keys to be changed;
Procedure:
1: **if** T is empty **then**
2: return the empty L
3: **else if** n is the child of the root of T **then**
4: remove current root node of T and n
5: the sibling node of n becomes the root of T
6: **else**
7: find the parent node P of n , and the parent Gp of P
8: the sibling node of n becomes the child of Gp
9: remove n and P
10: **end if**
11: update node information along the removal path and add the nodes into L
12: return L

- First, in our M_AVL tree, there are two types of nodes: internal nodes and leaf nodes [37]. We allow mixed type descendants on internal nodes which reduces the number of keys to change when re-keying is necessary.
- Second, we define the internal nodes as logical nodes for the purpose of Key Encryption Keys (KEKs). The leaf nodes which represent a member have the KEKs along the path to the root.
- Finally we note three differences between our Modified AVL tree and the original AVL tree. The original AVL is a balanced binary search tree while the M_AVL is a general balanced tree. Another difference is that the group member is stored only at the leaf nodes. The internal nodes are logical nodes which save the Key Encryption Key (KEK) information. A third difference is that the M_AVL tree does not have to keep the order of the members.

Figure 8 shows a M_AVL key tree structure. Although the root node does not conform to the original AVL balance condition, the tree is general balanced ($N=5$, $c=1$, $b=1$). If node U5 is a newly inserted node, using M_AVL can minimize the keys to be changed since the inserting path is short. When a member leaves, our algorithm checks the longest path to the deepest node, determines the highest relative balanced subtree, and determines the shallowest most unbalance subtree. We switch the heads of two subtrees to rebalance the key tree.

Algorithm 3 Remove

Input: A `TreeNode` n , a `M_AVLTree` T the node to be removed;
Output: A list L of tree nodes with keys to be changed;
Procedure:

```

1: if batch removal is supported then
2:   if batch condition is not met then
3:     add  $n$  in tree  $T$ 's deleting list
4:     return empty  $L$ 
5:   end if
6: else
7:   Invoke RemoveOneNode to remove node  $n$ 
8:   Invoke Rebalance routine
9:   Combine the output and add the nodes into  $L$ 
10:  return  $L$ 
11: end if
12: for every node to be removed do
13:   Invoke RemoveOneNode
14: end for
15: Invoke Rebalance routine
16: update node information along the removal path, combine with rebalance output and add the nodes into  $L$ 
17: return  $L$ 

```

Algorithm 4 Rebalance

Input: A `M_AVLTree` T to be rebalanced;
Output: A list L of tree nodes with keys to be changed;
Procedure:

```

1: while  $T$  is not general balanced do
2:   find the highest deepest balanced subtree with root node  $Ht$ 
3:   find the shallowest most unbalance subtree above  $Ht$  with root node  $St$ 
4:   swap the two subtrees  $Ht$  and  $St$ 
5:   update node information along the two paths and add new nodes into  $L$ 
6: end while
7: return  $L$ 

```

We support two basic key tree operations: *Insert* and *Remove*. The internal node stores the path information to the deepest and shallowest nodes. We use an auxiliary data structure, member hash table, in our implementation. The member hash table allows us to locate any member node in constant time, given the member ID.

To further improve the re-keying performance of the rebalancing algorithm, we support batch removal in which re-keying upon a member leaving is only performed after a time threshold or count threshold.

The *Insert*, *Remove*, *Rebalance* and *Batch Removal* algorithms are shown in Algorithms 1 – 4.

VII. SYSTEM IMPLEMENTATION

In this section we discuss the design and implementation of the components for our location based system. Our LBS system is designed to be network and protocol independent. We take an approach to separate the application service logic and the network functionality as shown in Figure 9. We implement our Location Based Service system on Java 2 Standard Edition (J2SE) because it is widely supported by laptops and desktops; a similar implementation may be done with J2ME.

We first describe the implementation of the network functions, namely the Location Server and the Group Server, which provide common interfaces to the user and the Application Server to develop various LBS. Finally we describe two services, *LocateFriends* and *BuddyAlert*, which are implemented on an enhanced Instant Message Server (LeIM Server); we also include the location enhanced instant message client.

Fig. 9. System components and interfaces.

A. Location Management

An LBS Application Server has to interact with the Location Server for accessing a user's location to deliver required services. Different approaches can be followed for realizing the location service interface. Existing location interface implementations typically work with only one network and are not extensible. Our interface is implemented using SOAP/RPC over HTTPs web service. This is available for most wireless users using an Internet service model. We delegate the interaction with the Location Server to the Group Server in our system which takes advantage of the group key management introduced in Section V. The Application Server simply joins a user's location group to receive the location information distributed from the Group Server. The Location Server provides the Common APIs for the location service interface which basically includes location update and location query.

The Location Server provides location management logic and a location service interface as shown in Figure 10. The positioning functionality on the client is also demonstrated. The basic location service common APIs are further summarized in Table IV. Two different categories of query services are supported: pull and push, with respect to how the location information is accessed. Figure 10 shows a representative example of location update and pull based location information access. Method *publishLocation* is used by the end users to update their location to the Location Server. Methods *subscribeLocation* and *getUserLocation* are used by applications to request location subscription and get one-time location information, respectively. Figure 10 is easily modified to support push based scenarios in which the location information is pushed to the application servers at a specified frequency or in response to a specified event. Methods *subscribeLocation*, *startLocNotification*, and *endLocNotification* are used for this purpose.

We support the GPS location tracking functionality in the client location agent with the *Location* package. A nice feature of our location tracking module is the wrapper class which is used as an abstract interface independent of underlying positioning techniques. A standard NMEA GPS sentence is collected through methods

Fig. 10. Location management framework.

TABLE IV
SERVICE COMMON API

Role	Location Service APIs	Group Service APIs
Requester	<i>subscribeLocation</i> <i>getUserLocation</i>	<i>joinGroup</i> , <i>leaveGroup</i> , <i>bulkJoinGroup</i> <i>pullUnicastInfo</i> , <i>bulkPullUnicastInfo</i>
Target	<i>publishLocation</i> <i>startLocNotification</i> <i>endLocNotification</i>	<i>createGroup</i> , <i>deleteGroup</i> , <i>updateAuthList</i> <i>startMulticastInfo</i> <i>endMulticastInfo</i>

implemented based on the Java Native Interface (JNI), thus allowing platform-specific location drivers to be written in another language and accessed by Java. The Rayming TN200 USB GPS receiver is used as the external GPS device attached to the laptop on which we deploy the client applications.

B. Group Management

Group management is the core part of our LBS system. The cryptographic key distribution method and the multicast service described in Section V are implemented in this component. IP multicast is used to efficiently deliver location information to the users in a location group.

Figure 11 shows the group management software framework in detail. We decompose the features into function managers on the server and relative function agent modules on the user equipment and Application Servers.

The Group Server provides the Common APIs for the group service interface which are summarized in Table IV. In our LBS system the Group Server delegates the location server access for the application server. Similar to the Location Server introduced in Section VII-B, pull based location information access is also supported through unicast. Push based location information dissemination exploits the advantages of IP multicast. We describe the implementation of two basic services in Section VII-C.2. We only highlight the APIs below.

Methods *startInfoMulticast* and *endInfoMulticast* are invoked by users to start and end the distribution of information. Alternatively we support pull based location information unicast with methods *pullUnicastInfo* and *bulkPullUnicastInfo*. Methods *joinGroup*, *leaveGroup*, *bulkJoinGroup*, and *bulkLeaveGroup* are used by

Fig. 11. Group management framework.

end users or Application Servers to join and leave the multicast group(s) delegated in the Group Server.

The *GServer* package implements the group management server functions as shown in Figure 11. Multiple threads are forked by a message dispatcher thread in order to handle multiple requests from the members. Class *MulticastCommManager* implements the global multicast communication manager which controls the multicast sockets and threads for each group indexed by the group name. A set of multicast sockets and threads are used when we consider coding by information, coding by group and flat coding schemes. The nested coding scheme uses only one multicast socket and thread since the information is encoded in a nested way. *SecureMulticast* class implements the multicast interfaces for join (*sendJoinKEKs()*) and leave re-keying (*sendLeaveKEKs()*) and location information distribution (*sendLocationMulticastMessage()*).

The *KeyTree* package implements the LKH extension and our four key tree schemes. The M-AVL key tree algorithm introduced in Section V is included in this package. Class *KeyTreesManager* implements the global key tree manager which provides the *insertKeyTree()*, *deleteKeyTree()*, and *getKeyTree()* interfaces used by the *GServer* package. Interface *KeyTree* defines the key tree APIs including *insert()* and *remove()* and other utility APIs which can be used by the *GServer* package to manipulate the group key trees.

The Java Cryptographic Extension (JCE) provides a framework for encryption/decryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms.

C. Application Server and User Client

1) *Multi-level Location Sharing to Improve Mobile Network Services Experience:* In this section we discuss how multi-level location sharing can enable more sophisticated social experience in Location Based Services.

When games are designed for mobile devices, the player's location (even if relative) and movement can be incorporated into the game. This might include tracking a phone as it moves in addition to monitoring players' direction, velocity and acceleration. When multiple players are cooperating the location information

Fig. 12. Location privacy control and location sharing.

Fig. 13. *LocateFriends* client location display snapshot.

can be shared among them together with other private information. Botfighters2 and CitiTag make use of single level of players' location (or the most precise location) and collaboration among members when they are in proximity [38]. It might be interesting to share different levels of location information with different sub-groups inside a guild. Guilds are often cited as an important factor contributing to MMORPG's popularity. For example, merging neighbor groups by sharing coarse level location information can improve the group power and social experience.

2) *Applications Implementation*: To exercise our system, we implement two services, *LocateFriends* and *BuddyAlert*, on the application server as enhancements to instant message services. These may be viewed as stand-alone services, or building blocks for the games previously discussed.

The two services belong to push-based LBSs. The friends' location information are multicast to users periodically after they join the group. The core logic provides continuous sharing of location information at different granularity. We implement the services using the XMPP (Message and Presence Protocol) protocol. We use the Wildfire IM Server and Spark client development package [39] [40].

LocateFriends is an example in which the user application interacts with the group server directly for the basic location sharing. The application directly uses the group service framework APIs as described in Section VII-B. The UE component in Figure 11 shows the detail functionalities on the client.

The roster information provided by the Spark IM Client development package is used to edit the authorization list and subscription list information for a client, i.e., its location information groups. The configuration information is composed into a *createGroup* message and sent to the group server when the user clicks the Invite button. The user is a member of their own location groups by default and the initial keys are returned with the response. The end user will encrypt the location information using the group keys. Clients can configure the subscription list and include it in the *joinGroup* message sent to the group server. The client will get a Join Reply message which contains the initial key information along the key tree insertion path.

Fig. 14. Location sharing in IM Services.

A simplified flow for privacy control and location sharing is shown in Figure 12. The user may receive re-keying or location multicast messages from the group server after it joins the group. Figure 13 shows the location information displayed on different *LocateFriends* clients at different granularity.

The *BuddyAlert* service is also a push-based continuous tracking service. The Application Server component shown in Figure 11 demonstrates the software framework to build the services. The application server joins the multicast group of the buddies and receives the multicast message pushed from the group server periodically. Then the application checks the proximity of the user and its buddies. The users get an alert message from the application server when the predefined buddies are within the proximity of each other. This is an application example which uses the group service framework and adds custom service logic on the application server.

The *IMServerPushExt* packages implement the *BuddyAlert* service on the IM Server based on the models explained above. Push based group service framework APIs are used to implement the basic logic similar to the *LocateFriends*. Simple service logic is designed to alert buddies when they are within a requested proximity, for example, in the same city.

The *plugin.locationgroups* package implements the client plugin to support the location extension based on the Spark IM Client. The client uses the same interface to configure its location sharing profile which includes its buddies and the application server.

3) *Re-keying Enforcement For Location Sharing Policy in IM Services:* Our system is scalable in the face of high group dynamics. In this section we describe a practical re-keying scheme for Location Sharing in IM Services.

In social networks such as Facebook, people are usually involved in several community groups. In general, tightly coupled groups are small (< 100) in which people share precise location and require timely location updates. The sharing relationship is stable so that key management overhead is not prominent. Using key trees allows flexible location privacy control as discussed in previous sections, but the rebalance operation can be disabled. Loosely coupled groups are normally large, and people share coarse location information and require less frequent location updates.

The end user still has control over their location information access. For each group the end user can define a batch period for re-keying, or mandate join/leave re-keying. For each period the user can define service activation/deactivation time for each member or sub-group. The end user can specify different policies when members become offline to avoid unwanted logging. The end user can also define how often this level of location information should be updated/pushed to the group.

For example, Olly is in several social communities and has co-workers, friends, family members and club people on his buddy list. He defines his location information sharing groups. Figure 14 shows the social network relations and the mapping of location information groups. The members under the same community are inserted into the same subtree; this can help to simplify the group management and the re-keying.

We introduce an activation/deactivation time for each member or community group. When it is time to deactivate a member, re-keying occurs but the node is not removed from the tree. Instead, the node is marked and an activation timer is set for the node. At the same time it is inserted into the activation pending list.

A similar scheme is applied when members logout. The user will receive new keys encrypted with the old key when they become active again. If there is no re-keying during this interval the node is removed from the pending list.

For example, Olly configures his location profile for his managers which will be activated/deactivated on 9:00AM/6:00PM during weekdays. When his manager becomes offline, re-keying will occur to avoid offline location tracking. When Olly comes to work at 9:00AM, the members on the pending list will be re-keyed. The latest keys on the path to the root will be encoded with the old key which could be cached on the member nodes. The managers now can see Olly's location without any explicit rejoining. The rebalance feature is disabled since the social network of a user is stable.

TABLE V
SIMULATION PARAMETERS

Mean inter-arrival time for user joins, exponential distribution	20 s
Mean duration of group membership, exp. distribution	2 hours
Observation intervals	15 minutes intervals for over 17 hours
Group size	800 users
Ratio of subscribers with access to different levels of location information(5 classes)	1:1:1:1:1

VIII. PERFORMANCE EVALUATION

In the following we present the results of the load tests performed on our system for large group applications. The four different key distribution schemes are considered. We evaluate the system in two parts: key management overhead and memory cost for group management; location update overhead for service performance.

A. Simulation Model and Test Bed

We assume a situation in which the system is deployed in a company with a large number of employees. We focus on the performance of the system considering the typical dynamics of group applications. Thus we evaluate the performance over a period of time in which we model the behavior of users based on previous studies of group dynamics in the Mbone [41] [42]. This does not apply to the flexible privacy enforcement model we discussed in Section VII-C.3 which incurs less overhead. We demonstrate a case in which one person shares their location information with a large group of 800 people at different granularity. The user could be a person serving for a department such as a secretary or a lab support. It can also be used to deploy event based location sharing in large social networks and community services.

The *LocateFriends* client is used as the simulation application for the load test. To allow fine-grained location privacy control and demonstrate the group dynamics, we consider user login/logout as group joining/leaving events. This demonstrates system scalability in the worst case.

We assume people join the group during a concentrated period of time (e.g., when they arrive at work), and use the message service for about 2 hours each day. Therefore, the average inter-arrival time for users joining the group is relatively low (20 seconds); once a user has joined they remain in the group for an average of 2 hours. Once a user leaves the group, they do not re-join. The user inter-arrival time and membership duration are exponentially distribution in a relatively concentrated period of time [41]. We assume that the location information class that the users desire is uniformly distributed.

Our measurements were made in 15 minute intervals for 17 hours, the first 10 of which are presented here. We limit our presentation to the first 10 hours because this is when the network dynamics are highest. The

Fig. 15. Global distribution

results we shown here are not meant to be taken as absolute, but to show the trends and comparative results. We summarize these parameters in Table V.

To evaluate the performance of the system we set up a test bed consisting of three machines running windows XP. The Group Server is hosted on a machine with a Pentium M 2.00 GHz processor and 1.50 GB RAM. The Location Server and Location enhanced Instant Message Server are installed on a second machine with a Pentium M 1.50 GHz processor and 1.0 GB RAM. A third machine with a Pentium M 1.30 GHz processor and 1.0 GB RAM is used to simulate the user behavior of a large group.

B. Key Management Overhead

In the following subsections we first evaluate the rebalancing algorithm and batch removal scheme. Then we show the re-keying overhead in terms of the average number of key changes and rebalance operations. Furthermore, the bandwidth cost and the average server processing time of re-keying are measured and analyzed in order to identify the benefits and performance impact of the different schemes.

Figure 15 shows the size of the groups and number of key tree nodes (including internal and external nodes) over the simulation period. Figure 16 shows the number of users leaving in each 15 minute interval. We do not show the number of users joining over time due to space limitations. Please refer to Figure 16 when interpreting the remainder of the results.

1) Key Tree Rebalance Performance: We first evaluate the performance of the rebalancing algorithm and the batch removal optimization with the coding by group key distribution scheme. As seen in Figure 17 and 18, the tree height with rebalancing is close to the ideal value, which is an improvement of about 15% over when rebalancing is not used. This helps achieve better performance with fewer key changes when a member joins or leaves a group. Furthermore, if batch removal is applied with a threshold of five minutes, multicast traffic is reduced by over 50%. The tree height with batch removal is still similar to the ideal case as shown in Figure 17.

Fig. 16. Member leave rate.

Fig. 17. Rebalance and batch removal performance: tree height compared to the ideal tree height

Fig. 18. Rebalance and batch removal performance: re-key bandwidth overhead.

Fig. 19. Re-key key changes benchmark.

2) *Re-keying Key Changes Benchmarks*: The re-keying benchmarks are conducted to characterize the load for different key distribution schemes. Figure 19 shows the average number of re-keys the server needs to perform due to members leaving a group according to the intensities of Figure 16. The results for users joining the group are similar.

The three schemes (group, nested, and flat) that accommodate hierarchical coding of location information all have fewer key changes than the direct application of LKH (coding by information class). Due to the M_AVL algorithm, the key trees are general balanced and the key changes are lower than the height of the tree. When a member leaves the group, the rebalancing function may incur additional key changes because of the swap operations. We found that the average number of swap operations per leave was 0.46 with a variance (σ^2) of 0.35. This did not degrade the performance of the rebalancing algorithm.

The most efficient scheme in this respect is coding by group, in which each user group receives all data under a single key. This is intuitively the most efficient for as it requires re-keying only for users directly affected by the group membership change.

3) *Re-keying Bandwidth Overhead*: Figure 20 depicts the bandwidth overhead of the four schemes incurred for users leaving a group with the leave rate shown in Figure 16. Similar results for users joining a group are not shown due to space limitations. Except for the flat coding scheme, the bandwidth cost conforms to the key

Fig. 20. Re-key bandwidth overhead with respect to member leave rate.

Fig. 21. Re-key processing time on the server.

change curves of the different schemes. The re-keying bandwidth cost of the flat scheme is relatively higher than the nested scheme, whereas in Figure 19 the re-key key changes of the two schemes are almost the same. The main reason is that for the flat scheme, one group key needs to be multicast multiple times to subgroups. The most efficient schemes in this respect are coding by group and the nested coding.

4) *Server Re-keying Processing Time*: The server processing time measured here is exclusively the server side cost for the group management. In the group server implementation, multiple threads are forked in order to serve multiple requests. Since the system resources are restricted, the average processing time tends to increase when more users join or leave the system. We exclude the SSL handshake cost and the transmission time in our measurements because these are independent of our schemes.

Figure 21 shows the average re-keying processing time on the server for the different schemes. The figure depicts the average server processing overhead for each leave operation over time for the leave rates shown in figure 16. These results show that the processing overhead on the server is very low.

C. Group Management Memory Overhead

Figure 22 shows the average memory cost on the server for the different schemes over time according to the leave rates of Figure 16.

The client must save the Key Encryption Keys and the group keys to decode the re-keying and location messages. The number of initial keys after joining the group owned by a client is in general comparable to the height of the tree for the three schemes (group, nested, and flat). The coding by information class scheme is the worst in this respect.

D. Location Update Overhead

In this section we consider the scalability of the system in the face of user mobility. Recall, we have a single user generating location information that is sent to a group of up to 800 people. Each time the user moves

Fig. 22. Server key tree memory overhead.

Fig. 23. Location update overhead.

sufficiently to change the location information of one or more location classes, the information in the database must be updated. For tracking applications, this information must also be disseminated to users.

We choose two simple models, high mobility and low mobility, to represent the single user's location update traffic. For example, a high mobility user changes their highest level of granularity (level 5) location information every 0.5 hours on average. The user's level 4 and level 3 location information are changed every 2 hours and 4 hours, respectively. A low mobility user only changes their level 5 and level 4 location information every 2 hours and 4 hours on average, respectively. The location update interval conforms to an exponential distribution.

The updated information is multicast to the group members as soon as it changes due to the user mobility. These results are therefore applicable to push based services. Figure 23 shows the results. The flat scheme does not scale to high mobility because the location information with lower granularity needs to multicast to multiple subgroups. The other three schemes are comparable and scale to high mobility.

From the server's perspective, only one multicast message that includes all levels of the updated information needs to be sent for the nested scheme. This means fewer multicast groups and multicast sockets are managed on the server. For the other three schemes, multiple multicast messages are sent and each of the messages includes only one level of location information.

From the clients' perspective, the group scheme is desirable since they only need to handle one multicast message and decode once to get all the information. With nested coding the client needs to handle one multicast message but still has to decode multiple times to get all classes of location information. For the flat scheme, the client handles one multicast message and also has to decode multiple times to get all the information. For the information class schemes, the client handles multiple multicast messages and also has to decode multiple times to get all the information.

Nested coding incurs overhead for the user when they desire only a single class of location information because all classes up to the desired class must be decoded. To make nested coding more efficient, we can make the following change: location information of different classes are encoded separately and organized in

a flat way and sent in one multicast message.

E. Discussion

In this section we discuss the trade-offs of the various schemes described above.

The coding by class scheme does not scale well from the key management perspective if multiple information levels are extensively used. The scheme is only effective if a single group is dominant in which case the LKH scheme directly applies.

The flat coding scheme is the most flexible from the user or the application server perspective. In this scheme users or application servers only join the groups directly corresponding the classes of information they desire. The scheme does not scale to high mobility but may be used for pull based applications where the impact of the mobility is not critical.

The coding by group and the nested coding solutions are attractive for the following reasons. First, they both result in low re-keying overhead. Second, each user receives the data at and below their maximum class level. This potentially simplifies service logic because the recipients of the high granularity data do not have to process it to derive the lower granularity information. For example, when a high priority user receives the (x,y) coordinates of a peer, it will also receive the state location of the peer and not have to derive it itself. Finally, the key storage requirements on each user device are low.

The processing of messages is efficient for users in the coding by group scheme, because they only receive the data they require and decode once. This is desirable for devices with limited system resources and computation power. On the contrary, for nested coding the processing overhead on high priority users is high because they must decrypt data c times to retrieve all classes of data.

Nested coding is attractive in that only a single multicast group, thus a single multicast address and socket, is maintained for data dissemination and re-keying on the server. This simplifies the implementation logic and reduces the system resources required on the server.

IX. OPEN SERVICE ARCHITECTURE ENHANCEMENT

Next generation networks are expected to support services across heterogeneous networks regardless of the terminal used. The lack of a global standard for location information transport is a hindrance for service providers. The intent of the Open Service Access/Parlay (OSA/Parlay) [43] is to provide an object-oriented architectural model which can be viewed as an abstract network layer to expose network services in a secured and controlled manner. This architecture is targeted to bridge traditional telecommunication networks and IP based networks to provide a mobile virtual network from different operators.

ParlayX provides web service interfaces [43] which can be used by loosely coupled application servers to stimulate service development outside of traditional telecommunication players. These interfaces include 3rd party call delivery, conferencing, messaging, and user presence and location information.

Each OSA/Parlay API provides an interface to the network and the application. The former, implemented by the network operator, interacts with core network elements, while the latter is deployed by applications and services in order to interact with the network. The communication between the network and application follows an asynchronous client/server model using standard middleware infrastructure. When applications perform remote method invocations, call back objects are generated in the application side in order to receive the network results. Two possible middleware options for implementation are the Common Object Request Broker Architecture (CORBA) or the SOAP over HTTP Web Service technology. OSA/Parlay has specified the interfaces using both the Interface Definition Language (IDL) for CORBA and the Web Services Description Language (WSDL). The architectural model is also enhanced with security capabilities, offered by the OSA/Parlay framework API to support authentication and authorization mechanisms. The SOAP approach has higher response time and uses plain text transmission while CORBA outperforms the former. On the other hand, SOAP is easy to develop and deploy. Both middleware gateways consist of international standards, which guarantee interoperability between different platforms or implementations.

Our system framework is designed to be extensible and provides common APIs as described in Section VII-B and VII-A. The APIs can be mapped to ParlayX methods [43] and our location server can function as a proxy in between the application server and the core network.

X. CONCLUSION

In this paper, we present a framework to support privacy enhanced location based services and a practical LBS system implementation. The key idea behind the system is to hierarchically encrypt location information under different keys, and distribute the appropriate keys only to group members with permission to see the location information at that granularity. The common APIs of the framework allow applications to access the location information at different levels of granularity regardless of the network and protocols.

We propose re-keying through the application of the LKH protocol. We modify MIKEY to accommodate LKH and the four different hierarchical location information dissemination methods. We implement the proposed extensions in our LBS system and conduct the load test to examine the system scalability and efficiency. We then discuss the extension of the system architecture to an open service architecture. This architecture is ideal for a service model in which a single operator does not have direct control over all service elements.

REFERENCES

- [1] J. Arkko and E. Carrara, "Mikey: Multimedia internet keying," RFC 3830, IETF, 2004.
- [2] S. Rafaeeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 309–329, 2003.
- [3] D. Wallner, E. Harder, and R. Agee, "Key management for multicast: Issues and architecture," Internet Draft, IETF, 1998.
- [4] "Pols:privacy observant location system," <http://pols.sourceforge.net/>.
- [5] "Functional stage 2 description of location services (lcs)," 3GPP, TS 23.271 Version 6.8.0 (2006-06), 2006.
- [6] IETF, "Geopriv working group," <http://www.ietf.org/html.charters/geopriv-charter.html>.
- [7] U. Hengartner and P. Steenkiste, "Exploiting information relationships for access control," in *Third IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*. IEEE, March 2005, pp. 269–278.
- [8] D. Quercia, S. Hailes, and L. Capra, "B-trust: Bayesian trust framework for pervasive computing," in *iTrust. LNCS*. Springer, May 2006, pp. 269–278.
- [9] —, "B-trust: Bayesian trust framework for pervasive computing," in *iTrust. LNCS*. Springer, May 2006, pp. 269–278.
- [10] M. Langheinrich, "A privacy awareness system for ubiquitous computing environments," in *the 4th international conference on Ubiquitous Computing*. Springer-Verlag, 2002, pp. 237–245.
- [11] G. Myles, A. Friday, and N. Davies, "Preserving privacy in environments with location-based applications," *IEEE Pervasive Computing*, vol. 2, no. 1, pp. 56–64, 2003.
- [12] M. Gruteser and X. Liu, "Protecting privacy in continuous location-tracking applications," in *IEEE Security and Privacy*, vol. 2, no. 2, March 2004, pp. 28–34.
- [13] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. of 1st Intl. Conf. Mobile Systems, Applications, and Services (MobiSys)*. ACM, May 2003, pp. 31–42.
- [14] A. R. Beresford and F. Stajano, "Location privacy in pervasive computing," *IEEE Pervasive Computing*, vol. 2, no. 1, pp. 46–55, 2003.
- [15] Q. He, D. Wu, and P. Khosla, "Quest for personal control over mobile location privacy," *IEEE Communications Magazine*, vol. 42, no. 5, pp. 130–136, 2004.
- [16] B. Schilit, J. Hong, and M. Gruteser, "Wireless location privacy protection," *IEEE Computer*, vol. 36, no. 12, pp. 135–137, 2003.
- [17] M. Prabaker, J. Rao, I. Fette, P. Kelley, L. Cranor, J. Hong, and N. Sadeh, "Understanding and capturing peoples privacy policies in a people finder application," in *the UbiComp Workshop on Privacy*. ACM, 2007.
- [18] L. P. Cox, A. Dalton, and V. Marupadi, "Smokescreen: Flexible privacy controls for presence-sharing," in *the 5th Intl. Conf. Mobile Systems, Applications, and Services (MobiSys)*. ACM, 2007.
- [19] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: One-way function trees and amortized initialization," Internet Draft, IETF, 2000.
- [20] A. Perrig, D. Song, and J. D. Tygar, "Elk, a new protocol for efficient large-group key distribution," in *IEEE Symposium on Security and Privacy*, May 2001, pp. 247–262.
- [21] O. Rodeh, K. Birman, and D. Dolev, "Using AVL Trees for Fault Tolerant Group Key Management," Cornell University, Tech. Rep. 2000-1823, 2000.
- [22] M. Moyer, J. Rao, and P. Rohatgi, "Maintaining balanced key trees for secure multicasts," Internet Draft, IETF, 1999.
- [23] G. Hao, N. V. Vinodchandran, and B. Ramamurthy, "A balanced key tree approach for dynamic secure group communication," in *the 14th Intl. Conf. on Computer Communications and Networks*. IEEE, 2005, pp. 345–350.
- [24] S. Zhu, S. Setia, and S. Jajodia, "Performance optimizations for group key management schemes," in *the 23rd Intl. Conf. on Distributed Computing Systems*, May 2003, pp. 345–350.
- [25] "Myspace," <http://en.wikipedia.org/wiki/Myspace>.
- [26] "Friendster," <http://en.wikipedia.org/wiki/Friendster>.
- [27] "Facebook.statistics.retrieved on 2007-12-20," <http://www.facebook.com/press/info.php?statistics>.
- [28] R. Gross and A. Acquisti, "Information revelation and privacy in online social network (the facebook case)," in *the Workshop on Privacy in the Electronic Society*. ACM, 2005, pp. 71–80.
- [29] "Mmorpg," <http://en.wikipedia.org/wiki/MMORPG>.
- [30] "Botfighters2," <http://www.Botfighters2.com>.
- [31] "Cititag," <http://cnm.open.ac.uk/projects/cititag/>.
- [32] "World of warcraft surpasses 9 million subscribers worldwide," blizzard Entertainment press release, July, 2007.
- [33] N. Ducheneaut, N. Yee, E. Nickell, and R. J. Moore, "Alone together? exploring the social dynamics of massively multiplayer online games," in *the International Conference for Human-computer Interaction (CHI)*. ACM, 2006, pp. 407–416.
- [34] M. Baugher, B. Weis, and etc., "The group domain of interpretation," RFC 3547, IETF, 2003.
- [35] H. Harney, U. Meth, and etc., "Gsakmp: Group secure association key management protocol," RFC 4535, IETF, 2006.
- [36] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001.
- [37] A. Andersson, "General balanced trees," *Journal of Algorithms*, vol. 30, no. 1, pp. 1–28, 1999.
- [38] Y. Vogiazou and B. Raijmakers, "Urban space as a large-scale group playground," in *the UbiComp Workshop on UbiComp in the Urban Frontier*. ACM, 2006, pp. 45–58.
- [39] "Jive software:wildfire instant message server," v3.1.1, <http://www.jivesoftware.org/wildfire/>.
- [40] "Jive software:spark instant message client," v2.0.0.2, <http://www.jivesoftware.org/spark/>.
- [41] K. Almeroth and M. Ammar, "Multicast group behavior in the internet's multicast backbone (mbone)," *IEEE Communications*, vol. 35, no. 6, pp. 124–129, 1997.
- [42] S. Alouf, E. Altman, C. Barakat, and P. Nain, "Optimal estimation of multicast membership," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2165–2176, 2003.
- [43] "Open service access (osa); parlay x web services," 3GPP, TS 29.199 Version 7.0.0 (2006-06), 2006.