# The New Casper: Query Processing for Location Services without Compromising Privacy *

Mohamed F. Mokbel[1] Chi-Yin Chow[1] Walid G. Aref[2]

[1]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN
[2]Department of Computer Science, Purdue University, West Lafayette, IN

## ABSTRACT

This paper tackles a major privacy concern in current location-based services where users have to continuously report their locations to the database server in order to obtain the service. For example, a user asking about the nearest gas station has to report her exact location. With untrusted servers, reporting the location information may lead to several privacy threats. In this paper, we present *Casper*[1]; a new framework in which mobile and stationary users can entertain location-based services without revealing their location information. *Casper* consists of two main components, the *location anonymizer* and the *privacy-aware query processor*. The *location anonymizer* blurs the users' exact location information into cloaked spatial regions based on user-specified privacy requirements. The *privacy-aware query processor* is embedded inside the location-based database server in order to deal with the cloaked spatial areas rather than the exact location information. Experimental results show that *Casper* achieves high quality location-based services while providing anonymity for both data and queries.

## 1. INTRODUCTION

The explosive growth of *location-detection* devices (e.g., cellular phones, GPS-like devices, and RFIDs) results in a wide spread of location-based applications. Examples of these applications include location-based store finders (*"Where is my nearest restaurant"*), traffic reports (*"Let me know if there is congestion within ten minutes of my route"*), and location-based advertisements (*"Send e-coupons to all cars that are within two miles of my gas station"*). Registered users with location-based services continuously send their location information to the location-based database server. Upon requesting a service, a registered user has to issue a location-based query that is executed at the server based on the knowledge of the user location [24, 33, 36, 45].

Although location-based applications along with the location-based query processing promise safety and convenience, they threaten the privacy and security of their customers. The location-based query processor relies mainly on the implicit assumption that users agree to reveal their private locations. In order to get a location-based service, a user has to report her location. In other words, a user trades her privacy with the service. If a user wants to keep her private location information, she has to turn-off her location-aware device and (temporarily) unsubscribe from the service. With untrustworthy servers, such model provides several privacy threats. For example, an employer may check on her employee behavior by knowing the places she visits and the time of each visit, the personal medical records can be inferred by knowing which clinic a person visits, or someone can track the locations of his ex-friends. In fact, in many cases, GPS devices have been used in stalking personal locations [15, 43]. The traditional approach of *pseudonymity* (i.e., using a fake identity) [37] is not applicable to location-based applications where a location of a person can directly lead to the true identity. For example, asking about the nearest Pizza restaurant to my home using a fake identity will reveal my true identity, i.e., a resident of the home.

In this paper, we tackle such privacy-leakage models as we propose *Casper*[1]; a novel framework that turns traditional location-based servers and query processors to provide anonymous service to their customers. In *Casper*, mobile users can entertain location-based services without the need to reveal their private location information. Upon registration with *Casper*, mobile users specify their convenient level of privacy through a user-specified *privacy profile*. A user privacy profile includes two parameters $k$ and $A_{min}$. $k$ indicates that the mobile user wants to be $k$-anonymous, i.e., not distinguishable among other $k$ users while $A_{min}$ indicates that the user wants to hide her location information within an area of at least $A_{min}$. Large values for $k$ and $A_{min}$ indicate more strict privacy requirements.

*Casper* mainly consists of two components, namely, the *location anonymizer* and the *privacy-aware query processor*. The *location anonymizer* is a trusted third party that acts as a middle layer between mobile users and the location-based database server in order to: (1) receive the exact location information from mobile users along with a *privacy profile* of each user, (2) blur the exact location information into

*Mohamed Mokbel's research is supported by the Grants-in-Aid, University of Minnesota. Walid Aref's research is supported in part by the National Science Foundation under Grants IIS-0093116 and IIS-0209120.

[1] *Casper*, the friendly ghost, is a 1960's cartoon character for a friendly ghost that can hide its location and help people. (http://www.toontracker.com/casper/newcaspr.htm).


*VLDB '06,* September 12-15, 2006, Seoul, Korea.

cloaked spatial areas based on each user *privacy profile*, and (3) send the cloaked spatial areas to the location-based database server. The *privacy-aware query processor* is embedded inside the location-based database server to tune its functionality to deal with anonymous queries and cloaked spatial areas rather than the exact location information. We identify three novel query types that are supported by *Casper*: (1) Private queries over public data, e.g., *"Where is my nearest gas station"*, in which the person who issues the query is a private entity while the data (i.e., gas stations) are public, (2) Public queries over private data, e.g., *"How many cars in a certain area"*, in which a public entity asks about personal private locations, and (3) Private queries over private data, e.g., *"Where is my nearest buddy"* in which both the person who issues the query and the requested data are private. With this classification in mind, traditional location-based query processors can support only public queries over public data. Due to the lack of the exact location information at the server, the anonymous query processor provides a *candidate list* of answers instead of a single exact answer. We prove that our candidate list is *inclusive*, i.e., contains the exact answer, and is *minimal*, i.e., a high quality answer is given to the users. In general, the contributions of this paper can be summarized as follows:

- We introduce *Casper* as a novel paradigm that allows mobile users to anonymously entertain location-based services by specifying their anonymity requirements through a user *privacy profile*.
- We introduce the *location anonymizer* that blurs the location information for mobile users into cloaked spatial areas that match each user privacy profile.
- We identify three novel query types that are not supported by existing location-based servers, namely, *private queries over public data*, *public queries over private data*, and *private queries over private data*.
- We introduce a *privacy-aware query processor* that provides a unified framework for supporting the new query types. We prove that our *privacy-aware query processor* results in an *inclusive* and *minimal* answer.
- We provide experimental evidence that our proposed paradigm *Casper* is *efficient* in terms of query processing time and cloaking time, is *scalable* in terms of supporting large numbers of mobile users, and is *privacy-aware* where it provides a high-quality answer without the need for the exact location information.

The rest of the paper is organized as follows. Section 2 highlights the related work to *Casper*. The *Casper* architecture is outlined in Section 3. The two main components of *Casper*, the *location anonymizer* and the *privacy-aware query processor* are described in Sections 4 and 5, respectively. Extensive experimental evaluation of *Casper* is presented in Section 6. Finally, Section 7 concludes the paper.

## 2. RELATED WORK

The $k$-anonymity model [39, 40] has been widely used in maintaining data privacy (e.g., see [7, 29, 30, 31]). The main idea is to have each record in the database $k$-anonymous, i.e., not distinguishable among other $k$ records. Although, we aim for similar $k$-anonymity model for our mobile users, none of these techniques can be applied to the case of location-based queries, mainly for the following four reasons: (1) These techniques preserve the privacy of the stored data. In *Casper*, we aim not to store the data at all. Instead, we store *perturbed* versions of the data. Thus, data privacy is managed before storing the data. (2) These approaches protect the data not the queries. In *Casper*, we aim to protect the person who issues the query. For example, a person who wants to ask about her nearest ATM machine needs to protect her location while the ATM location information is not protected. (3) These approaches guarantee the $k$-anonymity for a snapshot of the database. In location-based environments, data and queries are continuously updated with high rates. Such dynamic behavior calls for continuous maintenance of the $k$-anonymity model. (4) These approaches assume a unified $k$-anonymity requirement for all the stored records. In *Casper*, $k$-anonymity is a user-specified privacy requirement which may have a different value for each user.

Motivated by the privacy threats of location-detection devices [1, 6, 8, 44], several research efforts are dedicated to protect the user location privacy (e.g., false dummies [27], landmark objects [21], and location perturbation [13, 16, 17]). However, none of these approaches have addressed the query processing issue, i.e., getting an anonymous service from location-based applications. Several architectures have been explored to provide secure data transformation from the client to the server machines (e.g., secure-multiparty communication [12], minimal information sharing [3], untrusted third party [14], and trusted third party [2, 23]). Among these models, *Casper* employs the trusted third party model as it requires less computation overhead and is more suitable for real-time query processing. The trusted third party model is already utilized by existing location privacy techniques (e.g., [8, 16, 17]) and is commercially applied in other fields. For example, the Anonymizer [4] is responsible for private web surfing to internet users.

While our *privacy-aware query processor* is unique in nature, there are previous attempts to provide similar functionalities to our *location anonymizer*. The closest approaches to ours are the spatio-temporal cloaking [17] and the CliqueCloak algorithm [16]. The spatio-temporal cloaking assumes that all users have the same $k$-anonymity requirements. For each user location update, the spatial space is recursively divided in a KD-tree-like format till a suitable subspace is found. Such technique lacks scalability as it deals with each single movement of each user individually. The CliqueCloak algorithm assumes a different $k$-anonymity requirement for each user where it combines a set of users together and constructs a clique graph to decide that some users can share the cloaked spatial area. The cloaked spatial area for combined users is their minimum bounding rectangle. This approach suffers from a major privacy threat where it reveals some information about the possible user locations (e.g., some users should be lying on the rectangle boundary). In addition, due to the computation overhead of computing the clique graph, this approach is limited to a small number of users with small $k$-anonymity requirements, i.e., $k$ is from 5 to 10. Our *location anonymizer* distinguishes itself from these two approaches as it: (1) Provides a customizable privacy profile for each mobile user that contains the $k$-anonymity and minimum cloaked area $A_{min}$ requirements, (2) Scales well to a large number of mobile users with arbitrary privacy profiles, and (3) Cannot be reverse engineered to give any information about the exact user location.

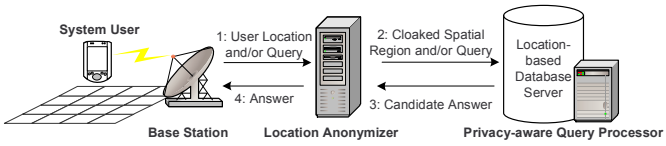



**Figure 1: The *Casper* system architecture**

# 3. CASPER SYSTEM ARCHITECTURE

Mobile users register with *Casper* by a certain *privacy profile* that outlines the privacy requirements of each user. A user privacy profile is defined as a tuple $(k, A_{min})$, where $k$ indicates that the user wants to be $k$-anonymous, i.e., not distinguishable among other $k$ users, while $A_{min}$ is the minimum acceptable resolution of the cloaked spatial region. $A_{min}$ is particularly useful in dense areas where even a large $k$ would not achieve higher privacy requirements. Mobile users have the ability to change their *privacy profiles* at any time. Our employed privacy profile matches the privacy requirements of mobiles users as depicted by several social studies (e.g., see [6, 18, 20, 26, 32]).

Figure 1 depicts the *Casper* system architecture which has two main components: the *location anonymizer* and the *privacy-aware query processor*. The *location anonymizer* receives continuous location updates from mobile users, blurs the location updates to cloaked spatial areas that match each user privacy profile $(k, A_{min})$, and sends the cloaked spatial areas to the location-based database server. While *cloaking* the location information, the anonymizer also removes any user identity to ensure the pseudonymity of the location information [37]. Similar to the exact point locations, the *location anonymizer* also blurs the query location information before sending a cloaked query area to the location-based database server.

The *privacy-aware query processor* is embedded inside the location-based database server to anonymously deal with cloaked spatial areas rather than exact point locations. Instead of returning an exact answer, the *privacy-aware query processor* returns a *candidate list* of answers to the location-based query through the *location anonymizer*. Mobile users would locally evaluate their queries given the candidate list. The *privacy-aware query processor* guarantees that the candidate list is of minimal size and contains the exact query answer. The size of the candidate list heavily depends on the user *privacy profile*. A strict privacy profile would result in a large candidate list. Using their *privacy profiles*, mobile users have the ability to adjust a personal trade-off between the amount of information they would like to reveal about their locations and the quality of service that they obtain from *Casper*. Location-based queries processed at the *privacy-aware* location-based database server may be received either from the mobile users or from public administrators. Queries that come from mobile users are considered as *private queries* and should pass by the *location anonymizer* to hide the query identity and blur the location of the user who issues the query. Location-based queries that are issued from public administrators are considered as *public queries* and do not need to pass through the *location anonymizer*, instead, they are directly submitted to the location-based database server. The database server will answer such public queries based on the stored blurred location information of all mobile users.

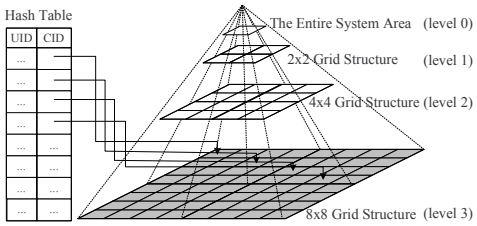



**Figure 2: The *basic* location anonymizer.**

# 4. THE LOCATION ANONYMIZER

As depicted in Figure 1, the *location anonymizer* blurs the exact point location information $p$ of each mobile user to a spatial region $R$ that satisfies each user privacy profile. To avoid the drawbacks of previous location anonymizers [16, 17], we identify the following four requirements that we aim to satisfy in our *location anonymizer*:

1. **Accuracy.** The *cloaked* region $R$ should be of area $A_R$ and contain $k_R$ users that satisfy and as close as possible to the user profile (i.e., $k_R \gtrsim k$, $A_R \gtrsim A_{min}$).
2. **Quality.** An adversary can only know that the exact user location information could be equally likely anywhere within the spatial cloaked region $R$.
3. **Efficiency.** The *cloaking* algorithm should be computationally efficient and scalable. It should be able to cope with the continuous movement of large numbers of mobile users and real-time requirements of spatio-temporal queries.
4. **Flexibility.** Each registered user with the location anonymizer should have (1) the ability to specify her own privacy requirements and (2) the ability to change her requirements at any time.

Notice that the spatio-temporal cloaking algorithm in [17] can support only the quality requirement, while the CliqueCloak algorithm in [16] can partially support the accuracy and flexibility requirements in terms of the $k$-anonymity parameter. In the rest of this section, we present two alternative techniques for our *location anonymizer* that satisfy the above four requirements, namely, the *basic* location anonymizer and the *adaptive* location anonymizer.

## 4.1 The Basic Location Anonymizer

**Data structure.** Figure 2 depicts the data structure for the *basic* location anonymizer. The main idea is to employ a grid-based complete pyramid data structure [41] that hierarchically decomposes the spatial space into $H$ levels where a level of height $h$ has $4^h$ grid cells. The root of the pyramid is of height zero and has only one grid cell that covers the whole space. Each pyramid cell is represented as $(cid, N)$ where $cid$ is the cell identifier while $N$ is the number of mobile users within the cell boundaries. The pyramid structure is dynamically maintained to keep track of the current number of mobile users within each cell. In addition, we keep track of a hash table that has one entry for each registered mobile user with the form $(uid, profile, cid)$ where $uid$ is the mobile user identifier, $profile$ is the user's privacy profile, and $cid$ is the cell identifier in which the mobile user is located. $cid$ is always at the lowest level of the pyramid (the shaded level in Figure 2).

**Algorithm 1** Bottom-up cloaking algorithm

```
1: Function BottomUp-Cloaking(k, A_min, cid)
2: if cid.N ≥ k and cid.Area ≥ A_min then
3:    return Area(cid);
4: end if
5: cid_V ← The vertical neighbor cell of cid.
6: cid_H ← The horizontal neighbor cell of cid.
7: N_V = cid.N + cid_V.N, N_H = cid.N + cid_H.N
8: if (N_V ≥ k OR N_H ≥ k) AND 2cid.Area ≥ A_min then
9:    if (N_H ≥ k AND N_V ≥ k AND N_H ≤ N_V) OR N_V <
      k then
10:      return Area(cid) ∪ Area(cid_H);
11:   else
12:      return Area(cid) ∪ Area(cid_V);
13:   end if
14: else
15:   BottomUp-Cloaking((k, A_min), Parent(cid));
16: end if
```

**Maintenance.** Due to the highly dynamic environment of location-based applications, any employed data structure should be sustainable to frequent updates. A location update is sent to the *location anonymizer* in the form $(uid, x, y)$ where $uid$ is the user identifier, $x$ and $y$ are the spatial coordinates of the user's new location. Once the update is received at the *location anonymizer*, a hash function $h(x, y)$ is applied to get the user cell identifier $cid_{new}$ at the lowest grid layer. Then, the user entry in the hash table is checked to get its original cell identifier $cid_{old}$. If the old cell identifier matches the new one ($cid_{old} = cid_{new}$), then there is no need to do any more processing. If there is a change in the cell identifier ($cid_{old} \neq cid_{new}$), three operations should take place: (1) Update the new cell identifier in the hash table, (2) Update the counters $N$ in both the old and new pyramid grid cells, and (3) If necessary, propagate the changes in the cell counters $N$ for higher pyramid layers. If a new user is registered, a new entry will be created in the hash table and the counters of all the affected grid cells in the pyramid structure are increased by one. Similarly, if an existing user quits, its entry is deleted from the hash table and the counters of all affected grid cells are decreased by one.

**The *cloaking* algorithm.** Algorithm 1 depicts a bottom-up cloaking algorithm for the grid-based pyramid structure. The input to the algorithm is the user privacy profile $(k, A_{min})$ and the cell identifier $cid$ for the gird cell that the user is currently in. $k$ and $A_{min}$ should be less than the total number of users registered in the system and the total spatial area, respectively. If the initially given cell already satisfies the user privacy requirement, i.e., $cid.N \geq k$ (the number of users within the cell is greater than $k$) and $cid.Area \geq A_{min}$ (the cell area is larger than $A_{min}$), we return the cell $cid$ as the spatial cloaked area (Line 3 in Algorithm 1). If this is not the case, we check for the vertical and horizontal neighbor cells to cell $cid$. Two cells are considered neighbors to each other if they have the same parent and lie in a common row (horizontal neighbor) or column (vertical neighbor). Thus, each cell has only one horizontal and one vertical neighbors. If the combination of the cell $cid$ with any of its neighbors yields a spatial region that satisfies the anonymity requirement, we return the combination that gives closer value to $k$ (Lines 5 to 13 in Algorithm 1). If none of the neighbors can be combined with cell $cid$, then the algorithm is recursively executed with the parent cell of $cid$ till a valid cell is returned (Line 15 in Algorithm 1).

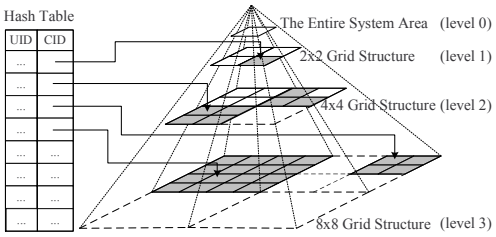



**Figure 3: The *adaptive* location anonymizer.**

## 4.2 The Adaptive Location Optimizer

The basic location anonymizer incurs high cost for both location updates and cloaking time. For location updates, when a user changes her cell from $cid_1$ to $cid_2$, a set of updates need to be propagated from $cid_1$ and $cid_2$ at the lowest level till a common parent is reached. For the cloaking time, Algorithm 1 has to start from the lowest level regardless of the user privacy requirements. To avoid these drawbacks, we introduce the *adaptive* location anonymizer where the pyramid structure is *adaptively* maintained to certain levels that match the privacy requirements of existing users.

**Data Structure.** Figure 3 depicts the data structure for the *adaptive* location anonymizer that mainly utilizes an incomplete pyramid structure [5]. The contents of each grid cell and the hash table are exactly similar to those of Figure 2. The main idea of the incomplete pyramid structure is to maintain only those grid cells that can be potentially used as cloaking regions for the mobile users. For example, if all mobile users have strict privacy requirements where the lowest pyramid level would not satisfy any user privacy profile, the *adaptive* location anonymizer will not maintain such level, hence, the cost of maintaining the pyramid structure is significantly reduced. The shaded cells in Figure 3 indicate the lowest level cells that are maintained (Compare to Figure 2 where all shaded cells are only in the lowest level). As an example, in the second topmost level, the right bottom quadrant is shaded indicating that all users in this quadrant have strict privacy requirements that will not be satisfied by any lower grid cell. Notice that it is not necessary to extend a whole quadrant. For example, in the lowest level, there are four shaded cells in the upper-right corner to indicate that these cells have the most relaxed user privacy requirements. Instead of having the hash table pointing to the lowest pyramid level as in Figure 2, in the *adaptive* location anonymizer, the hash table points to the lowest maintained cells which may not be at the lowest pyramid level.

**Maintenance.** In addition to the regular maintenance procedures as that of the *basic* location anonymizer, the *adaptive* location anonymizer is also responsible on maintaining the shape of the incomplete pyramid. Due to the highly dynamic environment, the shape of the incomplete pyramid may have frequent changes. Two main operations are identified to maintain the efficiency of the incomplete pyramid structure, namely, cell *splitting* and cell *merging*.

*Cell splitting.* A cell $cid$ at level $i$ needs to be split into four cells at level $i + 1$ if there is at least one user $u$ in $cid$ with a privacy profile that can be satisfied by some cell at level $i + 1$. To maintain such criterion, we keep track of the most relaxed user $u_r$ for each cell. If a newly coming object $u_{new}$ to the cell $cid$ has more relaxed privacy requirement than $u_r$, we check if splitting cell $cid$ into four cells at level

$i+1$ would result in having a new cell that satisfies the privacy requirements of $u_{new}$. If this is the case, we will split cell *cid* and distribute all its contents to the four new cells. However, if this is not the case, we just update the information of $u_r$. In case one of the users leaves cell *cid*, we just update $u_r$ if necessary.

*Cell merging.* Four cells at level $i$ are merged into one cell at a higher level $i-1$ only if all the users in the level $i$ cells have strict privacy requirements that cannot be satisfied within level $i$. To maintain this criterion, we keep track of the most relaxed user $u'_r$ for the four cells of level $i$ together. If such user leaves these cells, we have to check upon all existing users and make sure that they still need cells at level $i$. If this is the case, we just update the new information of $u'_r$. However, if there is no need for any cell at level $i$, we merge the four cells together into their parent cell. In the case of a new user entering cells at level $i$, we just update the information of $u'_r$ if necessary.

The high cost of cell splitting and merging is amortized by the huge saving in continuous updates and maintenance of large numbers of non-utilized grid cells as in the case of the *basic* location anonymizer. However, the basic assumption is that mobile users are moving at reasonable speeds. Thus, cell splitting and merging are not very frequent events. In the case of extremely high speed, e.g., each user movement results in a cell change, the *basic* location anonymizer would have less maintenance cost than the *adaptive* one.

**The *cloaking* algorithm.** The cloaking algorithm for the *adaptive* location anonymizer is exactly similar to Algorithm 1. The only difference is that the input to the algorithm is a cell *cid* from the lowest maintained level rather than a cell from the lowest pyramid level. Thus, the number of recursive calls of the algorithm (Line 15 in Algorithm 1) is greatly reduced. In fact, in many cases, we may not need any recursive calls.

## 4.3 Discussion

Both the *basic* and *adaptive* location anonymizers satisfy the four requirements that we have set at the beginning of this section, namely, *accuracy*, *quality*, *efficiency*, and *flexibility*. In terms of ***accuracy***, the ability to maintain large numbers of small size grid cells along with searching for horizontal and vertical neighbor grid cells help in achieving higher accuracy for large number of users of various privacy requirements. For ***quality***, the fact that we utilize a pre-defined space partitioning scheme (i.e., the pyramid structure) guarantees that the cloaked spatial area is completely independent form the data. Thus, an adversary cannot guess any information about the exact user location other than that the probability that the user is located at a certain point in the cloaked region $R$ is $\frac{1}{Area(R)}$ for all points within $R$, i.e., the possible user location is uniformly distributed over the cloaked region $R$. Compared to previous algorithms [16, 17], the ***efficiency*** of the pyramid-based location anonymizer is obvious as its pre-computed grid cells scale well to large numbers of mobile users. The main idea is that the *cloaking* time is considerably low as the space is already partitioned while the update time is optimized through the efficient grid structure. With respect to ***flexibility***, our location-anonymizer allows each user to specify her convenient privacy requirements through the privacy profile $(k, A_{min})$. In addition, a user has the flexibility to change her privacy profile anytime.

# 5. PRIVACY-AWARE QUERY PROCESSING

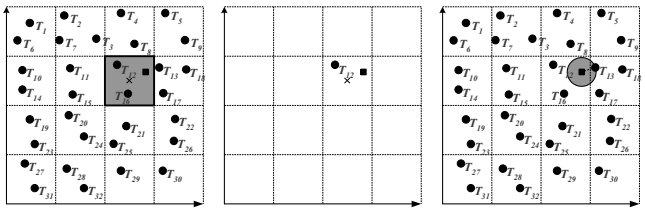
As depicted in Figure 1, the *privacy-aware* query processor is embedded inside the location-based database server. The main goal of the *privacy-aware query processor* is to provide highly efficient, accurate, and anonymous location-based services based on the knowledge of the cloaked spatial areas rather than the exact location information. Two data types are stored in the *privacy-aware* location-based database server, *public* data and *private* data. *Public* data includes stationary objects such as hospitals, restaurants, and gas stations or moving objects such as police cars and on-site workers. Such persons and facilities do not want to hide their location information. Thus, they are stored directly in the location-based database server without interference from the location anonymizer. *Private* data mainly contains personal information of mobile or stationary users with a *privacy profile* of non-zero $k$ or non-zero $A_{min}$. Such data are received at the *privacy-aware* location-based database server as cloaked spatial regions from the location anonymizer. Based on the stored data, we identify three novel query types that are supported in *Casper* through its *privacy-aware query processor*:

- ***Private queries over public data.*** Examples of these queries include a person (private query) asking about her nearest gas station (public data). In this case, the *privacy-aware query processor* does not have the exact location of the person who issues the query while the exact locations of gas stations are known.

- ***Public queries over private data.*** Examples of these queries include an administrator (public query) asking about the number of mobile users (private data) in a certain area. In this case, the *privacy-aware query processor* knows the exact query information, yet it does not know the exact locations of mobile users.

- ***Private queries over private data.*** Examples of these queries include a person (private query) asking about her nearest buddy (private data). Both the exact locations of the person and her buddies are not available at the *privacy-aware query processor*.

With this classification in mind, traditional location-based database servers (e.g., [19, 35, 45]) can support only *public queries over public data* where the complete knowledge of location information of both data and queries are available. In the rest of this section, we will focus on the first and third query types that involve *private queries*. The second query type can be considered as a special case of the third query type where the query area is exactly known. Without loss of generality, in this section, we will focus on nearest-neighbor queries as it is one of the most important and challenging location-based queries. Extensions of the proposed approaches to other location-based spatio-temporal queries, e.g., range queries and aggregates are straightforward. Also, we focus mainly on evaluating a snapshot answer of the above novel query types. Supporting continuous queries and large numbers of outstanding queries can be achieved by seamless integration of the *Casper* framework into any scalable and/or incremental location-based query processor (e.g., see [22, 25, 36, 34, 47, 48, 49]).



(a) The Server (b) Client I (c) Client II

**Figure 4: Naive privacy-aware query processing.**

## 5.1 Private Queries over Public Data

In this section, we will consider a nearest-neighbor query issued by a user $u$ in the form "What is my nearest gas station?". In this case, the *privacy-aware query processor* does not know the exact location information of $u$. Instead, the query processor knows only a cloaked spatial area that $u$ is located in. On the other hand, the exact location information of gas stations is known. Figure 4a depicts such scenario by showing the data stored at the server side. There are 32 target objects, i.e., gas stations, $T_1$ to $T_{32}$ represented as black circles, the shaded area represents the cloaked spatial area of the user who issued the query. For clarity, the actual user location is plotted as a black square inside the cloaked area. However, such information is not stored at the server.

Figures 4b and 4c give two naive approaches that represent two different extremes for evaluating private queries over public data. In the first approach, the server computes the nearest target object to the center of the cloaked area ($T_{12}$) and sends the answer directly to the client. Although this approach minimizes the data transmitted from the server to the client, it gives an inaccurate answer where the actual nearest target to the client is $T_{13}$. In the second approach, the server sends all target objects to the client. Then, the client evaluates her query locally to get $T_{13}$ as its exact answer. Although this approach provides exact answer, it is not practical due to the overhead of transmitting large numbers of target objects and the limited capabilities at the client side.

Our approach is considered a compromise between these two extremes. The main idea is to compute a *candidate list* of answers to be sent to the client. Then, the client will evaluate her query locally over the received *candidate list* in order to get the exact answer. We prove that our approach is scalable and efficient by proving that the computed *candidate list* is *inclusive*, i.e., contains the exact answer, and is of *minimal* size.

### 5.1.1 Algorithm for Nearest-Neighbor Queries

The main idea of our algorithm is to initially select a set of *filter* target objects that can be used to prune the search over the whole set of target objects. Using the *filter* objects, we identify a spatial search space $A_{EXT}$ that covers all the possible areas that may include a potential answer to the nearest-neighbor query regardless of the exact user location in the cloaked area $A$. Finally, all target objects that lie inside $A_{EXT}$ are returned to the client as the *candidate list*.

Figure 5 gives a running example for a private nearest-neighbor query over public data where it presents a zoom

**Algorithm 2** Private NN Queries over Public Data

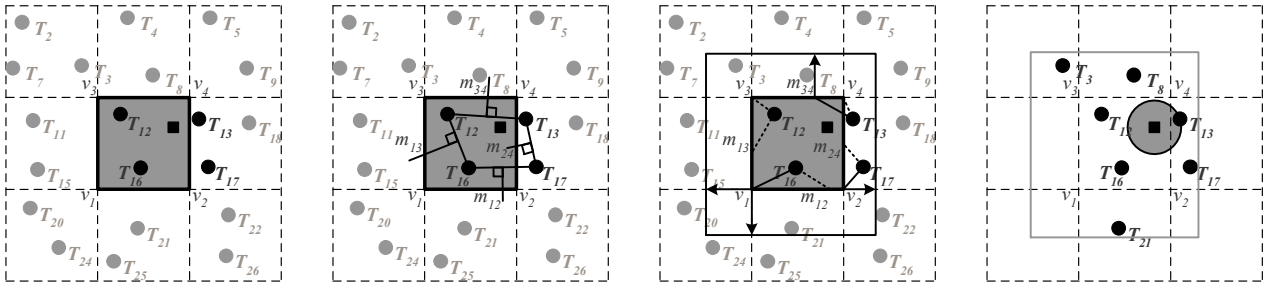
```
1: Function PRIVATE_NN_PUBLIC_DATA(Cloaked Area A)
2: A_EXT is an extended area and initially set to A
3: for each vertex v_i in region A do
4:    t_i ← is the nearest target object to v_i
5: end for
6: for Each edge e_ij = v_i v_j of region A do
7:    if t_i = t_j then
8:       m_ij ← NULL
9:    else
10:      L_ij is a line connecting t_i and t_j
11:      P_ij is a line that divides and is orthogonal to L_ij
12:      m_ij is the intersection point of P_ij and e_ij
13:   end if
14:   d_m ← Distance(t_i,m_ij) = Distance(t_j,m_ij)
15:   d_i ← Distance(v_i,t_i)
16:   d_j ← Distance(v_j,t_j)
17:   max_d ← MAX(d_m, d_i, d_j)
18:   Expand A_EXT by distance max_d in v_i v_j direction
19: end for
20: candidate_list ← All target objects inside A_EXT.
21: return candidate_list
```

view of the shaded area of Figure 4a along with its neighbor cells. Algorithm 2 gives the pseudo code for private nearest-neighbor queries over public data. The input to Algorithm 2 is the cloaked spatial area $A$ that is received from the *location anonymizer* while the output is a *candidate list* of answers to be sent to the user who issued the query. In general, Algorithm 2 has the following four steps:

**STEP 1: *The filter step.*** In this step, four *filter* target objects are chosen as the nearest object $t_i$ for each vertex $v_i$ in the cloaked area $A$ (Lines 3 to 5 in Algorithm 2). In our example, (Figure 5a), the four *filters* are $T_{16}$, $T_{17}$, $T_{12}$, and $T_{13}$ where they are the nearest target objects to the vertices $v_1$, $v_2$, $v_3$, and $v_4$, respectively.

**STEP 2: *The middle point step.*** In this step, we aim to find a point $m_{ij}$ for each edge $e_{ij} = v_i v_j$ such that $m_{ij}$ divides $e_{ij}$ into two segments $v_i m_{ij}$ and $m_{ij} v_j$. The main idea is that any point in the first segment $v_i m_{ij}$ will have $t_i$ as its nearest *filter* target object and any point in the second segment $m_{ij} v_j$ will have $t_j$ as its nearest *filter* target object while point $m_{ij}$ is of equal distance from both targets $t_i$ and $t_j$ (Lines 7 to 13 in Algorithm 2). We distinguish between two cases based on whether the two vertices $v_i$ and $v_j$ have the same filter or not. If $v_i$ and $v_j$ have the same filter $t$, then point $m_{ij}$ does not exist as all points on edge $e_{ij}$ will have $t$ as their nearest target object (Line 8 in Algorithm 2). If $t_i$ and $t_j$ are different, $m_{ij}$ is found by connecting $t_i$ and $t_j$ through a line $L_{ij}$. Then, another line $P_{ij}$ is plotted that is perpendicular to $L_{ij}$ and divides $L_{ij}$ into two equal segments. Finally, $m_{ij}$ will be the intersection point between $P_{ij}$ and the edge $e_{ij}$ (Lines 10 to 12 in Algorithm 2). Figure 5b depicts this step in our running example where points $m_{12}$, $m_{13}$, $m_{24}$, and $m_{34}$ are plotted.

**STEP 3: *The extended area step.*** In this step, for each edge $e_{ij}$, we aim to find the largest distance $max_d$ from any point on $e_{ij}$ to its nearest filter target object (Lines 14 to 18 in Algorithm 2). Only three points on $e_{ij}$ can be candidates to have the distance $max_d$ to their nearest *filter* object, $v_i$, $v_j$, or $m_{ij}$. Thus, we compute the three distances $d_i$, $d_j$, and $d_m$ that represent the distances from points $v_i$, $v_j$, and $m_{ij}$ to targets $t_i$, $t_j$, and $t_i$, respectively. Notice that in case $m_{ij}$ does not exist, the distance $d_m$ will be 0. Then, the distance $max_d$ is computed as the maximum distance



(a) Step 1: Filters (b) Step 2: Middle points (c) Step 3: $A_{EXT}$ (d) Step 4: Client

**Figure 5: Example of a private query over public data.**

of $d_i$, $d_j$, and $d_m$ (Line 17 in Algorithm 2). Finally, the area $A_{EXT}$ is expanded by the distance $max_d$ in the same direction of edge $e_{ij}$ (Line 18 in Algorithm 2). Figure 5c depicts this step where all the computed distances for the four edges are plotted. Only those distances that contribute to $max_d$ are plotted as solid while other distances are plotted as dotted lines. An arrowed line is plotted from each edge to represent its $max_d$ extension to plot $A_{EXT}$. The intersection of the arrowed line with its edge represents the point that has contributed to $max_d$.

**STEP 4:** ***The candidate list step.*** In this step, the server issues a range query that returns all target objects within the area $A_{EXT}$ as the *candidate list*. The *candidate list* is sent to the client where the query can be evaluated locally (Lines 20 to 21 in Algorithm 2). Figure 5d depicts this step where the *candidate list* has only seven objects that include the exact query answer $T_{13}$. Notice the difference between Figure 5d and Figure 4c, where in the former, the client needs to evaluate her query on only 7 targets as opposed to 32 targets in the latter case.

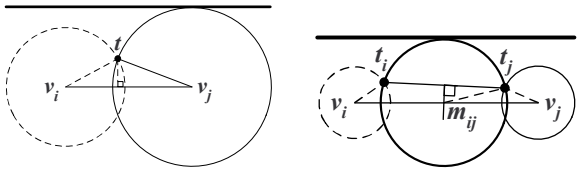
Our approach is independent from the nearest-neighbor and range query algorithms used in both the *filter* and *extended area* steps to find the nearest target to each vertex and the objects within area $A_{EXT}$, respectively. These algorithms are assumed to be implemented in traditional location-based database servers. We do not have any assumptions about these algorithm, it can be employed using R-tree or any other methods. In fact, our approach can be seamlessly integrated with any traditional location-based database servers to turn them to be *privacy-aware*.

### 5.1.2 Proof of Correctness

In this section, we prove the correctness of Algorithm 2 by proving that: (1) it is *inclusive*, i.e., it returns the exact answer within its candidate list, and (2) it is *minimal*, i.e., area $A_{EXT}$ is of minimal size given the set of filter targets.

THEOREM 1. *Given a cloaked area A for a user u located anywhere within A, Algorithm 2 returns a candidate list that includes the exact nearest target to u.*

PROOF. Assume that there is a user $u$ located in area $A$ where $u$'s nearest target $t_u$ is not included in the *candidate list*. Assume further that the exact location of $u$ within $A$ is on the edge $e_{ij} = v_i v_j$. Based on the *filter* objects $t_i$ and $t_j$ of $v_i$ and $v_j$, respectively, we identify two cases:



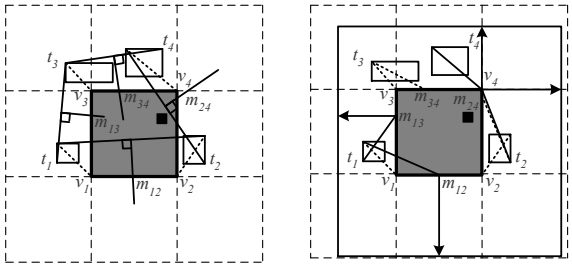
(a) $t = t_i = t_j$ (b) $t_i \neq t_j$

**Figure 6: Private queries over public data.**

`Case 1`: $t_i = t_j$. Figure 6a depicts this case. The dotted circle represents the distance $d_i$ from $v_i$ to $t$ while the solid circle represents the distance $d_j$ from $v_j$ to $t$. The bold line that is tangent to the solid circle represents the boundary of the area $A_{EXT}$ that will include the candidate list. For any location $p_u$ of the user $u$ on $v_i v_j$, if there is a target $t_u$ that is nearest to $p_u$ than $t$, then $t_u$ would be within the union area of the two circles. Thus, $t_u$ should be below the solid line, hence is included in $A_{EXT}$ and is returned within the *candidate list*.

`Case 2`: $t_i \neq t_j$. Figure 6b depicts this case. The dotted and solid circles represent the distances $d_i$ and $d_j$, respectively. The bold circle represents the distance $d_m$ from $m_{ij}$ to $t_i$. The bold line that is tangent to the bold circle represents the boundary of the area $A_{EXT}$ that will include the candidate list. Then, the location $p_u$ of the user $u$ is either in the line segment $v_i m_{ij}$ or $m_{ij} v_j$. In the former case, similar to `Case 1`, if there is a target $t_u$ that is nearest to $p_u$ than $t_i$, then $t_u$ would be within the union area of the dotted and bold circles. Thus, $t_u$ should be below the solid line, hence is included in $A_{EXT}$ and is returned within the *candidate list*. Similarly, if $p_u$ is on the line segment $m_{ij} v_j$, then if there is a target $t_u$ that is nearest to $p_u$ than $t_j$, then $t_u$ would be within the union area of the solid and bold circles. Thus, $t_u$ should be below the solid line, hence is included in $A_{EXT}$ and is returned within the *candidate list*.

From `Cases 1` and `2`, we conclude that if user $u$ is on the boundary of the cloaked region $A$, then any target object $t_u$ that is nearest to $u$ would be included in the *candidate list*. Trivially, the proof would be valid if the user $u$ is within the area $A$ rather than on its boundaries. □




(a) Server side (b) Server side

**Figure 7: Private query over private data.**

THEOREM 2. *Given a cloaked area $A$ for a user $u$ and a set of filter target objects $t_1$ to $t_4$, Algorithm 2 issues the minimum possible range query to get the candidate list.*

PROOF. In Figures 6a and 6b, the tangent point between the bold line and the largest circle may contain a target that is nearest to one location in $v_iv_j$ (point $v_j$ in Figure 6a and point $m_{ij}$ in Figure 6b). Thus, if there is another line that is lower than the bold line, it will cross the largest circle and may end up in missing target objects. Thus, the bold line presents the minimum possible expansion for the line $v_iv_j$ in order to include all possible nearest target objects that are closer than $t_i$ and $t_j$. Applying the same proof for the four edges of the area $A_{EXT}$ concludes that $A_{EXT}$ is the smallest possible area that contains all possible nearest target objects given a set of target filters objects. □

## 5.2 Private Queries over Private Data

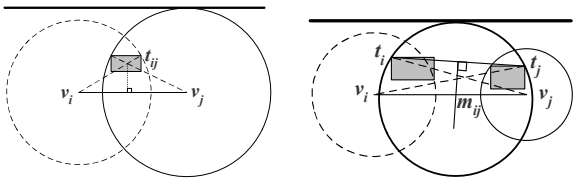
In this section, we extend our approach for private queries over public data to deal with private data that are represented by cloaked regions rather than by exact locations. Similar to Section 5.1 and without loss of generality, we focus on private nearest-neighbor queries over private data.

### 5.2.1 Algorithm for Nearest-Neighbor Queries

The same idea of Algorithm 2 can be applied for private data with the following changes:

**STEP 1:** ***The filter step.*** Similar to the filtering step in Algorithm 2, the four filter target objects are chosen as the nearest target object $t_i$ to each vertex $v_i$. The only difference is that for each vertex $v_i$, we consider that the exact location of a target object within its cloaked area is the furthest corner from $v_i$. Figure 7a shows only the nearest four target objects $t_1, t_2, t_3$, and $t_4$ to the cloaked area. Target objects are drawn as regions since they represent private data. For clarity, we draw these rectangles smaller in size than the query area. However, this is not a necessary condition for our approach. The dotted line from each vertex $v_i$ to its nearest target $t_i$ represents the distance that we measure in determining the nearest target.

**STEP 2:** ***The middle point step.*** The main idea is similar to that of Algorithm 2. The only difference is that the line $L_{ij}$ that connects $t_i$ and $t_j$ would consider the furthest corner of $t_i$ and $t_j$ from the reverse vertex $v_j$ and $v_i$, respectively. Figure 7a depicts this step. For example, consider edge $v_3v_4$, the line $L_{34}$ connects the furthest corner of $t_4$ from vertex $v_3$ to the furthest corner of $t_3$ from $v_4$.




(a) $t_i = t_j$ (b) $t_i \neq t_j$

**Figure 8: Private queries over private data**

**STEP 3:** ***The extended area step.*** The main idea is similar to that of Algorithm 2. However, the three distances that are taken into consideration are $d_i$ as the distance from $v_i$ to the furthest corner of $t_i$ from $v_i$, $d_j$ as the distance from $v_j$ to the furthest corner of $t_j$ from $v_j$, and $d_m$ as the distance from $m_{ij}$ to any of the end points of $L_{ij}$. Figure 7b depicts such distances. The bold lines represent the largest distance computed for each edge. The arrow lines represent the $max_d$ distance used to expand the area $A$.

**STEP 4:** ***The candidate list step.*** This step is exactly similar to that of Algorithm 2. To accommodate data areas rather than exact point locations, we will return any target object that has an area overlapped with $A_{EXT}$. More sophisticated techniques of probabilistic queries and data uncertainty can take place. For example, we may choose to return only those target objects that have more than $x\%$ of their cloaked areas overlap with $A_{EXT}$. Our approach is completely independent from deciding whether an object is considered within an area or not. Thus, our approach can be seamlessly integrated with any approaches for probabilistic query processing (e.g., [10, 11, 28, 38, 42, 46]).

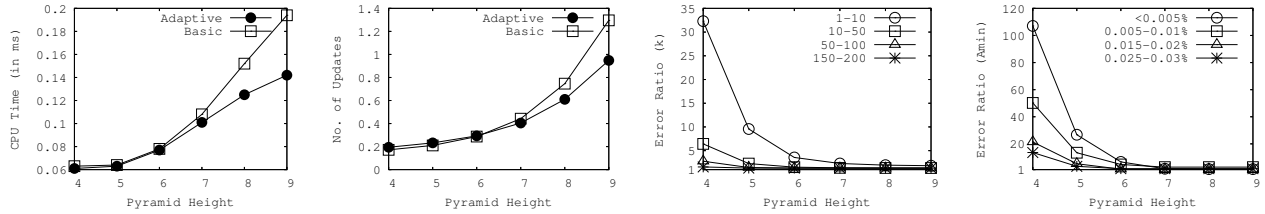
### 5.2.2 Proof of Correctness

THEOREM 3. *Given a cloaked area $A$ for a user $u$ located anywhere within $A$ and a set of target objects represented by their cloaked areas, Algorithm 2 with the modifications in Section 5.2.1 returns a candidate list that includes the exact nearest target to $u$.*

PROOF. The proof is similar to that of Theorem 1 with the difference of dealing with rectangular areas of the target objects rather than with exact point locations. Figures 8a and 8b show the two cases of $t_i = t_j$ and $t_i \neq t_j$, respectively. As in the proof of Theorem 1a, in Figure 8a, any target object that could be a nearest neighbor to any point on $v_iv_j$ should be inside the union area of the dotted and solid circles. Thus, it would be returned with the candidate list. Similarly, in Figure 8b, any target object that could be nearest to any point $p_u$ in $v_iv_j$ should overlap the union of the dotted and bold circles if $p_u$ is in $v_im_{ij}$ or overlap with the union of the bold and solid circles if $p_u$ is in $m_{ij}v_j$. Thus, it would be returned within the *candidate list.* □

THEOREM 4. *Given a cloaked area $A$ for a user $u$ and a set of filter objects $t_1$ to $t_4$ represented by their cloaked areas, Algorithm 2 with the modification in Section 5.2.1 issues the minimum possible range query to get the candidate list.*

PROOF. The proof is exactly the same as that of Theorem 2. The tangent bold lines in Figures 8a and 8b serve the same purpose as those tangent lines in Figures 6a and 6b □




(a) Cloaking Time (b) Update Cost (c) $k$ Precision (d) $A_{min}$ Precision

**Figure 10: The height of the pyramid structure**

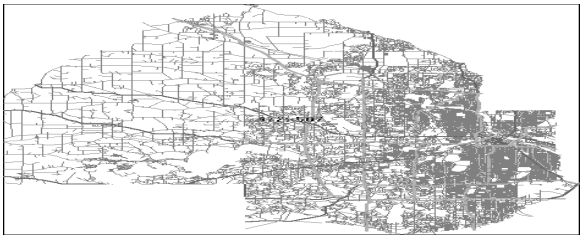


**Figure 9: Map of Hennepin County, MN, USA.**

# 6. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the performance of the *Casper* framework, by evaluating its two main components, the *location anonymizer* (Section 6.1) and the *privacy-aware query processor* (Section 6.2). In addition, we evaluate the end-to-end performance for the user interaction with *Casper* (Section 6.3). In all the experiments of this section, we use the *Network-based Generator of Moving Objects* [9] to generate a set of moving objects. The input to the generator is the road map of Hennepin County in Minnesota, USA (Figure 9). The output of the generator is a set of moving objects that move on the road network of the given city. Target objects are chosen as uniformly distributed in the spatial space.

## 6.1 Location Anonymizer

In this section, we evaluate and compare the efficiency and scalability of both the *basic* and *adaptive* location anonymizers with respect to the cloaking time, maintenance cost, and accuracy. We were unable to perform comparison with other approaches for the location anonymizer [16, 17] as these approaches are limited either for small numbers of users [17] or for privacy requirement ($k$ is from 5 to 10) [16]. Unless mentioned otherwise, the experiments in this section consider 50K registered mobile users in a pyramid structure with 9 levels. We generate a random privacy profile for each user where $k$ and $A_{min}$ are assigned uniformly within the range [1-50] users and [.005,.01]% of the space, respectively.

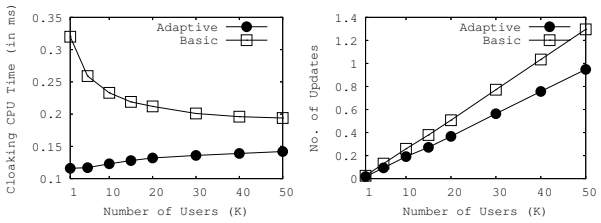
### 6.1.1 The Pyramid Height

Figure 10 gives the effect of the pyramid height on the performance of both the *basic* and *adaptive* location anonymizers. The pyramid height varies from four to nine levels. Figure 10a gives the effect of the pyramid height on the average cloaking time per user request (Algorithm 1). For more than six pyramid levels, the performance of the *adaptive* approach is clearly better. The main reason is that the *adaptive* approach smartly decides about the lowest maintained level such that the number of pyramid levels to be searched is minimized. Figure 10b gives the effect of the pyramid height on the average number of updates required for each location update. For lower pyramid levels, the *basic* approach has a lower update cost as the cost of splitting and merging in the *adaptive* approach prevails the savings in updating the cell counters. However, for higher pyramid levels, the *adaptive* approach has less update cost as it encounters huge savings in updating large numbers of cell counters.

Optimally, a user wants to have a cloaked region that exactly matches her privacy profile. Due to the resolution of the pyramid structure, the *location anonymizer* may not be able to provide an exact match. Instead, a more restrictive cloaked region will be given to the user which may result in a lousy service that the user is not comfortable with. Figures 10c and 10d give the effect of the pyramid height on the accuracy of the cloaked region in terms of $k$ and $A_{min}$, respectively. Both the *basic* and *adaptive* approaches yield the same accuracy as they result in the same cloaked region from Algorithm 1. In Figure 10c, the accuracy is measured as $k'/k$, where $k'$ is the number of users included in the cloaked spatial region while $k$ is the exact user requirement. We run the experiment for various groups of users with most relaxed privacy requirements ($k$ is from 1 to 10) to restrictive users ($k$ is from 150 to 200) while setting $A_{min}$ to zero. Lower pyramid levels give very inaccurate answer for relaxed users. However, higher pyramid levels give very accurate cloaked region that is very close to one (optimal case) even for relaxed users. Similar behavior is depicted in Figure 10d where the accuracy is measured as $A'/A_{min}$, where $A'$ is the computed cloaked spatial region while $A_{min}$ is the required one. Also, we run the experiment for several groups of users with various $A_{min}$ requirements while setting $k$ to one.

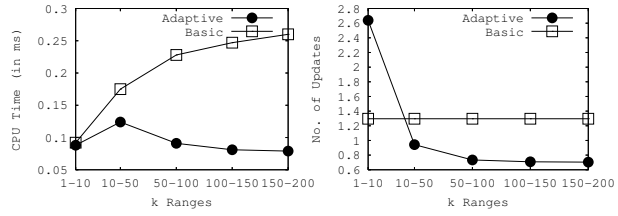
### 6.1.2 Scalability

Figure 11 gives the scalability of the *basic* and *adaptive* location anonymizers with respect to varying the number of registered users from 1K to 50K. With respect to the cloaking time (Figure 11a), the performance of the *basic* location anonymizer is greatly enhanced with the increase of the number of users. The main idea is that by increasing the number of users, the privacy requirements of mobile users will be likely to be satisfied in lower pyramid levels, i.e., less recursive calls to Algorithm 1. This is not the case for the *adaptive* location anonymizer where the large number of users increases the number of maintained grid cells to ac-



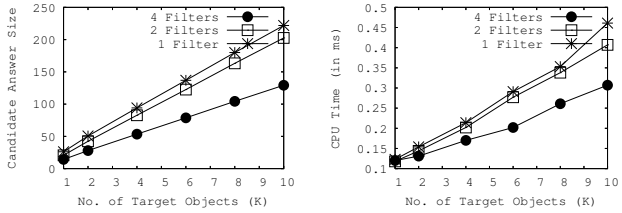

(a) Cloaking Time (b) Update Cost

**Figure 11: Number of users**



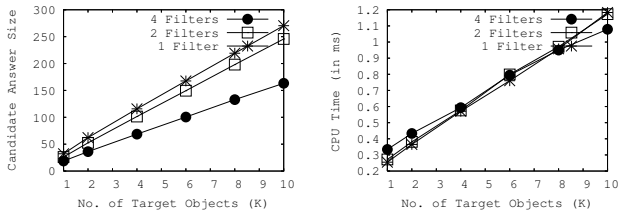

(a) Cloaking Time (b) Update Cost

**Figure 12: $k$ ranges**




(a) Candidate List (b) Processing Time

**Figure 13: Number of *public* target objects**




(a) Candidate List (b) Processing Time

**Figure 14: Number of *private* target objects**

commodate users with various requirements. However, the cloaking time of the *adaptive* approach is always less than that of the *basic* approach. For the update cost (Figure 11b), with the increase of the number of users, the performance of the *adaptive* approach is always better than that of the *basic* approach due to the less number of maintained cells.

### 6.1.3 Effect of Privacy Profile

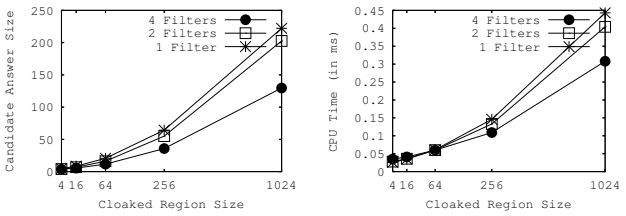
Figure 12 gives the effect of increasing the $k$ anonymity parameter on both the *basic* and *adaptive* approaches. The range of $k$ varies from [1-10] to [150-200]. For the cloaking time (Figure 12a), the *basic* location anonymizer incurs high cloaking cost with more restrictive privacy requirements as it has to traverse more pyramid levels in order to get the desired cloaked region. The *adaptive* location anonymizer has similar cost trend for relaxed users ($k < 50$). However, with more restrictive privacy profile, the performance of the *adaptive* approach gets much better as mobile users tend to cluster in higher pyramid levels, thus decreasing the cloaking time. For the update cost (Figure 12b), the *basic* approach is not affected by the privacy profile as it always maintains a complete pyramid structure. On the other hand, the *adaptive* approach has high update cost only for relaxed users as this will require maintaining lower pyramid levels in addition to the splitting and merging cost. However, the *adaptive* approach adopts its structure with more strict privacy requirements to give a much better performance than that of the *basic* approach. Similar figures and experiments give similar results for the case of changing $A_{min}$ (not shown due to space limitation).

## 6.2 Privacy-aware Query Processor

In this section, we study the efficiency and scalability of the *privacy-aware query processor* with respect to the size of the returned *candidate list* and the query processing time. As our approach is unique in nature where it is the first query processor that deals with private data, we evaluate our *privacy-aware query processor* based on the number of used filters. We compare among three alternatives of our approach when using one, two, or four filters in the first step of Algorithm 2. When choosing one filter, we choose the nearest target to the center of the cloaked area while in the case of two filters, we choose the nearest target to any two reverse corners in the cloaked area. For the case of four filters, we choose the nearest target to each corner as depicted in Algorithm 2. Notice that all the theorems and proofs in Section 5 are valid for the three cases. For traditional nearest-neighbor and range query algorithms used within our *privacy-aware query processor* (Steps 1 and 3 in Section 5), we use similar techniques to [36] and [34], respectively. However, as we have mentioned earlier our framework is completely independent from these approaches as it can be integrated with any existing algorithms for nearest-neighbor and range queries. Unless mentioned otherwise, all experiments in this section have 10K target objects, user privacy profile of $k$ in [1-50], $A_{min}$ in [.005-.01]. Private target objects has a region of [1-64] cells.

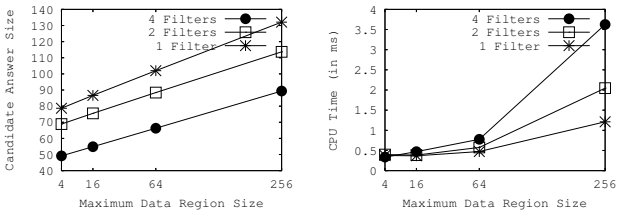
### 6.2.1 Scalability

Figures 13 and 14 give the scalability of the *privacy-aware query processor* when increasing the number of public and private targets from 1K to 10K, respectively. For the case of public targets, the size of the candidate list is greatly reduced when using more filters (Figure 13a). For the case of 10K targets, using four filters results in a candidate list about half the size of the one returned using only one fil-




(a) Candidate List (b) Processing Time

**Figure 15: Cloaked region size (public data)**




(a) Candidate List (b) Processing Time

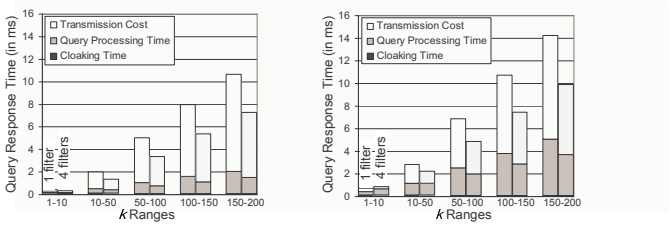
**Figure 16: Data region size**

ter. With respect to the query processing time (Figure 13b), the computational overhead from computing four filters is amortized by the huge pruning in the search space to get the *candidate list*. Thus, using four filters always achieves better performance.

For the case of private target data, (Figure 14), the performance of the candidate list size is similar to that of Figure 13. However, with respect to the query processing time (Figure 13b), using four filters always results in a higher cost. The main reason is that searching private data (represented as cloaked regions) is an expensive task. However, the small size of the candidate list does not affect only the query processing cost at the server, but it also significantly reduces the transmission cost of the candidate list to the client along with the client computation time. Thus, even the case of four filters gives higher query processing cost at the server size, it is greatly preferred as it saves in the transmission time due to the less candidate list size. Section 6.3 will elaborate more on the effect of the transmission cost.

### 6.2.2 Privacy Profile of Queries and Data

Figure 15 gives the effect of the user privacy profile on the query performance. The cloaked query area varies from 4 to 1024 cells while the target objects are public. Using more filters consistently results in a better performance in terms of both the candidate list size and the query processing time. Similar performance is achieved when considering private target objects (not shown due to the space limitation). Figure 16 gives the effect of the privacy profile of target objects when varying the cloaked region for target objects from 4 to 256 cells. Using four filters results in significant decrease in the candidate list size while an increase in the query processing time.




(a) Public data (b) Private data

**Figure 17: End-to-End Performance**

## 6.3 The CASPER End-to-End Performance

This section evaluates the overall performance of the *Casper* framework based on the $k$-anonymity. Figure 17 measures the total end-to-end time from submitting a private nearest-neighbor query to *Casper* and getting back the result for both public and private data. The total time is divided into three components based on the consumed time at the *location anonymizer*, the *privacy-aware query processor*, and the transmission time of the *candidate list*. For transmission time, we assume that a data record is of size 64 bytes transmitted over a channel of bandwidth 100 Mbps. We use the *adaptive* location anonymizer and a privacy-aware query processor with *four filters*. We consider 10K mobile users and 10K target objects that are represented by a cloaked area [1-64] cells for the case of private data.

Figure 17a shows the performance for groups of $k$ from [1-10] till [40-50] while Figure 17b gives the same experiment but for larger values of $k$ up to [150-200]. It is apparent that the location cloaking time at the location anonymizer is the lowest overhead as it appears only for small values of $k$. For strict privacy requirements, the *location anonymizer* time is almost negligible. The main reason is that the *location anonymizer* spends its time wisely in maintaining the pyramid structure in order to provide very small cloaking time. For lower privacy profiles ($K < 10$ for public data and $k < 30$ for private date), the query processing time (using four filters) dominates the transmission time. However, for strict privacy requirements, the transmission time is the dominant factor in affecting the end-to-end performance. When using less than four filters in the *privacy-aware query processor*, the number of candidate list significantly increases, hence the transmission time increases and greatly dominates and degrades the overall performance. Although less than four filters reduces the query processing time as depicted in Figures 14b and 16b, yet it will not increase the total performance. Similar performance is given when varying the $A_{min}$ parameter (Experiments of filters and $A_{min}$ are not shown due to space limitation).

# 7. CONCLUSION

This paper introduces *Casper*; a novel framework in which mobile users can entertain location-based services without the need to disclose their private location information. Mobile users register with *Casper* by a user-specified *privacy profile*. *Casper* has two main components, the *location anonymizer* and the *privacy-aware* query processor. The *location anonymizer* acts as a third trusted party that blurs

the exact location information of each user into a cloaked spatial area that matches the user privacy profile. Four requirements from the location anonymizer are outlined: accuracy, quality, efficiency, and flexibility. Two alternatives of the *location anonymizer* that achieve these requirements are proposed: the *basic* and *adaptive* location anonymizer. The *privacy-aware query processor* is embedded into traditional location-based database servers to tune their functionalities to be *privacy-aware* by dealing with cloaked spatial areas rather than exact point information. Three novel query types that are supported by *Casper* are identified, *private queries over public data*, *public queries over private data*, and *private queries over private data*. We have provided a framework for dealing with these queries that returns a *candidate list* of answers rather than an exact answer. We have proved that the returned candidate list contains the exact answer and is of minimal size. Extensive experimental evaluation studies all the components of *Casper* and shows its efficiency, accuracy, and scalability with large number of mobile users and various privacy requirements.

# 8. REFERENCES


[1] L. Ackerman, J. Kempf, and T. Miki. Wireless location privacy: A report on law and policy in the united states, the european union, and japan. Technical Report DCL-TR2003-001, DoCoMo Commuinication Laboratories, USA, 2003.

[2] G. Aggarwal. et al. Vision Paper: Enabling Privacy for the Paranoids. In *VLDB*, 2004.

[3] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information Sharing Across Private Databases. In *SIGMOD*, 2003.

[4] Anonymous surfing. http://www.anonymizer.com.

[5] W. G. Aref and H. Samet. Efficient Processing of Window Queries in The Pyramid Data Structure. In *PODS*, 1990.

[6] L. Barkhuus and A. K. Dey. Location-Based Services for Mobile Telephony: a Study of Users' Privacy Concerns. In *INTERACT*, 2003.

[7] R. J. Bayardo and R. Agrawal. Data Privacy through Optimal k-Anonymization. In *ICDE*, 2005.

[8] A. R. Beresford and F. Stajano. Location Privacy in Pervasive Computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[9] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[10] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying Imprecise Data in Moving Object Environments. *TKDE*, 16(9), Sept. 2004.

[11] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis. Probabilistic Spatial Queries on Existentially Uncertain Data. In *SSTD*, 2005.

[12] W. Du and M. J. Atallah. Secure Multi-Party Computation Problems and their Applications: A Review and Open Problems. In *New Security Paradigms Workshop*, 2001.

[13] M. Duckham and L. Kulik. A Formal Model of Obfuscation and Negotiation for Location Privacy. In *Pervasive*, 2005.

[14] F. Emekci, D. Agrawal, A. E. Abbadi, and A. Gulbeden. Privacy Preserving Query Processing using Third Parties. In *ICDE*, 2006.

[15] Foxs News. Man Accused of Stalking Ex-Girlfriend With GPS. http://www.foxnews.com/story/0,2933,131487,00.html.

[16] B. Gedik and L. Liu. A Customizable k-Anonymity Model for Protecting Location Privacy. In *ICDCS*, 2005.

[17] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *MobiSys*, 2003.

[18] M. Gruteser and X. Liu. Protecting Privacy in Continuous Location-Tracking Applications. *IEEE Security and Privacy*, 2(2):28–34, 2004.

[19] R. H. Güting, V. T. de Almeida, D. Ansorge, T. B. Z. Ding, T. Höse, F. Hoffmann, M. Spiekermann, and U. Telle. SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching. In *ICDE*, 2005.

[20] U. Hengartner and P. Steenkiste. Protecting Access to People Location Information. In *Proceeding of the International Conference on Security in Pervasive Computing, SPC*, 2003.

[21] J. I. Hong and J. A. Landay. An Architecture for Privacy-Sensitive Ubiquitous Computing. In *MobiSys*, 2004.

[22] H. Hu, J. Xu, and D. L. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *SIGMOD*, 2005.

[23] N. Jefferies, C. J. Mitchell, and M. Walker. A Proposed Architecture for Trusted Third Party Services. In *the Intl. Conf. on Cryptography: Policy and Algorithms*, 1995.

[24] C. S. Jensen. Database Aspects of Location-Based Services. In *Location-Based Services*, pages 115–148. Morgan Kaufmann, 2004.

[25] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective Density Queries of Continuously Moving Objects. In *ICDE*, 2006.

[26] E. Kaasinen. User needs for location-aware mobile services. *Personal and Ubiquitous Computing*, 7(1):70–79, 2003.

[27] H. Kido, Y. Yanagisawa, and T. Satoh. An Anonymous Communication Technique using Dummies for Location-based Services. In *Intl. Conf. on Pervasive Services, ICPS*, 2005.

[28] I. Lazaridis and S. Mehrotra. Approximate Selection Queries over Imprecise Data. In *ICDE*, 2004.

[29] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *ICDE*, 2006.

[30] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient Full-Domain K-Anonymity. In *SIGMOD*, 2005.

[31] A. Meyerson and R. Williams. On the Complexity of Optimal K-Anonymity. In *PODS*, 2004.

[32] M. F. Mokbel. Towards Privacy-Aware Location-Based Database Servers. In *International Workshop on Privacy Data Management, PDM*, Apr. 2006.

[33] M. F. Mokbel and W. G. Aref. PLACE: A Scalable Location-aware Database Server for Spatio-temporal Data Streams. *IEEE Data Engineering Bulletin*, 28(3):3–10, 2005.

[34] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[35] M. F. Mokbel, X. Xiong, W. G. Aref, S. Hambrusch, S. Prabhakar, and M. Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams (Demo). In *VLDB*, 2004.

[36] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *SIGMOD*, 2005.

[37] A. Pfitzmann and M. Kohntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[38] D. Pfoser, N. Tryfona, and C. S. Jensen. Indeterminacy and Spatiotemporal Data: Basic Definitions and Case Study. *GeoInformatica*, 9(3), Sept. 2005.

[39] L. Sweeney. Achieving k-anonymity Privacy Protection using Generalization and Suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[40] L. Sweeney. k-anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[41] S. L. Tanimoto and T. Pavlidis. A Hierarchical Data Structure for Picture Processing. *Computer Graphics and Image Processing*, 4(2), 1975.

[42] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing Uncertainty in Moving Objects Databases. *TODS*, 29(3), Sept. 2004.

[43] USAToday. Authorities: GPS system used to stalk woman. http://www.usatoday.com/tech/news/2002-12-30-gps-stalker_x.htm.

[44] J. Warrior, E. McHenry, and K. McGee. They Know Where You Are . *IEEE Spectrum*, 40(7):20–25, 2003.

[45] O. Wolfson, H. Cao, H. Lin, G. Trajcevski, F. Zhang, and N. Rishe. Management of Dynamic Location Information in DOMINO. In *EDBT*, 2002.

[46] O. Wolfson and H. Yin. Accuracy and Resource Concumption in Tracking and Location Prediction. In *SSTD*, 2003.

[47] T. Xia and D. Zhang. Continuous Reverse Nearest Neighbor Monitoring. In *ICDE*, 2006.

[48] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate Nearest Neighbor Queries in Road Networks. *TKDE*, 17(6), 2005.

[49] X. Yu, K. Q. Pu, and N. Koudas. Monitoring K-Nearest Neighbor Queries Over Moving Objects. In *ICDE*, 2005.