This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Protection of Query Privacy for Continuous Location Based Services

Aniket Pingley*, Nan Zhang*, Xinwen Fu†, Hyeong-Ah Choi*, Suresh Subramaniam*, and Wei Zhao‡
*The George Washington University †The University of Massachusetts, Lowell ‡The University of Macau
Email: {apingley, nzhang10}@gwu.edu, xinwenfu@cs.uml.edu, {hchoi, suresh}@gwu.edu, weizhao@umac.mo

***Abstract*—Location-based services (LBS) have become an immensely valuable source of real-time information and guidance. Nonetheless, the potential abuse of users' sensitive personal data by an LBS server is evolving into a serious concern. Privacy concerns in LBS exist on two fronts: *location privacy* and *query privacy*. In this paper we investigate issues related to query privacy. In particular, we aim to prevent the LBS server from correlating the service attribute, e.g., bar/tavern, in the query to the user's real-world identity. Location obfuscation using spatial generalization aided by anonymization of LBS queries is a conventional means to this end. However, effectiveness of this technique would abate in continuous LBS scenarios, i.e., where users are moving and recurrently requesting for LBS. In this paper, we present a novel query-perturbation-based scheme that protects query privacy in continuous LBS even when user-identities are revealed. Unlike most exiting works, our scheme does not require the presence of a trusted third party.**

I. Introduction

In this paper, we develop novel techniques to protect the privacy of service attributes (e.g., bar, hospital, church) in continuous location-based-service (LBS) queries issued by a moving user. Our techniques are computationally and communication-wise efficient, require minimum storage footprint, and do not affect the accuracy of LBS query answers. Most importantly, we achieve a privacy guarantee while requiring neither a trusted third party nor a pre-known trajectory of the user's future movements - a feature which none of the existing techniques can achieve.

A. Motivation

Enabled by positioning infrastructures such as GPS, location-based services (LBS) are becoming an increasingly important component of not only leisure travel but also critical applications such as emergency response, public safety, etc. A typical example of LBS is a search engine for the nearest Points-of-Interest (POIs) - e.g., Google Maps. In this case, a user submits its current location and the service attribute (i.e., POI type such as museum) of interest to an LBS server, which returns a small number of POIs that match the user-specified service attribute value and are geographically close to the user's current location. LBS is enabled by techniques developed on multiple fronts - e.g., networking, database, and information retrieval - and is being actively studied in these research communities.

Despite the benefits provided by LBS, potential abuse of users' sensitive information by the LBS server may inhibit a user from availing these services. Henceforth, we refer to a malicious LBS server as the *adversary*. Privacy concerns in LBS exist on two fronts: *location privacy* and *query privacy*. Location privacy is related to the disclosure and misuse of user's location information. An example of its implication is that if a user issues an LBS query from a location within hospital premises then the adversary can associate a medical condition with the user. Query privacy, on the other hand, is related to disclosure of the service attribute. For example, frequent queries for a bar/tavern may lead the adversary to infer that the user is alcoholic.

Although distinct, location privacy and query privacy are closely related. In particular, disclosure of location may in turn reveal the service attribute to the adversary. Consider the case where a user anonymously, e.g., using Anonymizer (*www.anonymizer.com*) or Tor (*www.torproject.org*), issues an LBS query from his/her home. Ideally, the usage of anonymizer should dissociate the user from the service attribute, i.e., provide query privacy even if the location is disclosed. However, this is not true in practice - the adversary can usually find publicly available "external information" (e.g., white pages) to link the user's identity with his/her (home) location, and thereby compromise the user's query privacy. Such inference is called "restricted space identification" [11].

Location k-anonymity is a traditional technique to provide both location and query privacy simultaneously [1], [7], [11], [16], [23]. With this technique, a LBS query is issued to the LBS server via a trusted third party. The third party augments a user's location to a *cloaking region*, which geographically covers not only the user who issues the query but also $k-1$ other users, and then transmits the query to the LBS server. Since all the k users report the same cloaking region in their queries, the adversary cannot distinguish the location or service attribute of any user from the received queries.

A major challenge to location k-anonymity arises from the *continuous* nature of LBS queries - i.e., a user may repeatedly issue queries with the same service attribute (e.g., an alcoholic may query for a bar/tavern from different locations several times a week). Chow et al. [4] found that, to handle a continuous LBS query, the set of k users in a cloaking region must remain grouped together *over all snapshots* of the query, because otherwise the adversary may intersect users in different snapshots to infer the commonly present user as the query issuer. Unfortunately, it is highly unlikely for different users in a cloaking region to travel in the same direction. Thus, to cover the k users for a prolonged period of time (note that

many real-world continuous LBS queries - e.g., the alcoholic example discussed above - may last for months), the cloaking region often becomes extremely large and therefore fails to provide accurate answers to LBS queries.

A location obfuscation technique proposed by Xu et al. [32] may be used to protect location privacy for continuous LBS queries, as it utilizes a log of historical user locations rather than the real-time location information to generate cloaking regions. Nonetheless, this techniques offers no protection for the service attribute - i.e., query privacy. In this paper, we consider query privacy protection techniques that can hide the service attribute of continuous LBS queries even when the user's identity has been compromised by the adversary.

B. Outline of Technical Results

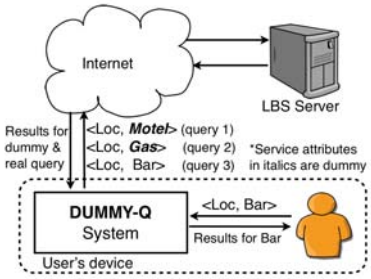


Fig. 1: Privacy-Preserving LBS with DUMMY-Q

We develop DUMMY-Q, a user-centric technique for query privacy protection which operates solely on the user side and does not require any trusted third party. The key idea is to confuse the adversary by issuing multiple counterfeit queries with varying service attributes but the same (real) location, henceforth referred to as *dummy queries*, along with each real query issued by the user. Figure 1 illustrates the usage of DUMMY-Q in LBS. While the notion of dummy queries appears simplistic, the challenges are plenty and intricate, especially in the scenario of continuous LBS queries:

- A critical requirement for dummy generation is that the dummy service attribute values must be generated in a judicious manner so as to remain consistent with the *query context* - i.e., the location where query is issued. For example, while users on a coastal location may often query for beaches, the same service attribute value may be quite rare around a desert area. If such adherence to the trend of queries is shunned, then the adversary may be able to exclude certain service attributes according to common sense and thereby identify the real query.
- In addition, one must insert the same (dummy) service attribute values over different snapshots of a continuous LBS query, in order to prevent the adversary from inferring the most frequent value as the real one (note that the real value has to be included at all snapshots). This, combined with the first challenge, requires the dummy insertion process to take into consideration the user's possible future locations, as otherwise the dummy values inserted in the beginning may later be excluded by the adversary according to the context of future queries.
- One must minimize the number of inserted dummy queries because each consumes additional overhead for issuing the query and waiting for the answer. Note that (1) the (real and dummy) queries have to be issued sequentially, because many LBS servers (e.g., Yahoo!) deny queries issued with time interval shorter than a threshold; and (2) the real query cannot always be issued before the dummies as otherwise the adversary may identify the real query based on timing.
- Finally, a resource challenge facing our approach is the limited storage and computational capacity of mobile devices, from which many LBS queries are issued and therefore privacy protection must be enforced. In particular, the first challenge requires the dummy insertion process to be aware of the query context, which has to be stored locally on a mobile device because of our design choice of a user-centric approach. To this end, a major challenge is to store and retrieve the query context information in an efficient manner, i.e., with minimal storage *and* computational overhead.

The main technical result in the paper is *Pool-Builder*, a dummy query generation algorithm which takes into account two inputs for generating the dummy service attribute values: the query context, i.e., the set of service attribute values which may be issued from a given location, and the user's motion model, i.e., the set of locations the user may travel to in future snapshots of the continuous LBS query. Based on the inputs, Pool-Builder randomly selects a set of dummy service attribute values such that, even after the exclusion of "unreasonable" dummy values according to all future snapshots of the query, the adversary still cannot compromise the real service attribute value with probability exceeding a pre-determined threshold. Hence, query privacy is guaranteed. Note that Pool-Builder can be readily integrated with any motion model. For example, we consider in the experiments a worst-case model which allows the user to travel to anywhere within a certain distance.

To address the resource limitation of mobile devices, we use a quad-tree based scheme to transform and store the query context information as a bit stream which achieves a high compression ratio and supports efficient retrieval. While the quad-tree scheme was used in our recent work [24], the difference between query context compression and the previous work is subtle and shall be discussed in Section V-B.

C. Contribution and Scope of the Paper

The contributions of this paper are summarized as follows:

- We consider a *novel problem* of query privacy protection for continuous LBS queries.
- We develop DUMMY-Q, a *novel solution* which protects query privacy by generating dummy queries that take into account query context and user motion models. Our scheme is transparent to the service provider, i.e., multiple LBS queries[1] are answered individually and independently.

[1]Users with limited data plans and/or paid LBS may have to bear extra cost(s) due to extraneous data of dummy queries. Unfortunately, a user must make this tradeoff to use DUMMY-Q.

- We also devise a quad-tree based scheme to significantly reduce the storage and computational capacity requirements of DUMMY-Q.
- Our contribution also includes a comprehensive set of experiments which demonstrate the effectiveness of our approach on query privacy protection.

The rest of the paper is organized as follows. We introduce preliminaries in Section II. In Section III, we overview the architecture of DUMMY-Q and define the privacy and utility measures. Sections IV and V are devoted to the Pool-Builder algorithm and the quad-tree based storage scheme, respectively. Section VI presents experimental results, followed by the related work in Section VII and conclusion in Section VIII.

II. Preliminaries

We start by defining continuous LBS queries and describing the problems of existing query privacy protection techniques on addressing such queries. Then, we discuss preliminaries of two inputs taken by DUMMY-Q: user motion models and historical query trends. Finally, we define the adversary model considered in the paper.

A. Continuous LBS Query

An LBS system involves two types of communicating parties - the LBS users and the LBS server, which may also be the adversary. A typical LBS user carries a GPS-enabled mobile device and issues to the LBS server queries of the form

SELECT TOP(k) FROM POI WHERE type = U_{poi} ORDER BY DISTANCE(POI.location, loc) ASC;

after obtaining its location loc. Here POI is the remote database of POIs, k is a pre-determined parameter (e.g., 5 or 10), and U_{poi} is the service attribute value. Let the set of all possible values for U_{poi} be $\mathcal{P}_{\text{all}}$. Since k is fixed, we can represent the query as a 2-tuple $\langle loc, U_{\text{poi}} \rangle$. The query answer returned by the LBS server includes (at most) k POIs in the database which match U_{poi} and are closest to loc.

Continuous LBS queries are recurrent over a period of time and contain the same service attribute value. For example, a user may continuously query for Italian restaurants while driving from office to home, leading to a sequence of LBS queries with the same service attribute value. We therefore define a continuous LBS query as a sequence of its snapshots:

Definition II.1. *(Continuous LBS Query) A continuous LBS query $\mathcal{Q}$ consists of a sequence of 2-tuples $q_1 : \langle loc_1, U_{\text{poi}} \rangle$, $q_2 : \langle loc_2, U_{\text{poi}} \rangle$, ... $q_n : \langle loc_n, U_{\text{poi}} \rangle$, such that $\forall i \in [1, n-1]$, $\langle loc_i, U_{\text{poi}} \rangle$ is issued before $\langle loc_{i+1}, U_{\text{poi}} \rangle$.*

B. Motion Models

An important input to DUMMY-Q is the user's motion model - i.e., a prediction of locations from which the user may issue future snapshots of the continuous query. Formally, we partition the geographical region under consideration into a grid $\mathcal{R}$ with M cells $C_1, \ldots, C_M \in \mathcal{R}$. We shall explain details of the partitioning process in Section III-A. For a continuous LBS query, the motion model predicts $\mathcal{C} = \{C_{t_1}, \ldots, C_{t_n}\}$, where C_{t_i} is the cell which loc_i is predicted to be in. We consider three types of motion models:

1) Precise Trajectory: A user's trajectory $\mathcal{C}$ is known in advance before the first snapshot of a continuous LBS query is issued. This model applies when the user enters its destination to the mobile device (and issues LBS queries along the route).

2) Worst-Case Model: No information about the user's trajectory is available - i.e., random-world assumption applies so $\forall i \in [1, n]$, C_{t_i} is uniformly distributed over all cells in $\mathcal{R}$. The alcoholic example discussed in Section I is a typical scenario where this model applies. Since there can be numerous locations from which an alcoholic queries for bar/tavern over a period of months, one can make no assumption but that the locations are within the boundary of an area (e.g., city).

3) Predictive Trajectory Model: The model generates a probability distribution of $\mathcal{C}$ based on many factors, e.g., user's current location, speed, time-of-day, weather, traffic, etc. Please refer to Section VII for a brief discussion of the existing work for this model.

C. Query Context

A natural method to determine whether a query may be issued from a given location is to check the historical query logs - i.e., queries issued in the past. Specifically, we consider the log as a set of 2-tuples ⟨location, service attribute value⟩. One can compute from the log the *frequency* of a service attribute value U_{poi} for a given cell $C_i \in \mathcal{R}$, i.e., the number of historical queries issued from the cell which feature U_{poi}. We denote the frequency by $\lambda(U_{\text{poi}}, C_i)$. In practice, if no such historical log is available, a simple way to approximate $\lambda(U_{\text{poi}}, C_i)$ is to count the number of POIs with type U_{poi} within C_i, as a user is likely to query for POIs that reside around its location.

D. Privacy Concerns: The Adversary Model

The adversary we consider in the paper is a malicious LBS server which aims to associate a user's identity with the service attribute value U_{poi} specified by the user. Note that when the LBS server receives a user-issued query, it learns not only the payload information, i.e., loc_i and U_{poi}, but also *control information* such as the user's IP address or account information (if login is required for using the LBS). As discussed in Section I, for the purpose of this paper, we consider the worst-case scenario where the user's identity is exposed to the adversary through such control information. Another worst-case assumption we make is that loc_i is transmitted to the LBS server without revision - so as to guarantee that the accuracy of query answer is not affected. Note that this worst-case scenario represents the state of practice of existing LBSes.

One can see that the adversary can identify all n snapshots of the continuous LBS query issued by the user, and link each of them with its exact issuing location. The adversary successfully intrudes privacy if it can infer the real value of U_{poi} from the n snapshot queries. In addition to the queries, the adversary also has pre-knowledge of the query context

information - i.e., $\lambda(U_{\text{poi}}, C_i)$ for all possible values of U_{poi} and $C_i \in \mathcal{R}$.

III. Overview of DUMMY-Q

A. Requirements of Dummy Insertion

DUMMY-Q issues queries with dummy service attributes along with the real query. For each real query $\langle loc_i, U_{\text{poi}} \rangle$ issued by the user, we denote the corresponding set of dummy queries by $\langle loc_i, \mathcal{D}_{\text{pool}} \rangle$ where $\mathcal{D}_{\text{pool}} \subseteq \mathcal{P}_{\text{all}}$ is the set of dummy service attribute values chosen by DUMMY-Q. Recall from Section I that the dummy values must be consistent across different snapshots of the continuous LBS query. Thus, $\mathcal{D}_{\text{pool}}$ does not change over different values of i - i.e., each value in $\mathcal{D}_{\text{pool}}$ is issued as a separate (dummy) LBS query during every snapshot of the continuous query.

Recall from Section I that $\mathcal{D}_{\text{pool}}$ must satisfy two requirements:

- *Utility Requirement:* it must be small enough so as not to incur excessive overhead for LBS query processing, and
- *Privacy Requirement:* the inserted dummy service attribute values should not be identified and removed by the adversary according to the query context information.

To properly address the privacy requirement, it is critical to understand how the adversary may distinguish a dummy query from the real one. Intuitively, the adversary considers a query $\langle loc_i, U \rangle$ as dummy if the number of historical queries which feature U and are issued from the cell C_{t_i} (which contains loc_i) is below a pre-determined threshold. Nonetheless, a key factor here that one must rigidly analyze is how large an area should C_{t_i} cover - e.g., whether it is uniform across the area $\mathcal{R}$ or has a varying size depending upon its location.

Our key observation here is that the size of a cell $C_i \in \mathcal{R}$ should vary according to the *density* of historical queries issued from the cell - e.g., the cell size should be smaller at a downtown area than a rural area. Intuitively, the historical query distribution may change significantly over a short distance in a downtown area. For example, the frequency for "art gallery" to be queried from Greenwich Village, Manhattan, NYC may be much higher than that from Wall Street 2 miles away. To accurately capture the query context information in this case, one needs to define the cell to be smaller than 2×2 miles. On the other hand, such a cell may be too small for a rural area as it might not include any historical query at all.

More generally, the cell size assignment should make each cell contain a roughly equal number of historical queries, such that the query frequency $\lambda(U_{\text{poi}}, C_i)$ is approximately proportional to the probability for U_{poi} to be specified in an LBS query issued from C_i. We shall describe the detailed construction of C_i in Section V-A.

Given a design of $C_1, \ldots, C_M$, we are now ready to formally define how an adversary can distinguish a dummy service attribute value from a real one. In particular, the adversary concludes U to be a dummy iff there exists $i \in [1, n]$ such that $\lambda(U, C_{t_i}) < \sigma$, where σ is a pre-determined parameter which we refer to as the *popularity threshold*. The objective of DUMMY-Q is to ensure that no dummy value in $\mathcal{D}_{\text{pool}}$ can be identified by the adversary as a dummy.

B. System Architecture of DUMMY-Q

Figure 2 depicts the architecture of DUMMY-Q. It consists of four components, *trajectory prediction*, *POOL-BUILDER*, *grid/context retrieval*, and *query generation*.

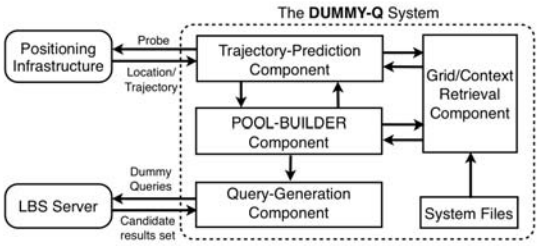


Fig. 2: System Architecture of DUMMY-Q

The trajectory prediction component receives as input the user's current location and, optionally, its previous trajectory from the mobile device's positioning infrastructure. The objective is to generate a set of cells $\tilde{\mathcal{C}} \subseteq \mathcal{R}$ which the user is predicted to travel to and issue future snapshots of the continuous LBS query from. The key technical part of this component is the motion model. Again, while we shall test all three types of motion models discussed in Section II-B in the experiments, the design of these motion models is not the focus of this paper. Note that to transform the predicted trajectory to a set of cells, the trajectory prediction component needs to learn the cell design (i.e., how $\mathcal{R}$ is partitioned into $C_1, \ldots, C_M$) from the grid/context retrieval component.

The POOL-BUILDER component receives the set of possible cells from the trajectory prediction component, and produces $\mathcal{D}_{\text{pool}}$ as its output. To do so, it has to take into account the query context information by learning $\lambda(U, C_i)$ from the grid/context retrieval component.

As such, the responsibility of the grid/context retrieval component includes the storage and retrieval of grid partition as well as query context information. Due to the resource constraint of mobile devices, it must minimize the required storage capacity and the retrieval overhead.

Finally, the query generation component takes $\mathcal{D}_{\text{pool}}$ as input from the POOL-BUILDER component, and generates the corresponding dummy queries which are then issued to the LBS server along with the real query (in a random order). The query generation component receives from the LBS server answers to all issued queries, and then selects the answer to the real query and returns it to the user. Note that the user can be oblivious to the underlying query privacy protection process, and the query accuracy is not affected.

One can see that POOL-BUILDER and grid/context retrieval are the two main components of DUMMY-Q. We shall describe their design in the next two sections, respectively.

IV. POOL-BUILDER Component

In this section, we develop POOL-BUILDER, our algorithm for generating dummy service attribute values $\mathcal{D}_{\text{pool}}$. We first

define the utility and privacy guarantees POOL-BUILDER aims to achieve, and then describe its detailed design.

A. Utility and Privacy Guarantees

Utility Guarantee: Since the query processing overhead is vital to the usability of DUMMY-Q, we maintain a Φ-*query utility guarantee* that the number of dummy values $|\mathcal{D}_{\text{pool}}| \leq \Phi - 1$, i.e., the number of (real and dummy) queries issued to the LBS server at each snapshot is at most Φ. Note that Φ is a pre-determined parameter.

Privacy Guarantee: Ideally, we would like to ensure that the adversary cannot distinguish between any of the Φ received values - i.e., for POOL-BUILDER to achieve the following Φ-*diversity privacy guarantee*.

Definition IV.1. *For a given set of predicted cells $\tilde{\mathcal{C}}$, Φ-diversity is achieved iff at least Φ values in $\mathcal{D}_{\text{pool}}$ exceed the popularity threshold σ over all cells in $\tilde{\mathcal{C}}$ - i.e.,*

$$|\{U|U \in \mathcal{D}_{\text{pool}} \text{ and } \forall C_i \in \tilde{\mathcal{C}}, \lambda(U, C_i) \geq \sigma\}| \geq \Phi. \quad (1)$$

A tacit assumption made in the definition is that the real service attribute value, i.e., U_{poi}, exceeds the popularity threshold over all predicted cells in $\tilde{\mathcal{C}}$. This is a reasonable assumption because otherwise the cell should not be predicted as a possible location from which a future snapshot is issued.

Note that Definition IV.1 is quite stringent when combined with the Φ-query utility guarantee, because all values in $\mathcal{D}_{\text{pool}}$ must then exceed the popularity threshold over all cells in $\tilde{\mathcal{C}}$. This may not be achievable for certain real-world scenarios - e.g., although rare, there might exist a cell which has 95% of its historical queries dominated by a single service attribute value. To address such cases, we consider a relaxation of Φ-diversity privacy guarantee by introducing an additional parameter $\delta \in [0, 1]$, the *factor of commonality*. The following (δ, Φ)-diversity guarantee requires the existence of at least $\delta \cdot |\tilde{\mathcal{C}}|$ predicted cells, where $|\tilde{\mathcal{C}}|$ is the number of cells in $\tilde{\mathcal{C}}$, over which Φ values in $\mathcal{D}_{\text{pool}}$ exceed the popularity threshold.

Definition IV.2. *$\mathcal{D}_{\text{pool}}$ achieves (δ, Φ)-diversity iff $\exists \tilde{\mathcal{C}}_0 \subseteq \tilde{\mathcal{C}}$ such that $|\tilde{\mathcal{C}}_0| \geq \delta \cdot |\tilde{\mathcal{C}}|$ and*

$$|\{U|U \in \mathcal{D}_{\text{pool}} \text{ and } \forall C_i \in \tilde{\mathcal{C}}_0, \lambda(U, C_i) \geq \sigma\}| \geq \Phi. \quad (2)$$

Note that (δ, Φ)-diversity guarantee is reduced to Φ-diversity when $\delta = 1$.

B. Algorithm POOL-BUILDER

The input to POOL-BUILDER includes $|\mathcal{P}_{\text{all}}|$ sets of cells $\mathcal{K}_1, \ldots, \mathcal{K}_{|\mathcal{P}_{\text{all}}|}$ which satisfy $\forall i \in [1, |\mathcal{P}_{\text{all}}|]$,

$$\mathcal{K}_i = \{C_j | C_j \in \mathcal{R}, \lambda(U_i, C_j) \geq \sigma\}. \quad (3)$$

For given parameters Φ and δ, the objective is to find a subset of $\{\mathcal{K}_1, \ldots, \mathcal{K}_{|\mathcal{P}_{\text{all}}|}\}$ with size Φ, denoted by $\mathcal{K}_{g_1}, \ldots, \mathcal{K}_{g_\Phi}$, such that

$$\left| \bigcap_{i=1}^{\Phi} \mathcal{K}_{g_i} \right| \geq \delta \cdot |\tilde{\mathcal{C}}|. \quad (4)$$

If such a subset cannot be found, then the dummy insertion process fails, and DUMMY-Q has to block the real query from being issued. We first prove that this SUBSET-INTERSECTION problem is NP-complete through reduction from the BALANCED BICLIQUE problem [15].

Theorem IV.1. *SUBSET-INTERSECTION is NP-complete.*

Proof: It is easy to see that SUBSET-INTERSECTION is in NP. To prove NP-completeness, we consider reduction from BALANCED BICLIQUE which, for a given bipartite graph $\langle V, E \rangle$, decides if it contains a biclique subgraph with k vertices on each side - i.e., whether there exists two disjoints sets of k vertices $A, B \subseteq V$, such that $\forall a \in A$ and $b \in B$, $(a, b) \in E$. For a given bipartite graph $\langle V, E \rangle$, let the two independent subsets of vertices be V_1 and V_2. Construct $|V_1|$ service attribute values and $|V_2|$ cells to form $\mathcal{P}_{\text{all}}$ and $\mathcal{R}$, respectively. If the two vertices corresponding to U and C are connected in the graph, then set $\lambda(U, C) = \sigma$. Otherwise $\lambda(U, C) = 0$. To decide whether a k-balanced-biclique subgraph exists, set $\Phi = k$ and $\delta = k/|V_2|$. One can see that a k-balanced-biclique exists iff there exists a subset of $\{\mathcal{K}_1, \ldots, \mathcal{K}_{|\mathcal{P}_{\text{all}}|}\}$ with size Φ such that (4) holds. Since BALANCED BICLIQUE is NP-complete [15], SUBSET-INTERSECTION is NP-complete. ■

To address this problem, we devise POOL-BUILDER on heuristic. In particular, we consider a greedy algorithm which chooses $\mathcal{K}_{g_1}, \ldots, \mathcal{K}_{g_\Phi}$ in order, and selects the value of $\mathcal{K}_{g_i}$ which maximizes $|\cap_{j=1}^{i-1} \mathcal{K}_{g_j}|$. We shall demonstrate in the experiments section the effectiveness of POOL-BUILDER.

C. Limitations

We now discuss two known limitations of POOL-BUILDER. Consider a scenario where the server found matching results only for the dummy service attribute values but not the real value U_{poi}. The user might choose to issue another query (e.g., by broadening the service attribute from "Chinese restaurant" to "Asian restaurant") until matching results for U_{poi} are returned. In this case, the adversary may infer from the timing of received queries that the real query in the previous batch must be the one that returns empty - thereby breaching query privacy. A simple solution is to block the second user query from being issued at the same location. Nonetheless, this degrades the utility of LBS.

Another limitation of POOL-BUILDER arises from the following scenario: a user issues an LBS query from the location of a POI that was recently returned in the result of a previous query. In this case, the adversary can infer that the service attribute of the previous query is highly likely to be the type of POI on the user's current location. A possible solution to this problem is *location perturbation* - e.g., the perturbation scheme proposed in our prior work [24] can be readily integrated with DUMMY-Q as it also does not require a trusted third-party.

V. Grid/Context Retrieval Component

Recall that the responsibility of grid/context retrieval component is two-fold: One is to determine the cell $C_i \in \mathcal{R}$ a

given location loc falls into. The other is to compute query context - i.e., retrieve $\lambda(U, C_i)$ which is the frequency of a service attribute value U over cell C_i. We consider these two tasks respectively in the following two subsections.

A. Storage and Retrieval of Grid Partition Information

Recall from Section III-A that the objective of grid partitioning is to ensure that each cell has a roughly constant number of historic queries. To achieve this objective, we consider a recursive partitioning technique as follows: We start with the entire region $\mathcal{R}$ being a cell. We recursively partition a cell into four equal-sized cells iff the number of historical queries issued from within the original cell is greater than a pre-determined threshold. One can see that each cell contains roughly σ or fewer historic queries. Figure 3 depicts an example of such a partition.

As demonstrated in Figure 3, the partitioning scheme can be readily represented as a quad-tree. In particular, each node in the tree has only two possibilities: either it is a leaf node, or it contains 4 children. Thus, to efficiently store the tree, we conduct a breath-first traversal of the tree, storing 1 bit for each node indicating whether it is a leaf node or not. To make this storage scheme scalable to a very large tree, we consider a divide-and-conquer approach with which each file stores a subtree with depth of at most h. Figure 3 shows an example of three divided files when $h = 3$. One can see that with the divide-and-conquer approach, the size of each file is at most $(4^h-4)/3$ bits. The total storage overhead is $O(M+h\cdot M/4^h)$, where M is the total number of cells in the grid. Note that the second item is due to the requirement of storing the size for each file, a cost of the divide-and-conquer approach.

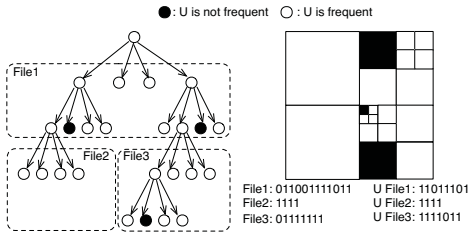


Fig. 3: Example of Grid/Context Storage

In order to retrieve the partition information - i.e., to identify the cell id corresponding to a given location - one needs to conduct a *drill-down* over the quad-tree. In particular, we start from the root node, and find its child that covers the given location. We repeat this process until reaching a leaf node which indicates the cell id corresponding to the location. During this process, we access the files storing all nodes on the path from the root to the cell and reconstruct their corresponding subtrees. One can see that the retrieval process requires access to at most $d/(h-1)$ files, where d is the depth of the tree, leading to time complexity of $O(4^h \cdot d/h)$.

B. Storage and Retrieval of Query Context Information

The storage of query context information can be done alongside the grid partitions. In particular, for each service attribute value U, we only need to store 1-bit information for each *leaf* node in the quad-tree, indicating whether more than σ historic queries of type U have been issued from the cell. This can also be integrated with the divide-and-conquer approach, so that for each file storing the partitioning information, there is a corresponding file for each service attribute value U. An example is shown in Figure 3, with black (resp. white) nodes representing $\lambda(U, C_i) < \sigma$ (resp. $\geq \sigma$). One can see that the storage overhead for query context information is $O(M)$ for each service attribute value, where M is the total number of cells in $\mathcal{R}$. The quad-tree scheme was used in our prior work [24] for storing road density information. A key difference here is the divide-and-conquer approach.

VI. Experiments

A. Experimental Setup

Datasets: We evaluated the performance of DUMMY-Q over the state of Connecticut (5,543 sq. miles), USA. Recall from Section V-A that to partition $\mathcal{R}$ we ideally need historical query logs to compute the total number of queries issued from a cell. To simulate such information, we consider the number of historic queries in a cell to be proportional to its road density - i.e., the total length of roads in the cell. Intuitively, similar to the number of historic queries, road density is higher in downtown areas than rural (or suburban) ones. The road density information for Connecticut was retrieved from the second edition of the Topological Integrated Geographic Encoding and Referencing (TIGER)[2] system published by the US Census Bureau in 2006.

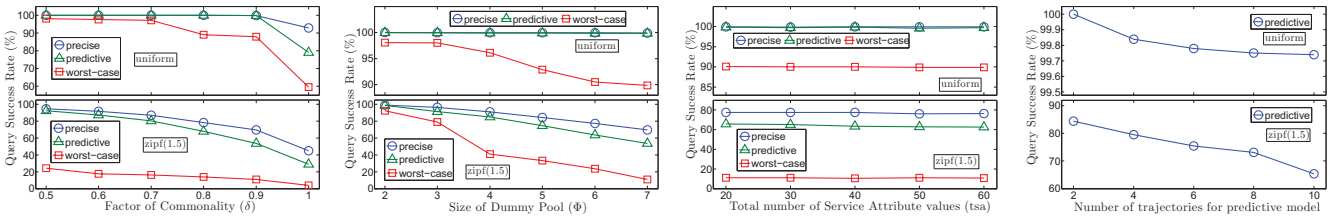
To simulate a continuous LBS query, one needs to generate both the service attribute value and the location for all snapshots. We randomly generated the service attribute value from 60 possible values according to Uniform and/or Zipf (exponent values 1.0 and 1.5) distributions. We chose these two distributions because they are often used to model real-world phenomena. To generate the location snapshots, we used Brinkhoff's Generator [2] to produce 3 million user trajectories for training (i.e., for computing the historic frequencies of each service attribute value) and 200,000 ones for testing the performance of DUMMY-Q.

With our quad-tree based storage scheme, the grid partition and query context information is encoded into a file of 659KB.

Performance Metrics: Since DUMMY-Q does not affect the accuracy of query answers, we considered two performance metrics, both for privacy protection: *Query Success Rate* (QSR) depicts the probability for POOL-BUILDER to successfully generate $\mathcal{D}_{\text{pool}}$ which achieves a given (δ, Φ)-diversity guarantee. The *Average size of Candidate Set* (ACS), on the other hand, captures the average number of service attribute values from which $\Phi - 1$ values can be chosen into $\mathcal{D}_{\text{pool}}$ to achieve (δ, Φ)-diversity guarantee. Intuitively, ACS+1 indicates an upper bound on Φ for a given factor of commonality δ.

Parameters: We varied four parameters in our experiments: i) Factor of Commonality, i.e., δ, with a default value of 0.9,

[2]http://www2.census.gov/geo/tiger/tiger2006se/CT/



(a) QSR vs δ (b) QSR vs Φ (c) QSR vs $\mathcal{P}_{\text{all}}$ (d) QSR vs # of Precise-trajectories

Fig. 4: Comparative study of Motion Models

ii) Size of $\mathcal{D}_{\text{pool}}$, i.e., Φ, with a default value of 7, iii) Length of user's trajectory (in miles), with a default value of 5 miles, and iv) Popularity threshold, i,e. σ, with a default value of 20. In addition, we make our evaluation thorough by varying total number of service attributes values, i.e, $|\mathcal{P}_{\text{all}}|$.

B. Experimental Evaluation

Most of our experiments were performed with the precise-trajectory motion model (Refer Section II-B1). Nonetheless, we start by presenting a comparative evaluation of three different types of motion models where predictive trajectories are generated using historical precise trajectories.

1) Comparing Motion Models: In this section we perform comparative study of our motion models. Intuitively, due to the random-world assumption in the worst-case model, attaining (δ, Φ)-diversity guarantee becomes much more challenging as compared to case where precise-trajectory is known. This intuition clearly resounds in Figures 4(a), 4(b) and 4(c). In addition, it is evident from the aforementioned figures that the predictive-trajectory model performs (much) better as compared to the worst-case model.

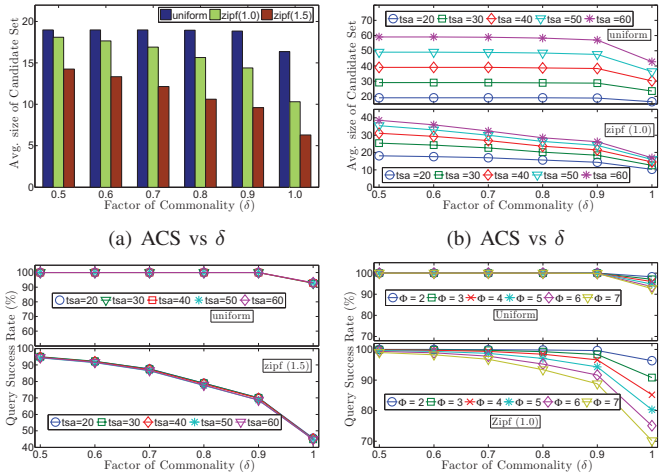
An interesting observation can be made from Figure 4(c) where varying number of service attribute values does not induce any significant change/degradation in the the QSR for any of the motion models. Thus, it can be concluded that DUMMY-Q is readily adaptable if new service attribute values are introduced *or* gain popularity in the LBS market.

Figure 4(d) demonstrates the degradation of QSR for predictive-trajectory model when larger number of precise-trajectories are involved. It can be understood that if the predictive-trajectory model involves a very large number of precise-trajectories in its construction, then the achieved QSR would witness near-equal degradation as in the worst-case model.

2) Effect of Factor of Commonality (δ): Recall from Section IV-A that δ was introduced to relax the stringent requirement posed by the Φ-diversity guarantee. In a nutshell, the higher the δ the greater the challenge to guarantee (δ, Φ)-diversity. It can be clearly observed from Figure 5(a) that even for zipf(1.5) query distribution with the most stringent privacy requirement (i.e., $\delta = 1$) (δ, Φ)-diversity guarantee, a $(1.0, 5+1)$-diversity guarantee is attainable, which perhaps is "reasonable" in practice.

Figure 5(b) depicts the relationship between δ and ACS for different values of $|\mathcal{P}_{\text{all}}|$. Even in this figure $(1.0, 5+1)$-diversity guarantee is attained for all the values of total number of service attributes. Figure 5(c) reaffirms the usability of DUMMY-Q. In particular, when $\delta = 0.8$ and query distribution is zipf(1.5), for at least 75% (i.e., for 150k) trajectories, dummy queries could be successfully issued. Here Φ was set to 7.

We elaborately demonstrate the effect of δ on QSR for various values of Φ in Figure 5(d). It can be observed that the degradation of QSR is more pronounced only when $\delta = 1.0$ for both uniform and zipf distribution of queries. From the study in this section it can be concluded that DUMMY-Q provides "reasonable" usability even for higher values of δ.



(a) ACS vs δ (b) ACS vs δ

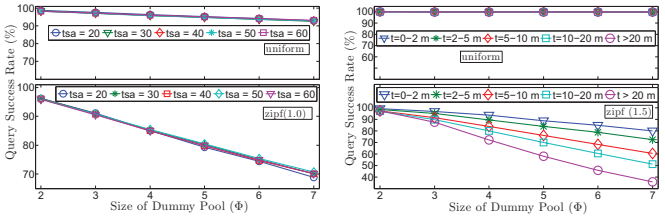
(c) QSR vs δ (d) QSR vs δ

Fig. 5: Effect of Factor of Commonality

3) Effect of Size of Dummy Pool (Φ): As previously observed in Figure 5(d), increase in value of Φ causes degradation of QSR. Nonetheless, Figure 6(a) demonstrates that such degradation is more-or-less similar for different values of $|\mathcal{P}_{\text{all}}|$. As stated previously in Section VI-B1, this behavior depicts our system's adaptability to change/increase in $|\mathcal{P}_{\text{all}}|$.

We also tested for effect of Φ on QSR for varying length of user's trajectory (in miles). We will discuss in detail the effect of length of user's trajectory on ACS and QSR in the following section. Nonetheless, it can be clearly observed from Figure 6(b) that decreasing trend (for zipf(1.5)) in QSR with larger Φ is consistent over trajectories of various lengths.

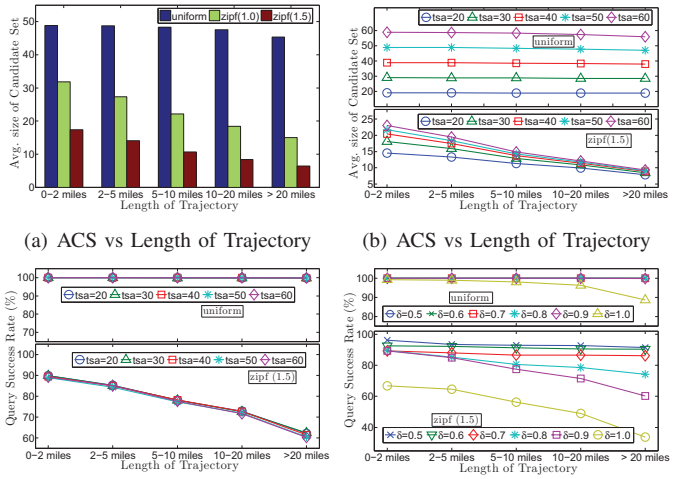
4) Effect of Trajectory Length: Intuitively, it should be more challenging to attain (δ, Φ)-diversity guarantee for a longer trajectory. The reason in simple - a longer trajectory comprises more *cells*, i.e., larger $|\tilde{\mathcal{C}}|$. This intuition is confirmed from all the figures in this section. In Figure 7(a),



(a) QSR vs Φ (b) QSR vs Φ

Fig. 6: Effect of Size of Dummy Pool

it can be clearly observed that ACS drops, marginally for uniform distribution, and highly for zipf distribution, as length of trajectory increases. Here Φ was set to 7. Figure 7(b) depicts the consistency of drop in ACS with increasing length of trajectory for different values of $|\mathcal{P}_{\text{all}}|$.

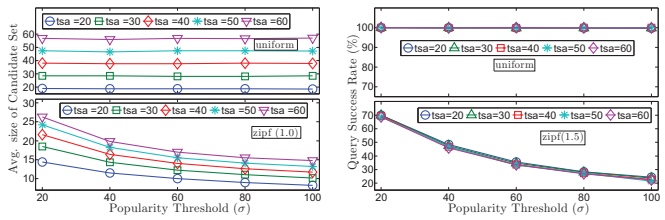


(a) ACS vs Length of Trajectory (b) ACS vs Length of Trajectory

(c) QSR vs Length of Trajectory (d) QSR vs Length of Trajectory

Fig. 7: Effect of Trajectory Length

Figure 7(c) depicts the previously observed behavior of degradation of QSR is more-or-less the same for different values of $|\mathcal{P}_{\text{all}}|$. In addition, we tested for varying values of δ. Figure 7(d) depicts this experiment. It can be clearly observed in Figure 7(d) that setting δ to 1.0 causes rapid degradation of QSR.



(a) ACS vs σ (b) QSR vs σ

Fig. 8: Effect of Popularity Threshold

5) Effect of Popularity Threshold (σ): By intuition, increasing the popularity threshold σ should make it more difficult to achieve higher values of ASC and QSR. In an extreme case, setting $\sigma = 0$ should enable all service attributes in $\mathcal{P}_{\text{all}}$ to be included in $\mathcal{D}_{\text{pool}}$. Contrarily, a very high value σ might prohibit the generation of $\mathcal{D}_{\text{pool}}$ even if δ and Φ are set very low. Figures 8(a) and 8(a) demonstrate the degradation in ACS and QSR respectively for increasing value of σ.

VII. Related Work

Location obfuscation aided by query anonymization has been the traditional approach ([1], [4], [5], [8]–[11], [14], [16], [22], [23], [25]–[27], [30]–[32], [35]) to preserving location and/or query privacy. While most works in this realm utilize a trusted middleware, many other works exist otherwise ([5], [8]–[10], [24], [33]). While works in [5], [9], [10] employ spatial cloaking using P2P infrastructure, technique presented in [24] employs user-centric protection using location perturbation, and the work in [8] employs cryptographic techniques using private information retrieval (PIR) protocol.

Albeit most of the techniques mentioned above are limited to snapshot queries, many works investigate privacy issues in continuous LBS. Pioneering work [12] presented by Gruteser et al., propose to identify sensitive and non-sensitive areas to protect location privacy in continuous location tracking applications. Xu et al. in [31] argued that spatial intersection of different cloaking regions *and* the distribution of users in them indicates user's true location. They proposed entropy-based measure to calculate the level of anonymity achieved. Chow et al. in [4] proposed the property of *memorization* for query privacy is continuous LBS. Here, they suggest that successive cloaking regions must also include locations of all users cloaked previously. This property may however result in very large cloaking regions thus inducing high processing overhead for the server. Since the requirements of continuous location k-anonymity are very stringent, use of historical user-locations was proposed in [32] and [22], in order to generate relatively smaller cloaking regions.

However, it was argued in [29] that if all queries in the user's anonymity set have the same service attribute value, then query privacy will be breached. To this end, the authors proposed to apply the well-known principle of ℓ-diversity [21] in data privacy. In specific, it was proposed that the cloaking region to be sent in the LBS request must comprise of multiple queries that have diverse service attribute values, which are categorized as sensitive and non-sensitve. Liu et al. in [19] however argued over the subjectiveness of the service attributes being sensitive and non-sensitive. To this end, they categorize service attribute values (e.g., burger and pizza can be categorized as fast food) and generate anonymity sets with queries belonging to diverse categories. However, both these techniques focus only on snapshot queries.

Query privacy issues in continuous LBS are similar to challenges in incremental datasets [3] *or* re-publication of data [28]. Specifically, besides the required consistency over recurrent user anonymity sets, maintaining the same set of (diverse) service attributes must be enforced. Recent work by Dewri et al. [6] utilizes the principle of m-*invariance* [28] to generate spatial cloaking regions to realize the aforesaid enforcement. To generate the cloaking regions, they adopted and modified *HilbertCloak* algorithm introduced in [16]. Although effective, their technique requires a trusted middleware/anonymizer.

Moreover, the aforesaid enforcement results in larger cloaking regions and higher anonymization times.

Even though the use of dummies is not popular in LBS privacy literature, few works ([17], [20], [34]) do utilize them. However, differently from our scheme, these works send dummy locations to the LBS server. It was argued by You et al. in [34] that long term tracking of user's trajectory can lead to privacy breach even when dummy locations are utilized. To this end, it was proposed that dummy trajectories cross path with original trajectory, which results in increase in number of possible routes taken by the user. Similar work was presented by Hoh et al. in [13], where path perturbation is performed by crossing paths of users when they are nearby each other.

Destination prediction of a moving user will be a valuable augmentation to our current work since precise-trajectories (refer Section II-B1) may not necessarily always be used. Some works to this end exist [18], [36]. Krumm et al. in [18] presented methodology that considers the likelihood of users visiting previously unobserved locations based on trends in the survey data, e.g., driving efficiency, trip times etc., *and* background properties of locations, e.g., ground and water cover. Ziebart et al. in [36] presented a system title PROCAB that predicts destination(s) based on a given user's partially traveled route and data collected from 25 taxi cab drivers.

VIII. Conclusion

In this paper we have initiated an investigation of the protection of query privacy for continuous LBS queries. We proposed DUMMY-Q, a dummy query generation scheme which takes into account the user's motion model and the query context information as identified from the historical query logs. We proved that the problem of achieving both utility and privacy guarantee for dummy insertion is NP-complete, and devised POOL-BUILDER on heuristic to generate the dummy queries. We also developed a quad-tree based storage/retrieval scheme for query context information to significantly reduce the storage and computational overhead of DUMMY-Q. Finally, we described a comprehensive set of experiments that demonstrate the effectiveness of our approach over a real-world map and various motion models, including a synthetic traffic generator.

Acknowledgement

This research is supported in part by the US NSF under grants 0324988, 0329181, 0721571, 0808419, 0852673, 0852674, 0907964, 0915795, 0915834, 0942113, 0943479, 0958477, by the 973 Program of China under grant 2011CB302800, and by George Washington University under a Research Enhancement Fund. Any opinions, findings, conclusions, and/or recommendations expressed in this material, either expressed or implied, are those of the authors and do not necessarily reflect the views of the sponsors listed above.

References

[1] C. Bettini, X. Wang, and S. Jajodia. Protecting privacy against location-based personal identification. In *SDM (VLDB Workshops)*, 2005.

[2] T. Brinkhoff. A framework for generating network-based moving objects. In *GeoInformatica*, 2002.

[3] J. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *SDM (VLDB Workshops)*, 2006.

[4] C. Chow and M. Mokbel. Enabling private continuous queries for revealed user locations. *SSTD*, 2007.

[5] C. Chow, M. F. Mokbel, and X. Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *ACM GIS*, 2006.

[6] R. Dewri, I. Ray, I. Ray, and D. Whitley. Query m-invariance: Preventing query disclosures in continuous location-based services. In *MDM*, 2010.

[7] B. Gedik and L. Liu. A customizable k-anonymity model for protecting location privacy. In *ICDCS*, 2005.

[8] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, Tan, and Kian-Lee. Private queries in location based services: Anonymizers are not necessary. In *SIGMOD*, 2008.

[9] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Mobihide: a mobilea peer-to-peer system for anonymous location-based queries. In *SSTD*, 2007.

[10] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Prive: anonymous location-based queries in distributed mobile systems. In *WWW*, 2007.

[11] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *MobiSys*, 2003.

[12] M. Gruteser and X. Liu. Protecting privacy, in continuous location-tracking applications. *Security and Privacy*, 2004.

[13] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. In *SecureComm*, 2005.

[14] H. Hu and J. Xu. Non-exposure location anonymity. In *ICDE*, 2009.

[15] D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 8(3):438–448, 1987.

[16] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *TKDE*, 2007.

[17] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *ICPS*, 2005.

[18] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp*, 2006.

[19] F. Liu, K. Hua, and Y. Cai. Query l-diversity in location-based services. In *MDM*, 2009.

[20] H. Lu, C. S. Jensen, and M. L. Yiu. Pad: privacy-area aware, dummy-based location privacy in mobile services. In *MobiDe*, 2008.

[21] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[22] S. Mascetti, C. Bettini, X. Wang, D. Freni, and S. Jajodia. Providenthider: An algorithm to preserve historical k-anonymity in lbs. In *MDM*, 2009.

[23] M. F. Mokbel, C. Y. Chow, and W. G. Aref. The new casper: Query processing for location services without compromising privacy. In *VLDB*, 2006.

[24] A. Pingley, W. Yu, N. Zhang, X.Fu, and W. Zhao. CAP: A Context-Aware Privacy-Preserving LBS System. In *ICDCS*, 2009.

[25] D. Riboni, L. Pareschi, C. Bettini, and S. Jajodia. Preserving anonymity of recurrent location-based queries. In *TIME*, 2009.

[26] T.Xu and Y.Cai. Feeling-based location privacy protection for location-based services. In *CCS*, 2009.

[27] T. Wang and L. Liu. Privacy-aware mobile services over road networks. In *VLDB Endowment*, 2009.

[28] X. Xiao and Y. Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *SIGMOD*, 2007.

[29] Z. Xiao, J. Xu, and X. Meng. p-sensitivity: A semantic privacy-protection model for location-based services. In *PALMS (MDM Worskshops)*, 2008.

[30] J. Xu, X. Tang, H. Hu, and J. Du. Privacy-conscious location-based queries in mobile environments. *TPDS*, 2009.

[31] T. Xu and Y. Cai. Location anonymity in continuous location-based services. In *GIS*, 2007.

[32] T. Xu and Y. Cai. Exploring historical location data for anonymity preservation in location-based services. In *INFOCOM*, 2008.

[33] M. L. Yiu, C. Jensen, X. Huang, and H. Lu. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *ICDE*, 2008.

[34] T. You, W. Peng, and W. Lee. Protecting moving trajectories with dummies. In *PALMS (MDM Worskshops)*, 2007.

[35] G. Zhong and U. Hengartner. A distributed *k*-anonymity protocol for location privacy. In *PerCom*, 2009.

[36] B. Ziebart, A. Maas, A. Dey, and J. Bagnell. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior. In *UbiComp*, 2008.