Longitude: Centralized Privacy-Preserving Computation of Users' Proximity

Sergio Mascetti, Claudio Bettini, and Dario Freni

Università degli Studi di Milano
DICo - EveryWare Lab

Abstract. A "friend finder" is a Location Based Service (LBS) that informs users about the presence of participants in a geographical area. In particular, one of the functionalities of this kind of application, reveals the users that are in proximity. Several implementations of the friend finder service already exist but, to the best of our knowledge, none of them provides a satisfactory technique to protect users' privacy. While several techniques have been proposed to protect users' privacy for other types of spatial queries, these techniques are not appropriate for range queries over moving objects, like those used in friend finders. Solutions based on cryptography in decentralized architectures have been proposed, but we show that a centralized service has several advantages in terms of communication costs, in addition to support current business models. In this paper, we propose a privacy-aware centralized solution based on an efficient three-party secure computation protocol, named *Longitude*. The protocol allows a user to know if any of her contacts is close-by without revealing any location information to the service provider. The protocol also ensures that user-defined minimum privacy requirements with respect to the location information revealed to other buddies are satisfied. Finally, we present an extensive experimental work that shows the applicability of the proposed technique and the advantages over alternative proposals.

1 Introduction

Location-aware social networks are social network applications in which the geographical position of participants can be used to enable new services. The ability to access social network applications through mobile devices and the availability of precise positioning technologies are likely to make this new generation of social networks very popular. As in many social networks, each user is part of one or more groups of users, called *friends* or *buddies*. Among the enabled services, proximity services alert a user when one of her buddies is in the vicinity, possibly enacting other activities like the visualization on a map of the approximate position, or starting a communication session. These services, often called *friend finder*, are currently available on the Internet, with specific client-side software or plugin to be installed on mobile devices[1]. From a data management point

[1] Examples are Google Latitude, Loopt, iPoki.

of view, a proximity service involves the computation of a sequence of range queries over a set of moving entities issued by a moving user, where the range is a distance threshold value decided by the user.

Currently available services are based on a centralized architecture in which location updates are acquired from mobile devices[2] by a location server in the service provider (SP) infrastructure; proximity is computed based on the acquired locations. These services are not currently offered by mobile phone operators, that usually have approximate location information about their users from the network infrastructure. Indeed, in this paper we assume that the SP is an untrusted entity that has no location information about users except the one acquired through the service itself. We also consider "proximity" a user-dependent concept: User A defines a distance threshold δ_A, and considers any buddy B as being in proximity if the following condition holds:

$$dist(loc(A), loc(B)) \leq \delta_A \tag{1}$$

where $dist(loc(A), loc(B))$ denotes the Euclidean distance between the reported locations of A and B. Optimization strategies for location updates and proximity computations have been proposed [1], but they are not the focus of this paper.

The problem we are considering is to offer such a service, in an analogous centralized architecture, while providing formal guarantees to each user about her location privacy. The launch of friend finder services on the Internet has indeed generated a lot of concerns by the potential users about the release of precise location data, that in current systems can be easily associated with the user identities, as demonstrated by privacy research in LBS [2, 4]. There are actually two different concerns: the first is related to the location data disclosed to the SP. Indeed, many users do not have complete trust in the SP, and they are also concerned about their location data being possibly accessed later on SP data stores by untrusted parties. Hence, the first privacy requirement we aim to satisfy is not to reveal *any* location information to the SP. The second concern regards the location data disclosed to the buddies: a user may wish not to provide the exact location to her buddies, although she may be willing to reveal if she is in proximity. In general, the level of location privacy can be represented by the uncertainty that an external entity has about the position of the user, and this uncertainty can be formally represented as a geographic area in which no point can be ruled out as a possible position of the user. In principle, each user should be able to express her privacy preferences by specifying for each other user (or class of users perceived as adversaries) a partition of the geographical space defining the *minimal uncertainty regions* that she wants to be guaranteed. For example, Alice specifies that Bob should never be able to find out the specific building where Alice is within the campus, i.e., the entire campus area is a minimal uncertainty region. Current services have very limited support to fine tune the location privacy with respect to buddies.

[2] While a variety of positioning technologies and communication infrastructure can be used, here we assume GPS-enabled devices with always on 3G data connections.

Considering the related research literature, the available privacy preserving solutions for location based services are not straightforwardly applicable to this problem, since they are either focused on guaranteeing the anonymity of requests or limited to k-NN (Nearest Neighbor) queries or range queries over static resources. At the time of writing, we are aware of only a few proposals for privacy-aware proximity detection [6, 7, 5].

The benefits and vulnerabilities of applying distance preserving transformations have been investigated in privacy preserving data mining [3]. In the specific topic of privacy-aware proximity services, distance preserving transformations have been used to hide the user positions to the SP [6]. However, this specific solution seems to be subject to vulnerabilities, since the SP acquires information on the exact distance between two buddies and hence the SP can exclude some places as their possible locations. On the contrary, *Longitude* does not preserve the exact distance but, instead, it preserves what we call *modular distance* that prevents the disclosure of any location information to the SP.

The release of location information to the SP is also avoided in the three protocols proposed in [7]. They are based on a two-party secure computation exploiting public key cryptography. The solutions suggest a decentralized architecture, and each user does have to contact every buddy each time she needs to know which ones are in proximity; this can result in high communication costs for large number of buddies. In our approach we take advantage of the presence of the SP to significantly reduce the communication costs of a user. An experimental comparison in terms of service precision and communication cost with the algorithm named Pierre[3] is shown in Section 4. An important conceptual difference from both of these papers, is that we more formally consider the privacy with respect to buddies, through the definition and enforcement of user-defined minimal uncertainty regions. In their approach, the location information revealed to buddies only depends on the proximity threshold used in the queries, while in our case it also depends on the minimal uncertainty region defined by each user. This leads to two advantages: an explicit privacy guarantee of the protocol regarding buddies, and a better quality of service as it will be clearer from the details of the protocol.

Finally, in our previous work on this topic [5] we presented an obfuscation-based solution in which the SP is allowed to acquire user location information, but only limited to a user-defined precision. The *SP-Filtering* procedure proposed in [5] provides an approximate answer to the proximity problem exploiting spatial granularities, and may also be used as a pre-processing step of *Longitude*, as well as of the algorithms in [7].

Longitude is based on a three-party secure computation involving only communication between each buddy and the server. Each time a user location is sent to the SP, it is first generalized to a two-dimensional area A whose dimension depends on the user privacy requirement with respect to buddies. Our solution considers a mapping from the two-dimensional space in which users move into

[3] This has been selected since it is the algorithm implemented by the authors in the NearbyFriend service.

a toroidal space. A solid transformation is applied to the projection of A in the toroidal space and the result is then sent to the SP. Each user shares a (possibly different) secret with each of her buddies that determines the solid transformation. The SP computes proximity in the toroidal space and communicates the result to the participating buddies, which can then compute the proximity in the two-dimensional space. In order to avoid correlations in time, the above transformation changes at each run. The knowledge of the distance in the toroidal space does not disclose to the SP any location information about the buddies.

The contributions of this paper can be summarized as follows:

- We design a protocol enabling a service to compute users' proximity in a centralized architecture, without the service provider acquiring any location information;
- We allow each user to specify minimum privacy requirements with respect to the location data released to other buddies, and show the correctness of the protocol with respect to these privacy requirements;
- We experimentally evaluate precision and performance of the proposed protocol by using a realistic simulation of user movements. We also experimentally compare our solution with the only available implementation of a privacy-preserving friend finder.

The main goal of this paper is to illustrate an innovative technique for privacy-aware proximity computation and not to illustrate all the technical details of the protocol. Despite extensions can be applied to deal with more involved scenarios, the basic protocol we describe does not assume particularly powerful adversaries. For example, our aim is not to contrast complex cryptanalytic attacks, and we assume that potential adversaries could only acquire the messages exchanged as part of the protocol, without a-priori knowledge about particular distributions of individuals or spatio-temporal trajectories.

The rest of the paper is organized as follows. In Section 2 we illustrate the *Longitude* protocol, and in Section 3 its formal properties are analyzed. In Section 4 we report experimental results, and in Section 5 we conclude the paper pointing out some interesting future work.

2 Privacy Preserving Proximity Computation

In this section we first describe how buddies can specify their privacy requirements, and then illustrate the *Longitude* protocol.

2.1 Minimum Privacy Requirements

We assume that each user A can specify her *minimum privacy requirements* with respect to other buddies by defining a particular grid G_A partitioning the spatial domain, such that each cell c_A of the grid represents a minimum uncertainty region for A. Cells can either be shaped as squares or rectangles, and all the cells within a grid are required to have the same shape and the same dimension.

We conservatively assume that the minimum privacy requirement defined by a user is public and hence that it is possibly known by the SP or by the other buddies.

The two extreme cases in which a user requires no privacy protection, and maximum privacy protection can be naturally modeled. For example, if a user A does not want her privacy to be protected with respect to other buddies (in this case A can tolerate other buddies to know her location at the maximum available precision) then A will set G_A to the finest grid (the one having as cells the basic elements, or pixels, of the spatial domain). Similarly, if A wants to impose the maximum privacy protection, then A sets G_A to the grid having a single cell covering the entire spatial domain.

The first privacy requirement identified in Section 1 regarding the information released to the SP can be formalized by considering this maximum protection grid as a protection against the SP. This requirement would be clearly violated if locations are sent to the SP; but the second requirement, regarding buddies would be easily violated as well. Indeed, suppose A has a buddy B who sets a value of δ_B in a way such that the circular region of radius δ_B (centered at B's location) is properly contained in a cell of G_A. Then, if A happens to enter that circular region, the SP will notify B, and A's minimum privacy requirement would be violated.

2.2 The *Longitude* Protocol

For the sake of simplicity, the protocol will be illustrated considering a user A issuing a proximity request with respect to a single buddy B, but the extension to multiple buddies is trivial, and the experiments in Section 4 consider a large number of buddies. The main steps of the *Longitude* protocol are the following: each time A wants to check whether B is in proximity, A runs the *encryptLocation* procedure to encrypt the cell c_A of G_A where A is located and sends it to the SP. Referring to the intuitive protocol description given in introduction, the encryption is equivalent to project c_A in the toroidal space, and then apply the solid transformation. Upon receiving the request, the SP sends a message to B requiring a location update. B runs the *encryptLocation* procedure to encrypt the cell c_B of G_B where B is located and sends the result to the SP. Note that B uses the same key as A, generated from their common secret. The encryption function is designed in such a way that the SP, upon receiving a request from A and an answer from B with cells encrypted with the same key, can compute, through the *computeProximity* procedure, the distance in the toroidal space, which we call *modular distance*. The SP compares the modular distance with the proximity threshold and sends the result as a boolean value to the requester A that computes whether B is in proximity or not through the procedure *getResult*.

The encryption function is such that, if A sends her location cell to the SP using the same encryption key in different instants and while being in different cells, and the SP is aware of this, he can possibly learn some information about the movement of A, and hence about her location. For this reason, A changes the encryption key each time she communicates her location cell to the SP.

The following is a simple protocol to achieve this, but many optimizations and different solutions can be devised without affecting the main results of this paper. We assume A and B share a secret K; the actual key used to encrypt the location information is composed by a pair of integers generated with a pseudo-random number generator (PRNG) with seed K. An integer i, locally stored by A, is incremented at each proximity request, and it is used to select the generated keys with index $2i$ and $2i+1$. Its value is also included in the proximity request, since the locations of other buddies will need to be encrypted with a key selected according to i.

2.3 The *encryptLocation* Procedure

The procedure is schematically illustrated as Procedure 1. It is used to issue requests for proximity as well as to send responses to location requests by the SP. The inputs are the location l of the user running the procedure, the grid G chosen to protect the privacy of the user, the seed K, the parameter $lastIndex$ that takes the value of i (i.e., the index of the last key generated by the PRNG by the user running the procedure), and the optional parameter $newIndex$ that is only defined when the procedure is used to respond to a proximity request issued by another buddy; In this case, the value of $newIndex$ is the index of the key used by the issuing buddy. If the procedure is used to issue a request for proximity, the index is incremented. If the procedure is used to send a response to a location request, it first checks if the index used by the buddy issuing the request has ever been used. If this is the case, using the same index again could compromise the user's privacy and the procedure simply terminates, hence ignoring the request incoming from the SP. Otherwise, the key with this index is generated with the PRNG.

The next three steps consist in defining the encryption key k_i as the pair of integers k_x and k_y generated with the PRNG. Then, the cell c in which the user is located is encrypted using the function E with parameter k_i. Finally, the result is sent to the SP together with the value of i that is also stored on the client for the next run of the procedure.

Before describing the encryption function, we first introduce some notation. In our approach we assume that users are moving in a two-dimensional space W which consists in a rectangular grid of $size_x \times size_y$ points. For each point $p \in W$, we denote with p_x and p_y the projection of p on the x and y axis, respectively.

The encryption function E we propose is based on a "modular translation". The idea is to apply, to each point of c, a translation followed by a modulus operation in such a way that no point is moved outside W. For example, if a point is moved by the translation right above the top boundary of W, the modulus operation moves it right above the bottom boundary of W and hence still within W (see Figure 1(a)).

The translation shift value is represented by $\alpha = \langle \alpha_x, \alpha_y \rangle$ which is computed from the key $k_i = \langle k_x, k_y \rangle$ as follows: $\alpha_x = k_x \mod size_x$, $\alpha_y = k_y \mod size_y$.

Procedure 1 *encryptLocation*

Input: a location l, a grid G, the seed K, the value $lastIndex$, the optional value $newIndex$.

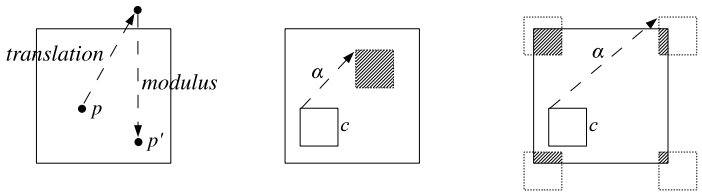
Procedure:

if (issuing request for proximity) **then**
  $i = lastIndex + 1$
else {responding to a proximity request}
  if ($newIndex \leq lastIndex$) **then return**
  $i = newIndex$
end if
k_x is the $2i$-th number generated by the PRNG with seed K
k_y is the $(2i+1)$-th number generated by the PRNG with seed K
$k_i = \langle k_x, k_y \rangle$
c is the cell of G that contains the location l
$c' = E_{k_i}(c)$
send $\langle i, c' \rangle$ to the SP.
store i {for the next execution}

The encryption function E_{k_i} is then specified as:

$$E_{k_i}(c_A) = \bigcup_{p \in c_A} \langle (p_x + \alpha_x) mod\ size_x, (p_y + \alpha_y) mod\ size_y \rangle$$

In practice, $c'_A = E_{k_i}(c_A)$ is computed by applying a transformation to each point of c_A. On the x axis, the transformation consists in shifting the point by α_x and then in applying the module $size_x$. On the y axis the transformation is analogous. It is worth noting that, depending on α and c_A, $E_{k_i}(c_A)$ could be a set of contiguous points (see Figure 1(b)) as well as a set of non-contiguous points (see Figure 1(c))



(a) Translation of a point (b) Contiguous points (c) Non-contiguous points

Fig. 1. Examples of modular translations of a point and of a cell. $E_{k_i}(c)$ represented in gray.

2.4 The *computeProximity* Procedure

The *computeProximity* procedure (see Procedure 2) is run by the SP when it receives two locations encrypted with the same key.

Procedure 2 *computeProximity*

Input: $\langle i, c'_A \rangle$ received from A, which issued a proximity request, and $\langle i, c'_B \rangle$ received from B, which is responding to the request.
Procedure:
1: $dist = mmd(c'_A, c'_B)$ {minimum modular distance}
2: send the boolean value ($dist \leq \delta_A$) to A

The first step of the procedure consists in computing the "minimum modular distance" between c'_A and c'_B as follows:

$$mmd(c'_A, c'_B) = \min_{p \in c'_A, p' \in c'_B} modDist(p, p')$$

where *modDist* is the *modular distance* between p and p'. Intuitively, the modular distance is the Euclidean distance computed as if W were "circular" on both axis. For example, consider two points p and p' (see Figure 2(a)), with the same horizontal position such that p is close to the top boundary of W and p' is close to the bottom boundary. The Euclidean distance of the two points is about $size_y$, while the modular distance is close to zero. The same holds for the other axis (see Figure 2(b)) and also for the combination of the two axis (see Figure 2(c)). Formally, given two points p and p', $\Delta_x = |p_x - p'_x|$ and $\Delta_y = |p_y - p'_y|$, the modular distance is defined as:

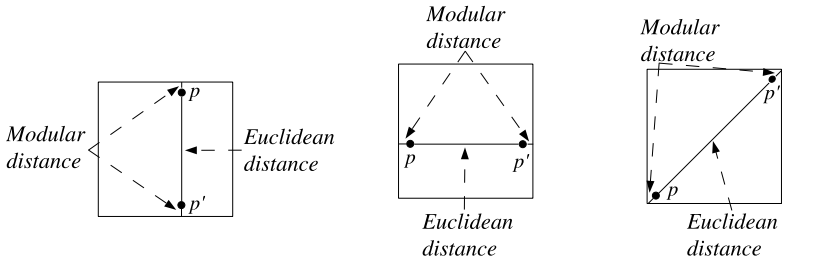
$$modDist(p, p') = min(\sqrt{(\Delta_x)^2 + (\Delta_y)^2}, \sqrt{(size_x - \Delta_x)^2 + (\Delta_y)^2},$$
$$\sqrt{(\Delta_x)^2 + (size_y - \Delta_y)^2}, \sqrt{(size_x - \Delta_x)^2 + (size_y - \Delta_y)^2})$$

The final step of *computeProximity* consists in comparing the minimum modular distance between c'_A and c'_B with δ_A, the proximity threshold of A. The boolean value of this comparison is sent to A.

2.5 The *getResult* Procedure

In the *getResult* procedure (see Procedure 3) user A, which is running the procedure, decides whether B is in proximity or not. This result is obtained considering the boolean value received from the SP and the relative position of the cell c_A, where A is located, with respect to a region called "certainty region" of A. This region, denoted by CR_A, is the set of points of W that are farther than δ_A from the boundaries of W (see Figure 3).

The correctness of the result computed by the *getResult*, as well as the approximation introduced by the protocol and its safety are discussed in Section 3.



(a) On the vertical axis (b) On the horizontal axis (c) On both axis

Fig. 2. Examples of modular distance

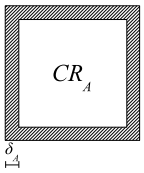


Fig. 3. Example of the certainty region CR_A

3 Analysis of the *Longitude* Protocol

In this section we first discuss the safety of the *Longitude* protocol with respect to privacy protection and then we analyze its correctness and the approximation it introduces. We first introduce a formal proposition that will be used in the protocol analysis.

Proposition 1. *Given two cells c_A and c_B and a key k_i, the encryption function E is such that:*

$$mmd(c_A, c_B) = mmd(E_{k_i}(c_A), E_{k_i}(c_B))$$

Proposition 1 intuitively states that the encryption function E presented in Section 2.3 does not alter the minimum modular distance between c_A and c_B.

3.1 Safety

We first analyze the privacy that the *Longitude* protocol provides to a user with respect to another buddy and with respect to the SP under the assumptions that the SP and the buddies do not collude. Then, we discuss the location information that is disclosed in case collusion occurs.

During the execution of the protocol the only message that A receives containing information related to the location of a buddy B is the boolean value received from the SP as a response to A's request for the proximity of B. When

Procedure 3 *getResult*

Input: The boolean value *res* received from the SP, the cell c where the user running the procedure is located, the certainty region CR of the user running the protocol, the user B which responded to the proximity request.

Procedure:

1: **if** (res = **True** AND $c \subseteq CR$) **then**
2: B is in proximity
3: **else**
4: B is not in proximity
5: **end if**

A receives **True** from the SP (i.e., $mmd(c'_A, c'_B) \leq \delta_A$), due to Proposition 1, A learns that B is located in a cell c_B of G_B such that $mmd(c_A, c_B) \leq \delta_A$. Since A knows c_A and G_B, she can compute the set of cells where B is possibly located. Formally, A cannot exclude B is located in any cell c of G_B such that $mmd(c_A, c) \leq \delta_A$. Analogously, when A receives **False** from the SP A cannot exclude B is located in any cell c of G_B such that $mmd(c_A, c) > \delta_A$. Consequently, the minimum privacy requirement of B with respect to A are guaranteed.

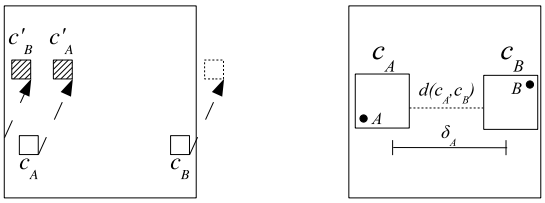
For what concerns the privacy protection with respect to the SP, it is easily seen from the protocol that the SP only learns the minimum modular distance between c'_A and c'_B and hence, due to Proposition 1, the minimum modular distance between c_A and c_B. Since this knowledge does not disclose any information about the location of A and B, we can conclude that *Longitude* guarantees that the SP does not acquire any information about the location of the buddies.

We now turn to consider collusion. If a user B considers all buddies as untrusted, he will probably use the same (coarse) grid for everybody. In this case, even if buddies collude, the minimum privacy requirements are guaranteed. However, if user B has different degrees of trust on her buddies (hence using different grids), and these buddies collude, the location of B could be discovered with high precision by intersecting the location information about B acquired by the colluding buddies. This can be easily avoided by imposing the following constraint on the relationship among the spatial grids used as privacy preferences: *cells from different grids never partially overlap*. In this case, the location of B is never disclosed with a precision higher than the finest grid among those defined for the colluding buddies. In other words, the minimum privacy requirement defined for the most trusted buddy among the colluding ones is guaranteed. Collusion with the SP is not likely in the service model we are considering, since the SP is considered untrusted, while a certain degree of trust is assumed among the participating buddies that indeed share a secret. In the worst case in which the trust model is broken by a buddy A of B colluding with the SP, the SP can obtain and share with A the cell c_B where B is located each time B sends this information encrypted with the secret seed K shared with A. Note that the minimum privacy requirement with respect to A is guaranteed, and that the SP can only obtain the same location information about B available to A.

3.2 Service Precision

We now discuss the correctness of *Longitude* in terms of the service precision it provides. If A receives **False** from the SP then, according to the *computeProximity* procedure, $mmd(c'_A, c'_B) > \delta_A$. Due to Proposition 1, this means that $mmd(c_A, c_B) > \delta_A$. Since $mmd(c_A, c_B)$ is a lower bound to the real distance between A and B, it is guaranteed that B is not in proximity of A. Vice versa, if A receives **True**, it is not possible for A to conclude that B is in proximity, since two forms of approximation are introduced. We now explain the reason for these approximations, and our choice for the conditions under which the protocol declares B's proximity; we will show in Section 4 through extensive experiments the impact these approximations have in practice.

One form of approximation, which we call the *modular-shift error* is due to the fact that the encryption function does not preserve the distance. Indeed, as shown in Figure 4(a), it can happen that, while c'_A is close to c'_B, c_A is far from c_B. This would imply that, when the SP sends **True** to A (i.e., $mmd(c'_A, c'_B) \leq \delta_A$) A does not actually know whether B is in proximity or not. However, it is easily seen that when c_A is in the certainty region CR_A, $mmd(c_A, c_B)$ is equal to the minimum distance between c_A and c_B. In this case A can exclude the *modular-shift error*. Consequently, A knows that $minDist(c_A, c_B) \leq \delta_A$ and considers B as in proximity whenever **True** is returned by the SP, and c_A is contained in CR_A (lines 1-2 of the *getResult* procedure). If **True** is returned but c_A is not contained in CR_A, then A cannot conclude that B is in proximity. As we shall see in our experimental results, this case is very rare and, as a practical and efficient solution, procedure *getResult* returns in this particular case B as not being in proximity. Clearly, this leads to some possible false negative responses. A technical solution to avoid this approximation at some extra cost is to apply a P2P protocol between A and B, whenever this case arises [5].



(a) *modular-shift* error (b) Cell approximation

Fig. 4. Two forms of approximation introduced by the *Longitude* Protocol.

The second form of approximation, which we call *cell approximation*, is due to the fact that B may not be in proximity of A even if $minDist(c_A, c_B) \leq \delta_A$.

Figure 4(b) shows an example of this situation. The consequence of *cell approximation* is that, even if A knows that $minDist(c_A, c_B) \leq \delta_A$, she cannot be sure whether $dist(loc(A), loc(B)) \leq \delta_A$. Nevertheless, in this case A assumes B to be in proximity. This can lead to some false positive cases. In our experimental evaluation we show that for many practically useful grids G_A and G_B, *cell approximation* only minimally affects quality of service.

4 Experimental Evaluation

We performed an extensive experimental evaluation of our solution and we compared it with the Pierre protocol proposed in [7]. In our tests we evaluated the *service precision*, measured as the percentage of correct answers given by the protocol, the *privacy*, measured in terms of the size of the region in which an adversary cannot exclude any of the points as a possible location of a user, and the *system costs*, in terms of communication and computational costs.

Experimental setting. For the tests, we used an artificial dataset of user movements which was obtained using the *MilanoByNight* simulation[4]. We carefully tuned the simulator in order to reflect a typical deployment scenario of a friend finder service, i.e. $100,000$ potential users of this service moving from their homes to entertainment places on the road network of Milan during a weekend night. All the test results shown in this section are obtained as average values computed over $1,000$ users, each of them using the service during the 4 hours of the simulation. Locations are sampled every 2 minutes. The total size of the map is 215 km^2 and the average density is 465 users/km^2. Our techniques were implemented in Java, and tests were performed on a 64-bit Windows Server 2003 machine with $2,4$Ghz Intel Core 2 Quad processor and 4GB of shared RAM.

To represent different levels of privacy, we considered eleven different *levels* of grids. A grid of level 0 consists of 1024×1024 cells having a square shape with an edge of about 15 meters. The grid covers the whole map. Grids of level l are obtained by grouping together 2^l cells of the level 0 grid on each dimension. For example, the grids of level 2 are obtained by grouping together 4×4 cells of the level 0 grid, starting from the cell positioned at the lower left corner. The grid of level 10 contains only one cell covering the entire map.

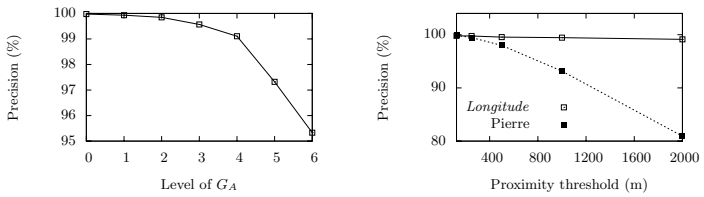
For the sake of simplicity, in our tests we assume that all the users share the same parameters. In particular, in each test we fix a single value of δ (the proximity threshold) and G_A for all the users. Table 1 shows the most relevant parameters of our experimental evaluation. Default values are denoted in bold.

Evaluation of service precision. Figure 5(a) shows the service precision for different levels of G_A using our protocol. It can be observed that for small values of G_A the percentage of correct answers is close to 100%. In particular, using our parameters, the service precision is always above 95% when the size of a cell of G_A is less than 1 km^2.

[4] `http://everywarelab.dico.unimi.it/lbs-datasim`

Table 1. Parameter values

Parameter	Values
δ	125m, 250m, **500m**, 1000m
Level of G_A grid	0, 1, 2, **3**, 4, 5, 6, 7, 8, 9, 10
Average number of buddies	10, 20, 30, 40, **50**, 60, 70, 80, 90, 100

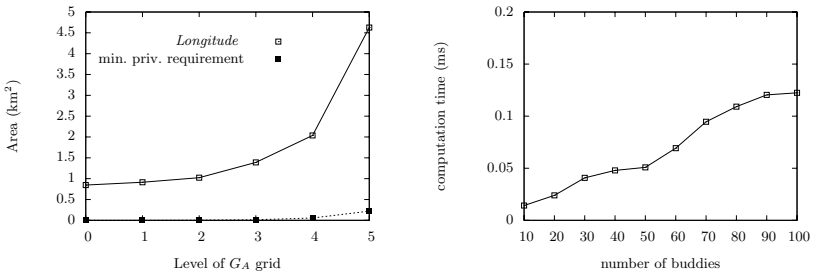


(a) Percentage of correct answers (b) Service precision with different δ

Fig. 5. Service precision

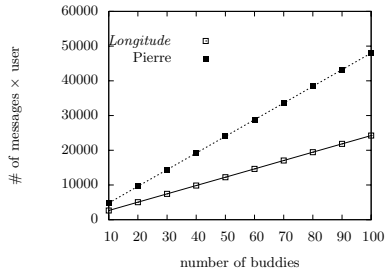
In Figure 5(b) we compare the service precision of the Pierre protocol and *Longitude* for different proximity thresholds. The idea of the Pierre protocol is that the plane is divided into a grid of cells, where the edge of a cell is equal to the proximity threshold δ_A requested by the issuing user A. After a two-party secure computation between A and another user B, A obtains to know whether B is located in the same grid cell or in one of the adjacent ones. The Pierre protocol is subject to a form of approximation similar to the *cell approximation*. However, in the case of the Pierre protocol, the approximation depends on the value of δ_A. Consequently, as shown in Figure 5(b), the precision of the Pierre protocol decreases for large values of the proximity threshold. In contrast, the precision of the *Longitude* protocol is not significantly affected by the proximity threshold.

Evaluation of privacy. Although the minimum privacy requirement is always guaranteed, using *Longitude*, it is desirable for a user to obtain as much privacy as possible. We measure the privacy as the size of the uncertainty region, i.e. the size of the region in which an adversary cannot exclude any of the points as possible location of a user. The larger this region is, the better. Figure 6(a) shows the privacy obtained by a user A for different levels of G_A with respect to another user B when the SP notifies B that A is in proximity. We can observe that *Longitude* always achieves more privacy than the minimum required. Even when using a G_A equal to zero, which is the minimum possible privacy requirement, the average area of uncertainty is around 0.85 km^2. This is approximately the area of the circle centered in B's location and having radius equal to the proximity threshold.



(a) Evaluation of privacy

(b) Client-side computation time



(c) Average number of sent/received messages

Fig. 6. Evaluation of privacy and of performance

Evaluation of system costs. To evaluate the computational costs, we analyze the computation time needed when a user updates her location, both on the client and the server sides. The main parameter affecting this cost is the number of buddies. In Figure 6(b) we can observe that, as expected, the computation time grows linearly with the number of buddies. It should be observed that the computation time is around 0.1 milliseconds with 100 buddies on the desktop machine we used in our tests. We are confident that the computational cost remains sustainable also on high-end mobile devices.

We also measured the server-side computation time and we observed that it grows linearly with the number of buddies and that, even when a user has 100 buddies, the server side computation time is less than 0.3ms.

The metrics we considered to evaluate the communication costs is the total number of messages exchanged by each user (see Figure 6(c)). It can be observed that the number of messages sent by the Pierre and the *Longitude* protocols grows linearly with the number of buddies. However, the number of messages required by the Pierre protocol is almost double with respect to *Longitude*. This is due to the fact that, given n the number of buddies, $2n$ messages are needed to a user when issuing a proximity query using Pierre, and other $2n$ messages are needed to that user to reply to the proximity requests issued by all the buddies. On the contrary, when using *Longitude*, only 2 messages are needed by

a user when issuing a proximity query, and other $2n$ messages are still needed to communicate the encrypted locations requested by the SP for all the buddies.

5 Conclusion and Future Work

We believe that the *Longitude* protocol presented and validated in this paper can be a technical solution for users that would like to enjoy proximity services, but do not necessarily trust the service providers or the security of their infrastructure, as well as for those that want to have more control on the information released to buddies.

The solution proposed in this paper is based on a centralized architecture that enables optimizations that are not possible with a P2P solution. Some optimizations have already been proposed, in centralized architecture, to enhance the system performance at the cost of revealing some location information to the SP (e.g., the *SP-Filtering* protocol, presented in [5]). The centralized architecture can be also exploited to provide other forms of optimization that do not reveal any location information to the SP. One form of optimization is based of the following idea: if the users in a set form a clique (i.e., for each pair of users in the set, the two users are buddies), then a single "group key" can be used, instead of a key for each pair of buddies. This can significantly reduce the number of locations that need to be sent to the SP, hence reducing computation and communication cost. We leave as a future work the evaluation of the performance improvement obtained with this optimization.

Several other issues deserve further investigation. The specification of privacy preferences in terms of spatial granularities requires a study of a user interface that should be at the same time intuitive and effective in graphically showing the uncertainty regions. From a technical point of view, several details need a deeper investigation, including the choice of an adequate PRNG, the refinement of the protocol to provide protection against sophisticated cryptanalysis, as well as time constraints on successive runs of the protocol to prevent attacks based on historical correlation. Another direction we are considering is the extension of our architecture to a hybrid architecture in which the *Longitude* protocol is coupled with P2P algorithms to improve service precision in particular situations.

Acknowledgments

The authors would like to thank the reviewers for their very helpful comments. This work was partially supported by Italian MIUR under grants PRIN-2007F9437X and InterLink II04C0EC1D, and by the National Science Foundation under grant CNS-0716567.

References

1. Arnon Amir, Alon Efrat, Jussi Myllymaki, Lingeshwaran Palaniappan, and Kevin Wampler. Buddy tracking - efficient proximity detection among mobile friends. *Pervasive and Mobile Computing*, 3(5):489–511, 2007.

2. Panos Kalnis, Gabriel Ghinita, Kyriakos Mouratidis, and Dimitris Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1719–1733, 2007.
3. Kun Liu, Chris Giannella, and Hillol Kargupta. An attacker's view of distance preserving maps for privacy preserving data mining. In *Proc. of 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, volume LNCS 4213, pages 297–308. Springer, 2006.
4. Sergio Mascetti, Claudio Bettini, Dario Freni, and X. Sean Wang. Spatial generalization algorithms for LBS privacy preservation. *Journal of Location Based Services*, 2(1):179–207, 2008.
5. Sergio Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, and Sushil Jajodia. Privacy-aware proximity based services. In *Proc. of the 10th International Conference on Mobile Data Management*, pages 31–40. IEEE Computer Society, 2009.
6. Peter Ruppel, Georg Treu, Axel Küpper, and Claudia Linnhoff-Popien. Anonymous user tracking for location-based community services. In *Proc. of the Second International Workshop on Location- and Context-Awareness*, volume LNCS 3987, pages 116–133. Springer, 2006.
7. Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *Privacy Enhancing Technologies*, volume LNCS 4776, pages 62–76. Springer, 2007.