

THE NATIONAL UNIVERSITY  
of SINGAPORE

School of Computing  
Lower Kent Ridge Road, Singapore 119260

**TRB7/06**

*PRIVE: Anonymous Location-Based Queries  
in Distributed Mobile Systems*

*Gabriel GHINITA, Panos KALNIS  
and Spiros SKIADOPOULUS □ □*

*July 2006*

# Technical Report

## Foreword

*This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.*

JAFFAR, Joxan  
Dean of School

# PRIVÉ: Anonymous Location-Based Queries in Distributed Mobile Systems

Gabriel Ghinita<sup>1</sup>

Panos Kalnis<sup>1</sup>

Spiros Skiadopoulos<sup>2</sup>

<sup>1</sup>Dept. of Computer Science  
National University of Singapore

{ghinitag, kalnis}@comp.nus.edu.sg

<sup>2</sup>Dept. of Computer Science & Technology  
University of Peloponnese, Greece

spiros@uop.gr

## Abstract

Nowadays, mobile users with positioning devices can access Location Based Services (LBS) and query about points of interest in their proximity. For such applications to succeed, privacy and confidentiality are essential. Encryption alone is not adequate; although it safeguards the system against eavesdroppers, the queries themselves may disclose the location and identity of the user. Recently, there have been proposed centralized architectures based on  $\mathcal{K}$ -anonymity, which utilize an intermediate anonymizer between the mobile users and the LBS. However, the anonymizer must be updated continuously with the current locations of all users. Moreover, the complete knowledge of the entire system poses a security threat if the anonymizer is compromised.

In this paper we address two issues: (i) We show that existing approaches may fail to provide spatial anonymity for some distributions of user locations and describe a novel technique which solves this problem. (ii) We propose PRIVÉ, a decentralized architecture for preserving the anonymity of users issuing spatial queries to LBSs. Mobile users self-organize into an overlay network with good fault tolerance and load balancing properties. PRIVÉ avoids the bottleneck caused by centralized techniques both in terms of anonymization and location updates. Moreover, the state is distributed in numerous users, rendering the system resilient to attacks. Extensive experimental studies suggest that PRIVÉ is applicable to real-life scenarios with large populations of mobile users.

## 1. Introduction

The increased popularity of mobile communication devices with embedded positioning capabilities (e.g., GPS, RFID) has triggered the development of location-based applications. General Motor’s OnStar navigation system, for example, combines the vehicle’s position with real-time in-

formation to avoid traffic jams and automatically alerts the authorities in case of an accident. More applications based on the users’ location are expected to emerge with the arrival of the latest gadgets (e.g., iPAQ hw6515, Mio A701) which combine the functionality of a mobile phone, PDA and GPS receiver.

Consider the following scenario: Bob uses his GPS enabled mobile phone to ask the query “Find the nearest hospital to my present location”. This query can be answered by a *Location-Based Service* (LBS) in a public server (e.g., Google Maps), which is *not* trusted. To preserve his privacy, Bob does not contact the LBS directly. Instead he submits his query via an intermediate trusted server which hides his ID (services for anonymous web surfing are commonly available nowadays). However, the query still contains the exact coordinates of Bob. One may reveal sensitive data by combining the location with other publicly available information. If, for instance, Bob uses his mobile phone within his residence, the untrustworthy owner of the LBS may infer Bob’s identity and speculate that he suffers from a medical condition.

In practice, users are reluctant to access a service that may disclose sensitive information (e.g., corporate, military), or their political/religious affiliations and alternative lifestyle. To preserve privacy in LBSs, recent research focused on adapting the well established  $\mathcal{K}$ -anonymity technique to the spatial domain.  $\mathcal{K}$ -anonymity [17, 19] has been used in statistical databases as well as for publishing census, medical and voting registration data. A dataset is said to be  $\mathcal{K}$ -anonymized, if each record is indistinguishable from at least  $\mathcal{K}-1$  other records with respect to certain identifying attributes. In the LBS domain, a similar idea appears in the work of Ref. [8, 9]. Both approaches employ *spatial cloaking* to conceal the location of the user: Instead of reporting the exact coordinates to the LBS, they construct an *Anonymizing Spatial Region* ( $\mathcal{K}$ -ASR) which encloses the locations of  $\mathcal{K}-1$  additional users. Ref. [12, 15] extend this method by proposing a framework for the entire process of query anonymization in LBSs.

All existing approaches utilize a *centralized* anonymizer. This is a trusted server that acts as an intermediate tier between the users and the LBS. All users subscribe to the anonymizer and continuously report their location while they move. Each user sends his query to the anonymizer, which constructs the appropriate  $\mathcal{K}$ -ASR and contacts the LBS. The LBS computes the answer based on the  $\mathcal{K}$ -ASR, instead of the exact user location; thus, the response of the LBS is a superset of the answer. Finally, the anonymizer filters the result from the LBS and returns the exact answer to the user.

Our work is motivated by the following shortcomings of existing anonymization approaches: (i) The centralized anonymizer is a bottleneck due to the handling of all query requests and the required post-processing, in addition to the frequent updates of user locations. Moreover, the anonymizer is a single point of failure; the system cannot function without it. (ii) The complete knowledge of the locations and queries of all users is a serious security threat, if the anonymizer is compromised. Even if there is no attack, the centralized anonymizer may be subject to governmental control, and may be banned or forced to disclose sensitive user information (similar to the legal case of the Napster file-sharing service). (iii) Independent of the centralized architecture, the hierarchical partitioning method for  $\mathcal{K}$ -ASR construction [9, 15] may fail to provide anonymity under certain conditions (see Section 3).

In this paper, we propose PRIVÉ, a distributed architecture for anonymous location-based queries, which solves the problems of existing systems. Our contributions are: (i) We develop a superior  $\mathcal{K}$ -ASR construction mechanism based on the Hilbert space-filling curve, that *guarantees* anonymity even if the attacker knows the locations of all users. (ii) We introduce a distributed protocol used by mobile entities to self-organize into a fault-tolerant overlay network. The structure of the network resembles a distributed  $B^+$ -tree (each mobile user corresponds to a data point), with additional annotation to support efficiently the Hilbert-based  $\mathcal{K}$ -ASR construction. In PRIVÉ, any participant can play the role of the anonymizer. Therefore, the bottleneck of the centralized server is avoided. Moreover, since the status of the system is distributed, PRIVÉ is resilient to attacks. (iii) We also conduct an extensive experimental evaluation. The results confirm that PRIVÉ achieves efficient anonymization and load balancing with low maintenance overhead, while being fault-tolerant. Therefore, it is scalable to a large number of mobile users.

The rest of the paper is organized as follows: Section 2 discusses the architecture of PRIVÉ. In Section 3, we introduce our Hilbert-based  $\mathcal{K}$ -ASR construction mechanism, whereas in Section 4 we describe the distributed protocol of the overlay network. Section 5 presents the experimental evaluation of our system. A brief survey of the related work

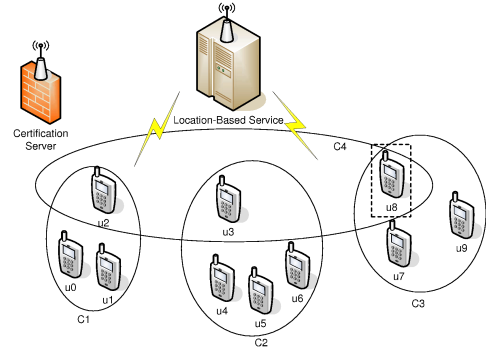


Figure 1. Architecture of PRIVÉ

is included in Section 6. Finally, Section 7 concludes the paper and discusses directions for future work.

## 2. System Architecture

Fig. 1 depicts the architecture of PRIVÉ. We assume a large number of users who carry mobile devices (e.g., mobile phones, PDAs) with embedded positioning capabilities (e.g., GPS). The devices have processing power and access the network through a wireless protocol such as WiFi, GPRS or 3G. Moreover, each device has a unique network identity (e.g., IP address) and can establish point-to-point communication (e.g., TCP/IP sockets) with any other device in the system through a base station (i.e., the two devices do not need to be within the range of each other). For security reasons, all communication links are encrypted.

In addition, we assume the existence of a trusted central *Certification Server* (CS), where users are registered. Prior to entering the system, a user  $u$  must authenticate against the CS and obtain a certificate. Users having a certificate are trusted by all other users. Typically, a certificate is valid for several hours; it can be renewed by recontacting the CS. Apart from the certificate, the CS returns to  $u$  the IP addresses of some users who are currently in the system.  $u$  employs this list to identify an entry point to the distributed network. Note that the CS does not know the locations of the users and does not participate in the anonymization process.

Each user corresponds to a peer. Peers are grouped into clusters, based on their location. Within each cluster, peers elect a cluster head, and the set of heads is grouped recursively to form a tree. To achieve load balancing, the cluster heads change periodically in a round-robin manner. By definition, cluster heads may belong to multiple levels of the tree. In Fig. 1, for instance, there is a two-level hierarchy, where users  $u_2$ ,  $u_3$ ,  $u_8$  are the heads of cluster  $C_1$ ,  $C_2$  and  $C_3$ , respectively; also  $u_8$  is the head of the upper layer cluster  $C_4$ .

Typically users ask Range or Nearest-Neighbor (NN) queries with respect to their position (e.g., “find the near-

est hospital”). Based on individual criteria, users select the required degree of anonymity  $\mathcal{K}$ .  $\mathcal{K}$  may vary among consecutive queries. Users collaborate to achieve spatial cloaking and construct the  $\mathcal{K}$ -ASR in a distributed manner. Then, a random user  $u'$  is selected to send the anonymized query to LBS, filter the result set and forward the actual answer to the query initiator. The result set from the LBS is a superset of the actual answer and its cardinality depends on the size of the  $\mathcal{K}$ -ASR.

PRIVÉ can collaborate with various *untrustworthy* spatial databases providing LBSs. The only requirement from the LBSs is to support NN queries of regions (i.e.,  $\mathcal{K}$ -ASRs) as opposed to points. Intuitively, the nearest neighbors of a region are all the data objects inside the region plus the NNs of every point in the perimeter of the region. Query processing at the LBS [10, 12, 15] is orthogonal to our work but outside the scope of this paper.

### 3. Spatial $\mathcal{K}$ -anonymity

A user  $u$  who issues a location-based query is considered to be  $\mathcal{K}$ -anonymous if his location is indistinguishable from the location of  $\mathcal{K}-1$  other users [9]. Formally:

**Definition [Spatial  $\mathcal{K}$ -anonymity]** Let  $H$  be a set of  $\mathcal{K}$  distinct user entities with locations enclosed in an arbitrary spatial region  $\mathcal{K}$ -ASR. A user  $u \in H$  is said to possess  $\mathcal{K}$ -anonymity with respect to  $\mathcal{K}$ -ASR if the probability  $P$  of distinguishing  $u$  among the other users in  $H$  does not exceed  $1/\mathcal{K}$ . We refer to  $\mathcal{K}$  as the required degree of anonymity.

Note that: (i) The definition assumes a *snapshot* of the users’ locations. Although PRIVÉ supports user mobility,  $\mathcal{K}$ -anonymity is undefined across multiple snapshots. (ii) Spatial  $\mathcal{K}$ -anonymity does *not* depend on the size of the  $\mathcal{K}$ -ASR. In the extreme case, the  $\mathcal{K}$ -ASR can degenerate to a point, if  $\mathcal{K}$  users are at the same location. In general, we prefer small  $\mathcal{K}$ -ASRs, in order to minimize the processing cost at the LBS and the communication cost between the LBS and the mobile user. Nevertheless, some applications impose a lower bound on the size of the  $\mathcal{K}$ -ASR [15]. In such a case, the  $\mathcal{K}$ -ASR can be trivially enlarged to satisfy the lower bound, by scaling it proportionally in all directions.

A naïve  $\mathcal{K}$ -ASR construction algorithm would choose a random  $\mathcal{K}$ -ASR. However, if the  $\mathcal{K}$ -ASR is too small it may contain fewer than  $\mathcal{K}$  users, whereas if it is larger than necessary, it will affect the query cost. Constructing the  $\mathcal{K}$ -ASR in the neighborhood of the querying user  $u$  (e.g., using the  $\mathcal{K}$  nearest neighbors of  $u$ ) is also inappropriate, because  $u$  tends to be closest to the center of the  $\mathcal{K}$ -ASR, thus easily identified.

An optimal  $\mathcal{K}$ -ASR construction algorithm must partition the user population into *static* buckets of  $\mathcal{K}$  users, such

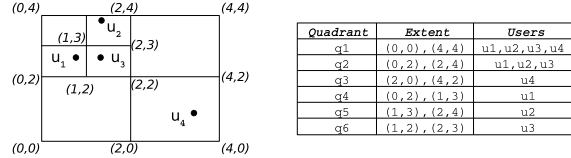


Figure 2. Limitations of QUADASR,  $\mathcal{K}=2$

that the sizes of the associated  $\mathcal{K}$ -ASRs are minimized. Every query from users that reside in the same bucket will have the same associated  $\mathcal{K}$ -ASR; therefore, the probability of a specific user issuing a query is  $1/\mathcal{K}$ . This is a *sufficient* condition to guarantee  $\mathcal{K}$ -anonymity. Even if the frequency of queries varies among users, an attacker will not be able to infer which user issued a particular query.

Static partitioning has three drawbacks: (i) Calculating the optimal partitioning is an NP-Hard problem [13]. (ii) Due to the fixed partitions, requests with anonymity degree larger than  $\mathcal{K}$  cannot be satisfied. Moreover, larger partitions would increase the average size of the  $\mathcal{K}$ -ASRs, affecting severely the processing cost. (iii) Finally, and more importantly, we assume mobile users, therefore static partitioning is not applicable.

In the following, we consider methods for on-the-fly  $\mathcal{K}$ -ASR construction. First, we explain the drawbacks of existing approaches. Then, we propose a novel method, HILBASR, which guarantees anonymity.

### 3.1 Drawbacks of Existing Approaches

The anonymization technique of Ref. [9] builds the  $\mathcal{K}$ -ASR of a given location using the PR-Quad-tree. When user  $u$  issues a query, the Quad-tree is traversed until a quadrant which contains  $u$  and less than  $\mathcal{K}-1$  other users is found. The parent of that quadrant is returned as the  $\mathcal{K}$ -ASR. A similar idea is used in Ref. [15]. We refer to this technique as QUADASR.

There are two drawbacks of QUADASR: (i) It may fail to achieve anonymity for some user distributions. Consider the example of Fig. 2. Each user resides in his own quadrant identified by its lower-left and upper-right coordinates. When any of the users  $u_1, u_2$  or  $u_3$  issues a query with degree of anonymity  $\mathcal{K}=3$ , the quadrant  $q_2 = ((0, 2), (2, 4))$  which encloses  $u_{1...3}$  will be returned as the  $\mathcal{K}$ -ASR. On the other hand, when the isolated user  $u_4$  issues a query with  $\mathcal{K}=3$ , the larger quadrant  $q_1 = ((0, 0), (4, 4))$  is returned. Note that if  $1 < \mathcal{K} \leq 3$ , the only reason to return quadrant  $q_1$  is that  $u_4$  issued a query. If an attacker knows the locations of the users in the area<sup>1</sup>, he will be able to pinpoint  $u_4$  as the query origin. (ii) A second drawback of QUADASR is that due to the non-uniform distribution of user locations,

<sup>1</sup>By triangulation, phone companies can estimate the location of a user within 50-300 meters, as required by the US authorities (E911).

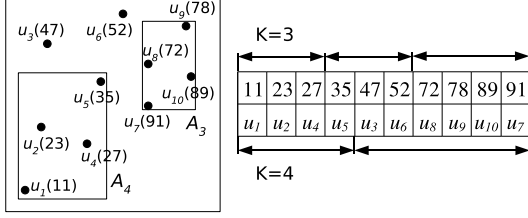


Figure 3. HILBASR,  $\mathcal{K}=3$  and  $\mathcal{K}=4$

the number of users enclosed by a  $\mathcal{K}$ -ASR may grow much larger than  $\mathcal{K}$  (as for  $u_4$  in the previous example). This corresponds to larger spatial extent of the  $\mathcal{K}$ -ASR, hence higher processing cost.

Both problems also exist if, instead of the Quad-tree, we use data partitioning spatial indices such as R-trees. Next, we describe our HILBASR method, which overcomes the above drawbacks.

### 3.2 The HILBASR Algorithm

Our HILBASR algorithm *guarantees* that the probability of identifying the query initiator is always bounded by  $1/\mathcal{K}$ , even if the attacker *knows* the locations of all users. HILBASR uses the Hilbert [6] ordering to group users into buckets of  $\mathcal{K}$ . The Hilbert space filling curve transforms the 2-D coordinates of each user into an 1-D value. With high probability, if two points are in close proximity in the 2-D space, they will also be close in the 1-D transformation.

To compute the  $\mathcal{K}$ -ASR, HILBASR employs an appropriate partitioning scheme that supports users' mobility and varying  $\mathcal{K}$  with minimal overhead. Intuitively, HILBASR computes and sorts the Hilbert values of all users. Then, the algorithm conceptually groups the sorted Hilbert values into  $\mathcal{K}$ -buckets that contain  $\mathcal{K}$  users, except from the last one which may contain up to  $2\mathcal{K}-1$  users. Let us consider a user  $u$  posing a query with anonymity degree  $\mathcal{K}$ . To compute the  $\mathcal{K}$ -ASR of user  $u$ , HILBASR computes the Hilbert value  $\mathcal{H}(u)$  of  $u$  and finds the  $\mathcal{K}$ -bucket that  $\mathcal{H}(u)$  belongs to. The minimum bounding box (MBB) of all the users in the  $\mathcal{K}$ -bucket corresponds to the  $\mathcal{K}$ -ASR.

For example, in Fig. 3, we illustrate the position of 10 users and their sorted Hilbert values. To compute the 3-ASR of user  $u_9$ , HILBASR first finds the  $\mathcal{K}$ -bucket which  $\mathcal{H}(u_9)$  belongs to. In our case, this consists of four users,  $u_8, u_9, u_{10}$  and  $u_7$ . Then, HILBASR returns the MBB of these users. Thus, the 3-ASR of user  $u_9$  is area  $A_3$ . Similarly, the 4-ASR of user  $u_5$  is area  $A_4$ .

Note that for a given snapshot, HILBASR returns the same  $\mathcal{K}$ -ASR for all users in the  $\mathcal{K}$ -bucket. This makes the  $\mathcal{K}$  users of the  $\mathcal{K}$ -bucket indistinguishable from each other. Thus, the probability of identifying the query initiator is bounded by  $1/\mathcal{K}$ .

As mentioned earlier, techniques that use fixed buckets

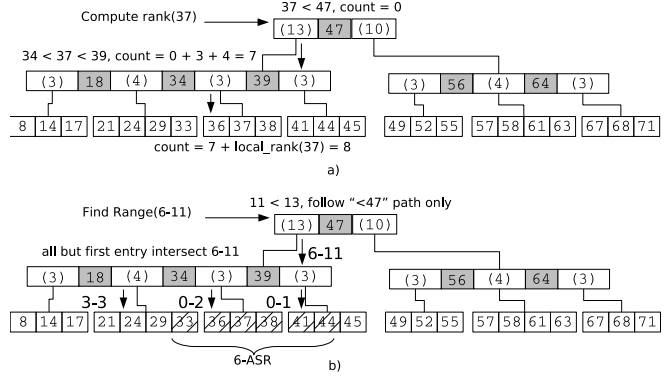


Figure 4. HILBASR with Annotated  $B^+$ -tree

suffer from lack of flexibility in accommodating queries with varying  $\mathcal{K}$ . Our method overcomes this limitation by avoiding to materialize the  $\mathcal{K}$ -buckets. Instead, it maintains a balanced sorting tree, which indexes the Hilbert values of users' locations. Let a user  $u$  initiate a query with anonymization degree  $\mathcal{K}_u$ . Our algorithm performs a search for  $\mathcal{H}(u)$  in the index and computes  $rank_u$ , which corresponds to the position of  $\mathcal{H}(u)$  in the in-order traversal of the tree. From  $rank_u$ , we calculate the *start* and *end* positions defining the  $\mathcal{K}$ -bucket which includes  $\mathcal{H}(u)$ , as:

$$\begin{aligned} start &= rank_u - (rank_u \bmod \mathcal{K}_u) \\ end &= start + \mathcal{K}_u - 1 \end{aligned} \quad (1)$$

The complexity of the in-order tree traversal is  $O(N)$ , where  $N$  is the number of indexed users. To compute  $rank_u$  efficiently, we use an annotated  $B^+$ -tree (similar to the aR-tree [16]), where each tree node stores the number of leaf nodes in each of its subtrees. Consider the example in Fig. 4. For each internal node entry  $e$ , we store the number of leaf entries that are rooted at  $e$ ; annotation counters are shown in brackets. Assume we want to determine a  $\mathcal{K}$ -ASR for entry 37, with  $\mathcal{K}=6$ . First, we compute the rank of entry 37 (Fig. 4a): we follow the path in the tree from root to the leaf that contains 37, and at each internal node we add to the rank value the sum of all counters in the node situated at the left of the followed pointer. At the leaf layer, we add to the rank the local rank value of key 37 in its leaf, and obtain rank 8 (ranks start from 0). Then, we calculate the bucket delimiters using Eq. (1), and obtain the interval [6..11]. Next (Fig. 4b), we perform a range search to locate the entries with ranks [6..11]. Observe that this operation uses the annotation, rather than the  $B^+$ -tree keys. Sub-ranges at each level are determined by splitting the initial range based on subtree sizes; the offset for the recursive call at entry  $e$  is determined as the initial start value minus the sum of counters of all entries in the node preceding  $e$ . The resulting  $\mathcal{K}$ -ASR is highlighted in the diagram.

The data structure is scalable, since the complexity of constructing the  $\mathcal{K}$ -ASR is  $O(\log N + \mathcal{K})$ , whereas search, insert and delete cost  $O(\log N)$ . Therefore, HILBASR is

applicable to large numbers of mobile users who update their position frequently and have varying requirements for the degree of anonymity  $\mathcal{K}$ .

#### 4. Distributed Anonymization in PRIVÉ

In this section, we introduce PRIVÉ, a distributed protocol which supports decentralized query anonymization using the HILBASR algorithm. PRIVÉ mimics the functionality of a  $B^+$ -tree in a distributed setting. Each mobile user  $u$  has an associated index entry consisting of an ID (e.g., IP address), and the Hilbert value  $\mathcal{H}(u)$  of his location as index key. A node (leaf or internal) in the  $B^+$ -tree, corresponds to a *cluster* of users, with size bounded between  $\alpha$  and  $3\alpha$ , where  $\alpha$  is a fixed system parameter. We use the terms *cluster* and *index node* interchangeably. The maximum cluster size is  $3\alpha$ , instead of the usual  $2\alpha$  for  $B^+$ -trees, to prevent cascading splits and merges (i.e., a split followed by a user departure), which are costly in the distributed environment.

Every user belongs to a leaf level cluster (level 0) of the  $B^+$ -tree, and the contents of each cluster are disjoint (see Fig. 5). The users of each cluster  $C$  elect a leader called *head*( $C$ ). The head (marked with an asterisk) handles all index operations on behalf of the users in the cluster. Cluster heads are recursively grouped to form a tree; therefore, they belong to multiple levels of the tree. We denote by  $C_u^i$ , the level  $i$  cluster which includes user  $u$ . In our example, user  $u_a$  is the head of cluster  $C_a^0$  at level 0. The same user is also the head of cluster  $C_a^1$  and  $C_a^2$ , therefore it belongs to every level of the tree. There is a single cluster at the top of the hierarchy, which we refer to as *top*. The cluster head of *top* is denoted by *root* ( $u_a$  in the example). In our protocol description, we use remote procedure call convention to specify interactions between users. The notation  $u.func(params)$  denotes the invocation of subroutine *func* with parameters *params* at user  $u$ .

Each cluster is associated with its *state* information. The state of a leaf level cluster consists of an ordered list of user key/address pairs and user coordinates, required to determine the  $\mathcal{K}$ -ASR bounding box. The state of an upper layer cluster with  $m$  elements consists of a list of  $m$  user addresses, separated by  $m - 1$  key values used to direct the search; the process is similar to a  $B^+$ -tree, with the role of memory pointers fulfilled by the IP addresses of users. Each internal node entry is annotated with a counter (depicted in parenthesis) representing the total number of users at the subtree under the entry. Only the head needs to know the state of the cluster. However, in our implementation, we replicate the state to every user within the cluster, in order to improve fault tolerance. In Section 5, we discuss the tradeoff between fault tolerance and maintenance cost. The PRIVÉ hierarchy has at most  $\log_\alpha N$  layers, where  $N$  is the total number of users. Since cluster size is bounded, and a

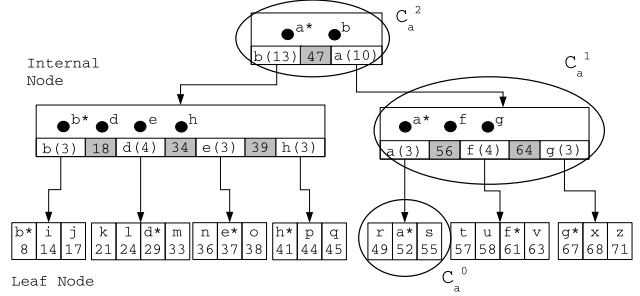


Figure 5. Distributed Index Structure,  $\alpha=2$

user may belong to at most one cluster at each level, there is an upper bound of  $O(\alpha \log_\alpha N)$  on the membership state stored at a user.

#### 4.1 Index Operations

The index supports four operations: *join*, *departure*, *relocation* and *anonymization*. We denote by  $\mathcal{K}$ -request a request for a  $\mathcal{K}$ -ASR with anonymization degree  $\mathcal{K}$ . We establish two performance metrics for PRIVÉ: (i) *latency*: the number of hops an index operation requires to complete. The latency is equal to the longest tree path followed as a result of the operation. Multiple paths may be followed in parallel during an operation. (ii) *communication cost*: the number of messages triggered by an index operation.

**Join.** User join corresponds to a  $B^+$ -tree insertion operation. Newly joining users authenticate at the certification server and receive the address of a user already inside the system. Without loss of generality, we assume that joining users know the root, since the root can be reached from any user in  $O(\log_\alpha N)$  cost. We stress that since we require an index structure with annotation, in order to determine the absolute ranks of users, all joins must occur through the root. To avoid overloading the root, we devise a load-balancing mechanism (Section 4.3). User join has  $O(\log_\alpha N)$  complexity in terms of latency and  $O(\log_\alpha N + \alpha)$  communication cost; the second term is for updating the cluster state in all the users of the affected cluster.

Consider user  $u_y$  with Hilbert value 46 that joins the index of Fig. 5:  $u_y$  contacts  $u_a$  (at the root level) who forwards the join request to  $u_b$  and updates  $u_b$ 's annotation counter in  $C_a^2$  to 14.  $u_b$  then forwards the request to  $u_h$ , whose annotation counter in  $C_b^1$  is updated to 4. Fig. 6(a) shows the join outcome. User join may trigger a cluster split, handled similarly to a  $B^+$ -tree node split; the head initiating the split leads one of the resulting clusters, and appoints a random initial cluster node to lead the other.

**Departure (informed).** User departure is similar to a  $B^+$ -tree deletion. The effect of deletion must be propagated to *root* to update the annotation counters. Deletion has  $O(\log_\alpha N)$  latency and  $O(\log_\alpha N + \alpha)$  communication cost. If cluster size decreases below  $\alpha$ , the cluster head trig-

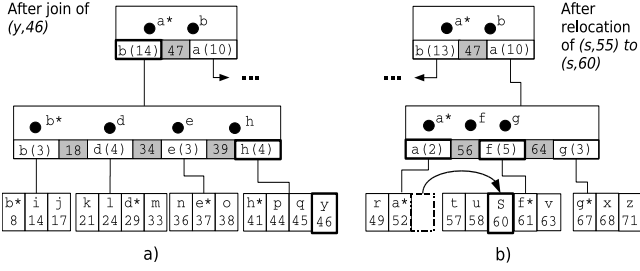


Figure 6. User Join and Relocation,  $\alpha=2$

gers a merge operation with the neighbor leaf-level cluster that has fewer members (to avoid a cascaded split). The head of the resulting cluster can be any of the initial heads, except if one of them is also head at the higher level. If so, it will be chosen as cluster leader, to minimize membership changes.

**Relocation.** User mobility is treated as an entry update, which in a  $B^+$ -tree translates into a deletion and an insertion. Since users are likely to change location often, we optimize this process by performing local reassignment of users to nearby clusters. Due to the good locality properties of Hilbert ordering, the number of clusters involved in relocation is likely to be small. Annotation counter updates are only performed by affected clusters; this way, updates are not propagated all the way to the root. The upper bound on relocation latency is  $O(\log_\alpha N)$ , but in most cases relocation only involves a few clusters, at the low layers of the index. The pseudocode for user relocation is given in Fig. 7.

---

**u.RelocateMyself()** /\*executed by moving user\*/

1. determine new key value  $\mathcal{H}_u = \text{Hilbert}(u.x, u.y)$
2. call  $\text{head}(C_u^0).\text{Relocate}(u, \mathcal{H}_u, 0)$

**u.Relocate(relocated\_user,  $\mathcal{H}, l$ )**

1. **if** ( $\mathcal{H}$  in indexed key range at level  $l$ )
2. **if** ( $l = 0$ )
3. add *relocated\_user* to leaf user list; **return**
4. **else**
5. let  $n$  be the next hop for  $\mathcal{H}$
6. call  $n.\text{Relocate}(relocated\_user, \mathcal{H}, l - 1)$
7. **else** call  $\text{head}(\text{parent}(C_u^l)).\text{Relocate}(relocated\_user, \mathcal{H}, l + 1)$

---

Figure 7. User Relocation

Consider user  $u_s$  from the Fig. 5 that relocates to a new position with Hilbert value 60. He forwards the request to  $u_a = \text{head}(C_s^0)$ .  $u_a$  cannot keep  $u_s$  within the same leaf entry, since the new value is outside the interval [49..55]. Since  $u_a = \text{head}(C_a^1)$ , with no additional message,  $u_a$  decides that  $u_s$  can be relocated to  $C_f^0$ , forwards the request to  $u_f$  and updates the annotation counters of  $u_a$  and  $u_f$  accordingly. Fig. 6(b) illustrates the relocation outcome.

**$\mathcal{K}$ -request.** A  $\mathcal{K}$ -request operation corresponds to the HILBASR algorithm described in Section 3. Consider the example in Fig. 8, where user  $u_m$  issues a  $\mathcal{K}$ -request with  $\mathcal{K}=6$ . The request follows the path:  $u_m \rightarrow u_d \rightarrow u_b \rightarrow u_a$  (solid arrows in Fig. 8(a)). The root  $u_a$  determines the  $\mathcal{K}$ -bucket (i.e.,  $start = 6, end = 11$ ) and sends a  $\mathcal{K}$ -ASR

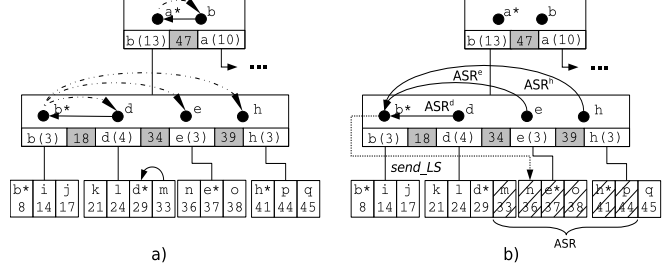


Figure 8.  $\mathcal{K}$ -request,  $\alpha=2, \mathcal{K}=6$

request to  $u_b$  (dotted arrows in Fig. 8(a)).  $u_b$  sends in parallel requests for partial  $\mathcal{K}$ -ASRs with ranges [6..6], [7..9] and [10..11] to  $u_d, u_e$  and  $u_h$ , respectively.  $u_b$ , which is the head of the lowest-layer cluster that completely covers the  $\mathcal{K}$ -bucket (shown hashed in Fig. 8(b)) collects the partial  $\mathcal{K}$ -ASRs and assembles the final query  $\mathcal{K}$ -ASR. Note that, the cluster head that covers the  $\mathcal{K}$ -bucket sustains the highest load among all other users involved in the query. This potential load imbalance issue is addressed in Section 4.3. A  $\mathcal{K}$ -request has  $O(\log_\alpha N) + O(\log_\alpha \mathcal{K})$  latency and  $O(\log_\alpha N) + O(\mathcal{K}/\alpha)$  cost. The pseudocode for  $\mathcal{K}$ -request is shown in Fig. 9.

---

**u.K-request()** /\*executed by query source\*/

1. determine key value  $\mathcal{H}_u = \text{Hilbert}(u.x, u.y)$
2. call  $\text{head}(C_u^0).\text{ForwardRequest}(\mathcal{H}_u, 0, 0)$

**u.ForwardRequest( $\mathcal{H}, count, l$ )**

1. **if** ( $l = 0$ )  $count = \text{rank}_{\mathcal{H}}$  in leaf entry
2. **else**  $count++ = \text{sum of annotation counters of keys } < \mathcal{H}$
3. **if** ( $u$  is root)
4. compute *start* and *end* using eq (1)
5.  $\mathcal{K}\text{-ASR} = \text{root}.\text{findMBR}(start, end, \text{root.height})$
6. **else** call  $\text{head}(C_u^{\text{level}+1}).\text{ForwardRequest}(\mathcal{H}, count, l + 1)$

**u.findMBR(*start*, *end*, *l*)**

1. **if** ( $l = 0$ ) /\*leaf level\*/
2. **return** MBR of members with local rank in [*start*, *end*]
3. find set of next hops  $U$  for range [*start*, *end*]
4.  $MBR = \emptyset$
5. **for**  $u' \in U$
6.  $MBR = MBR \cup u'.\text{findMBR}(start_{u'}, end_{u'}, l - 1)$
7. **return**  $MBR$

---

Figure 9.  $\mathcal{K}$ -request

**LBS communication.** Once the  $\mathcal{K}$ -ASR is determined, the anonymized query must be sent to LBS; this can be done by any user. Nevertheless, the communication with the LBS is expensive, since the response is a superset of the actual answer and may contain redundant information. This translates to financial cost, since mobile phone operators typically charge users by the amount of transferred data. Therefore, a fair solution would require the query initiator (i.e.,  $u_m$  in our example) to handle the communication with the LBS. However, this would reveal the identity of  $u_m$ . In our implementation, we adopt the following solution: once the  $\mathcal{K}$ -bucket is selected, we choose a random user from within the bucket to contact the LBS. Therefore, the probability of identifying  $u_m$  is still  $1/\mathcal{K}$ , and at the same time  $u_m$  is



more likely to handle his own request, than if a random user from the entire system were chosen. The potential unfairness among the users of the  $\mathcal{K}$ -bucket may be handled by a credit mechanism; this is outside the scope of the paper.

In Fig. 8(b), user  $u_n$  is randomly selected to communicate with the LBS. The selection process is integrated in the  $\mathcal{K}$ -request process as follows:  $u_b$  who is the  $\mathcal{K}$ -ASR-dominating user, generates a token and randomly passes it to only *one* of the partial  $\mathcal{K}$ -ASR requests to the lower layer. The process is repeated recursively down to the leaf layer. There,  $u_n$  is assigned to contact the LBS, and its IP address is sent back to  $u_b$  together with the partial  $\mathcal{K}$ -ASR. Then,  $u_b$  sends one additional message to inform  $u_n$  about the entire  $\mathcal{K}$ -ASR.  $u_n$  queries LBS, filters the result and returns the exact answer to the query initiator  $u_m$ .

## 4.2 Fault Tolerance

PRIVÉ implements a soft-state based mechanism to deal with user failures or disconnections without notification. Each cluster leader sends periodically (i.e., every  $\delta t$  seconds) a *membership\_update* message to all cluster members. The message contains the membership list of the current cluster  $C$  and that of  $parent(C)$ . Cluster members respond to these messages; if a cluster member does not respond to two consecutive messages, it is considered disconnected and removed from the cluster. The change is broadcast by the cluster head to the remaining cluster members.

If a non-head cluster member  $u$  does not receive a *membership\_update* from its head for a  $2\delta t$  period, it initiates a *leader election* process. Alternatively, when  $u$  attempts to initiate a operation such as query or relocation, and cannot contact the cluster head for two consecutive attempts, it triggers the leader election protocol without waiting for the timer to expire.  $u$  checks the membership it had at the last update, and chooses as leader (i.e., *new\_head*) the user with the smallest identifier. It then sends a *transfer\_head* message to *new\_head*, which in turn sends a membership update message to all cluster users and also contacts  $head(parent(C))$  to notify for the change in leadership. *new\_head* will replace the old head in all layers where the latter was leader before disconnection.

## 4.3 Load Balancing

The hierarchical structure can cause significant differences between the load sustained by cluster heads and ordinary cluster members, as well as among cluster heads at different layers of the hierarchy<sup>2</sup>. To alleviate the inherent imbalance, we propose a cluster head *rotation* mechanism, where users take turn in fulfilling the cluster head role. Since the promotion to cluster head translates into

<sup>2</sup>The issue of load imbalance due to LBS communication is not a result of internal index operations, and has been addressed in Section 4.1.

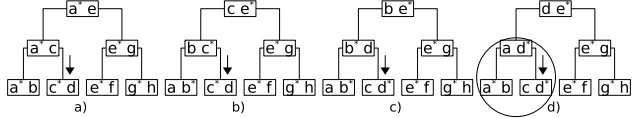


Figure 10. Load balancing mechanism

presence at a higher layer of the hierarchy, the rotation also ensures that users equally share the load at different layers.

Rotation is triggered when a node reaches a certain load threshold, denoted by *load unit*. In wireless devices the communication cost is dominant. It is also important from the user's perspective, since mobile phone operators charge by the amount of transferred data. Therefore, in PRIVÉ the load is best represented by the number of messages sent and received by the user.

When a user  $u$  reaches one load unit, it triggers a head rotation in all the clusters it currently heads, starting with its highest layer. For each node along the path to its level 0 cluster, the member with the least load is appointed as new head. Note that, since  $u$  stores the membership state about all clusters it belongs to at different layers, the appointment of a new leader can be done directly by  $u$ , without the need for a complex protocol or additional messages. Choosing the cluster member with the lowest load prevents the newly appointed head to start a fresh rotation soon after promotion.

Fig. 10 illustrates the rotation mechanism. For simplicity, all clusters have size 2; cluster heads are marked with an asterisk. Assume all queries originate at user  $u_d$  (marked with an arrow) with  $\mathcal{K}=4$ . After user  $u_a$  reaches one load unit, it hands over the root role to user  $u_e$  (at layer 2) from the right-hand subtree. Also, at layer 1, user  $u_c$  becomes the head and is automatically promoted to layer 2. Similarly, at layer 0, user  $u_b$  becomes the head and is promoted to layer 1; the result is shown in Fig. 10(b)). Next, node  $u_c$  reaches its load unit, because more requests pass through it (it must inject queries and collect partial  $\mathcal{K}$ -ASRs). User  $u_c$  triggers a rotation, but at layer 1, appointing  $u_b$  as cluster head (see Fig. 10(c)). Subsequently, user  $u_b$  may be the next one to reach the load threshold, and start a new rotation in the left subtree. Observe that at step (d), the left subtree has already performed a complete rotation round, while the right subtree has only performed one rotation. Hence, our rotation mechanism alleviates hotspots (an entire subtree shares the load generated by user  $u_d$ ) and at the same time provides a degree of *fairness*, not allowing a localized hotspot to affect a large partition of the index.

The granularity of load unit choice is important in practice, in order to achieve a good tradeoff between load balancing and cost, since a rotation may incur a number of messages as large as  $O(\alpha \log_\alpha N)$ . We further discuss this issue in Section 5.

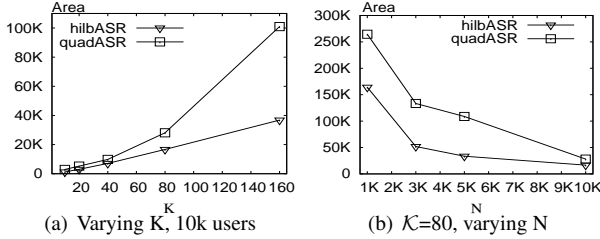


Figure 11.  $\mathcal{K}$ -ASR area

## 5. Experimental Evaluation

We present the experimental evaluation of the HILBASR anonymization technique and PRIVÉ anonymization system. To evaluate PRIVÉ, we implemented an event-driven packet-level simulator in C++. Since we are mostly interested in the overlay-layer performance, we consider a full mesh topology with lossless 500ms round-trip time links between any pair of users. Our workload consists of user locations and movement patterns, and is created using IAPG [5] which models user movement on public road networks. For user movement, we consider velocities ranging from 18 to 68km/h. We present our results for a data set consisting of the San Francisco bay area (Fig. 12(a)), with number of users  $N$  varying from 100 to 10000. We varied the anonymization degree  $\mathcal{K}$  from 10 to 160. We considered both uniform and Zipf distribution of queries over the users.

In the first experiment, we compare HILBASR against QUADASR in terms of spatial extent (i.e., area) of the generated  $\mathcal{K}$ -ASR. We consider a time snapshot of user locations and generate a number of queries equal to population size  $N$ . Each query is initiated by a random user. Fig. 11(a) shows the results for varying  $\mathcal{K}$  and 10K users. HILBASR is better in all cases. A similar trend was observed for  $\mathcal{K}$ -ASR perimeter. In Fig. 11(b) we set  $\mathcal{K}=80$  and vary the number of users. The decrease in  $\mathcal{K}$ -ASR size with increasing  $N$  is explained by the higher user density in the same dataspace (i.e.,  $\mathcal{K}$  users can be located in a smaller region). HILBASR again outperforms QUADASR in terms of  $\mathcal{K}$ -ASR extent, which translates into reduced execution cost at the LBS and communication cost between the LBS and the user. Furthermore, HILBASR guarantees anonymity in all cases. Therefore HILBASR is superior to QUADASR in all aspects (Ref. [12] gives more detailed experimental results).

The following experiments evaluate the performance of PRIVÉ for user join/departure,  $\mathcal{K}$ -ASR construction, user mobility, fault-tolerance and load-balancing.

**Join and Departure.** In a system with  $N$  users, we perform  $0.1N$  random user joins, followed by  $0.1N$  random user departures. Fig. 12(b) shows the join latency measured as hop count from the time a user issues a join request until it receives a join response message from its leaf-level head. We observe that the latency is lower than the theoretical  $1 + \log_\alpha N$ , as a user may appear in multiple levels

and can avoid sending redundant messages to himself. The communication cost (i.e., total messages) per join and departure operation (Fig. 12(c)) varies linearly with  $\alpha$ , since every join/departure translates into a *membership\_update* broadcast message within one leaf-level cluster. Note the role of  $\alpha$  in the latency-cost tradeoff: an increase of  $\alpha$  decreases latency as  $\log_\alpha N$ , but triggers a linear cost increase in membership notification. A larger  $\alpha$  also increases the cost of periodic cluster membership maintenance.

**$\mathcal{K}$ -request.** Fig. 12(d) and 12(e) show the variation of  $\mathcal{K}$ -request latency and communication cost with  $\alpha$ ,  $\mathcal{K}=40$ . Larger  $\alpha$  decreases the latency as the height of the index decreases. The communication cost also decreases, as fewer leaf-level cluster heads need to be contacted to build the  $\mathcal{K}$ -ASR. However, the choice of  $\alpha$  cannot grow very large from index maintenance considerations. Fig. 12(f) and 12(g) show the latency and communication cost variation with anonymization degree  $\mathcal{K}$ ,  $\alpha = 5$ . Latency is only marginally affected by  $\mathcal{K}$  (the dominant factor in latency is  $\log_\alpha N$ , since in practice  $\mathcal{K} \ll N$ ), while the communication cost grows linearly with  $\mathcal{K}$ . The percentage of the user population involved in answering a single  $\mathcal{K}$ -request operation is shown in Fig. 12(h) and 12(i). For small  $N$  values, at most 2% of all users are needed to answer a  $\mathcal{K}$ -request, while for larger  $N$ , less than 0.5% of the users are required.

**Relocation.** PRIVÉ addresses user mobility by using an index update algorithm that attempts to solve relocation at the lowest levels of the hierarchy, in order to reduce both latency and communication cost. In our simulated scenario, we consider 10000 users across 20 consecutive time frames, with half of the indexed users moving at each time frame according to a pattern generated by IAPG [5]. We consider three velocities: 68, 40 and 18km/h. Fig. 12(j) and 12(k) show that relocation is efficiently handled: for the moderate  $\alpha = 10$  value, the relocation is done on average in 2.5 hops for fast-moving users and 1.5 hops for slow-moving users. The dominant communication cost is that of the membership change propagation; for  $\alpha = 10$  this cost is roughly a quarter that of an index deletion followed by insertion for the 68km/h case, and 1/8 for 18km/h. Fig 12(l) shows the frequency of relocations completed at various levels of the hierarchy for a 6-level,  $\alpha = 3$ , 10000 users system. Most relocations are solved at the low levels of the hierarchy: for slow movement, 70% are solved at the leaf level and 86% at levels 0 and 1; for fast movement, 32% of relocations are completed at the leaf level, 63% at levels 0 and 1, and 86% at levels 0, 1 or 2.

**Fault-tolerance.** Starting with a system having correct cluster membership system, we fail simultaneously 10, 20 or 30% of the nodes. We use maintenance timer values of 30 seconds for refreshing cluster membership and 60 seconds for purging a failed member. Fig. 12(m) shows the evolution of membership state correctness over time (1 rep-

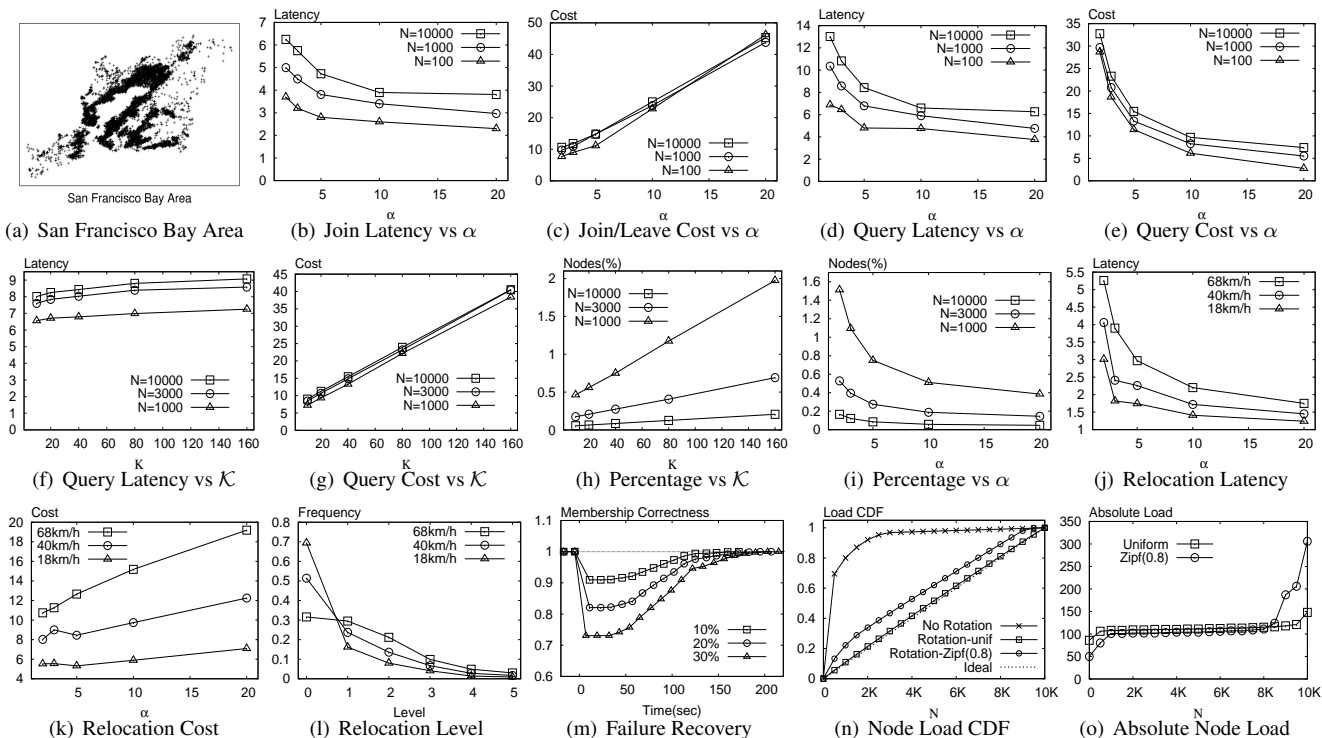


Figure 12. PRIVÉ Experimental Evaluation

resents completely correct state). The system recovers to a correct state within 3 purge cycles (138 sec) for 10% failure and 4 purge cycles (197 sec) for 30% failure.

**Load-balancing.** We measured the load incurred by each user for a 10000 users system,  $\alpha = 5$ ,  $\mathcal{K}=80$ , *load unit* = 200 messages per  $\mathcal{K}$  and a simulated time of 1 hour, during which an average of 8 queries/user were generated. We considered both uniform and skewed (Zipf 0.8) query source distribution. Fig. 12(n) shows the cumulative distribution function (CDF) of *sorted* user loads. The load is highly unbalanced if no rotation is performed, with 10% of users sustaining more than 80% of the load. With rotation, for uniform query distribution, the load is close to the ideal one (i.e., diagonal line). For skewed query distribution, most of the users share equal load, while part of the users (roughly 10%) share a slightly higher load, as dictated by the fairness requirement discussed in Section 4.3. This is illustrated better in Fig. 12(o) which shows the absolute load of each user.

## 6. Related Work

$\mathcal{K}$ -anonymity was first discussed in relational databases where published statistical data (e.g., census, medical) should not be linked to specific persons. Samarati and Sweeney [17, 19] proposed the following definition: A relation satisfies  $\mathcal{K}$ -anonymity if every tuple in the relation is indistinguishable from at least  $\mathcal{K}-1$  other tuples with respect to every set of quasi-identifier attributes. Quasi-identifiers

are sets of attributes (e.g., date of birth, gender, zip code) which can be linked to publicly available data to uniquely identify individuals. Two techniques are used to transform a relation to a  $\mathcal{K}$ -anonymized one: *Suppression*, where some of the attributes or tuples are removed and *generalization*, which involves replacing specific values (e.g., phone number) with more general ones (e.g., only area code). Both techniques result to information loss. Ref. [4] and Ref. [13] discuss efficient algorithms for anonymizing an entire relation while preserving as much information as possible. In Ref. [20] the authors consider the case where each individual requires a different degree  $\mathcal{K}$  of anonymity, while Aggarwal [1] shows that anonymizing a high-dimensional relation results to unacceptable loss of information due to the dimensionality curse. Finally, Machanavajjhala et al. [14] propose  $\ell$ -diversity, an anonymization method under the assumption that the attacker has domain-specific knowledge.

$\mathcal{K}$ -anonymity has also been adopted in the LBS domain: in Ref. [8, 9], the location of the user is concealed by constructing an *Anonymizing Spatial Region* ( $\mathcal{K}$ -ASR) which encloses the locations of the query source and  $\mathcal{K}-1$  additional users. However, their methods of  $\mathcal{K}$ -ASR construction are inefficient, they do not discuss how queries are executed, and anonymization may fail for some data distributions. Ref. [12, 15] extend further these ideas and present a framework for the entire process of anonymization and query processing at the LBS. Nevertheless, all previous methods assume a centralized anonymizer, which may be

come a bottleneck or a security threat.

Key and range search has been studied extensively in distributed environments. Several structured Peer-to-Peer systems (e.g, Chord [18]) support distributed key search with  $O(\log N)$  complexity. Closer to our work, the P-tree [7] supports range queries by embedding a B<sup>+</sup>-tree on top of an overlay network. No global index is maintained; instead each node maintains its own B<sup>+</sup>-tree-like structure. BATON [11] also addresses range queries, by embedding a balanced binary tree onto an overlay network. It uses additional cross-links to prevent hotspots, and achieves  $O(\log N)$  complexity for both search and maintenance. The drawback of all these systems is that they cannot support efficiently node annotation. However, annotation is essential for our HILBASR algorithm. PRIVÉ, on the other hand, supports search, maintenance and annotation in  $O(\log N)$ .

Hierarchical clustering in distributed environments has been an active research topic in recent years. In Ref. [3], a hierarchical-clustering routing protocol for wireless networks is presented. The NICE project [2] proposes a scalable application-layer multicast protocol, based on delivery trees built on top of a hierarchically connected control topology. Nodes participating in a multicast group are organized into a multi-layer hierarchy of clusters with bounded size. NICE trees obtain delays in the order of  $O(\log N)$ , where  $N$  is the size of the multicast group, and there is an upper bound of  $O(\log N)$  in terms of control state maintained per node. PRIVÉ also uses hierarchical clustering of mobile users, but the requirements of total ordering and annotation impose particular challenges that have not been addressed by existing research.

## 7. Conclusions

In this paper we introduced PRIVÉ, a distributed system for query anonymization in LBSs. In PRIVÉ, mobile users wishing to issue location-based queries, organize themselves into a hierarchical overlay network and anonymize queries in a fully decentralized fashion. PRIVÉ supports our HILBASR anonymization technique, which guarantees anonymity under any user distribution. We show experimentally that our system is efficient, scalable, fault tolerant and achieves load balancing.

LBSs for mobile users are already a reality in Japan, where most of the new mobile phones contain a positioning device, and high-speed wireless networks are common. As such applications gain popularity, privacy and confidentiality concerns are expected to rise. In the future, we plan to address anonymity of continuous spatial queries, and extend our algorithm to trajectories, as opposed to points. We also plan to deploy PRIVÉ in infrastructure-less environments, such as ad-hoc wireless networks (Wi-Fi, Bluetooth), without point-to-point links between all users.

## References

- [1] C. C. Aggarwal. On k-Anonymity and the Curse of Dimensionality. In *VLDB*, pages 901–909, 2005.
- [2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, 2002.
- [3] S. Banerjee and S. Khuller. A Clustering Scheme for Hierarchical Control in Wireless Networks. In *Proc. of IEEE INFOCOM*, 2001.
- [4] R. Bayardo and R. Agrawal. Data Privacy through Optimal k-Anonymization. In *Proc. of ICDE*, pages 217–228, 2005.
- [5] T. Brinkhoff. A framework for generating network-based moving objects. *Geoinformatica*, 6(2):153–180, 2002.
- [6] A. R. Butz. Alternative Algorithm for Hilbert’s Space-Filling Curve. *IEEE Trans. on Computers*, pages 424–426, April 1971.
- [7] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying P2P Networks using P-trees. In *Proc. of WebDB*, pages 25–30, 2004.
- [8] B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *Proc. of ICDCS*, pages 620–629, 2005.
- [9] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proc. of USENIX MobiSys*, 2003.
- [10] H. Hu and D. L. Lee. Range Nearest-Neighbor Query. *IEEE TKDE*, 18(1):78–91, 2006.
- [11] H. V. Jagadish, B. C. Ooi, and Q. H. Vu. BATON: a Balanced Tree Structure for P2P networks. In *Proc. of VLDB*, 2005.
- [12] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving Anonymity in Location Based Services. Technical Report TRB6/06, National University of Singapore, 2006.
- [13] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient Full-Domain K-Anonymity. In *Proc. of ACM SIGMOD*, pages 49–60, 2005.
- [14] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkatasubramanian. l-Diversity: Privacy Beyond k-Anonymity. In *Proc. of ICDE*, 2006.
- [15] M. F. Mokbel, C. Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *Proc. of VLDB*, 2006.
- [16] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proc. of SSTD*, pages 443–459, 2001.
- [17] P. Samarati. Protecting Respondents’ Identities in Microdata Release. *IEEE TKDE*, 13(6):1010–1027, 2001.
- [18] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.
- [19] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *Int. J. of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.
- [20] Y. Tao and X. Xiao. Personalized Privacy Preservation. In *Proc. of ACM SIGMOD*, 2006.