

Mobile Social Services via Virtual Individual Servers

Amre Shakimov
Duke University
Durham, NC
shan@cs.duke.edu

Harold Lim
Duke University
Durham, NC
harold@cs.duke.edu

Landon P. Cox
Duke University
Durham, NC
lpcox@cs.duke.edu

Ramón Cáceres
AT&T Labs
Florham Park, NJ
ramon@research.att.com

ABSTRACT

Mobile social services enable increasingly popular forms of social interaction among mobile-device users, such as finding out when people of interest are nearby. However, today’s services raise important privacy concerns because they concentrate location information for many users under a single administrative domain. We present a privacy-preserving framework for these services in which each person maintains her own location history in her own Virtual Individual Server (VIS), a personal virtual machine running in a utility computing infrastructure. We enable location sharing among VISs through self-organizing overlay networks, one per social group with which VIS owners wish to share information. We describe an implementation of this framework that exploits skip graphs and Z-order space-filling curves to provide efficient and scalable operations on distributed location data. Finally, we demonstrate the feasibility of our approach by evaluating our implementation using real-world location traces gathered for this purpose. Our decentralized approach makes large-scale privacy breaches much less likely than in centralized architectures, and gives people fine control over what location information they share with whom.

1. INTRODUCTION

Programmable consumer devices such as mobile phones have placed computation within arm’s reach at all times and in all places. *Mobile social services* take advantage of the nearly constant physical proximity of devices to their owners to enable a wide range of new social interactions among mobile users, for example finding out when friends are nearby. Unfortunately, most of today’s mobile social services raise important concerns about user privacy and autonomy. In particular, most services concentrate personal data for many users under a single administrative domain. Users send periodic location updates to a trusted service provider that, in turn, coordinates interactions among users based on these locations. This centralization simplifies service architectures, but is inherently vulnerable to large-scale privacy breaches from intentional and unintentional data disclosures.

Consider Loopt [17], a commercial service that allows friends to track each others’ current location. Although the service provides control over which users can track one’s location, end-users’ privacy is not protected from the central server itself. Our prior work on Micro-Blog [9] suffers from similar drawbacks. In exchange for access to “virtual sticky notes” and the ability to issue geographically-scoped queries to live users, participants allow a central coordinator to track their location in real time. For both services, end users are required to treat the service provider as an omniscient guardian of their location data.

Unfortunately, recent lessons from online social networks demonstrate that such centralized services are prone to privacy violations. Sensitive user data can be inadvertently leaked [25], and can be abused by internal administrators, as some have accused Facebook employees of doing [20]. Even worse, social services can leak inappropriate information directly to users’ close relations. For instance, Facebook’s Beacon program caused an uproar when it began actively reporting users’ online activity such as purchases and visited web pages directly to their friends [18]. As social services become more entwined with users’ lives and gain access to their detailed location histories, the threat of privacy violations will grow.

In this paper we present a decentralized architecture for mobile social services that avoids these privacy vulnerabilities. Under our scheme, each person maintains her location history in a *Virtual Individual Server (VIS)*, a personal virtual-machine instance hosted by a utility computing infrastructure such as Amazon Elastic Compute Cloud (EC2) [1]. VISs allow individual control of sensitive location information along with the associated software and access-control policies. Our approach distributes responsibility for personal information among many administrative domains, one domain per user, and thus makes a large-scale privacy breach much less likely than in centralized architectures.

We believe that individuals will adopt virtualized utility computing for many of the same reasons that enterprises are adopting it: VISs unburden users from having to maintain their own high-availability systems without forcing them to give up control of their data, software, and policies. In contrast to free mobile social services, paid utility-computing services such as EC2 let users retain ownership of the content that users place on these services [2].

We enable location sharing among VISs through self-organizing, distributed data structures called *skip graphs* [3]. Skip graphs are similar to distributed hash tables (DHTs) and enjoy the same scal-

ability advantages, but are more appropriate for managing location information because they support efficient range queries. We further use *Z-order space-filling curves* [22] to map two-dimensional geographic coordinates to the one-dimensional name spaces supported by skip graphs. Location range queries are critical to core features of mobile social services such as friend finding, virtual sticky notes, and live geo-queries.

VISs thus form overlay networks based on skip graphs, one overlay per social group with which VIS owners wish to share location information. A VIS can protect its owner’s privacy in a number of ways. One, it can refuse to divulge her location. Two, it can lie and publicize a fictitious location. Three, it can publish different location information to different social groups. For instance, it can give location at one accuracy to one group and at another accuracy to another group (e.g., 10 square meters vs. 10 square miles). Recent human-factor studies have shown that it is important for people to retain such deniability, flexibility, and control regarding their location information [5][13][16].

More generally, we argue that the interests of individuals would be well served by using Virtual Individual Servers to store and share the growing variety and volume of personal data generated on their mobile devices: photos, microblog entries, location histories, etc. A mobile device and a VIS belonging to the same person can generally place a higher degree of trust in each other than on systems owned by other people and organizations. In addition, VISs enhance the capabilities of mobile devices in important ways since they reside on fixed infrastructure. For example, VISs are highly available, strongly connected, and not subject to battery life-related concerns. Using VISs as trusted and resource-rich proxies for mobile devices thus has advantages over serving personal data from either centralized services or the devices themselves.

This paper makes the following contributions:

- It proposes the use of Virtual Individual Servers as trusted and resource-rich proxies for personal mobile devices.
- It presents the design of a privacy-preserving framework for mobile social services based on overlay networks of VISs.
- It describes an implementation of this framework, including a companion application for mobile phones, that provides efficient and scalable operations on distributed location data.
- It demonstrates the feasibility of our approach by evaluating our implementation using location traces gathered for this purpose from a population of mobile phone users.

In a recent paper [29], we explored the use of VISs to support online social networks that do not have a mobile or location-based component. For example, our earlier system did not support range queries on two-dimensional data. It used traditional DHTs to support only point queries on one-dimensional data. In addition, that work did not involve mobile users, mobile applications, mobile devices, or mobility traces. In contrast, this work focuses on supporting mobile social services through location range queries.

Experimental results using our prototype implementation indicate that our approach to managing location information is a viable and desirable alternative to the prevailing mobile social service architecture. The latency of common operations on mobile social groups grows slowly with the size of the group. Similarly, the memory and bandwidth required by a VIS is manageable and grows with the size and number of groups to which its owner belongs.

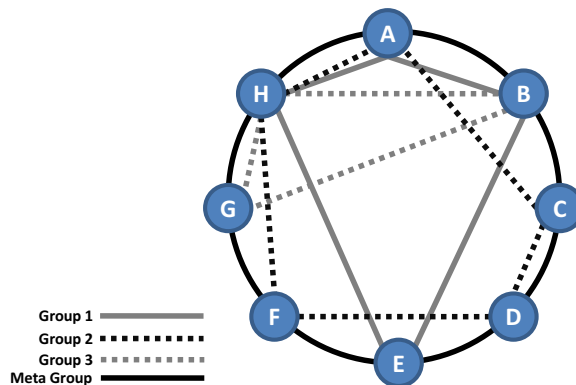


Figure 1: Example Vis-à-Vis network with eight Virtual Individual Servers, three groups, and the Meta Group

2. BACKGROUND

The work presented in this paper builds on three enabling technologies: our Vis-à-Vis architecture for online social networks, skip-graph distributed data structures, and Z-order space-filling curves. This section provides an overview of these technologies.

2.1 Vis-à-Vis

We recently introduced Vis-à-Vis as a decentralized architecture for online social networks (OSNs) based on the privacy-preserving notion of a Virtual Individual Server [29]. As mentioned earlier, a VIS is a personal virtual machine running within a cloud-computing utility. In Vis-à-Vis, each user manages personal information such as friend lists, photographs, and messages through his own VIS. In addition, VISs self-organize into groups corresponding to social groups with whom VIS owners wish to share information.

More specifically, VISs form overlay networks, one overlay per group. Figure 1 shows an example of a Vis-à-Vis network with eight VISs and three groups. A single VIS can join multiple Vis-à-Vis groups, just as a person can belong to multiple social groups. The figure also shows how all VISs join a well-known Meta Group through which other groups can be advertised.

Vis-à-Vis realizes this structure via two tiers of distributed hash tables. The top tier contains one DHT that embodies the Meta Group, and the bottom tier contains one DHT for each group. The top-level DHT maps keys consisting of a group descriptor to values consisting of a short list of representative VISs for that group. It is openly accessible to facilitate people finding groups of interest. Per-group DHTs have key-value semantics and access control policies that can be arbitrarily defined by the group members.

Vis-à-Vis supports a wide and extensible variety of groups. For example, it supports groups that anyone can join as well as groups with restricted membership. It also supports groups whose existence is openly advertised as well as groups whose existence is known only to their members. The use of DHTs enables efficient and scalable operations on groups. Our previous work defined four basic operations: create, join, leave, and query. Groups have a great deal of freedom in how they implement these operations for their particular type of group.

The generality of Vis-à-Vis in turn enables many popular OSN features, such as searching for groups, browsing groups, pro-actively suggesting groups and friends, and third-party plug-in applications.

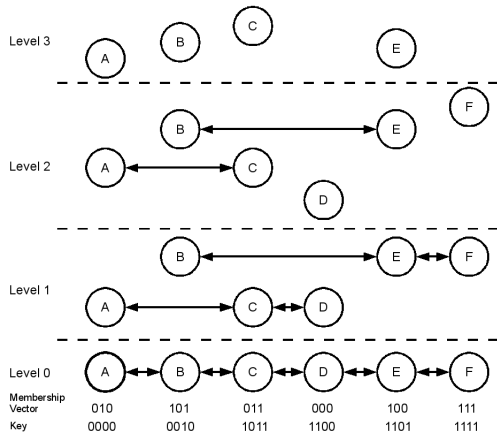


Figure 2: Example of a skip graph

In this work we extend Vis-à-Vis by adding location-based groups to enable mobile social services, as described in Section 3.

At the moment, the main drawback of relying on VISs is their monetary cost. For example, maintaining a modestly provisioned virtual machine running for a month at Amazon EC2 currently costs close to US\$75. We recognize that many people may choose to give up privacy in exchange for free service. However, we feel that it is valuable to explore the benefits and limitations of alternatives such as Vis-à-Vis, especially as public awareness of privacy issues grows and the price of utility computing drops.

2.2 Skip Graphs

The second enabling technology that we use to implement our mobile social services platform is *skip graphs*. Distributed skip graphs are based on skip lists [27] and provide efficient, fault-tolerant lookup and routing in volatile distributed environments. Nodes in a distributed skip graph are assigned two identifiers: a *membership vector* and a *key*. Membership vectors are generated randomly, and determine a node’s skip-list membership. A node’s key can be assigned arbitrarily, and is used to impose a partial ordering on the skip lists.

All nodes are members of the lowest-level list (level₀), and subsequent lists are composed of nodes with identical membership-vector prefixes. As a result, two nodes will be members of the same level_i list ($i \geq 0$) with probability $1/2^i$ for randomly assigned vectors. Lists are ordered based on each node’s key. Figure 2 shows a simple representation of a skip graph with six nodes (A, B, C, D, E, and F) as well as their membership vectors and keys.

Nodes in a skip graph can be located using either their membership vector or their key. To search for a node using its membership vector, a node initially traverses the level₀ list looking for a node that has the same first digit in its membership vector as the first digit in the membership vector of the destination node. Once this node is found, upper-level lists are incrementally traversed such that, at level_i the searching node seeks a node for which the i^{th} digit of its membership vector is equal to the i^{th} digit of the membership vector of the destination node. This procedure terminates when either the destination node is found or, if this node does not exist, the node with the closest membership vector is found. This prefix-based lookup is very similar to the algorithm used within DHTs,

and as with looking up a NodeID within a DHT, searching for a membership vector is probabilistically logarithmic in the size of the graph.

Consider F searching for vector 010 in Figure 2. F first looks in the lowest level, and passes the search to D since it is the first node F finds in level₀ whose vector begins with 0. D then looks for a node in level₁ with a prefix of 01 and finds C. C then uses its direct link to A in level₂ to finish the search.

To search for a key, a node first examines its neighbors in its highest-level skip list. If the destination key is greater than both its own key and the key of its right neighbor, then the search process is passed to the right neighbor. Similarly, if the key is less than both its own key and the key of its left neighbor, then the search process is passed to the left neighbor. If the destination node’s key fails both conditions, then the search continues among the node’s neighbors in the next-lowest skip list. This process continues until a node with the same key as the destination key receives the search process. If the search process reaches level₀, the node that receives the search process does not have the same key as the destination key, and neither its left nor right neighbors satisfy the passing condition, then the search fails. Searching for a key is also probabilistically logarithmic in the size of the graph.

Consider A searching for key 1101 in Figure 2. A first looks to its highest level. Since A does not have any qualifying neighbors at level₃, it looks at level₂, and sends the search process to C because 1101 is greater than both A and C’s key. Since C does not have a neighbor to its right, it drops the search to level₁, and sends the search process to D. When D receives the search process, it passes the search process to E at level₀. When E receives the search process, the search terminates, and A is notified of the result.

A node joins a skip graph by searching for the node with the key that is closest to its own in the graph. This process is similar to a single key query, and the node with the closest key terminates the search. It is possible for multiple nodes to have the same key. Starting at level₀, the joining node inserts itself between the closest node and the appropriate neighbor, ensuring that the partial ordering is maintained. The joining node then traverses the list at level₀ to find the closest node with the same membership vector for level₁. As before, the joining node inserts itself between the closest node and the appropriate neighbor at this level. This process continues until the joining node reaches a level where it does not have any neighbors. The closest node at each level also needs to check whether adding the joining node to the list causes the list’s state to change from a singleton to non-singleton. If this happens, the closest node must increase its level until it does not have any neighbors. The insert operation ends when all of the nodes in the skip graph are singletons at their highest levels.

To leave a skip graph, a node notifies each of its neighbors in all of its lists that it is leaving so that the neighbors can link with each other. If a node’s leaving causes one of its neighbors to become a singleton at that level, the neighbor decreases its own level since nodes can only be singletons in their highest level.

The relatively high cost of maintaining the partial ordering within lists whenever a node joins or leaves is the primary drawback of distributed skip graphs relative to DHTs. Otherwise, the lookup and routing performance of both data structures are comparable.

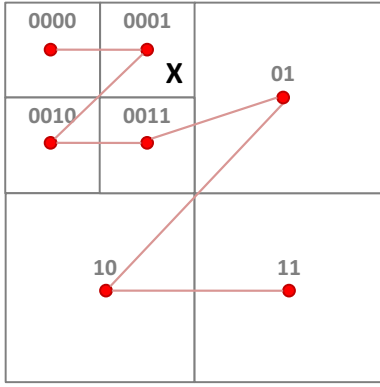


Figure 3: Z-value representation of a 2-D map

2.3 Z-Order Space-Filling Curves

The final enabling technology utilized by our distributed architecture is Z-order space-filling curves. Space-filling curves allow a high-dimensional space to be approximated by a lower-dimensional representation. These curves are useful for our purposes because they allow us to represent a two-dimensional space like latitude-longitude pairs in a one-dimensional space. Z-curves are a particularly appealing class of space-filling curve because they preserve much of the locality of the higher-dimensional space in the lower-dimensional space.

In Z-order space, a rectangular region is divided into 4 blocks (00, 01, 10 and 11), and the blocks are ordered in a Z-shaped curve. Large blocks can be recursively subdivided into four sub-blocks, with each sub-block computing its label by appending 00, 01, 10, or 11 to its parent's. In Figure 3, X has a Z-value of 0001, which can be derived by appending the label of its finer encompassing block (01) to that of its coarser encompassing block (00).

On a Z-curve, subdivision granularity is not fixed, and one block can have very fine-grained subdivisions, while others remain coarsely divided. Z-values are simply ordered based on the depth-first order of their prefixes. That is, finer-grained subdivisions of a block are traversed before moving on to the next block. In Figure 3, blocks are ordered as 0000, 0001, 0010, 0011, 01, 10, and 11.

3. DESIGN

Mobile social services use a continuous stream of location information from participants to coordinate social interactions such as notifying users when people of interest are nearby, delivering location-bound virtual sticky notes, and forwarding location-scoped queries to live mobile users. Unfortunately, all services of which we are aware suffer from the same drawback of concentrating users' sensitive location information under a single administrative domain. Our goal is to design a distributed platform for building mobile social services that provides stronger privacy protections than the dominant centralized architecture.

We realize this goal through a self-organizing, peer-to-peer architecture in which each peer manages a single user's location information and hosts higher-level services by directly accessing the location information managed by other peers. We assume a strong correspondence between administrative domains and individual participants, with no centralized aggregation of location information

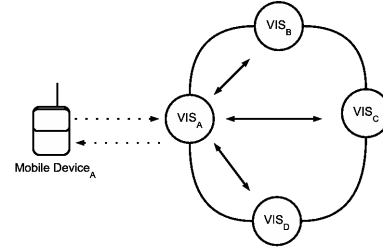


Figure 4: Query path from mobile device through its VIS to other VISs

from multiple domains. In the absence of widespread collusion, partitioning responsibility for location information among many domains reduces the likelihood of a large-scale privacy breach. At the same time, adding these privacy protections could adversely affect the features provided by mobile social services built within our framework. Forcing mobile social service providers to coordinate user interactions by gathering location information from multiple administrative domains adds complexity and can degrade performance.

We resolve this tension through three complementary techniques: 1) administrative domains in our peer-to-peer architecture are contacted via highly-available *VIS proxies*, which manage individuals' location information and query other participants on behalf of their owner, 2) proxies expose *location hints* rather than true locations to social groups to preserve individual control of location information, and 3) VISs are organized into *distributed skip graphs* to enable efficient retrieval of location information across domains.

In the rest of this section, we describe each of these techniques as well as how popular mobile social services' features can be implemented under our framework.

3.1 VIS Proxies

In our architecture, each administrative domain consists of a set of mutually trusting VISs and mobile devices. We imagine that the common case will be an individual participant in control of a single VIS and mobile phone. Secure communication among VISs and devices under the same domain is built on shared cryptographic state that can be distributed out of band in a scalable manner due to the limited number of nodes within a domain.

VISs are treated as *proxies* by their mobile devices, and mediate all access from other domains to an individual's location information. Devices are only responsible for uploading their location to their proxy as resources allow, and all inter-domain interactions occur between VISs. The path of a typical search request is shown in Figure 4. In this example, a request from Domain A originates on a mobile device, is forwarded to its VIS, and leads to direct queries of the VISs for Domains B, C, and D.

Placing the burden of storing and serving location information on VISs rather than mobile devices has two advantages. First, it saves devices' energy, storage, and compute resources. Second, serving location information from VISs avoids the complexities of building a highly-available service from intermittently-connected mobile devices and desktop PCs.

We can apply key-establishment techniques such as LoKey [23] to

provide mutual authentication between domains. LoKey distributes shared keys between a pair of users by taking advantage of two features that are inherent to mobile social services: 1) access to the closed Short Message Service (SMS) network via users’ mobile phones, and 2) a pre-established social connection such as one user’s knowledge of the other’s mobile phone number. Keys can be initially established between two user’s mobile phones over SMS, and then relayed to their proxy VISs.

This approach to establishing secure communication is attractive for two reasons. First, it avoids the Sybil attack [6] by binding identities to mobile phone numbers. Mobile phone numbers are expensive to acquire, which severely limits how much of the identity space an attacker can control. Second, reducing the problem of establishing cryptographic state out of band to distributing mobile phone numbers takes advantage of existing practices among friends and colleagues. Within a social network, users already commonly share their phone numbers, and even if a user’s number is not present in a phone’s address book, it may be accessible via an online social networking website like Facebook or LinkedIn.

Users in our prior work on Vis-à-Vis also use pair-wise authentication between peers to regulate the release of *restricted information* such as their friend lists, profile views, and messaging state. We would like to treat users’ location information similarly, with only explicit social relations being allowed to access users’ location. However, we would also like to support location-based interactions between non-friends such as browsing geo-tagged microblogs, sending queries to live mobile users, and notifying users when strangers with similar interests are nearby. This will require making some location information *searchable* as well restricted.

3.2 Location Hints

In Vis-à-Vis, searchable information is managed through the *typed group* abstraction. Typed groups represent users with a shared interest or attribute, and are named by a descriptor. Descriptors can be expressed as a $\langle \text{type}, \text{key} \rangle$ pair such as $\langle \text{FULLNAME}, \text{JaneDoe} \rangle$, $\langle \text{EDU}, \text{DukeUniversity} \rangle$, or $\langle \text{MUSIC}, \text{TheGratefulDead} \rangle$. Services running within a VIS can create, join, leave, and query groups.

One way to make users’ location searchable using the Vis-à-Vis framework is to multicast a location query to a group via the query operation. Unfortunately, this approach is inappropriate for location-based groups because it would deliver queries to all group members, generating a prohibitively large number of unnecessary requests for large groups. Instead, our framework extends the interface for typed groups by introducing two new operations:

- list `getlocation(descriptor, begin, end)`
- bool `setlocation(descriptor, location, uncertainty)`

For both operations, descriptor refers to the group’s identifier. For `getlocation`, begin and end define a rectangular geographic region, with begin referring to the northwest corner and end to the southeast corner of the region, respectively. For `setlocation`, location is a geographic coordinate, and uncertainty a distance. The `setlocation` operation allows a user to associate themselves with a geographic region that is within uncertainty meters of the specified location. The `getlocation` operation returns a list of group members whose associated location is a subregion of the query parameters. As we will describe in Section 3.3, we can use the regions passed into `setlocation` to efficiently identify group members within an area of interest.

We must be careful that `setlocation` does not undermine our privacy guarantees. If the semantics of `getlocation` are such that it returns all trusted and untrusted group members who are currently within a particular region, then location information would no longer be accessible only through authenticated VIS queries. On the other hand, if `getlocation` returns a list of only trusted group members within a region, then it will be impossible to coordinate interactions among strangers.

We resolve this tension by treating the regions defined by `setlocation` as *location hints*. Hints are a well-known technique in systems [14]; we use location hints only as an optimization, and provide no guarantee that they reflect the truth. Thus, the semantics of the `getlocation` operation are that it returns a list of group members who are *likely* to be within the specified region. These semantics decompose location queries into two phases: an initial unauthenticated phase in which hints are collected via a `getlocation` call, and a second authenticated phase in which individual VISs are queried directly.

This split strikes a balance between features and privacy. Services can use hints to coordinate best-effort interactions between strangers, and limit the number of VISs that must be queried when true locations are required. Authenticated follow-up queries to individual VISs preserve users’ control over their true location information.

A full discussion of policies for managing location privacy among social groups and relations is beyond the scope of this paper, but the `setlocation` and `getlocation` operations are general enough to support a wide range of policy options. For example, users can post different hints to different groups at different granularities, depending on how much they want their hint to reveal about their location to other group members. If a user is interested in meeting fans of a particular band, she might periodically post a location hint to the band’s group that closely corresponds to her actual location. On the other hand, if she is uninterested in being located by alumni of her alma mater, she could post a hint to her college’s group corresponding to its campus. Groups can also define a default location for its members. A natural default location for members of a group composed of Seattle residents like the “Seattle, WA” group on Facebook would be the city limits.

3.3 Distributed Skip Graphs

In Vis-à-Vis, typed groups are implemented as self-organizing, decentralized distributed hash tables (DHTs). DHTs are scalable, fault-tolerant, and balance their load equally among all members. Unfortunately, while DHTs support $O(\log n)$ key lookup and routing, they offer little support for range queries such as those needed to implement `setlocation` and `getlocation`. Thus, we implement our location-based groups by organizing VISs into *distributed skip graphs* rather than DHTs. Distributed skip graphs are similar to DHTs in many ways, but provide much better range-query performance.

As we noted in Section 2.2, nodes in a distributed skip graph are assigned two identifiers: a *membership vector* and a *key*. Membership vectors are generated randomly, and determine a node’s skip-list membership. Searching for a node by its membership vector is nearly identical to performing a lookup in a DHT. As a result, we can easily implement the create, join, leave, and query typed-group operations from Vis-à-Vis by swapping in skip graphs for DHTs, and treating a VIS’s membership vector like a DHT NodeID identifier.

While the membership vectors determine the skip lists in which a node is placed, a node’s key identifier is used to sort the skip lists. Keys induce a partial ordering on all nodes such that for any skip list, the key of a node’s left neighbor is less than or equal to its own, and the key of a node’s right neighbor is greater than or equal to its own.

Because keys can change over time and multiple nodes can safely occupy the same key, we would like to use a user’s location hint as its VIS’s skip-graph key. Unfortunately, the skip-graph key space is one-dimensional, and geographic locations such as GPS coordinates are two-dimensional. To transform two-dimensional locations into one-dimensional values we utilize the Z-order space-filling curves described in Section 2.3. Z-values are ideal for encoding location hints within a skip graph since they are one-dimensional, can be partially ordered, and naturally accommodate different location granularities. We now describe how to implement setlocation and getlocation using skip graphs with Z-value keys.

Consider a user who periodically computes her location via GPS, WiFi, or GSM-cell localization on her mobile phone, and forwards her VIS a setlocation request for each group hint she wants to update. Note that there may be good reasons to factor functionality between a device and VIS differently, but without loss of generality, we will assume that VISs receive periodic setlocation calls from their mobile devices for the discussion below.

Upon receiving the location updates, the VIS logs the location parameter, and decides whether to update each group’s location hint. These decisions are based the user’s previous location hints and each setlocation request’s location and uncertainty parameters.

Z-order curves do not have a fixed block granularity, so a user can set her location hint to a low-bit Z-value to obscure her location within a relatively large block, or can use a higher-bit Z-value for a finer-grained location hint. These bit levels are determined by the uncertainty parameter. Using boundaries for the geographic region associated with the group (e.g., the city limits of Seattle for a group of Seattle residents), the VIS recursively partitions geographic space into square blocks. The recursion stops when the next step would create blocks with width smaller than the uncertainty specified by setlocation. This ensures that the user’s location hint will properly obscure its location. It should be noted that in practice, obscuring a user’s location may be more challenging than described above since large portions of a block may be unlikely to contain people (e.g., in the ocean within a coastal block). Dealing with such complexities is beyond the scope of this paper.

Next, the VIS computes the updated location’s Z-value. If the Z-value of the new location and the old are the same, the VIS does nothing. If the Z-values differ, the VIS removes itself from the group’s skip graph and reinserts itself using the new Z-value as its key. As noted in Section 2.2, joining and leaving a skip graph are relatively expensive operations since they involve resorting all of the skip lists. However, humans often cling to long-term destinations, or *hubs* [10], which should make skip-graph reordering rare.

Now consider a user who wishes to retrieve the identities of all group members within a geographic area through getlocation. As before, we will assume that the request originates on a user’s mobile device, and is sent to the user’s VIS. The begin and end parameters of a getlocation request specify the upper-left point and bottom-right point of a rectangular region, respectively. As with

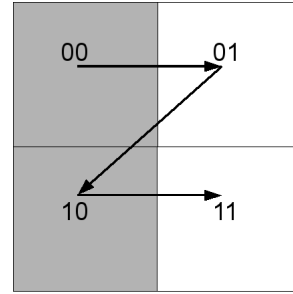


Figure 5: Reducing false positives by limiting range searches to subregions of interest

setlocation, the VIS recursively partitions geographic space into square blocks using the geographic region associated with the group. However, this time the recursion stops when the next step would create blocks with width smaller than the finest-grained block desired.

A naive way of implementing getlocation is to search for the finest-grained Z-value corresponding to the upper-left point, and then collect VIS-hint pairs along the level₀ skip list until the search reaches the finest-grained Z-value corresponding to the lower-right point. Due to the ordering of Z-values, walking the level₀ list will identify all location hints contained in the region of interest, regardless of their granularity.

Unfortunately, just following the Z-curve can generate false positives. Consider the example in Figure 5. If the user is interested in the grey region, simply traversing the Z-curve will also include 01, which is not part of the desired region. Although getlocation results are best-effort by definition, the additional skip-list traversals will increase latency, since walking the list requires synchronous message exchanges with each node on the path. A better strategy is to perform searches in parallel by breaking down the desired region into continuous ranges of the Z-curve, and issuing concurrent searches on each range. In Figure 5, we would perform parallel queries for blocks 00 and 10.

VISs can perform this optimization using the Z-order *next-match* algorithm [15]. This algorithm begins with the lowest Z-value in the specified region and continues on the Z-curve until it reaches a Z-value that lies outside the region of interest. All points reached until this exit make up the first continuous range. To find the next continuous range, next-match increments the Z-value of the exit block until it re-enters the region. This entry Z-value represents the start of the second continuous range. This process continues until the last Z-value in the region is reached. Each continuous region can be searched as parallel key-range queries.

Concurrency improves search latency, but may come at a cost. First, if each parallel search is assigned a thread, large searches may overload a VIS. Second, because each parallel search involves $O(\log n)$ messages to reach the beginning of each range, issuing many parallel searches will generate more network traffic than a simple sequential search. For example, in the extreme case when a region is composed of m blocks and each range query contains a single Z-value, parallel searches will generate $O(m * \log n)$ messages while a simple sequential search will only generate $O(m)$ messages.

We can balance search latency and network traffic by exploring regional partitions under different block sizes. Increasing the block size reduces the number of parallel searches, but increases the number of false positives since bigger blocks may extend beyond the region of interest. More experience is needed to understand the tradeoffs among search latency, network traffic, and false positive rate.

3.4 Supported Features

In this section, we briefly discuss how common features of mobile social services can be implemented using `getlocation` and `setlocation`.

Friend Finder Friend finders notify a user when someone of interest is nearby. Identifying nearby friends is straightforward: a user's VIS simply queries her friends' VISs for their location directly. Since users have a limited number of social relations, this simple approach will scale well. A more interesting friend finder service is to be notified when a stranger with common interests is nearby. To support this feature, a user first specifies the size of the area around her in which she is interested. Using this area, her mobile phone can compute an appropriately-sized region around her current location, and submit `getlocation` requests for every group that captures the attributes she values in others.

Live Mobile Queries Live mobile queries allow a user to send a query to members of a group that fall within a physical region. To use this feature, a user first formulates the query text in her mobile device, and selects a region on a graphical map. She can then forward the query text, group identifier, and region to her VIS. The VIS can perform a `getlocation` for the group using the region. The VIS then sends the query text to every VIS in the returned list, aggregates the responses, and forwards them back to the user's mobile device.

Virtual Sticky Notes Virtual sticky notes allow a user to post a persistent location-based virtual note that another user can view at a later time when she is physically close to the note's associated location. For example, a user can post reviews or comments of a particular exhibit, and another user visiting the exhibit can view the post at a later time.

Virtual sticky notes can be implemented by storing sticky-note meta-data in the key-region of the skip graph corresponding to the data's geo-tag. This meta-data would point to actual sticky-note data such as text, video, and images files, which would be maintained by note authors. As VISs update their location hint, they would become responsible for different ranges of the key space, but only storing meta-data in the skip graph reduces the amount of data that has to be copied between VISs. To access the sticky notes for a location, VISs would submit `getlocation` requests, and then directly request pointers for associated sticky notes from each member of the returned lists of VISs.

4. IMPLEMENTATION

4.1 Additions to Vis-à-Vis Prototype

Our Vis-à-Vis prototype uses Pastry [28] to provide basic distributed hash table (DHT) functionality. It also uses Scribe [4] to provide multicast functionality on top of Pastry DHTs, but only in groups whose configuration options require multicast.

Each VIS runs an Apache Tomcat server in addition to the core DHT-based software. We deployed Java Server Pages (JSPs) and

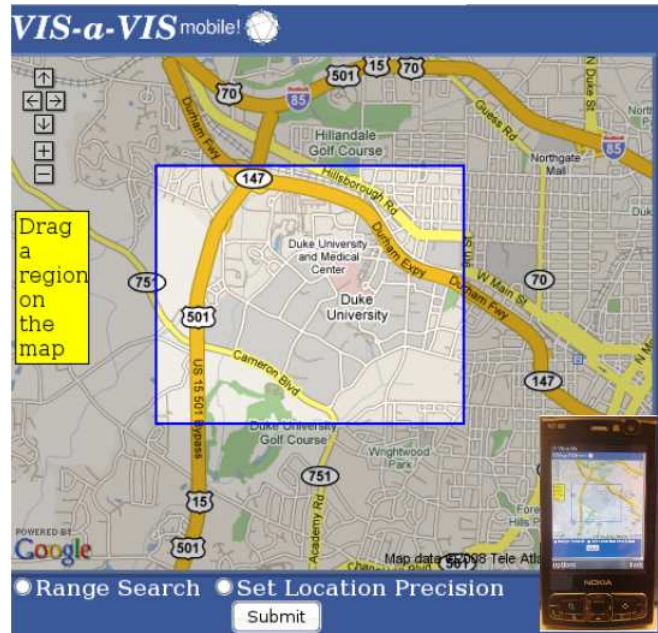


Figure 6: Mobile application running on a Nokia N95 phone

Servlets in the Tomcat server to implement external interfaces and their underlying logic. For example, the JSPs present web forms that a person can use to create, join, and leave a group, as well as advertise and search for information.

When deploying a VIS we encapsulate all the above software along with the requisite Java JDK inside a self-contained virtual machine image. We have successfully run VISs in a variety of virtualized computing environments, including EC2, Emulab, PlaceLab, and an experimental utility computing facility at Duke University.

In this work we added support for location-based groups to our previous Vis-à-Vis implementation, following the design described in the previous section. Location-based functionality is simply another configuration option when creating a Vis-à-Vis group. As a result, we were able to reuse a great deal of the existing Vis-à-Vis prototype. For example, all communication within location-based groups is done through Pastry and uses the same authentication scheme as for other Vis-à-Vis groups. The addition of skip-graph data structures and the related location-based operations added about 2,000 lines of code to our Vis-à-Vis implementation.

4.2 Mobile Application

We also created a mobile application for users to interact with our new location-based groups. Figure 6 shows a screen sample from this application running on a Nokia N95 phone. Using the Google Maps API, the application allows a user to define a rectangular region on a map. These regions can be used for two purposes: to set the precision of location updates sent to the mobile device's VIS, and to define the range of search queries sent to the VIS. The application converts two-dimensional geographic coordinates (latitude and longitude) into one-dimensional Z-values before submitting them to the VIS as part of an update or query.

We chose to calculate Z-values on the mobile device to reduce communication between the device and its VIS. For example, when the

mobile device senses a new location but the application discovers that the corresponding Z-value at the specified precision remains the same as for the previous location, there is no need to send an update to the VIS. Another approach would be for the mobile device to send raw coordinates to the VIS and have the VIS calculate the Z-values. There is a tradeoff here between the amount of processing performed by the mobile device and the number of requests sent from the device to its associated VIS. Exploring this tradeoff more fully is left for future work.

In the case of location range queries, our mobile application attempts to optimize the query in the following sense. It may submit the query as a single range of two Z-values, or it may partition the query into multiple, smaller, parallel searches. The tradeoff here is between the number of false positives returned and the number of requests sent. Our application estimates the cost of various alternatives and picks the least expensive. We explain our cost function and explore this tradeoff in greater detail in the following section.

4.3 Batch Application

The mobile application just described is used by live users to update their location and issue range queries from their devices. For evaluation purposes we implemented a separate batch application that parses mobility traces and submits location updates to VISs on behalf of emulated mobile devices. This multithreaded application is able to send updates in real time or at a configurable submission rate. It can also send range queries. We used this batch application to drive many of the experiments described in the following section.

5. EVALUATION

In evaluating our implementation of location-based groups for mobile social services, we wanted to answer the following questions:

- How many location updates does a user generate in a typical day?
- How does our implementation of location-based groups for Vis-à-Vis perform using data from real mobility traces?
- What is the latency of common operations on our location-based groups?
- How scalable is our location-based Vis-à-Vis prototype?
- How much skip-graph maintenance traffic does a VIS generate?
- How much memory overhead do our location-based groups incur?
- What are the tradeoffs between various range search schemes?

5.1 User Mobility Characterization

5.1.1 Mobility Traces

We set out to characterize mobility behavior of real users in a location-based group. In particular, we are interested in characterizing the amount of movement a user exhibits in a typical day to give insight into the expected number of location updates per user our system needs to handle.

To characterize user mobility, we used two live traces: one collected by us through the Duke University Smart Home as part of this work, and the other the existing University of California in San Diego (UCSD) Wireless Topology Discovery (WTD) trace [32].

Duke University Smart Home Trace. In the Smart Home program, sponsored by the Duke University Pratt School of Engineering, a group of ten engineering students live in a 6,000 square-foot house [7]. The centerpiece of the house is a laboratory that allows students to conduct various hands-on engineering experiments.

Data Type	Sampling Rate (in seconds)
GPS	10
GSM cell tower	60
WiFi AP	120
Bluetooth	300

Table 1: Sampling rate of the location data types

For our mobility trace experiment, we provided each student with a Nokia N95 8GB mobile phone pre-installed with Nokoscope, a mobile application developed by Nokia Research [24]. This application keeps a log of a wide variety of information available from the mobile device and opportunistically uploads the data to a central database. We configured the application to log only the information pertaining to GPS, GSM cell tower, WiFi access points sensed and Bluetooth devices sensed. Table 1 shows the sampling rates of the information the participants' devices logged.

For this paper, we only used the GPS and WiFi data. The data pertaining to WiFi access points only contain the MAC addresses of the access points, and not the actual location. To get a rough estimate of a participant's location from the WiFi access points, we used an API provided by Skyhook Wireless [31], which allows us to get the physical location from the MAC address by sending an HTTP request to Skyhook. The location of a user at a particular sample time is the centroid of the location of all access points detected.

The mobility trace experiment started on November 15, 2008, and is still ongoing. It should be noted that the participants can opt-out of the experiment at any time. Moreover, the participants at any time can reconfigure the Nokoscope application to divulge only information that they are comfortable releasing.

UCSD Wireless Topology Discovery Trace. The 11-week WTD trace was gathered by the UCSD SysNet Group, which recently made the trace data publicly available. In this experiment, wireless PDAs pre-installed with WTD software were given to around 300 freshmen students at UCSD [32]. The WTD software samples and logs nearby WiFi access points every 20 seconds.

5.1.2 Mobility Statistics

As mentioned in Section 3, users can choose the granularity of their location hint. The level of granularity is inversely proportional to the number of location updates, since in our prototype, the mobile device sends a location update request to its VIS only when the user moves out of her defined region. For both traces, we characterized a user's mobility with location information granularity of 80 meters, 160 meters and 320 meters.

Figure 7 shows the empirical cumulative distribution function (CDF) of the number of location updates a user generates on a typical day. In the Duke University smart home trace, even with a granularity of 80 meters, 90% of the users have fewer than or equal to 240 location updates in a day. Likewise, in the UCSD wireless topology

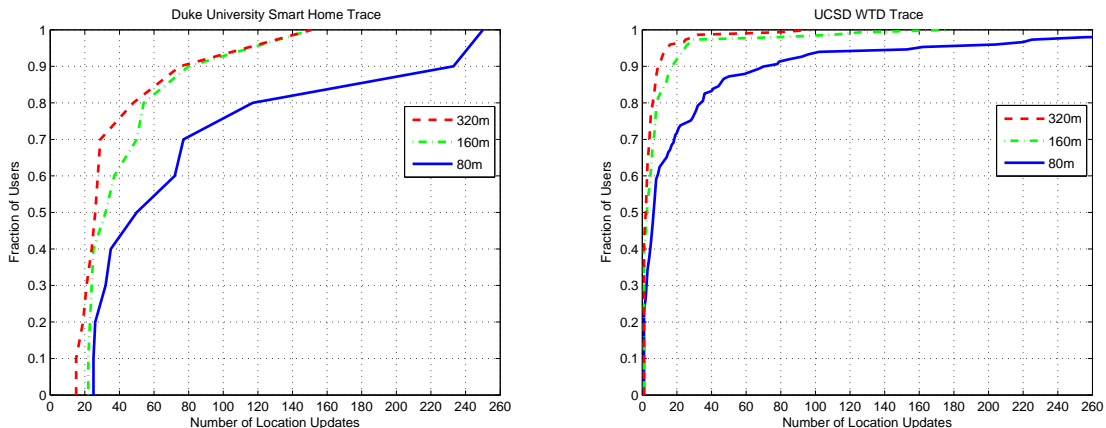


Figure 7: Empirical CDF of the number of location updates a user has on a typical day

discovery trace, 90% of the users have fewer than 80 location updates in a day. From these results, we can see that on a typical day a user’s location does not change very often; hence, the rate at which skip lists must be resorted due to location updates is expected to be low.

5.2 Location Update Operation from Traces

We are also interested in the latency and the number of hops required to perform a setlocation operation under the two mobility traces: the Duke University Smart Home trace and the UCSD wireless topology discovery trace. Latency measures the time to complete an update of a user’s location hint, though there are other external factors that also affect latency. Hence, we also measured the number of hops needed to complete the setlocation operation. In this metric, we measured the number of times a location update message gets forwarded to another group member before the operation completes.

5.2.1 Experimental Setup

We deployed our Vis-à-Vis prototype on an experimental utility computing cluster at Duke University, where each user’s VIS runs in a node on the cluster. Each node has 1GB of memory and a 2GHZ CPU. We drove the experiments with our batch application that, instead of processing location information from sensor readings on a mobile device, reads the location information from a trace. The batch application can also be configured to emulate the actual time intervals of the entries in the trace data. We ran the experiment on trace data equivalent to one day. Moreover, for this experiment we picked a day from each trace that had significant aggregate location update activity from the collection of participants. The traces used for this experiment contain 10 active participants from the Duke trace and 20 active participants from the UCSD trace.

5.2.2 Results

Table 2 shows the result of our experiment. On real data, the average time it takes to complete a location update operation is between 4 and 6 seconds, which seems acceptable for social networking purposes. Also, for 10 and 20 active participants, the average number of hops of each group’s location-update request is roughly the same.

5.3 Scalability of Operations

We are interested in whether the size of the group will degrade a user’s experience. To capture the effects of the size of the group on application response time, we analyze our prototype’s primitive operations, specifically the insert, leave, lookup, and range-search operations. We focus on these primitive operations because other operations are a combination of these. To test the scalability of the insert and search primitives, we measured the latency and number of hops, using a synthetic workload which will be explained in the next subsection. Due to the way we implemented the leave and range-search operation, those primitives’ performance can be analytically derived.

The leave operation’s computational complexity is probabilistically $O(2 * \log n)$ because a node can have $\log n$ levels and for each level a node has to notify its left and right neighbor of its intention to leave. The range search is a constant addition to the complexity of a search operation because we first search and then traverse the nodes in level₀.

5.3.1 Experimental Setup

We deployed our Vis-à-Vis prototype on Emulab [33], a testbed that provides resources for experiments on distributed systems. All of the resources are located on a single site and users can request a set of virtual machines. Moreover, Emulab provides users an interface to customize the network behavior and topology of the virtual machines.

For the insert and lookup operations, we ran our experiment 25 times on a group of size n , where n is 20, 40, 60, 80 and 100. For the insert operation, we first created a group of size n from n randomly chosen nodes, and then measured the latency and the number of hops it takes for the $n + 1^{th}$ (randomly chosen) node to successfully insert itself to the skip graph. The experiment for the lookup operation also involves creating a group of size n and then measuring the latency and the number of hops of a randomly picked node to perform the desired operation.

5.3.2 Results

Figure 8 and Figure 9 show the results of our experiment. Even with 100 nodes, the mean latency of the insert operation is still only less than 2 seconds longer than that of a group size of 20 nodes,

	Average Number of Hops	Standard Deviation of Hops	Average Latency	Standard Deviation of Latency
Duke Trace	9.1	3.9	4.7s	1.8s
UCSD Trace	8.6	3.4	5.7s	3.0s

Table 2: Performance of Location-Update Operation

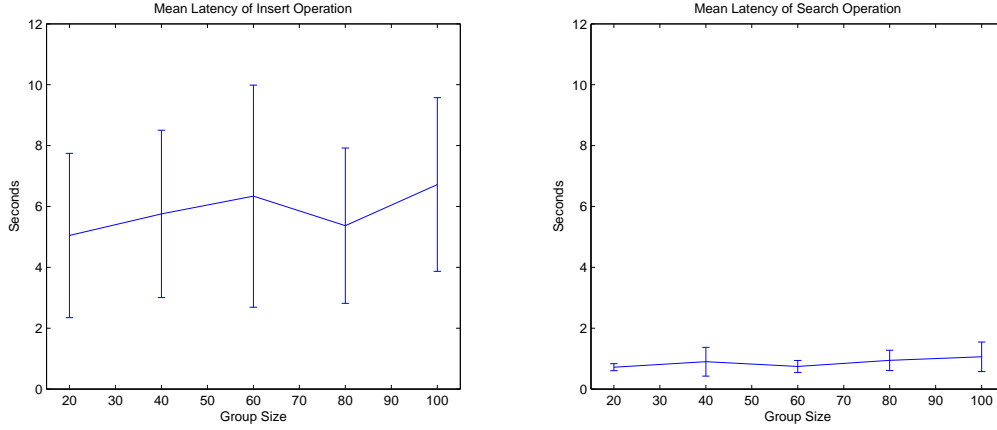


Figure 8: Mean Latency of insert and search operations. Error bars show the standard deviation.

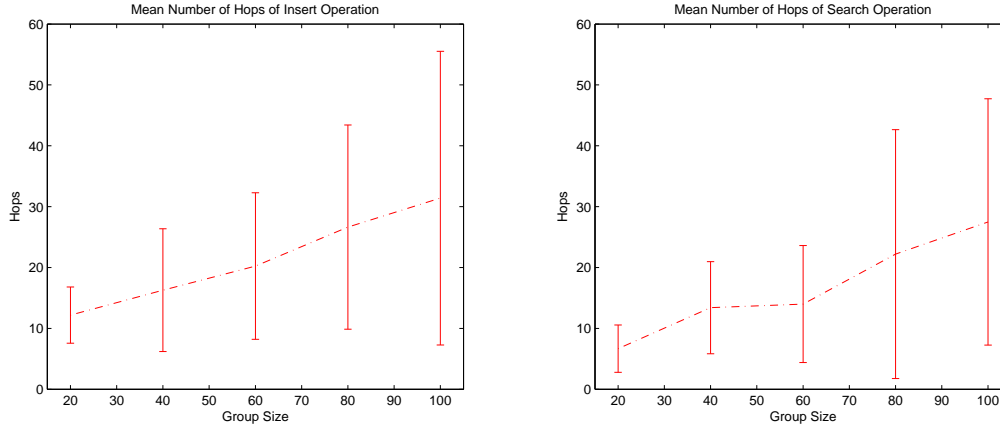


Figure 9: The mean number of hops of insert and search operations. Error bars show the standard deviation.

which shows that the insert operation is scalable. This is encouraging since our prior work on characterizing groups within Facebook found an average group size of close to 250 [29]. The results of our experiment show that the performance of the search operation is also not greatly affected by group size. Furthermore, it can be seen in Figure 9 that the number of hops increases slowly as the size of a group increases. It can also be seen that the standard deviation of each operation grows steadily with the size of the group. However, this behavior is to be expected because the membership vector of each node is generated randomly. Hence, the distribution of nodes in the lists at a particular level is not equal. It is possible that a particular list at some level can be densely populated while another list is sparsely populated. Depending on the number of members in a particular list, a node’s operations can be faster or slower based on its ability to skip over other nodes.

5.4 Network Maintenance Traffic

Although we have already characterized the mobility of users in the previous subsection and determined that the churn rate due to

location updates is low, we are still interested in the actual amount of network maintenance traffic our prototype generates. Similar to our previous experiment, we used an experimental utility computing cluster at Duke University and replayed a day’s worth of the Duke Smart Home trace data. For each of the machines, we logged the number of messages sent and received by our prototype. Since we know the size of these messages, we can estimate the amount of data being sent and received between nodes.

	Average	Min	Max
IN	1.14MB	.1MB	3.6MB
OUT	1.16MB	.1MB	4MB

Table 3: Average network maintenance traffic per day

Table 3 shows the result of the experiment. On average, a user’s node only consumes and generates around 1MB of data per day. Even if a user updates her location information frequently (in our trace the most active user had 151 location updates per day), the

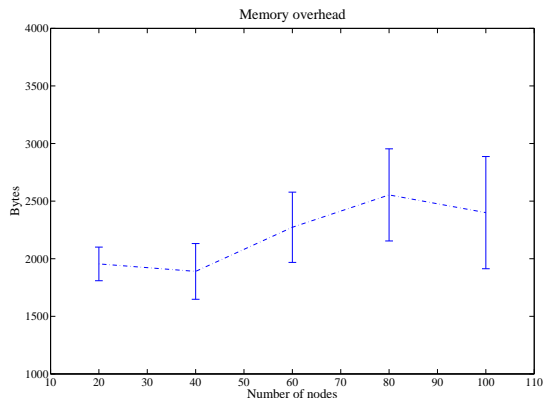


Figure 10: Memory overhead for maintaining a skip graph

amount of data generated is still only 4MB per day. This amount of traffic would be inexpensive to support. For example, Amazon EC2 currently charges \$0.10/GB for incoming traffic and \$0.17/GB for outgoing traffic, so that on average a user would only pay \$0.0003 per day for network traffic.

5.5 Memory Overhead

The memory overhead of our prototype is another important metric because it gives insight into how much memory per location-based group a VIS needs to maintain. We are interested in seeing how much memory is consumed as the size of the group grows and whether or not the memory overhead is feasible for a VIS to handle.

To get an estimate of the memory overhead, we serialized and measured the relevant Java objects in our prototype. We then, measured the sizes of these objects for group sizes of 20, 40, 60, 80 and 100. Figure 10 shows the result of this experiment. As it can be seen, memory overhead is less than 3KB for these groups, even for a group size of 100.

5.6 Analysis of Range-Search Schemes

As mentioned in Section 3, we used a next-match algorithm to reduce the number of false positives in the range-search operation. Consider the two range search queries on Figure 11. We can see that using the next-match algorithm at a fine granularity increases the number of parallel searches, and potentially decreases the number of false positives. However, though performing many parallel searches reduces latency and the false positive rate, it increases global network traffic because it creates more messages between nodes in the system.

One way to balance these issues is to make the search precision coarser. By choosing a bigger location region, we may be able to reduce network traffic without adversely affecting latency or accuracy. To characterize these tradeoffs, we compare the cost of using the next-match algorithm with other range-search schemes.

We use real mobility traces and parameters from our experiments to analytically quantify the tradeoffs of the following schemes:

- Linear search
- Naive range search
- Coarse-grained Next-match range search
- Fine-grained Next-match range search

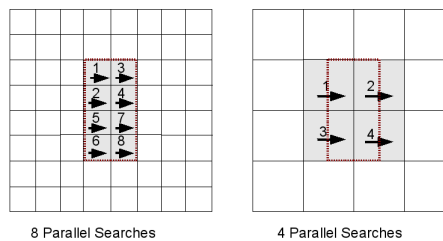


Figure 11: Number of parallel searches vs. granularity of next-match range searches

Linear search simply traverses the level₀ list and finds all users within the range. The naive range search performs a lookup of the minimum value of the range on the skip graph, and then traverses the level₀ list until the end of the range. Coarse- and fine-grained next-match range searches perform parallel search queries, with granularities of 640m and 160m, respectively.

In this experiment, we calculated the average number of false positives and the average number of parallel search requests. This requires location data of all users and the actual range search query. We used the UCSD wireless topology discovery trace for the evaluation since it has a higher number of users and location updates.

First, we set the location hints of our VISs by randomly taking a snapshot of the mobility trace and extracting the locations of all users at the time of the snapshot. Then, we generated range searches by randomly picking a region on the map with the following restriction: the range had to be at least 25% of the map to ensure that there are enough VISs within the search region.

Using this data we calculated the number of false positives and the number of parallel search requests for different range search schemes. To get a good sample size, we ran the experiment 5000 times in the Duke University computing environment. We then obtained the average number of false positives and the average number of parallel search requests for each range search scheme. The results are reported in Table 4.

We define two functions that can give insight into the cost in hops and time of using a particular range search scheme:

$$C_{hops} = (k \times O_{h1}) + (fp \times O_{h2}) \quad (1)$$

$$C_{time} = O_{t1} + \frac{1}{k}(fp \times O_{t2}) \quad (2)$$

where k is the average number of parallel search requests; O_{h1} and O_{t1} is the number of hops and time overhead of finding the starting element of the range search, respectively; fp is the average number of false positives of a particular range search scheme; and O_{h2} and O_{t2} is the number of hops and time it takes to communicate to another node, respectively. Table 5 shows the parameters we used for calculating the cost of each of the range search schemes. These parameters are derived empirically from our experiments.

Table 6 shows the costs of the range search schemes after applying values from Table 5 to cost functions (1) and (2). As expected, linear search is the most expensive, in terms of hops and latency. Naive range search generates the least global network traffic but still has at least double the latency of the next-match range

Scheme	Average number of parallel searches	Average number of false positives
Linear Search	1	89
Naive Range Search	1	89
Coarse-grained Next-match Range Search	2	58
Fine-grained Next-match Range Search	6	53

Table 4: Average numbers of parallel searches and false positives

Scheme	O_{h1}	O_{t1}	O_{h2}	O_{t2}	k	fp
Linear Search	152 hops	60.8s	1 hop	.4s	1	89
Naive Range Search	31.4 hops	1.06s	1 hop	.4s	1	89
Coarse-grained Next-match Range Search	31.4 hops	1.06s	1 hop	.4s	2	58
Fine-grained Next-match Range Search	31.4 hops	1.06s	1 hop	.4s	6	53

Table 5: Parameters used for the cost functions

Scheme	Hops	Latency
Linear	241	96.4s
Naive	120.4	36.66s
Coarse-grained Next-match	120.8	12.66s
Fine-grained Next-match	241.39	4.59s

Table 6: Costs of Range-Search Schemes

searches. Fine-grained next-match range search has the least latency but at the same time generates significant global network traffic. Coarse-grained next-match range search seems to have a good balance between the amount of global network traffic and latency.

However, it is still not clear which scheme presents the best trade-off. From an individual user’s point of view, it is better to have the lowest latency possible. From a global point of view, it is better to generate the least network maintenance traffic possible. We leave for future work the issue of finding the best operating point. One possible way to approach this problem would be to train the system to dynamically find the range search scheme that gives the best tradeoff for the current situation.

6. RELATED WORK

This section discusses related work that has not already been mentioned elsewhere in this paper.

SkipNet [12] is an overlay network also based on skip graphs. The design of SkipNet focuses on maintaining the content and path locality of the overlay network. We build on the same underlying data structure to preserve locality. However, our motivation for Vis-à-Vis is quite different in that we focus on designing a privacy-preserving framework for mobile social services.

ZNet [30] is also another peer-to-peer system that uses skip graphs. Moreover, they also use Z-order space-filling curves to convert multi-dimensional data into keys. In ZNet, a node in the skip graph is responsible for information corresponding to a whole region. In contrast, a node in Vis-à-Vis is only responsible for its own information, which means that a user retains control of her own information. Furthermore, Vis-à-Vis uses a different approach to range search by using the next-match algorithm [15].

P2PR-tree [21] takes a different approach to indexing spatial data in peer-to-peer systems by extending the R-tree data structure to distributed systems. An R-tree splits a space into hierarchical bound-

ing rectangles and in P2PR-tree, each node only needs to maintain minimal information about the R-tree by having each node only maintain at least one peer at each hierarchical level. Unfortunately, P2PR-tree does not address some operations needed for location-based services, such as location update and remove operations.

Social VPN [8] is a virtual network that exploits social and overlay networks. It uses the infrastructure of social networks, such as Facebook, to exchange credentials of users. Social VPN takes advantage of the relationships formed in social networks to determine which pairs of people already trust each other, so as to automatically configure IPsec tunnels between machines belonging to these pairs. In contrast, Vis-à-Vis keeps track of the social relationships and handles the OSN operations. Moreover, the motivation of Social VPN is fundamentally different from the motivation of Vis-à-Vis. We avoid turning over sensitive information to a central server, since this would undermine the whole idea of Vis-à-Vis.

NOYB [11] takes a different approach to the privacy threats of centralized OSNs by encrypting some of the data that users hand to the service. This is appealing because it may allow users of existing OSNs to retain their existing profiles, while Vis-à-Vis requires users to completely divest themselves from existing OSNs.

NOYB partitions user data into small cryptographic units called atoms, which correspond to various attributes associated with a user. The technique allows users to cooperatively scramble their profile attributes with other trusted users’ via pseudorandom substitution so that friends can locate a user’s real attributes, but the OSN cannot. Another key feature of NOYB is that the OSN is oblivious to the presence of the encrypted data, because atoms stored in the OSN look like legitimate profile values.

NOYB represents a very appealing approach to privacy in OSNs, but there are several drawbacks compared to Vis-à-Vis. First, retaining any presence in a centralized OSN leaves users open to the “Beacon attack,” in which the service directly notifies a user’s friends of potentially sensitive activity in other corners of Internet. Vis-à-Vis users are not vulnerable to such attacks because their VISs control all data sent along their social links. Second, NOYB requires additional key-managing software to be installed on any client machine accessing encrypted profile data, including public kiosks and mobile phones where it may not be convenient. Vis-à-Vis only requires clients to have a web browser. Finally, it is unclear how well NOYB generalizes to non-textual information, while Vis-à-Vis can secure arbitrary data types.

Like NYOB, flyByNight [19] uses encryption to ensure that sensitive data is never posted to OSNs in unencrypted form. The prototype takes the form of a Facebook application that uses client-side JavaScript to encrypt and decrypt sensitive data. flyByNight is also appealing because it allows users to continue using existing OSNs and the social state that users have accumulated there. However, flyByNight remains vulnerable to several types of attack from within the OSN, which Vis-à-Vis avoids by doing away with centralized OSNs altogether.

Turtle [26] is an anonymous peer-to-peer network designed for private information sharing using pre-existing social relations. Turtle projects trust relationships from the real world into the overlay to eliminate fear of censorship or legal sanctions in open networks. To control sensitive data every node allows to assign a specific attribute set to each shared data item, such that only authorized people are able to get this item. All communications within Turtle are encrypted and search queries are broadcasted to all “friend nodes” using a hop count bit. Vis-à-Vis has a similar motivation as Turtle and has a few similar design decisions. Namely, groups in Vis-à-Vis also assume out-of-band mutual trust between nodes and every member of the group relies on all members of this group. Thus, betrayal of one member can compromise other members. Besides, sensitive data can be received only through friends, hence, eliminating the man-in-the-middle attack. However, Turtle does not try to position itself as a social network service, and, therefore, it does not provide any means for forming different groups or advertising points of interests without flooding all members. Moreover, Turtle’s underlying architecture is an unstructured overlay, hence, flooding a query may not be efficient when compared to multicast within a DHT. Finally, it is not clear how Turtle behaves in the presence of churn and unavailability of some friends.

7. CONCLUSION

We have presented a privacy-preserving alternative to the prevailing centralized architecture for mobile social services. In our approach, each person maintains her own location history in her own Virtual Individual Server (VIS), a personal virtual machine running in a utility computing infrastructure. Each VIS acts as a trusted and resource-rich proxy for its owner’s personal mobile device, which opportunistically sends location updates to the VIS. VISs can share this location information by organizing into overlay networks, one per social group with which VIS owners wish to share information. Our system exploits skip graphs and Z-order space-filling curves to provide efficient and scalable operations on this distributed location data. We have demonstrated that our prototype system has reasonable performance through an experimental evaluation driven by real-world location traces. Our decentralized approach to mobile social services makes large-scale privacy breaches much less likely than in centralized architectures, and gives people fine control over what location information they share with whom.

8. REFERENCES

- [1] Amazon Web Services LLC. Amazon elastic compute cloud (amazon ec2), 2008. <http://aws.amazon.com/ec2/>.
- [2] Amazon Web Services LLC. Amazon web services customer agreement, September 2008. <http://aws.amazon.com/ec2/agreement/>.
- [3] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37+, November 2007.
- [4] M. Castro, P. Druschel, A. marie Kermarrec, and A. Rowstron. Scalable application-level anycast for highly

- dynamic groups. In *In Proc. NGC 2003*, 2003.
- [5] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge. Location Disclosure to Social Relations: Why, When, and What People Want to Share. In *Proceedings of the Conference On Human Factors in Computing Systems (CHI)*, April 2005.
- [6] J. R. Douceur. The sybil attack. In *Workshop on Peer-to-Peer Systems*, 2002.
- [7] Duke University. Duke university smart home program. <http://www.smarthome.duke.edu/>.
- [8] R. Figueiredo, O. Boykin, P. St. Juste, and D. Wolinsky. Social VPNs: Integrating overlay and social networks for seamless P2P networking. In *Workshop on Collaborative Peer-to-Peer Systems (COPS)*, Rome, Italy, June 2008.
- [9] S. Gaonkar, J. Li, R. R. Choudhury, L. P. Cox, and A. Schmidt. Micro-Blog: Privacy-aware people-centric sensing. In *MobiSys*, June 2008.
- [10] J. Ghosh, M. Beal, H. Ngo, and C. Qiao. On profiling mobility and predicting locations of campus-wide wireless network users. In *Proceedings of REALMAN*, pages 55–62, May 2006.
- [11] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in online social networks. In *SIGCOMM Workshop on Social Networks*, 2008.
- [12] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.
- [13] G. Iachello, I. Smith, S. Consolvo, G. Abowd, J. Hughes, J. Howard, F. Potter, J. Scott, T. Sohn, J. Hightower, and A. LaMarca. Control, Deception, and Communication: Evaluating the Deployment of a Location-Enhanced Messaging Service. In *Proc. of the 7th Intl. Conf. on Ubiquitous Computing (UbiComp 2005)*, September 2005.
- [14] B. W. Lampson. Hints for computer system design. In *IEEE Software*, pages 33–48, 1983.
- [15] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. *Lecture Notes in Computer Science*, 1832:20–??, 2000.
- [16] S. Lederer, J. I. Hong, A. K. Dey, and J. A. Landay. Personal privacy through understanding and action: five pitfalls for designers. *Journal of Personal and Ubiquitous Computing*, 8(6):440–454, 2004.
- [17] Loopt, Inc. Your social compass | loopt. <http://www.loopt.com>.
- [18] Louise Story and Brad Stone. Facebook retreats on online tracking, November 2007. *The New York Times*.
- [19] M. M. Lucas and N. Borisov. flybynight: Mitigating the privacy risks of social networking. In *WPES '08: Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 1–8, New York, NY, USA, 2008. ACM.
- [20] Megan McCarthy. How Facebook employees break into your profile, November 2007. <http://www.valleywag.com>.
- [21] A. Mondal and Y. M. Kitsuregawa. P2pr-tree: An r-tree-based spatial index for peer-to-peer environments. In *Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (held in conjunction with EDBT)*, page 516. Springer-Verlag, 2004.

- [22] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., March 1966.
- [23] A. J. Nicholson, I. Smith, J. Hughes, and B. D. Noble. Lokey: Leveraging the sms network in decentralized, end-to-end trust establishment. In *Proc. Pervasive 2006*, Springer-Verlag, pages 202–219, 2006.
- [24] Nokia Corporation. Nokoscope. <https://alpha.nokoscope.com/eb2>.
- [25] Privacy rights clearinghouse. <http://privacyrights.org>.
- [26] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. In *Proc. of the 12th Cambridge Intl. Workshop on Security Protocols*, 2004.
- [27] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33:668–676, 1990.
- [28] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middleware*, November 2001.
- [29] A. Shakimov, H. Lim, L. P. Cox, and R. Cáceres. Vis-à-Vis: Online Social Networking via Virtual Individual Servers. submitted for publication, October 2008.
- [30] Y. Shu, B. C. Ooi, and K. lee Tan. Supporting multi-dimensional range queries in peer-to-peer systems. In *Fifth IEEE International Conference on Peer-to-Peer Computing, 2005. P2P 2005*, pages 173–180, 2005.
- [31] Skyhook Wireless, Inc. Skyhook wireless. <http://www.skyhookwireless.com/>.
- [32] UCSD wireless topology discovery trace. <http://ramp.ucsd.edu/wtd/>.
- [33] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, December 2002.