Privately Querying Location-based Services with SybilQuery

Pravin Shankar, Vinod Ganapathy, Liviu Iftode
Rutgers University

{spravin,vinodg,iftode}@cs.rutgers.edu

Abstract

To usefully query a location-based service, a mobile device must typically present its own location in its query to the server. This may not be acceptable to clients that wish to protect the privacy of their location. Much prior work has addressed the problem of protecting client privacy in such location-based queries using k-anonymity. In such schemes, the location-based server is unable to distinguish a querying client from a group of k clients.

However, prior work on k-anonymity-based schemes has typically required the use of an anonymizer—a proxy that intercepts and modifies client queries so as to achieve k-anonymity. The centralized nature of anonymizers makes them a single point of failure. Moreover, in the presence of mobile clients, anonymizers must be implemented so as to avoid correlation attacks, where an adversary compromises client location using that client's queries from multiple locations. Alternatives that eliminate the anonymizer either rely on the participation of k other peers, thus making the system reliant on these peers, or are based upon computationally-expensive cryptographic protocols that present scalability problems.

This paper presents the design and implementation of SybilQuery, a fully decentralized and autonomous k-anonymity-based scheme to privately query location-based services. SybilQuery is a client-side tool that generates $k-1$ Sybil queries for each query by the client. The location-based server is presented with a set of k queries and is unable to distinguish between the client's query and the Sybil queries, thereby achieving k-anonymity. We tested our implementation of SybilQuery on real mobility traces of approximately 500 cabs in the San Francisco Bay area. Our experiments show that SybilQuery can efficiently generate Sybil queries and that these queries are indistinguishable from real queries.

1 Introduction

Modern mobile devices contain global positioning systems (GPS) that let these devices precisely determine their location. Coupled with Internet connectivity via 3G and WiFi, GPS allows these mobile devices to make *location-based queries*. A querying client device sends its location to a server, which responds with answers specific to the client's location. Examples of such queries include real-time traffic queries, *e.g.,* "how congested is the route from my home to my office?" and points-of-interest queries, *e.g.,* "what are the restaurants close to my current location?" Several modern devices, such as the iPhone, are pre-installed with query interfaces to location-based services, such as Google Maps.

However, clients that wish to protect their privacy may not wish to reveal their current location to a location-based server (LBS). Client location can be misused by LBSs in ways that range from serious breaches of privacy, such as spying on the client's daily commuting patterns, to simple annoyances, such as flooding the client with location-based spam and advertising in response to a query. Recent evidence suggests that privacy concerns can deter the widespread use of LBSs. For example, EZPass RFID tags have seen a low adoption rate due to privacy concerns among drivers [8]. Similarly, the New York City cab drivers' group called a strike upon the introduction of a plan to fit all cabs with GPS tracking devices [29]. Such concerns motivate the need for techniques that protect the privacy of client location from LBSs.

Several prior techniques [2, 5, 10, 11, 13, 14, 16, 17, 19, 27] to protect the privacy of client location fall under the broad umbrella of k-anonymity [32, 34]. In a k-anonymity-based scheme, an LBS is unable to distinguish a querying client from a group of at least k clients, where k is a security parameter. Spatial cloaking [16] is one such k-anonymity-based scheme that sends cloaked regions, such as rectangular blocks, as location-based queries. These cloaked regions are chosen so that there are at least k clients in the region. The LBS responds with results for the entire region, which must be filtered to obtain the desired query result. For example, in response to a spatially-cloaked query on restaurants closest to a client, the LBS would respond with all restaurants in the cloaked region.

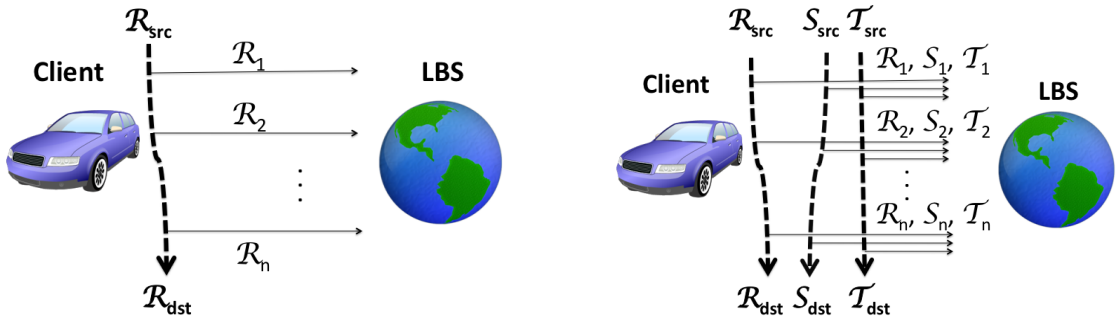
Unfortunately, most previously-proposed k-anonymity-based schemes require a trusted third party, called an *anonymizer*. The anonymizer accepts and forwards queries from a client to an LBS and ensures the k-anonymity property. For example, to implement spatial cloaking, the anonymizer would generate and send a cloaked region (containing at least k clients, including the querying client) to the server. With mobile clients, such as cell phones and vehicles with on-board GPS systems, the anonymizer must periodically regenerate queries (such as spatially cloaked regions) to ensure that the k-anonymity property continues to hold. These queries must be generated carefully to avoid *correlation attacks* [12], whereby a malicious LBS breaks k-anonymity by correlating multiple queries received from the anonymizer. In addition, because anonymizers must compute cloaked regions for each client and process the LBS' query results for these regions, they can cause performance and scalability bottlenecks. Recently proposed alternatives to eliminate anonymizers include peer-to-peer techniques [5, 13, 14] and protocols based on private information retrieval (PIR) [12]. Peer-to-peer techniques rely on the participation of k peers to ensure k-anonymity, in a manner akin to Crowds [30]. However, reliance on peers restricts the autonomy of such systems. Techniques based on PIR provide strong cryptographic guarantees, but are currently computationally expensive.

This paper presents the design and implementation of *SybilQuery*, a fully decentralized and autonomous k-anonymity-based system for location-based queries. SybilQuery is a client-side tool that operates by generating $k-1$ *Sybil queries* for each location-based query by a client. The Sybil queries contain locations that resemble the client's actual location. For example, the Sybil queries for a real query from a busy downtown area will also be from areas with similar traffic conditions. SybilQuery sends these k queries to the LBS, which is unable to identify the original query from the Sybil queries, thereby ensuring k-anonymity of the client's location. SybilQuery's design offers several advantages:

(**a**) **Performance.** SybilQuery eliminates the need for anonymizing servers, and the scalability and performance bottlenecks associated with centralized designs. For example, choosing higher values of k for better privacy leads to greater computational overheads at the anonymizer [2, 27]. SybilQuery's decentralized design offloads the computations needed to achieve k-anonymity to the clients themselves. Our experiments show that the costs of generating Sybil queries at the client are negligible.

(**b**) **Autonomy.** In contrast to prior work where queries generated by an anonymizer or using peer-to-peer techniques depend on *other* clients of the LBS, SybilQuery generates Sybil queries autonomously. For example, cloaked regions generated by spatial cloaking techniques [16] must have at least k clients; this in turn depends on the geographic distribution of clients [20]. Mobile clients further complicate the design of the anonymizer, because it must choose cloaked regions so as to prevent correlation attacks [12].

(**c**) **Ease of deployment.** A decentralized and autonomous design lends itself to easy deployment. Clients do not have to rely on service providers to deploy anonymizers or on peer participation in order to achieve k-anonymity. Unlike PIR-based techniques, SybilQuery does not require any changes to the LBS and only requires minor modifications to the querying client. We have integrated SybilQuery with LBSs such as Google Maps, Microsoft Live Maps and Yahoo! Maps.



(a) **Without SybilQuery, the client sends its actual location to the LBS as it moves along path $\mathcal{R}$.**

(b) **With SybilQuery, the client sends both its actual location along path $\mathcal{R}$ and along Sybil paths $\mathcal{S}$ and $\mathcal{T}$.**

Figure 1: Querying an LBS (a) without SybilQuery; and (b) using SybilQuery with k=3.

The reader may question the practicality of sending multiple queries to an LBS, which burdens the LBS with having to process k queries instead of one. While this is indeed the case, existing approaches place an *even greater burden* on the LBS. Spatial cloaking and peer-to-peer techniques require the LBS to process the *entire cloaked region* consisting of k clients, instead of k *points*, as in SybilQuery. Similarly, PIR-based protocols are computationally expensive. For instance, the PIR-based location privacy system of Ghinita *et al.* [12] imposes a computational cost of $O(n)$ at the server, where n is the size of the location database that the LBS must query (in addition to communication costs of $O(\sqrt{n})$). In contrast, SybilQuery ensures k-anonymity while imposing the least possible additional burden on the LBS (linear in k).

We have implemented a prototype of SybilQuery for vehicular networking applications. The input to SybilQuery is a path to be followed by a vehicle along which the vehicle may issue several queries to LBSs. SybilQuery outputs $k-1$ Sybil paths that statistically resemble the input path. The vehicle sends k queries when it accesses an LBS—its actual location, as well as $k-1$ waypoints derived from the Sybil paths (see Figure 1). We evaluated this prototype using real mobility traces of approximately 500 cabs in San Francisco [3]. Our experiments, which included both a user study with 15 volunteers and a quantitative evaluation of privacy, confirmed that Sybil paths produced by SybilQuery closely resemble real paths. For example, our user study showed that attempts at differentiating real paths from Sybil paths were no better than random guessing. In addition, we also evaluated the performance of SybilQuery, and compared its performance to spatial cloaking.

The rest of this paper presents the location privacy problem and threat model (Section 2), the design (Section 3) and implementation (Section 4) of the SybilQuery prototype and an evaluation on real traffic traces (Section 5). Section 6 presents related work in location privacy and Section 7 concludes.

2 Problem Definition and Assumptions

Abstractly, an LBS is a database that stores a set of tuples $< \ell, v >$, where ℓ's are geographic locations, *e.g.,* represented using latitudes/longitudes, and each v denotes value(s) associated with location ℓ, *e.g.,* restaurants or current traffic conditions at ℓ. We consider a model in which mobile clients periodically query the LBS as they move from a source to a destination. Each query contains the current location of the client, as shown in Figure 1(a), and the LBS returns the set of values associated with that location.

The problem of privately querying an LBS is for a client to issue location-based queries without revealing its current location to an adversary. The SybilQuery system aims to achieve this goal by sending $k-1$ *Sybil queries* along with each real query by the client. These queries contain locations along $k-1$ *Sybil paths*,

which are synthetically generated by SybilQuery, as shown in Figure 1(b). To prevent an adversary from identifying the client's actual location, SybilQuery must generate Sybil paths (and thereby, Sybil queries) that closely resemble real paths.

Our threat model assumes a powerful adversary, who can observe the contents of all queries made by the client. In fact, the LBS itself could be adversarial and could compromise client privacy by learning the client's past and current locations. In particular, we make the following assumptions about the adversary:

(**a**) **Adversary cannot access client's identifiers.** A fully-constructed client query to the LBS logically consists of three entities: (i) *a network identifier*, such as the IP address of the client's device that issues the location-based query; (ii) *an explicit user identifier*, such as the user's registered account name at the LBS or an HTTP cookie; and (iii) *the client's location*, as may be obtained by the client using GPS.

As in prior work on private location-based queries [2, 5, 10, 11, 12, 13, 14, 16, 17, 19, 27], we assume that the adversary does not have access to the client's network identifiers. Without this assumption, the adversary may be able to compromise client privacy, *e.g.,* using a reverse geo-lookup of the client's network identifiers. To hide network identifiers from the adversary, we assume that the client's service provider (*e.g.,* the cellular provider) and local access points, which know the client's current location, do not collude with the adversary (*e.g.,* the LBS). In particular, because the service provider and local access points already know the client's location, we assume that they are not adversarial. The client could optionally use an anonymizing network, such as Tor [35], although the network latencies imposed by Tor may be prohibitive for some applications (*e.g.,* see Saint-Jean *et al.* [31]); we do not use Tor in our prototype.

We also assume that the adversary (and in particular, the LBS) does not require the user to send explicit user identifiers in order to use the service. This is indeed a practical assumption (and is also a standard assumption in prior work); several popular LBSs, such as Google Maps, Microsoft Live Maps and Yahoo! Maps, do not require a registered user or HTTP cookies for a client to usefully query them. Without this assumption, an adversary with background knowledge about an individual, *e.g.,* the residential ZIP code of the querying client, could likely identify queries sent by that client.

Thus, we assume that the adversary only has access to an ordered list of locations from client queries. *The goal of the SybilQuery system is to hide the true location of the client from the adversary.* The client's location is privacy-critical, because an adversary could use the client's location to first identify the client's postal address and then use freely-available search engines to identify the client [21].

(**b**) **Adversary could be active or passive.** A passive adversary can observe the contents of all location-based queries as well as the responses received from the LBS. On the other hand, an active adversary can modify responses from the LBS as a way to compromise client privacy. For example, consider an adversarial LBS that reports real-time traffic information. Upon receiving a request for traffic conditions at a particular location ℓ, the LBS could return false information, *e.g.,* by reporting heavy traffic at ℓ. This response from the LBS may cause the client to behave differently, *e.g.,* take a detour to avoid heavy traffic. The SybilQuery system must therefore respond appropriately, *e.g.,* by appropriately modifying the Sybil queries to statistically match the real query.

The client could optionally prevent arbitrary third-parties from reading and modifying queries (and responses) sent to/from the LBS using an encryption scheme. However, this scheme will not protect the client from an adversarial LBS, which still has access to the location sent in the query.

(**c**) **Statistical background knowledge.** The adversary could have access to global statistical knowledge, such as the average traffic densities at different locations at different times during the day. This information can potentially be used to compromise client privacy.

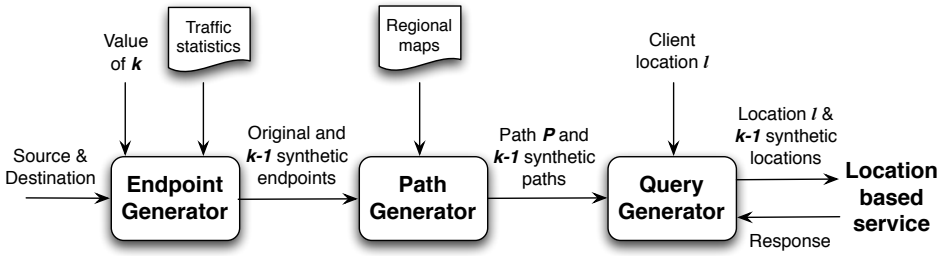


Figure 2: Basic design of SybilQuery. Design enhancements are discussed in Section 3.1.

The SybilQuery prototype can currently defend against several such statistical background knowledge attacks, particularly those that use knowledge about regional traffic. However, because we cannot anticipate all such attacks, we have designed SybilQuery to support an extensible architecture. In particular, it can incorporate several statistical features in its algorithm to generate Sybil paths, thereby making the system robust to future attacks.

SybilQuery however *cannot* defend against adversarial LBSs that have targetted background information about a specific client. If the LBS knows a particular client's daily commuting patterns or knows specific locations that the client visits, it can differentiate between real queries and Sybil queries sent by that client with high probability. For example, if the LBS knows a client's residential address (R) and his work address (W), it can identify the real path followed by the client by searching for a path that starts at R and ends at W.

3 Hiding Client Location using Sybil Queries

The SybilQuery system presents an interface akin to existing navigation systems. A user enters the address of the source and destination of a trip, and the system outputs a path P for the user to follow. Additionally, it requires as input a security parameter k, that it uses to generate $k-1$ Sybil paths. Each path is represented by the system as a sequence of waypoints enroute from the source to the destination. As depicted in Figure 2, SybilQuery consists of three modules: an endpoint generator, a path generator, and a query generator. We describe each of these modules in detail below.

The endpoint generator produces $k-1$ synthetic start and end points that statistically resemble the source and destination input by the user. To do so, the endpoint generator requires a database of regional traffic statistics. This database should reflect historic traffic trends at various geographic locations in the area (*i.e.,* it is not a database of real-time traffic conditions). The high level idea is that the endpoint generator processes this database to identify clusters of locations that share similar features, such as traffic density (details of specific features used in our implementation are deferred to Section 4). This process is a one-time activity and suffices to produce synthetic start and end points for multiple trips, *e.g.,* until the database is refreshed with newer statistics. The endpoint generator chooses synthetic start and end points from the clusters to which the actual source and destination belong. In doing so, it also ensures that the Euclidean distance between the source and destination of the actual path are within a threshold of the Euclidean distance between the synthetic start and end points.

The path generator uses the k start and end points, including the actual source and destination, to produce k paths, each represented as a sequence of waypoints. In generating paths, it consults a database of regional maps (as do existing on-board navigation systems). In fact, the path generator may be implemented using existing navigation systems, such as Google Maps, Yahoo! Maps, and on-board GPS devices, because

their interfaces are similar. Both the endpoint generator and path generator need to be invoked only once per trip.

The query generator is triggered when the user queries the LBS with his current location ℓ. Intuitively, it simulates the motion of users along the $k-1$ Sybil paths and generates $k-1$ Sybil locations. In its simplest form, the query generator simply computes the offset of ℓ from the source of the user's path P, and applies similar offsets to the sources of the Sybil paths to produce $k-1$ Sybil locations. However, it can also use current traffic conditions to more accurately simulate user movement along Sybil paths (*e.g.,* simulate slower movement if traffic is congested at one of the Sybil locations).

The modular design of SybilQuery offers several advantages. It lends itself to easy deployment, even with legacy devices. The endpoint generator is a standalone component that can possibly be installed as a user-level application either on the user's desktop or on his mobile device. The path generator, as noted above, can be implemented using off-the-shelf navigation systems. Using a navigation system enables robust path generation based on region maps, which allows SybilQuery to generate Sybil paths that respect features such as one-ways and road closures. It also allows SybilQuery to robustly handle detours from the path P suggested by the navigation system. Upon a detour, SybilQuery can simply trigger the path generator to produce a new path to the destination. Because paths are generated independent of each other, SybilQuery can also simulate detours in Sybil paths. For instance, it could randomly induce a detour in a Sybil path and request the path generator to produce a new path to the Sybil destination. SybilQuery's design only requires the query generator to be modified to send k queries to the LBS instead of one. Based upon the deployment scenario, even this modification should be relatively easy. For example, if an end-user employs a Web browser on his mobile phone to send location-based queries to an LBS, the query generator could be implemented as a browser extension.

3.1 Enhancements

The basic design presented above can be augmented in several ways to improve the robustness of the system to potential attacks.

(**a**) **Randomizing path selection.** Path generators implemented using navigation systems typically return the shortest route to the destination. However, real users may not always follow the shortest path to the destination because of factors such as detours and road closures. If SybilQuery always produces shortest Sybil paths and the user chooses a longer path to the destination, an adversary will be able to differentiate the real path from Sybil paths with high probability.

This problem can be addressed with path generators that can compute multiple paths to the destination (each with varying lengths). Instead of choosing the shortest path, the path generator could instead use a probability distribution (of the frequency with which users choose paths other than the shortest path) to make an appropriate choice from one of the available paths to the destination.

(**b**) **Handling active adversaries.** An actively adversarial LBS may return doctored query responses as an attempt to differentiate Sybil paths from a client's real path. For example, suppose that an adversarial LBS falsely reports traffic congestion at a particular location in response to a user query. In response, a real user will likely take a detour. If the SybilQuery system does not respond similarly to doctored responses (*e.g.,* by checking for congestion and triggering detours in Sybil paths) an adversary could likely distinguish the real path from Sybil paths.

SybilQuery can handle active adversaries using N-variant queries. In this technique, SybilQuery queries multiple LBSs and compares their responses in a manner akin to N-variant systems [6]. Unless all the LBSs queried collude with each other, adversarial LBSs are likely to be detected.

Note that an active adversary runs a greater risk of being detected by the user than a passive adversary. In its attempt to differentiate real paths from Sybil paths, the adversary is likely (with probability $1/k$) to send a doctored response to the actual user. If the user notices a difference between the response sent by the adversary and the expected answer, *e.g.,* the adversary falsely reports that there is traffic congestion at a location when there actually is no congestion, the user can conclude that an attack is in progress.

(**c**) **Providing path continuity.** Although the paths generated by SybilQuery statistically resemble each other, it is possible for the trip durations to differ (for both real and Sybil trips). If the real trip ends before some of the Sybil trips end, and the system stops sending queries, the LBS can differentiate the real path from Sybil paths. SybilQuery guards against this by being an "always on" tool that continues to simulate movement along Sybil paths even when the user's real trip is complete. This feature ensures security even when the lengths/durations of Sybil trips do not match those of real trips.

4 The SybilQuery Prototype

In this section, we describe the implementation of our SybilQuery prototype. The bulk of this section is devoted to the description of the endpoint generator, which is implemented as a python client that uses a PostgreSQL database backend with PostGIS spatial extensions to store regional traffic information. The path generator is implemented as a client that queries an off-the-shelf service [28] to find the set of waypoints corresponding to the shortest path, given a startpoint and an endpoint. Finally, the query generator simulates user movement along all the k paths.

4.1 The Endpoint Generator

As discussed in Section 3, the endpoint generator uses the source and destination addresses input by the user to generate $k - 1$ synthetic endpoints. To ensure that these synthetic endpoints statistically resemble the actual endpoints of the user's trip, the endpoint generator uses a regional traffic database (Section 4.1.1). The endpoint generator first preprocesses the database to identify key features of each geographic location (Section 4.1.2). When the user supplies a source and destination address, the endpoint generator returns $k - 1$ endpoints whose features closely resemble those of the user's source and destination (Section 4.1.3), which can then be used as inputs with the path generator.

4.1.1 Regional Traffic Database

For our prototype, we used a month-long (August 25, 2008—September 24, 2008) set of GPS traces from the Cabspotting project [3] as regional traffic database. The Cabspotting project tracks the mobility of cabs in the San Francisco Bay area.[1] Each cab that participated in this project was outfit with a GPS device that updated its location with a server each minute. Each of these updates was a quadruple:

<timestamp, cab ID, current location (latitude/longitude), flag>

Here, flag indicates whether the cab was metered (*i.e.,* in a trip) or empty. We used flag to convert the GPS traces into trips (*i.e.,* to demarcate sources and destinations during which the cab was metered) and stored these trips in a PostgreSQL database with PostGIS spatial extensions, which enabled us to make spatial queries on this database. Such spatial queries are used in the preprocessing step (Section 4.1.2) and to generate statistically similar endpoints (Section 4.1.3).

We used this database because it was easily available and the information that it contained accurately depicted traffic conditions at different geographic locations in the San Francisco Bay area. For example, in downtown areas that were crowded during the daytime, the trips tended to be slow. Similarly, this database also reflected both diurnal and daily variations in traffic conditions. Although we tailor our discussion in

[1] Consequently, our experiments also focus on trips in this geographic region. However, SybilQuery can automatically generate Sybil queries for any geographic region if provided with a similar traffic database for that region.

the rest of this section to this database, we emphasize that the techniques employed by SybilQuery are applicable to any traffic database. Overall, our database contains a total of 529,533 trips by 530 unique cabs; we observed at least 447 cabs on any given day.

4.1.2 Preprocessing the Traffic Database

An ideal implementation of the endpoint generator would use an annotated database of the local region to identify synthetic endpoints that resemble the endpoints input by the user. The annotated database would contain descriptive tags for each geographic location, such as "parking lot," "downtown office building," or "freeway." The endpoint generator would output synthetic sources and destinations whose tags match the corresponding tags of the source and destination supplied by the user. However, such annotated databases are laborious to create and even so are unlikely to be comprehensive or contain tags suitable for LBSs in different application domains.

The preprocessing step addresses this problem by automatically computing a feature set that describes each geographic location using information from the traffic database. These automatically-extracted features then serve as tags that the endpoint generator can use to find synthetic sources and destinations that statistically resemble the user's input.

Our implementation uses *traffic density* as the feature that characterizes each geographic location. We define the traffic density τ_ℓ of a geographic location ℓ as the number of trips that start, end, or traverse through ℓ in a fixed interval of time (we describe in detail below our how our system represents geographic locations; for simplicity, it suffices to think of them as fixed-size rectangular regions, *e.g.,* city blocks). We calculated τ_ℓ for each location ℓ using the PostGIS spatial extension of PostgreSQL. Because τ_ℓ can acquire a large number of discrete values, we used a simple clustering algorithm to group "similar" values of τ_ℓ into a single cluster. Although any clustering technique may be used, we found that even simple clustering algorithms, such as separating values of τ_ℓ into buckets, serve our purpose well. We empirically verified that these clusters of the San Francisco Bay area share similar semantic properties. For example, different shopping centers were grouped together into the same cluster as were residential areas.

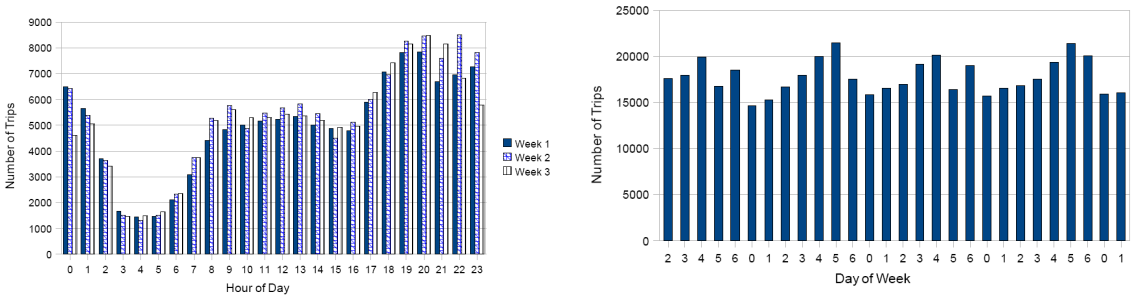
In addition to computing the traffic density of each geographic location ℓ, the preprocessing step also computes a probability distribution function (PDF) π_ℓ of the length of a trip that originates at ℓ. In particular, it computes the following quantity for each location ℓ for all values of *len*.

$$\pi_\ell(len) = \Pr[\text{trip length} = len] = \frac{\#\text{ trips of length } len \text{ that start in } \ell}{\#\text{ trips that start in } \ell}$$

This PDF is used by the endpoint generation algorithm to choose appropriate geographic locations as sources for Sybil paths, as described in Section 4.1.3.

Temporal traffic patterns. Traffic densities at a geographic location ℓ vary depending on the hour of the day and the day of the week, which results in a temporal pattern in the values of τ_ℓ. To better understand such temporal patterns, we calculated the values of τ_ℓ using both an hour as well as a day as the time interval. Figure 3(a) plots the values of τ_ℓ (averaged across all geographic locations ℓ) for each hour of the day for three weeks' worth of data from our traffic database. This figure shows that traffic density exhibits a strong diurnal pattern, *e.g.,* with high traffic density during peak hours (6pm-11pm) and low traffic density during the night (3am-6am). Figure 3(b) shows the temporal patterns on a daily basis. Although we expected to find higher traffic densities during weekdays, our data showed that this was not the case. This is likely because we used traces from the Cabspotter project, which only tracks the cab traffic, which may exhibit different patterns than regular traffic. A richer traffic database is likely to yield more temporal patterns.

To reflect these temporal patterns in the features of each geographic location, we leveraged the results in Figure 3 to define six kinds of temporal states, namely, "peak interval/weekday" (7pm-11pm),



(a) **Hourly Patterns** (b) **Daily Patterns (0=Friday)**

Figure 3: Temporal patterns of traffic density at different temporal granularities.

"normal interval/weekday" (6am-7pm and 11pm-3am), "off-peak interval/weekday" (3am-6am), "peak interval/weekend" (7am-11pm), "normal interval/weekend" (6am-7pm and 11pm-3am) and "off-peak interval/weekend" (3am-6am). The precomputation step computes six values of τ_ℓ for each location, corresponding to each of the temporal states above. When the user supplies a source and a destination, the endpoint generator chooses the value of τ_ℓ to compute synthetic endpoints based upon the timestamp in the user's input.

Representing geographic locations. To compute τ_ℓ, the preprocessing algorithm needs an appropriate data structure to represent the location ℓ. This data structure must simultaneously balance precision and scalability, *i.e.,* it must store precise values of τ_ℓ for each location ℓ and must readily scale to large geographic regions, such as the San Francisco Bay area.

To better illustrate this tradeoff, suppose that geographic locations are represented using fixed-size rectangular blocks, as was the case in an early implementation of our prototype. We found that a block size of 400m×400m resulted in a manageable number of blocks for the entire Bay area (about 100,000 blocks), but did not accurately represent traffic densities in crowded areas, such as the downtown region and the airport. On the other hand, a block size of 25m×25m (about the size of a city block) accurately represented traffic densities, but resulted in a large number of blocks (about 6.4 million blocks).

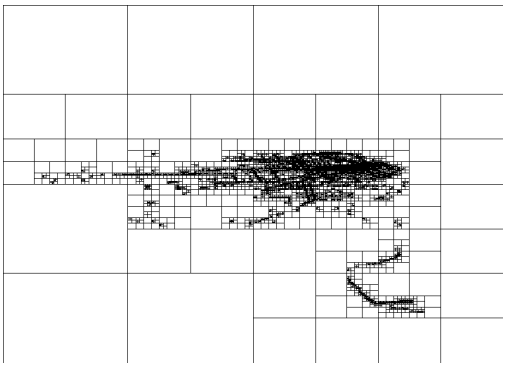
SybilQuery therefore uses an adaptive data structure, the *Quadtree* [9], to represent traffic densities. This data structure represents geographic locations using blocks of varying sizes (smallest size is 25m×25m) depending on the traffic density of that location. In particular, geographic regions with non-uniform distribution of traffic are represented using small block sizes, while regions with a uniform distribution of traffic are represented using larger block sizes. Using the Quadtree data structure, we were able to represent traffic densities for the entire San Francisco Bay area using just 16,000 blocks.

Algorithm 1 summarizes how the traffic database is preprocessed; for simplicity, it does not show the computation of τ_ℓ in each of the six temporal states, as discussed earlier. Intuitively, Algorithm 1 accepts a rectangular geographic region as input, and determines whether the traffic density in the region is uniform. In particular, the function is_uniform iterates over 25m×25m blocks of the geographic region and determines whether the traffic densities in these blocks are within a threshold value of each other. If this is indeed the case, the algorithm computes the traffic density of the entire geographic region and stores it in the TrafficDensity database. If not, it splits the geographic region into four equal-sized quadrants and recurses on each of them.

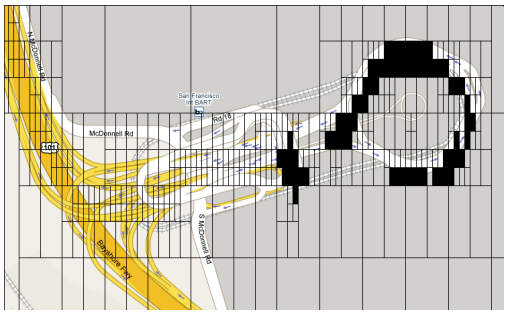
Algorithm : Extract_Location_Features (Traffic_Database, Geographic_Region)
Input : (a) Traffic_Database: Information about cab trips (Section 4.1.1); (b) Geographic_Region: A rectangular geographic region, represented by the extreme latitutes (LA) and longitudes (LO) along a diagonal: <LA_{left}, LO_{left}, LA_{right}, LO_{right}>.
Output : TrafficDensity: A database of tuples <ℓ, τ_ℓ> denoting the traffic densities for each location.

```
if is_uniform (<LA_left, LO_left, LA_right, LO_right>) or (sizeof(Geographic_Region) ≤ min_block_size (25m×25m)) then
    Compute τ_ℓ and π_ℓ for ℓ = (<LA_left, LO_left, LA_right, LO_right>) using Traffic_Database;
    Add <ℓ, τ_ℓ, π_ℓ> to TrafficDensity;
else
    Let <LA_mid, LO_mid> be the midpoint of Geographic_Region;
    Extract_Location_Features(Traffic_Database, <LA_left, LO_left, LA_mid, LO_mid>);
    Extract_Location_Features(Traffic_Database, <LA_mid, LO_left, LA_right, LO_mid>);
    Extract_Location_Features(Traffic_Database, <LA_left, LO_mid, LA_mid, LO_right>);
    Extract_Location_Features(Traffic_Database, <LA_mid, LO_mid, LA_right, LO_right>);
Cluster entries in TrafficDensity with similar values of τ_ℓ;
return TrafficDensity;
```

Algorithm 1: Preprocessing the traffic database.



(a) **San Francisco Bay area. The airport appears on the lower right corner of this figure.**



(b) **San Francisco airport. Black blocks have high traffic densities.**

Figure 4: Quadtree representations of geographic locations.

Output of preprocessing. Figure 4 pictorially represents the output of the preprocessing step. Figure 4(a) shows the quadtree representation of the entire San Francisco Bay area. The adaptive nature of the Quadtree data structure results in different block sizes for different geographic locations. Although each of these blocks has uniform traffic density, in our experience, areas with high traffic density tend to be represented with smaller block sizes. This effect is most pronounced in areas with dense traffic, such as the airport, depicted in Figure 4(b). Note that the regions with highest traffic density, such as the freeway and the pickup area in front of the airport are represented using small blocks, while areas inside the airport that have no traffic are represented using larger block sizes. Not all blocks have the same traffic density; in Figure 4(b), blocks with the highest traffic densities are filled in black.

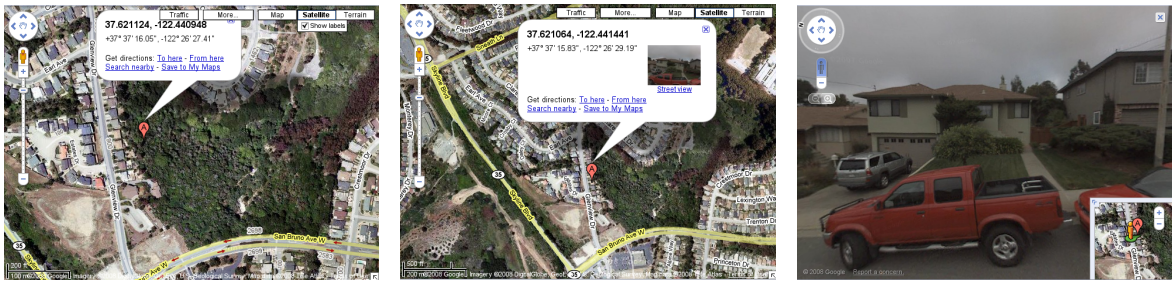
4.1.3 Generating Sybil Endpoints

When provided with a source and a destination input by the user, the endpoint generator computes a Sybil source/destination pair as shown in Algorithm 2 (our prototype adapts the same algorithm to generate $k-1$ Sybil source/destination pairs; for brevity, we only illustrate the algorithm for one pair). It first finds the geographic locations (in the Quadtree representation) ℓ_{src} and ℓ_{dst} that contain the source and destination addresses input by the user. Next, it computes the set of all source locations ℓ that satisfy two conditions: (1) the traffic density of ℓ is approximately that of ℓ_{src}; and (2) the probability of a trip of length dist

Algorithm : Generate_Sybil_Endpoints(src, dst)
Input : (a) src: Street address of user's source; (b) dst: Street address of user's destination.
Output : (src′, dst′): A Sybil source/destination pair.

1 dist = Euclidean distance between src and dst;
2 ℓ_{src}, ℓ_{dst} = geographic locations that contain src, dst;
3 Sources = Set of all locations ℓ with (i) $\tau_\ell \approx \tau_{\ell_{\text{src}}}$ and (ii) $\Pi_\ell(\text{dist}) \approx \Pi_{\ell_{\text{src}}}(\text{dist})$;
4 $\ell_{\text{src}'}$ = random location ∈ Sources;
5 Destinations = Set of all locations ℓ with (i) $\tau_\ell \approx \tau_{\ell_{\text{dst}}}$ and (ii) Euclidean distance between $\ell_{\text{src}'}$ and $\ell \approx$ dist;
6 $\ell_{\text{dst}'}$ = random location ∈ Destinations;
7 src′, dst′ = Reverse geocode random points in $\ell_{\text{src}'}$, $\ell_{\text{dst}'}$;
8 **return** (src′, dst′);

Algorithm 2: Generating Sybil endpoints.



(a) **Random point in geographic location.** (b) **Street address closest to random point via reverse geocoding.** (c) **Street view of Figure 5(b).**

Figure 5: Finding a point in a geographic location using reverse geocoding.

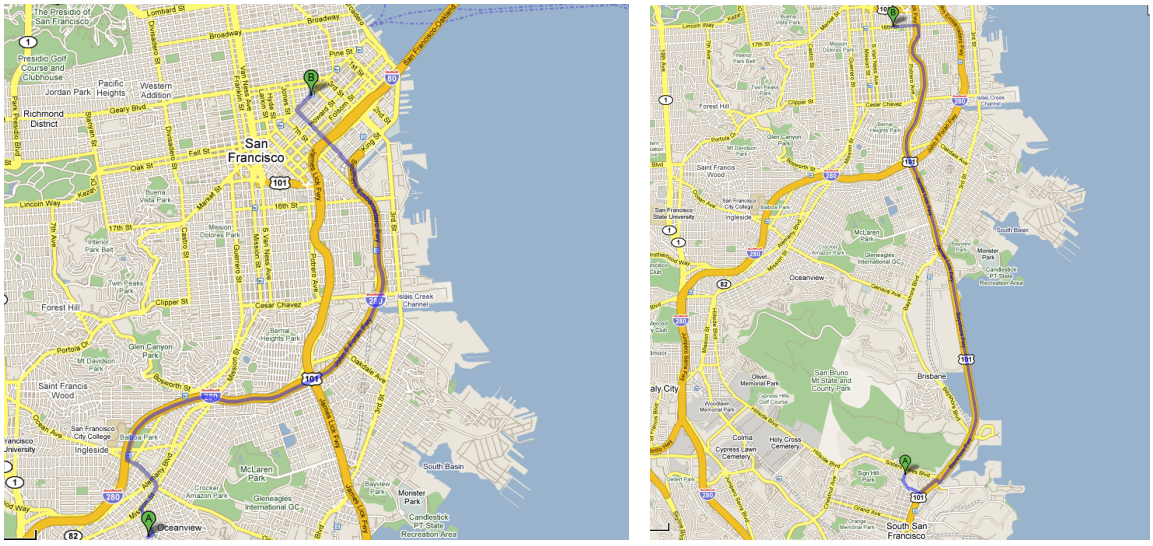
originating from ℓ matches that of ℓ_{src}. The key intuition behind this step is to find the set of source locations that closely resemble the source location input by the user. It randomly chooses a source location $\ell_{\text{src}'}$ from this set of locations, and finds a destination location $\ell_{\text{dst}'}$ within a radius dist of $\ell_{\text{src}'}$ whose traffic density matches that of ℓ_{dst}.

The last step is to identify actual addresses within the Sybil source/destination locations just identified. To do so, it randomly chooses a point within the geographic location. However, this point may reside in non-driveable terrain, as shown in Figure 5(a). If this point were returned as a source/destination, an adversary can easily identify this point as a Sybil address. To avoid such cases, we reverse geocode this point (using an API provided by Google Maps [15]) to find the street address closest to the point identified. Doing so greatly improves the quality of the endpoints generated by our algorithm. Figure 5(b) (Figure 5(c)) shows the closest street address (street view) of the point identified in Figure 5(a).

Finally, the endpoint generator caches the Sybil endpoints for the most common trips by the user. In addition to improving performance, *caching also improves security*. Suppose that a real path P frequented by the user (*e.g.,* commuter paths, such as home to office) is associated with multiple sets of Sybil paths. An LBS that observes paths over a period of time can statistically identify P as the real path. Instead, by associating P with the *same* set of Sybil paths, the LBS will observe the same set of paths each time the user traverses P, thereby preventing the attack.

4.2 The Path Generator

The SybilQuery prototype implements the path generator using the Microsoft Multimap API [28]. When supplied with a source and a destination, this API produces a sequence of waypoints representing the path to the destination. As noted earlier, SybilQuery admits the use of any off-the-shelf path generator, such as



(a) **Real trip. The values of τ_ℓ for the source and destination were 22 and 247, respectively.**

(b) **One of the Sybil trips. The values of τ_ℓ for the source and destination were 22 and 255, respectively.**

Figure 6: A real user trip and a Sybil trip generated by SybilQuery.

those used by on-board navigation systems. This feature helps SybilQuery produce paths that automatically account for environmental factors, such as road closures and one ways.

The Multimap API currently only produces the shortest path to the destination, thus making our prototype vulnerable to attack if the user follows a longer path to the destination. However, as noted in Section 3.1, this problem can easily be fixed with the use of an API that generates a choice of paths to the destination (or an API that allows the user to specify a waypoint that must be included enroute the destination). The path generation algorithm could then choose paths (both real and Sybil paths) based upon a probability distribution of the length of path that users typically follow to their destination.

4.3 The Query Generator

The basic version of SybilQuery's query generator simulates user movement along each path. It simulates the movement of users along Sybil paths at approximately the projected speed along that path. (Information about the average speed along a path is obtained is returned by most off-the-shelf path generators). When the user sends a query to the LBS, the query generator obtains the current location of the simulated users and sends these as Sybil queries to the LBS. We have also interfaced the query generator with the Yahoo! local API [1] to more accurately simulate movement along Sybil paths under the constraints of current traffic. Thus, if there is traffic congestion at a particular location, SybilQuery can either simulate slower movement in that location or simulate a detour (and request the path generator to generate a fresh path to the destination). Although querying an LBS that reports current traffic conditions renders the prototype vulnerable to malicious LBSs that report false traffic data, the query generator can use the N-variant queries approach described in Section 3.1 to counter this threat.

4.4 Example

Figure 6 presents SybilQuery in action. It shows a real user trip (Figure 6(a)), obtained from the Cabspotter traces for another month and one of the Sybil trips generated by SybilQuery (Figure 6(b)); k was set to 4 for

k	# questions	# correct	Probability
4	75	20	0.26
6	75	14	0.19

Figure 7: Results of a user study with 15 volunteers.

this example. The real trip (at approximately 6pm) originates at a home in Daly City and ends at a shopping area in downtown San Francisco. All the Sybil trips generated by SybilQuery also started in residential areas with similar traffic densities as the source of the real trip, and ended in a crowded region of San Francisco downtown. Figure 6 also reports the traffic densities for the sources and destinations for the real and Sybil trip generated by SybilQuery. A key point to note from this example is that traffic densities accurately capture semantic properties (*e.g.,* "residential area," "downtown," "shopping area") of geographic regions.

5 Evaluation

In this section, we present the results of our evaluation of the SybilQuery prototype. We evaluated both the privacy and the performance offered by SybilQuery.

5.1 Privacy

To evaluate the quality of the Sybil paths generated by our system, we conducted two studies. First, we qualitatively evaluated the Sybil paths via a user study. Second, we devised a metric to quantitatively evaluate the privacy offered by SybilQuery. For both studies, our methodology was similar; we picked real paths at random from the Cabspotter traces (different from the month-long traces that were used to seed our prototype's endpoint generator) and used SybilQuery to generate Sybil paths with different values of k.

5.1.1 User Study

We conducted a user study with 15 volunteers, each of whom had to answer ten questions. Each question presented k paths (one real path and $k-1$ Sybil paths) to the volunteer, who had to identify the real path from the Sybil paths. Five questions had $k = 4$ while the remaining five had $k = 6$. Each volunteer was presented with a different set of ten questions, *i.e.,* our study obtained responses for 150 questions in all. Volunteers could view paths using the Google Maps API, and were instructed to use any tools at their disposal, such as zooming into the map, street views, and local information about the San Francisco Bay area, in their attempt to distinguish real paths from Sybil paths. All our volunteers were computer science graduates familiar with the basic geography of the San Francisco Bay area.

Figure 7 presents the results of this study and shows the number of questions for which users were able to correctly identify the real path from the Sybil paths. As this figure indicates, volunteers were able to correctly guess the real path with a probability of 0.26 for $k = 4$ and 0.19 for $k = 6$. These probabilities are close to their expected values (0.25 and 0.17, respectively), leading us to conclude that the Sybil paths qualitatively resemble real paths.

5.1.2 Quantitative Estimation of Privacy

To quantitatively measure the privacy offered by SybilQuery, we devised a new metric. The intuition behind this metric is that an adversary who has access to a database $\mathcal{R}$ of real paths and a database $\mathcal{S}$ of Sybil paths corresponding to paths in $\mathcal{R}$ should be unable to differentiate between the two databases.

Recall that SybilQuery only generates *endpoints* that are statistically similar to each other and relies on off-the-shelf navigation systems to produce paths. Our metric, PathCmp, measures the similarity of two *paths* P and Q of approximately equal length, and is computed as follows. Suppose that the waypoints along

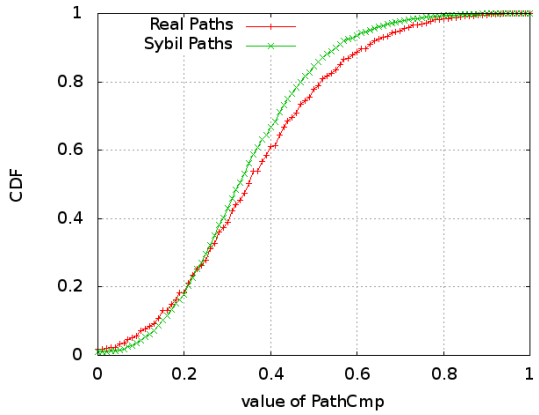


Figure 8: CDF of the values of PathCmp(R_i, R_j) and PathCmp(S_i, S_j) for each pair of paths in the database $\mathcal{R}$ of real paths and the database $\mathcal{S}$ of Sybil paths, respectively.

P are $p_1, \ldots, p_n$ and the waypoints along Q are $q_1, \ldots, q_n$.[2] PathCmp is the average value of the normalized difference between $\tau_\ell(p_i)$ and $\tau_\ell(q_i)$.

$$\text{PathCmp}(P, Q) = \sum_{i=1}^{n} \frac{|\tau_\ell(p_i) - \tau_\ell(q_i)|}{n \times D} \in [0, 1]$$

Here, D is the maximum difference between the values of τ_ℓ between *any* two geographic locations in our regional database. Intuitively, this metric computes overall path similarity by comparing each of the waypoints against each other as well. A value of zero indicates that paths P and Q are similar to each other, including at all waypoints, while a value of one indicates that they are dissimilar.

To evaluate SybilQuery, we randomly selected 1000 paths, 5 kilometers in length, from the Cabspotter traces and created a database $\mathcal{R}$. We then used SybilQuery to generate a database $\mathcal{S}$ of Sybil paths of each of the (real) paths in $\mathcal{R}$. We then computed PathCmp(R_i, R_j) and PathCmp(S_i, S_j) for each pair of paths in $\mathcal{R}$ and $\mathcal{S}$, respectively. Figure 8 shows the cumulative distribution function (CDF) of the values of PathCmp for both $\mathcal{R}$ and $\mathcal{S}$.

As this figure shows, the CDFs are similar to each other, thereby showing that paths in $\mathcal{S}$ exhibit the same features as those in $\mathcal{R}$. Because Sybil queries are generated using Sybil paths, this figure also indicates that Sybil queries will be similar to real queries. These results justify the design of SybilQuery because they indicate that *paths generated by off-the-shelf navigation systems using synthetic endpoints resemble actual user paths.*

5.2 Performance

We report the performance of several aspects of SybilQuery, including the time needed to preprocess the traffic database and the real-time performance of querying an LBS. We also compare the performance of SybilQuery against an implementation of spatial cloaking. All the results presented below were obtained on a 1.73GHz Pentium M laptop with 512 MB RAM. Each experiment was repeated 50 times, and the value of the privacy parameter k was fixed to be 4, unless otherwise indicated.

[2] In cases where a path has fewer than n waypoints, we designate randomly-chosen points along this path as waypoints.

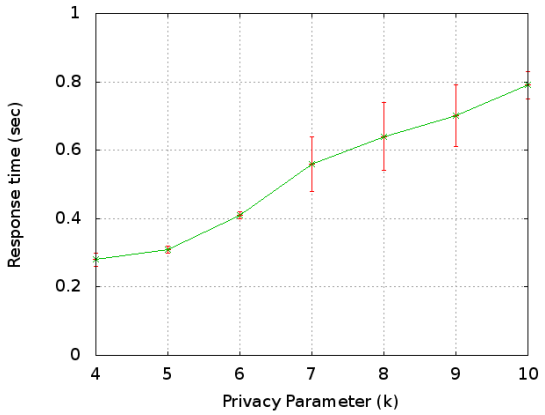


Figure 9: Query/response latency of SybilQuery.

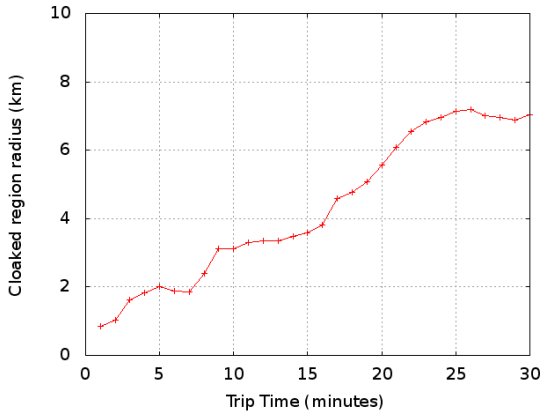


Figure 10: Spatial cloaking: Increase of cloaked region radius with trip length.

5.2.1 One-Time and Once-Per-Trip Costs

The one-time cost (offline step) for SybilQuery involves preprocessing of the traffic database, as described in Section 4.1.2. This step processed 529,533 trips and took approximately 2 hours and 16 minutes.

The once-per-trip costs for SybilQuery involve end-point generation and path generation, as described in Section 4.1.3 and Section 4.2. Generating a single pair of endpoints at the beginning of a user trip took an average of 5.47 seconds with a standard deviation of 1.02 seconds. To measure the cost of generating paths, we randomly chose trips with lengths varying between 5 and 30 minutes. The mean cost of computing a path (including the network latency to query the Microsoft MultiMap API) was 360 milliseconds, with a standard deviation of 150 milliseconds.

5.2.2 Query/Response Performance

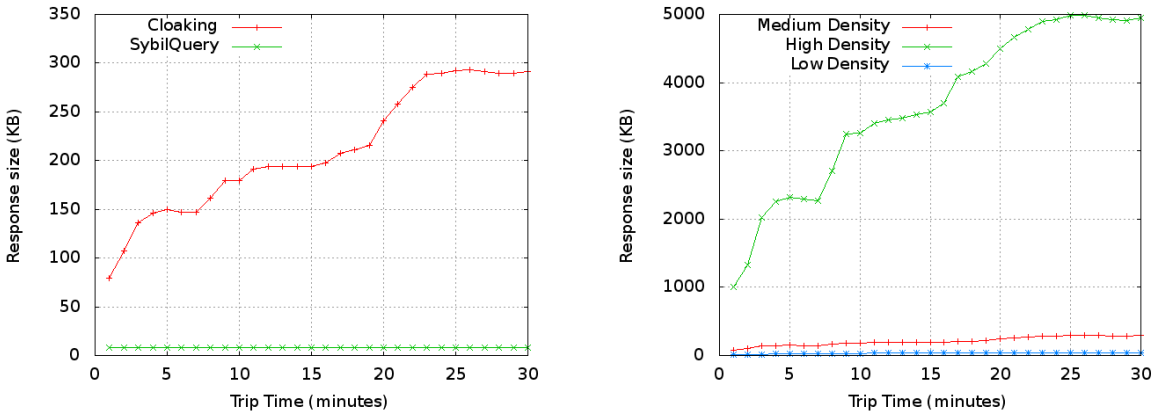
We measured the query/response latency of SybilQuery by integrating it with the Yahoo! Maps local search API [36]. Figure 9 shows the latency of sending k queries and receiving responses. As expected, the cost increases linearly with k, since k requests are sent to the LBS each time the client makes a query. Figure 9 is also indicative of the cost of using N-variant queries to defend against active adversaries; the cost increases linearly with $N \times k$.

5.2.3 Comparison with Spatial Cloaking

Spatial cloaking [16] is a state of the art technique that achieves k-anonymity by using an anonymizer to send the LBS cloaked regions containing at least k users. The LBS returns all points of interest (POIs) within the cloaked region that match the query to the anonymizer. Although spatial cloaking was originally proposed for static users, we considered a simple variant for mobile users in which cloaked regions grow as users move.[3]

We conducted experiments to compare spatial cloaking with SybilQuery. Because spatial cloaking uses an anonymizer to send cloaked regions containing k clients to the LBS, we can expect the cloaked regions to increase in size as clients move (because the cloaked region must contain the *same* set of clients to preserve k-anonymity). In addition, because an LBS returns query results for the cloaked region, we can expect

[3]Although we have not carefully analyzed whether this simple variant is vulnerable to correlation attacks [12], it suffices to compare the query/response performance of spatial cloaking with that of SybilQuery.



(a) **Response size returned by SybilQuery and spatial cloaking, for medium POI density.**

(b) **Response size returned by spatial cloaking, for different POI densities.**

Figure 11: Comparing the performance of SybilQuery with spatial cloaking for nearest neighbour queries.

that increased POI density will lead to increased query/response processing time at the anonymizer. Our experiments, reported below, confirmed these expectations.

Figure 10 presents the results of the experiment that shows that the size of cloaked regions increases as clients move. We randomly chose trips varying in duration from 1 minute to 30 minutes from the Cabspotter database. We then fixed $k = 4$, and selected the smallest cloaked region containing at least $k-1$ other clients. As Figure 10 shows, the size of the cloaked region increases with the duration of the trip.

To understand how this increase translates to query/response processing overhead, we studied the size of the response (in kilobytes) received from the LBS for a fixed nearest neighbor query: "return the Chinese restaurant closest to my current location." In spatial cloaking, the anonymizer sends the entire cloaked region to the LBS, and must process responses from the LBS to identify the closest Chinese restaurant for the client issuing the query. In contrast, because SybilQuery sends exact street addresses, the LBS sends the closest Chinese restaurant. Figure 11(a) compares the sizes of the query response for spatial cloaking and SybilQuery. As this figure shows, the query response size for SybilQuery is nearly a constant (at approximately 2KB); it only varies only with the value of k. However, the response size for spatial cloaking increases linearly. This is because the size of the cloaked region increases with trip length, which in turn directly translates to an increased number of POIs that match the user's query.

We also conducted an experiment to study the effect of increased POI densities on query/response performance. We chose three representative nearest neighbor queries that required the LBS to return the closest "restaurant," "Chinese restaurant," and "Hunan Chinese restaurant" to the client's location. For spatial cloaking, these queries represent, respectively, high, medium, and low POI densities. As Figure 11(b) shows, with spatial cloaking the size of the query results depends both on POI density *and* on the duration of the trip, and rises to about 5MB within 30 minutes for high POI densities. In contrast, with SybilQuery, the query response size remains fixed at 2KB (because only the closest restaurant is returned).

The experiments above used nearest-neighbour queries, for which the difference in query responses for spatial cloaking and SybilQuery are most pronounced. In contrast, range queries, such as "return the list of all Chinese restaurants in an x-mile radius," require the LBS to process the query over an entire geographic region. If a client issuing range queries uses SybilQuery to protect his location, the LBS must process k

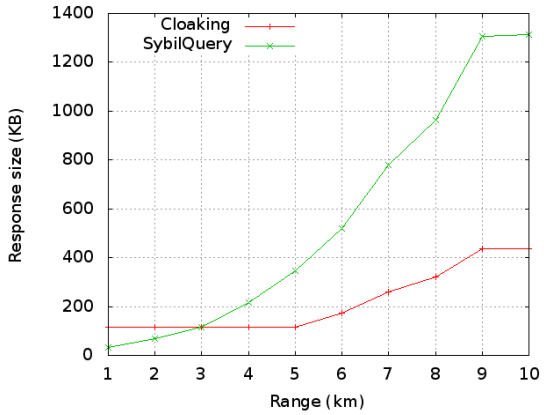


Figure 12: Comparing the performance of SybilQuery with spatial cloaking for range queries.

geographic regions, each of x-mile radius. In contrast, if the client uses spatial cloaking, the LBS must only process one geographic region whose radius is x miles larger than the cloaked region. We can therefore expect spatial cloaking to outperform SybilQuery as x increases.

Figure 12 shows that this is indeed the case. This figure plots the query response size against increasing values of x for both SybilQuery and spatial cloaking. For this experiment, we assumed a circular cloaked region with a fixed radius 5 kilometers, and used SybilQuery with $k = 4$. For smaller values of x, SybilQuery outperforms spatial cloaking because the combined area of k regions of radius x is smaller than the cloaked region. However, as x increases, spatial cloaking outperforms SybilQuery.

Nevertheless, we note that the query/response performance of SybilQuery can never be worse than k times the performance of spatial cloaking even for range queries.

6 Related Work

Cloaking schemes. Originally introduced by Gruteser and Grunwald [16], cloaking aims to achieve k-anonymity by hiding client location from the LBS both in space and time. These systems achieve spatial anonymity by sending a cloaked region containing at least k users to the LBS. They achieve temporal cloaking by delaying a query until at least k other users have also issued queries. This basic model has been extended in several ways. For example, work by Gedik and Liu [10, 11], Bamba *et al.* [2] and Mokbel *et al.* [27] allow users to personalize their privacy requirements, *e.g.,* by letting them decide thresholds for spatial and temporal resolution of cloaked regions. Similarly, work by Bamba *et al.* [2] extends the basic model to allow for ℓ-diverse [25] queries. Other related approaches include achieving k-anonymity using path confusion [17, 19], where the anonymizing system attempts to confuse the adversary by crossing paths where at least two users meet.

A common theme in all the above systems is the need for a trusted third-party anonymizer, that ensures k-anonymity (or ℓ-diversity). Using a third-party anonymizer has two disadvantages. First, the anonymizer is the central point of failure and presents scalability and performance bottlenecks. For example, an adversary could cripple the system with a denial-of-service attack on the anonymizer. Moreover, because the anonymizer has access to sensitive information from clients, a compromize of the anonymizer would result in a privacy breach. By offering a decentralized design, SybilQuery avoids these shortcomings. Second, most previous approaches only provide security guarantees for static snapshots and do not consider history of user movement (with the exception of work by Chow and Mokbel [4], which also uses an anonymizer). Therefore, if a user asks the same query from multiple cloaked regions as she moves, the LBS could compro-

mise her privacy by correlating queries from these regions. These attacks are called correlation attacks [12]. Although SybilQuery does not offer provable security guarantees against correlation attacks, it offers improved protection than cloaking schemes because it does not attempt to hide the actual path of the user from the LBS. Instead, it ensures that the adversary is unable differentiate user paths from Sybil paths.

Peer-to-peer schemes. Recent research has developed techniques based on peer-to-peer techniques to eliminate the need for an anonymizer [5, 13, 14]. Also related, although proposed as a mechanism for anonymous Web access, is Crowds [30], in which a query originates from a "crowd" of k users and the adversary is unable to identify the source of the query. These techniques have the advantage of being decentralized in nature. However, they still rely on the presence of at least k participating peers; in contrast, SybilQuery operates autonomously, independent of other querying peers.

PIR. Cryptographic techniques based on private information retrieval (PIR) [22] offer another alternative to eliminate anonymizers [12]. Using PIR to protect client location offers strong cryptographic guarantees on privacy that SybilQuery unfortunately cannot offer. However, the PIR-based scheme suffers from two drawbacks that limit its applicability. First, in spite of impressive recent advances, PIR remains computationally expensive [33]. For example, Ghinita *et al.* [12] employ a PIR protocol that imposes a cost of $O(n)$ at the server, in addition to client/server communication costs of $O(\sqrt{n})$. Practically, this translated to about 30 to 60 seconds of server processing time for each location query and communication of a few megabytes of data to process a single location-based query in their implementation. In contrast, SybilQuery is computationally-cheap and generates queries with sub-second latency. Second, PIR-based schemes require code modifications at both the client and the server to implement the PIR protocol. Thus, in contrast to SybilQuery, PIR-based schemes cannot easily be applied to legacy systems.

Other related research. SpaceTwist [24] is a recent location privacy system that aims to eliminate the need for an anonymizer. It uses a cloaking-like scheme to obfuscate the location of the client from the LBS and uses an incremental algorithm to process nearest neighbor queries. Like SybilQuery, SpaceTwist is also decentralized and autonomous. However, unlike SybilQuery, it does not ensure k-anonymity, does not handle mobile users, and only supports nearest neighbor queries.

The idea of introducing synthetic information to achieve privacy has previously been explored for anonymous Web access [18, 31]. In these systems, a user query to a Web server (*e.g.,* a search request) is hidden amongst synthetically generated queries. Recently, Machanavajjhala *et al.* [26] developed rigorous techniques based upon a variant of differential privacy [7] to add synthetic information to released databases, and applied it to a database of traffic commuting patterns. Adapting these techniques to enable real-time generation of synthetic queries is an interesting direction for future work.

7 Summary and Future Work

SybilQuery is an efficient, decentralized technique to hide user location from LBSs. Its modular design allows SybilQuery to be deployed with off-the-shelf client devices and without any changes to the LBS. We implemented a prototype of SybilQuery and integrated it with LBSs such as Google Maps, Yahoo! Maps and Microsoft Live Maps. Our experiments—both a qualitative user study and a quantitative metric to compare paths—show that Sybil paths generated by SybilQuery statistically resemble real paths.

In future work, we plan to enhance the SybilQuery prototype in several ways. Chief among these is modifying the tool to generate Sybil queries that also achieve stronger privacy guarantees, such as ℓ-diversity [25] and t-closeness [23]. ℓ-diversity ensures that there are at least ℓ values of a sensitive attribute in a k-anonymity set, while t-closeness ensures that statistical distributions of a sensitive attribute in a k-anonymity set are close to its global distribution. We propose to experimentally investigate whether the use of traffic density to automatically tag geographic locations achieves ℓ-diversity and t-closeness in the

queries generated by SybilQuery. Doing so requires a manual study that compares the semantic properties of each location (*e.g.,* "hospital," or "parking lot") to determine whether distribution of queries satisfies these stronger privacy requirements.

References

[1] Yahoo! local. `http://traffic.yahoo.com/traffic/`.

[2] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting anonymous location queries in mobile environments with PrivacyGrid. In *Proc. WWW'08: Intl. Conference on the World-Wide Web*, 2008.

[3] Cabspotting project. `http://cabspotting.org/`.

[4] C.-Y. Chow and M. F. Mokbel. Enabling private continuous queries for revealed user locations. In *Proc. SSTD'07: Advances in Spatial and Temporal Databases*, 2007.

[5] C-Y. Chow, M. F. Mokbel, and X. Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In *Proc. GIS'06: ACM International Symposium on Advances in Geographic Information Systems*, 2006.

[6] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, j. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: A secretless framework for security through diversity. In *Proc. USENIX Security Symposium*, 2006.

[7] C. Dwork. Differential privacy. In *Proc. ICALP'06: Intl. Colloquium on Automata, Languages and Programming*, 2006.

[8] EZPass: A pass on privacy? `"http://www.nytimes.com/2005/07/17/magazine/17WWLN.html"`.

[9] R. Finkel and J.L. Bentley. Quad trees: A data structure for retrieval on composite keys. 4(1):1–9, 1974.

[10] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *Proc. ICDCS'05: Intl. Conference on Distributed Computing Systems*, 2005.

[11] B. Gedik and L. Liu. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE Transactions on Mobile Computing*, 2007.

[12] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K-L. Tan. Private queries in location-based services: Anonymizers are not necessary. In *Proc. SIGMOD'08: 2008 ACM SIGMOD Conference*, 2008.

[13] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Mobihide: A mobile peer-to-peer system for anonymous location-based queries. In *Proc. SSTD'07: International Symposium on Spatial and Temporal Databases*, 2007.

[14] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous location-based queries in distributed mobile systems. In *Proc. WWW'07: Intl. Conference on the World-Wide Web*, 2007.

[15] Google maps geo API. `http://code.google.com/apis/maps/`.

[16] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. Mobisys'03: Intl. Conference on Mobile Systems, Applications and Services*, 2003.

[17] B. Hoh and M. Gruteser. Location privacy through path confusion. In *Proc. SecureComm'05: IEEE/CreateNet Intl. Conference on Security and Privacy for Emerging Areas in Communication Networks*, 2005.

[18] D. Howe and H. Nissenbaum. TrackMeNot: Resisting surveillance in Web search. *On the Identity Trail: Privacy, Anonymity and Identity in a Networked Society (Oxford University Press)*, 2008.

[19] R. Roy Choudhury J. Meyerowitz. Realtime location privacy via mobility prediction: Creating confusion at crossroads. In *Proc. HotMobile'09: Intl. Workshop on Hot Topics in Mobile Computing*, 2009.

[20] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1719–1733, 2007.

[21] J. Krumm. Inference attacks on location tracks. In *Proc. Pervasive'07: Intl. Conference on Pervasive Computing*, 2007.

[22] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proc. FOCS'97: IEEE Symposium on Foundations of Computer Science*, 1997.

[23] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proc. ICDE'07: Intl. Conference on Data Engineering*, 2007.

[24] M. L. Liu, C. S. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the tradeoffs among location privacy, query performance, and query accuracy in mobile services. In *Proc ICDE'08: Intl. Conference on Data Engineering*, 2008.

[25] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-Diversity: Privacy beyond k-anonymity. In *Proc. ICDE'06: Intl. Conference on Data Engineering*, 2006.

[26] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *Proc. ICDE'08: Intl. Conference on Data Engineering*, 2008.

[27] M. F. Mokbel, C-Y. Chow, and W. G. Aref. The new Casper: Query processing for location services without compromising privacy. In *Proc. VLDB'06: Intl. Conference on Very Large Databases*, 2006.

[28] Microsoft multimap API. `http://www.multimap.com/`.

[29] NYC cabs strike over GPS system plans. `http://www.engadget.com/2007/03/09/nyc-cab-drivers-say-no-thanks-to-gps-installation/`.

[30] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[31] F. Saint-Jean, A. Johnson, D. Boneh, and J. Feigenbaum. Private Web search. In *Proc. WPES'07: ACM Workshop on Privacy in the Electronic Society*, 2007.

[32] P. Samarati. Protecting respondents identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[33] R. Sion and B. Carbunar. On the computational practicality of private information retrieval. In *Proc. NDSS'07: Network and Distributed System Security Symposium*, 2007.

[34] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[35] Tor. `http://www.torproject.org/`.

[36] Yahoo maps local search API. `http://developer.yahoo.com/maps/`.