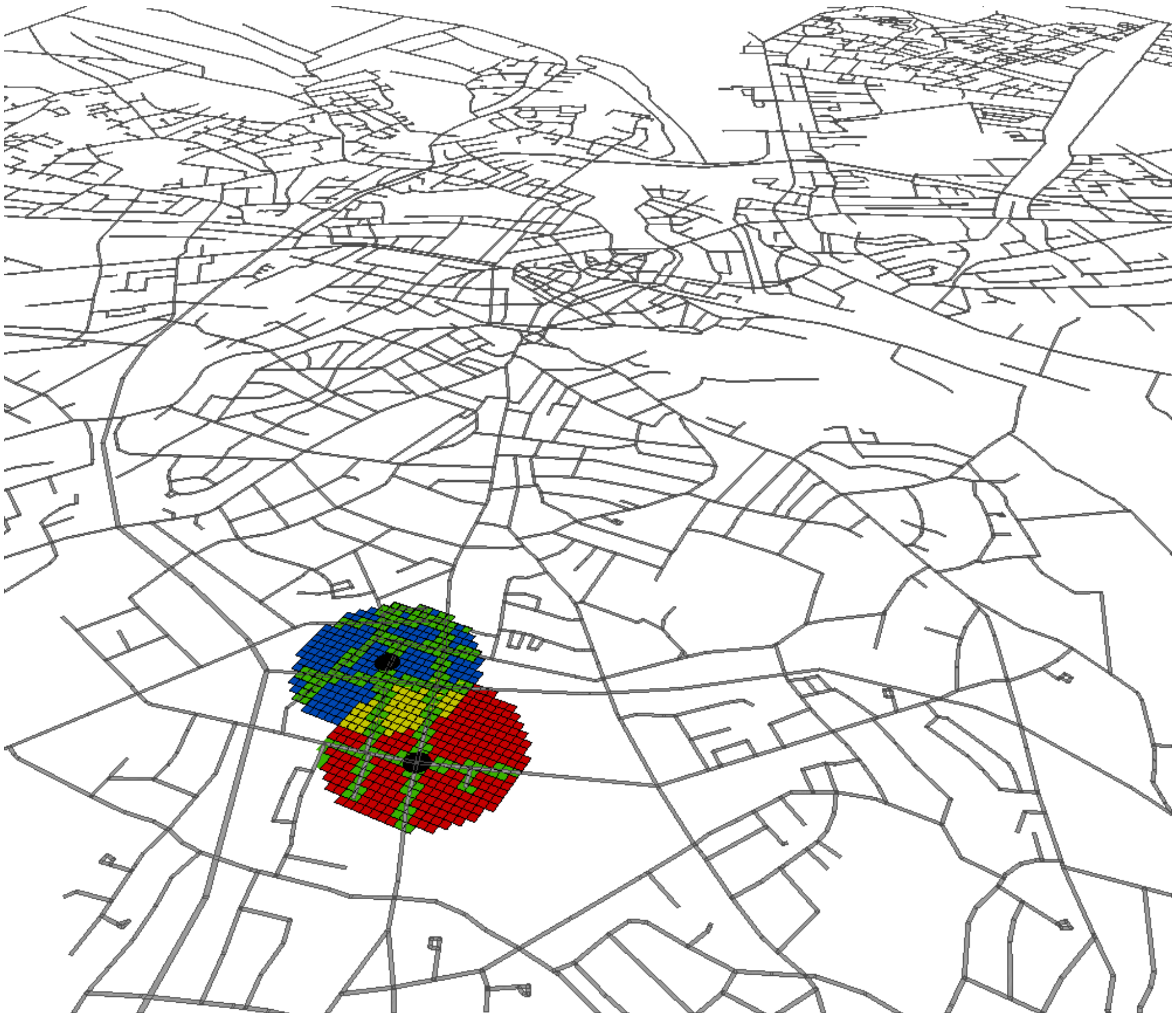
Frameworks For Private Identification Of Nearby Friends



June 2009

UNIVERSITY OF AALBORG

DEPARTMENT OF COMPUTER SCIENCE

Department of Computer Science

Aalborg University

DAT6/DE4

TITLE

Frameworks For Private Identification Of Nearby Friends

PROJECT PERIOD

DAT6/DE4

4. February - 11. June 2009

PROJECT GROUP

Computer Science, d622b

GROUP MEMBERS

Jeppe Rishede Thomsen

Laurynas Šikšnys

SUPERVISOR

Simonas Šaltenis

Man Lung Yiu

NUMBER OF COPIES 5

PAGES IN REPORT 63

PAGES IN APPENDIX 8

PAGES IN TOTAL 74

ABSTRACT

The privacy-aware proximity detection service determines if two users are close to each other without need to disclose their exact locations. Such kinds of services are getting increasingly popular, fueled by an increase in the availability of positioning devices like GPS.

This thesis presents two novel client-server proximity detection frameworks which offer a new set of features, lacked by alternative solutions. One of these is “blind query evaluation” which enables strong user location privacy. The server detects proximity between users by analyzing encrypted rather than spatial position data. Experimental results show that both frameworks are scalable to high number of users and convince that they are applicable for real world application.

Contents

Part I

Introduction

1.1 Motivation

With more than three billion out of the world's six billion people carrying mobile devices, the mobile Internet could soon tower over the PC-based Internet as we know it today. This expansion is enabled by the increased popularity of mobile devices, which are becoming more convenient, cheaper, and powerful. When these devices are equipped with geo-positioning technology, a variety of location-based services (LBSs) are available over the Internet to mobile device users.

Location-based services allow users to request nearest point-of-interest, such as an ATM or a restaurant, use turn-by-turn navigation with real-time traffic information, and locate other people. Users are interested in such services, thus the number of available location-based application is constantly increasing.

Recently emerged location-based mobile social networking services hold substantial potential and it is predicted that their revenues will reach \$3.3 billion by 2013. These services allow users to share real-life experiences via geo-tagged user-generated multimedia content, exchange recommendations about places, identify nearby friends and set up ad hoc face to face meetings.

When users use mobile social networking services (also other LBSs), they are usually required to provide their exact geographical locations. This sometimes threatens them, as many people clearly consider their location privacy a fundamental right. By using such services and sending their locations, users can endanger their security because, for example, an attacker can track them. This tracking capability of attackers opens up many crime possibilities (harassment, car theft, burglary, kidnapping, etc.).

The nearby-friends identification is becoming attractive feature of mobile social networking services as they allow users to see other users' locations on a screen, receive notifications about their proximity, and get their occupation status. Thus more and more commercial online services, like *Google Latitude*, *Loopt*, support it. In such services, when some user require a level of privacy and wants to hide his location on their friends' mobile devices, he usually disables location sharing with a service provider. This also prevents the user from being notified about the proximate friends, as well as user's friend to know about the user's proximity. Thus, due to poor location-privacy support, the nearby-friend detection is not always possible in existing mobile social networking products.

To challenge the problem we present two privacy-aware proximity detection approaches, the FriendLocator and the VicinityLocator. They both are general frameworks to ensure user location privacy in proximity-based services, where a distance between entities reaching some threshold triggers an action.

The FriendLocator and the VicinityLocator can be used to provide location privacy and nearby-friends identification in new or existing mobile social networking products. They may function as a component part such that the software on mobile devices may switch from complete user location sharing into our proposed protocol depending on user privacy requirements. The service in both cases would be able to inform the user about his nearby friends and let his friends to do the same.

1.2 Content

The FriendLocator and the VicinityLocator were explained in two distinct articles, which we here in combine and deliver as our Master's thesis. The work, which has been done during the spring semester of 2009, is introduced by the following time line:

February 1 - March 13 We have produced an 18 page article, introducing the FriendLocator. The article is based on a semester rapport done in the fall semester where primarily existing location privacy, proximity detection solutions were reviewed and the ideas of FriendLocator were presented. The article and previous semester paper were done with the additional co-author Ove Andersen. The article was submitted to SSTD 2009 conference at 13th of March and besides Ove Andersen, also introduces Simonas Šaltenis and Man Lung Yiu as co-authors.

April 27 - May 5 The 18 pages article of FriendLocator was accepted to SSTD 2009 as short research paper, thus during a period of 1 week we prepared 6 pages version of this article. The 6 pages article is included in the Appendix.

March 13 - June 11 Motivated by the additional knowledge about the work done with privacy and proximity detection, we develop a new approach, the VicinityLocator, where we combine ideas of the other authors' solution with ideas of our FriendLocator. We assemble our work into the Master's thesis.

In order to test both approaches we implemented two prototypes, the VicinityLocator, FriendLocator, and one additional competitor, on the .Net platform. Prototypes were tested on a dataset, containing simulated locations of objects, moving from sources to destinations, following the road-network of the German city Oldenburg. The dataset was generated with Brinkhoff's network-based generator.

Both FriendLocator and VicinityLocator approaches were developed to detect proximity between pairs of moving objects without requiring them to reveal any spatial information. Our approaches use client-server architecture and, unlike other solution, do not require users to communicate in peer-to-peer mode. Here the central server checks if two users are close to each other by "blindly' evaluating encrypted representations of user locations for equality.

In the FriendLocator any two users mutually agree on some threshold and then the system notifies them once the Euclidean distance between two users becomes lower than the threshold. On the contrary in the VicinityLocator, every user individually specifies so-called *vicinity regions* around their locations and the system notifies a user if any other user enters his vicinity region. These regions can be any arbitrary shape. Moreover, the precision of proximity detection in the FriendLocator is dependent on the mutually agreed upon distance threshold of two users, in the VicinityLocator however, users can select any level of precision they desire.

Implementations of these two approaches shares general concepts such as encryption, multiple spatial tessellations, mapping of user location into regions, but the understanding of proximity and principles of proximity detection are different.

In the following parts we present FriendLocator (Part II) and VicinityLocator (Part III), followed by conclusion (Part IV) and our submitted 6 page version of the FriendLocator paper in the appendix (Part V).

Part II

FriendLocator - A Location Privacy Aware Friend Locator

Abstract

The so-called friend-locator location-based service notifies a user if the user is geographically close to any of his or her friends. Services of this kind are getting increasingly popular, fueled by the rise of web-based social networking and increase in the availability of positioning devices like GPS. In practice, users require such services to satisfy user-defined privacy requirements. Existing commercial friend-locator applications only support the all-or-none privacy option, i.e., the users exact location is reported to all of the user's friends or it is hidden completely. In the mobile computing, efficient proximity detection methods have been proposed, however, they do not offer any privacy to the users. The challenge is to develop a communication-efficient solution such that (i) it detects proximity between a user and the user's friends, (ii) any other party (e.g., another user or the server) is not allowed to infer the location information of the user, and (iii) users have flexible choices of their proximity detection distances. To address this challenge, we develop a client-server solution for proximity detection based on an encrypted, grid-based mapping of locations. A user's location is converted to a 'meaningless' tuple that enables proximity detection among friend pairs, yet the tuple cannot be used to derive the user's exact location. Experimental results show that our solution is indeed efficient and scalable to a large number of users.

1.3 Introduction

Mobile devices with built-in geo-positioning capabilities are becoming cheaper and more popular [5]. Disclosing their location information (e.g., via Wi-Fi, Bluetooth, or GPRS) mobile users can enjoy a variety of location-based services (LBSs). In particular, mobile social-networking LBSs are predicted to become a multi-billion dollar industry over the next few years [1]. One type of such services is a *friend-locator* service, which shows users their friends' locations on a map and/or helps identify nearby friends. Several friend-locator services, like *iPoki*, *Google Latitude*, and *Fire eagle* [1] are now available on the Internet.

In existing services, the detection of nearby friends can be done only manually by a user, e.g., by periodically checking a map on the mobile device. In addition, this works only if the user's friends agree to share their exact locations or at least obfuscated location regions (e.g., downtown area). In case all users' friends require complete privacy, they disable their location sharing and this prevents the user from finding his or her nearby friends. Typically, LBS users require some level of privacy and may even feel unsecure, if it is not provided [9]. Thus, due to poor location-privacy support, the nearby-friend detection is not always possible in existing friend-locator products.

[1]http://www.ipoki.com; http://www.google.com/latitude; http://fireeagle.yahoo.net

The challenge is to design a friend-locator LBS that preserves the users' location privacy and enables automatic detection of nearby friends. Such a service must deliver notifications to a user when any of his or her friends is nearby. It must be efficient in terms of communication costs and users must have a flexibility to choose different preferences for nearby-friend detection.

In the research areas of databases and mobile computing, efficient proximity detection methods for moving objects have been proposed [2, 17]. However, these methods do not offer any location privacy. On the other hand, many location-privacy solutions exist [13, 6, 3, 7], but they mostly focus on querying the service provider for publicly known information (e.g., nearby restaurants or cinemas) using the 'hidden' location of a user as a query point. In contrast, in this paper, the user's locations are both query and data points and they need to be hidden from the service provider and other users.

To address the challenge, we develop a client-server, location-privacy aware friend-locator LBS, the FriendLocator. The proposed solution is based on both spatial cloaking and encryption. Any user location is mapped into a grid cell and then converted into an encrypted tuple before it is sent to the server. Having received the encrypted tuples from the users, the server can only compute proximity between them, but not deduce their actual locations. As the proximity detection is done at a central server, users do not know the exact locations of their friends. To optimize the communication cost, the FriendLocator employs a flexible region-based location-update policy where regions shrink and expand depending on the distance of a user from his or her closest friend.

The paper is organized as follows. We briefly review related work in Section 1.4 and then define our problem setting in Section 1.5. FriendLocator is presented in Section 1.6. Then, its resilience against two advanced types of attacks is studied in Section 1.7. Section 1.8 presents extensive experimental results of our proposed methods and Section 1.9 concludes the paper.

1.4 Related Work

In this section, we review relevant work on location privacy and proximity detection.

1.4.1 Location privacy

In the most common setting assumed in location-privacy research, an LBS server maintains a *public* set of points-of-interest (POI), such as gas stations. The goal is then to retrieve from the server the nearest POIs to the user, without revealing the user's *private* location q to the server. In contrast, the users' locations are both query locations and points-of-interest in our friend-locator problem setting.

Solutions for location privacy can be broadly classified into two categories: spatial cloaking and transformation. *Spatial cloaking* is applied to generalize the user's exact location q into a region Q', which is then used for querying the server. This approach can then be further divided into two sub-categories: spatial k-anonymity and obfuscation. Most k-anonymity methods [8, 13] employ a trusted third-party anonymizer to maintain the locations of all users. At query time, the anonymizer expands the user's location q into a k-anonymous region Q' such that it contains q and the locations of at least other $k-1$ users. The region Q' is then sent to the LBS server, which returns all the results that are relevant to any point in Q'. Even if the attacker knows the locations of all users at the time of the query, the identity of the querying user can be inferred only with the probability $1/k$.

The drawback of this approach is that the trusted anonymizer can become both a performance bottleneck and the single point of attack. In contrast, *obfuscation* techniques [6, 3] do not require any trusted anonymizer as the region Q' is computed at the client side. The recently proposed SpaceTwist [18] can also be viewed as an obfuscation technique because it does not employ any trusted anonymizer.

Finally, the *transformation* approaches [10, 7] map the user's location q and all POIs to a transformed space, in which the LBS server evaluates queries blindly without knowing how to decode the corresponding real locations of the users. However, the method of Khoshgozaran and Shahabi [10] does not guarantee the accuracy of query results whereas the method of Ghinita et al. [7] incurs high computational time at the server.

1.4.2 Proximity detection

Given a set of mobile users and a distance threshold ϵ, the proximity detection problem is to report, continuously, all events when a pair of mobile users comes within the distance ϵ of each other. Most of the existing solutions [2, 17] focus mainly on optimizing the communication and computation costs, rather than offering privacy to the users. Xu et al. [17] develop a centralized solution for a problem called multi-body constraint detection, which is a generalization of the proximity detection problem. However, the solution requires each mobile client to report his exact location to the server. Clearly, this violates the privacy requirements of users. Amir et al. [2] propose a peer-to-peer solution, called *Strips*, to reduce the communication cost of proximity detection. A strip of width ϵ is established midways between a pair of friends u and v. As long as both users do not travel across the strip, they are guaranteed to be not within proximity, so they do not need to communicate with each other. Otherwise, the users' u and v exchange their locations. Then, depending on the actual distance between them, they either establish a new strip or detect proximity. Unfortunately, the performance of the peer-to-peer method does not scale well in case each user has a large number of friends. Furthermore, the peer-to-peer approach requires users to exchange their exact locations, so it does not provide sufficient level of privacy for our application. These two issues motivate us to develop an efficient privacy-aware friend-locator solution.

Ruppel et al. [15] develop a centralized solution that supports proximity detection and provides the users a certain level of privacy. It first applies a *distance-preserving mapping* (a rotation followed by a translation) to convert the user's location q into a transformed location q'. Then, a centralized proximity detection method is applied to detect the proximity among those transformed locations. However, Liu et al. [11] points out that such distance-preserving mapping is not safe and the attacker can easily derive the mapping function and compute the users' original locations.

In contrast, our solution employs encrypted coordinates rather than distance-preserving mapping, rendering it difficult for the attacker to decode the true locations of the users.

Some of the above-mentioned privacy protection methods can be adapted to provide a certain level of privacy when used with the proximity detection methods. In particular, the experimental study described in Section 1.8 compares our proposed solution with a baseline solution, which combines the Strips technique [2] with a spatial cloaking method, such that (i) it correctly detects proximity among the users, (ii) it offers some privacy via spatial cloaking, and (iii) its communication cost is lower than Strips due to the use of a centralized (untrusted) server.

As described in the following, encrypted tuples are used in our proposed solution so it offers a much stronger notion of privacy than the baseline approach.

1.5 Problem Definition

In this section we introduce relevant notations and formally define the problems of proximity detection and privacy-aware friend locating.

We assume a setting where a large number of mobile device users form a social network. These devices, called mobile terminals (MT), are equipped with positioning and communication technology and can communicate with a central location server (LS) in an online mode. We use the terms *mobile terminals* and *users* interchangeably and denote the set of all MTs and their users in the system by $\mathbf{M} \subset \mathbb{N}$.

The friend-locator LBS notifies two users $u, v \in \mathbf{M} | u \neq v$ if u and v are friends and proximity between u and v is detected. Given distance thresholds ϵ and λ, proximity and separation of two users u and v is defined as follows [2]:

1. if $dist(u, v) \leq \epsilon$, users u and v are in proximity;

2. if $dist(u, v) \geq \epsilon + \lambda$, users u and v are in separation;

3. if $\epsilon < dist(u, v) < \epsilon + \lambda$, the service can freely choose to classify users u and v as being either in proximity or in separation.

Here, $dist(u, v)$ denotes the Euclidean distance between u and v. Parameter ϵ is called the *proximity distance*, and it is selected by the two users' u and v. Parameter $\lambda \geq 0$ is a service precision parameter and it introduces a degree of freedom in the service.

As different pairs of friends may want to choose different proximity distances, we use $\epsilon(u, v)$ to denote the proximity distance for the pair of users $u, v \in \mathbf{M}$. Note that $\epsilon(u, v)$ is defined only if u and v are friends. For simplicity, we assume all friendships to be mutual, i.e., if v is a friend of u, then u is a friend of v, and the proximity distance to be symmetric, i.e., $\epsilon(u, v) = \epsilon(v, u)$ for all friends $u, v \in \mathbf{M}$.

We assume that for any pair of friends, a proximity notification must be delivered to MTs as soon as the proximity is detected. The next proximity notification is sent to this pair of users only after their separation is detected followed by the new detection of proximity.

For simplicity, separation notifications are not considered in this paper, but it is trivial to augment the proposed algorithms to send such notifications.

The friend-locator LBS must be efficient in terms of mobile client communication and provide the following privacy guaranties for each user $u \in \mathbf{M}$:

- The exact location of u is not disclosed to any party, including any other user and the location server.
- User u permits the proximity to be detected only with his friends and does not allow anybody else to detect his proximity.

The following two sections present the proposed friend-locator algorithms that meet these requirements.

1.6 Our Proposed Solution

We propose a novel, incremental grid-based proximity detection algorithm. It is designed for a client-server system architecture and satisfies user location privacy requirements, defined in Section 1.5. First, we present a general proximity detection idea and its extension. Then, the algorithms of the FriendLocator, our privacy-aware friend-locator LBS, are presented. Table 1.1 summarizes the notations used in this paper.

1.6.1 Grid-Based Proximity Detection

Let us consider three parties: two friends, u_1 and $u_2 \in \mathbf{M}$, and the location server. Both users can send and receive messages to and from LS. User u_1 is interested in being informed by LS

Notation	Meaning
$\mathbf{M}$	A set of all MTs or their users
ϵ	The proximity distance
λ	The precision parameter
CM	A function that maps a 2D spatial location to an LMG cell
Ψ, E_Ψ	Ψ is a one-to-one encryption function, used by E_Ψ to encrypt LMG cell coordinates into a four-tuple of encrypted values
Γ	A function that detects proximity or separation given the four-tuples of encrypted values of two users
L	A function mapping a proximity level to an LMG cell size
$L_\epsilon(u, v)$	The proximity level selected by users u and v

Table 1.1: Table of Notations

when user u_2 is within proximity and vice versa.

Assume that users' u_1 and u_2 share a homogeneous grid with square cells of width $d > 0$. Each column (row) of this grid has associated a column-wise (row-wise) unique number, which we term an encryption number. The grid, together with the encryption numbers, constitutes a Location Mapping Grid (LMG). The LMG is generated by two users agreeing on a shared cell size d and a function Ψ. Function $\Psi : \mathbb{N} \mapsto \mathbb{N}$ is a one-to-one encryption function that maps a column or row number to a corresponding encrypted number[2]. The Ψ and d are secret and known only to u_1 and u_2. In practice, Ψ can be implemented as a keyed *secure hash function* (e.g., SHA-2) such that it is computationally infeasible for the attacker to break.

Each time u_1 or u_2 moves in the Euclidean space and enters a new cell of LMG, the following steps are taken:

- The user maps the current location (x, y) into an LMG cell (k,m)=$CM(x, y, d)$, where

$$CM(x, y, d) = (\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor). \tag{1.1}$$

- The user maps cell coordinates (k,m) into a 4-tuple of encrypted values $e = E_\Psi(k, m)$, where

$$E_\Psi(k, m) = (\Psi(k), \Psi(k + 1), \Psi(m), \Psi(m + 1)). \tag{1.2}$$

  Let the returned e be (α^-,α^+,β^-,β^+). Then, (α^-,α^+) and (β^-,β^+) are encrypted values of two adjacent columns k and $k + 1$ and two adjacent rows m and $m + 1$ respectively.

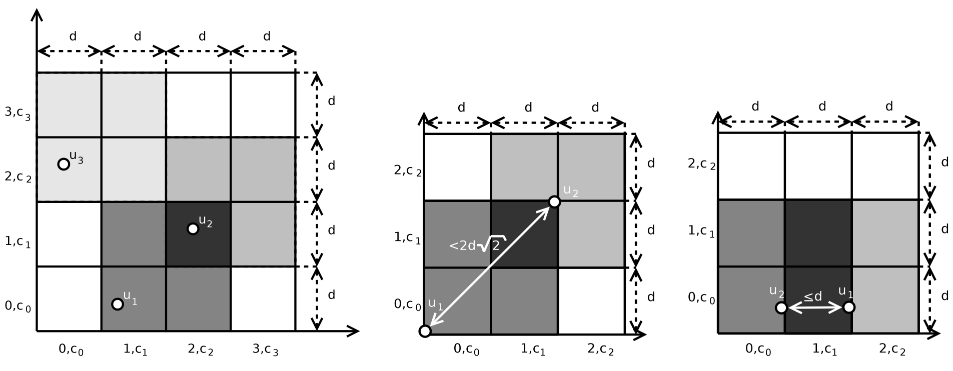
- The user sends the 4-tuple e to LS.

Due to u_1 and u_2 using identical LMG, with the same encrypted number assignments for

[2]For simplicity, we use the same encryption function for both columns and rows.

Col/ Row	Ψ
0	c_0 (5)
1	c_1 (9)
2	c_2 (1)
3	c_3 (7)

(a)



(b) (c) (d)

Figure 1.1: Example of proximity detection

each column and row, LS can detect proximity between them by checking if the following function is true:

$$\begin{aligned}\Gamma(e_1, e_2) &= \big((e_1.\alpha^- = e_2.\alpha^-) \vee (e_1.\alpha^- = e_2.\alpha^+) \vee (e_1.\alpha^+ = e_2.\alpha^-)\big) \\ &\quad \wedge \big((e_1.\beta^- = e_2.\beta^-) \vee (e_1.\beta^- = e_2.\beta^+) \vee (e_1.\beta^+ = e_2.\beta^-)\big)\,. \end{aligned} \tag{1.3}$$

Parameters e_1 and e_2 are 4-tuples delivered from users' u_1 and u_2 respectively. Note that since Ψ is a one-to-one mapping, Γ is evaluated to *true* if and only if k_{u_1} or $k_{u_1} + 1$ matches k_{u_2} or $k_{u_2} + 1$ and m_{u_1} or $m_{u_1} + 1$ matches m_{u_2} or $m_{u_2} + 1$, where (k_{u_1}, m_{u_1}) and (k_{u_2}, m_{u_2}) are LMG cells of users u_1 and u_2 respectively.

Figures 1.1a and 1.8b depict a simple example that explains how LS can detect proximity or separation between u_1, u_2, and u_3, utilizing 4-tuples of encrypted values and Γ. Here, three users share an LMG with cell sizes equal to d. Each row and column has encrypted values $c_0 \dots c_3$ assigned according to Ψ, which is shown in Figure 1.1a. The algorithm maps locations of users u_1, u_2, u_3 to cells $(1, 0)$, $(2, 1)$, $(0, 2)$ respectively. Using function Ψ, the corresponding 4-tuples of encrypted values, $e_1 = (c_1, c_2, c_0, c_1)$, $e_2 = (c_2, c_3, c_1, c_2)$, and $e_3 = (c_0, c_1, c_2, c_3)$, are computed and sent to LS. Due to $e_1.\alpha^+ = e_2.\alpha^- = c_2$ and $e_1.\beta^+ = e_2.\beta^- = c_1$, $\Gamma(e_1, e_2) = \mathit{true}$, thus LS reports proximity between u_1 and u_2. No proximity for u_3 is reported, as $\Gamma(e_3, e_1) = \Gamma(e_3, e_2) = \mathit{false}$.

Figures 1.1c and 1.1d visualize two proximity detection extremes. The locations of two users, u_1 and u_2, are shown relatively to LMG. In both cases, Γ is equal to *true*, however, in the case depicted in Figure 1.1c, Γ returns *false* when user u_2 moves away from u_1 on a diagonal and $dist(u_1, u_2)$ becomes greater or equal to $2d\sqrt{2}$, while, in Figure 1.1d, Γ returns *false* when user u_2 moves away from u_1 in a horizontal direction and $dist(u_1, u_2)$ becomes higher than d.

This means that, when two users u_1 and u_2 are considered proximate by the algorithm, the upper bound of the distance between them varies in the range $[d, 2d\sqrt{2})$ depending on how the locations of the users map to LMG. This is formalized in Lemmas 1, 2 and Theorem 3.

Lemma 1. *If LMG's cell size is d, e_1 and e_2 are 4-tuples of encrypted values from users u_1 and u_2, and $dist(u_1, u_2) \leq d$, then $\Gamma(e_1, e_2)$ is always true.*

Proof. Assume that user's u_1 location is (x_1, y_1) and it can be mapped to LMG cell $(k_1, m_1) = CM(x_1, y_1, d)$. When $dist(u_1, u_2) \leq d$, the user u_2 can be in location $(x_1 \pm d_x, y_1 \pm d_y)$, where $0 \leq d_x \leq d$ and $0 \leq d_y \leq d$. According to the definition of CM, this location can be mapped into cell $(k_2, m_2) = (\lfloor \frac{x_1}{d} \pm \frac{d_x}{d} \rfloor, \lfloor \frac{y_1}{d} \pm \frac{d_y}{d} \rfloor)$. As $\frac{d_x}{d} \leq 1$ and $\frac{d_y}{d} \leq 1$, (k_2, m_2) can only be equal to one of (k_1, m_1), $(k_1 \pm 1, m_1)$, $(k_1, m_1 \pm 1)$, or $(k_1 \pm 1, m_1 \pm 1)$. For the $\Gamma(e_1, e_2)$ to be true, current or adjacent column and row numbers of users u_1 and u_2 must match, i.e. $((k_1 = k_2 - 1) \lor (k_1 = k_2) \lor (k_1 = k_2 + 1)) \land ((m_1 = m_2 - 1) \lor (m_1 = m_2) \lor (m_1 = m_2 + 1))$. Observe that this holds for all cases of u_2's cell coordinates, proving that $\Gamma(e_1, e_2)$ is always true when $dist(u_1, u_2) \leq d$. □ □

Lemma 2. *If LMG's cell size is d, e_1 and e_2 are 4-tuples of encrypted values from users u_1 and u_2, and $dist(u_1, u_2) \geq 2\sqrt{2}d$, then $\Gamma(e_1, e_2)$ is always false.*

Proof. Let us start with a one-dimensional case and define user's u_1 location by coordinate x_1, which is mapped to $k_1 = \lfloor \frac{x_1}{d} \rfloor$. Let us vary x_1 in order to find a maximal distance $dx_{max} \geq 0$ from user u_1 to user u_2 such that u_1's and u_2's locations are mapped to the same or adjacent 1D cells, i.e., $dx_{max} = max\{r \in \mathbb{R}_{\geq 0} | \lfloor \frac{x_1}{d} + \frac{r}{d} \rfloor - \lfloor \frac{x_1}{d} \rfloor \leq 1\}$. In cases when $x_1 = n \cdot d$, $n \in \mathbb{N}$, we have $dx_{max} = 2d - \delta$, where $\delta > 0$ is a small real number. If we introduce another dimension, $dy_{max} = 2d - \delta$. Then, $d_{max} = \sqrt{2(2d - \delta)^2} < 2d\sqrt{2} - \delta$, the maximal distance between u_1 and u_2 in 2D space such that the two users fall within the same or adjacent cells (i.e., $\Gamma(e_1, e_2) = true$). As d_{max} is smaller than $2\sqrt{2}d$, the algorithm will always detect no proximity when $dist(u_1, u_2) \geq 2\sqrt{2}d$. □ □

Theorem 3. *The LMG-based proximity detection approach that uses LMG with cell size d detects proximity with parameter settings $\epsilon = d$ and $\lambda = \epsilon(2\sqrt{2} - 1)$.*

Proof. The theorem follows from the definition of proximity and separation in Section 1.5, Lemmas 1 and 2, and the observation that $dist(u, v) \geq d + d(2\sqrt{2} - 1)$ is equivalent to $dist(u, v) \geq 2\sqrt{2}d$. □ □

Note that an encrypted 4-tuple cannot be directly mapped back to the real world position without knowing Ψ and d. If Ψ is strong and secure, our approach provides privacy for user locations in case of a malicious LS. The approach also provides user location privacy in case of malicious friends, because none of the user's friends knows encrypted values of the user. However, user location privacy can be breached in cases of collectively malicious LS and some friend's MT. In this case, the attacker is able to find the real world region, containing the user. We address this type of attack in Section 1.7.2, but note that, even if such an attack is successful, the user's location is revealed only with a precision of a grid cell, which indirectly cloaks the exact location of the user.

1.6.2 Incremental Proximity Detection Approach

The idea presented in the previous section is not flexible, as it does not allow for pairs of users to choose individual proximity distances (ϵ). Also if ϵ (and d) is too small, too many location updates may be generated as the users change grid cells. To solve these problems, in the following, we extend the idea and present the incremental proximity detection algorithm that uses multiple grids.

We let all users in $\mathbf{M}$ share a list of LMGs with different cell sizes instead of a single LMG. Level zero LMG in that list has the largest grid cells. Cell sizes gradually decrease going from lower to higher level grids. Each user generates such a list of LMGs utilizing two shared private functions ψ and L. Function ψ was already defined in Section 1.6.1. Function $L : \mathbb{N} \mapsto \mathbb{R}$ is a strictly decreasing function which maps level number to a cell size of LMG at that level.

Theorem 3 and function L gives a correspondence between each level and proximity detection distance. An LMG at level $l \in \mathbb{N}$ can be used to detect proximity with the following settings $\epsilon = L(l)$, $\lambda = L(l) \cdot (2\sqrt{2} - 1)$. Thus, every two friend's $u_1, u_2 \in \mathbf{M}$ can choose an LMG level, called *proximity level*, that corresponds to their proximity detection settings best. Let us denote this proximity level $L_\epsilon(u_1, u_2) : \mathbf{M} \times \mathbf{M} \mapsto \mathbb{N}$. Again, as we have pairs of mutual friends and non-friends in the system, this function is defined only for friend pairs and it is symmetric, i.e., $L_\epsilon(u_1, u_2) = L_\epsilon(u_2, u_1)$ for all $u_1, u_2 \in \mathbf{M}$, if u_1 and u_2 are friends. In addition to the increased flexibility of choosing the proximity distance, multiple levels of grids enable incremental detection of proximity, which is made possible by the following theorem.

Theorem 4. *Let LMGs at all levels be aligned to coordinate axes and $L(l-1) \geq 2L(l)$ for $l \in \mathbb{N}$. Then, if proximity between users u_1, u_2 was detected at level l, i.e., Γ equals to true at that level, it will also be detected at level $l-1$.*

Proof. Using the fact that Γ is true at level l and reusing the proof of Lemma 2, we find a maximal distance between u_1 and u_2 along one of the coordinate axes, the $dx_{max} = 2L(l) - \delta$, where δ is a small positive number. Then, we can find $L(l-1)$ that guarantees proximity along the chosen coordinate axis for level $l-1$, i.e., $(\lfloor \frac{(x_1 + dx_{max})}{L(l-1)} \rfloor - \lfloor \frac{x_1}{L(l-1)} \rfloor) \leq 1$, where x_1 is a coordinate of u_1. The condition holds when $\frac{dx_{max}}{L(l-1)} \leq 1$, thus $\frac{2L(l)-\delta}{L(l-1)} \leq 1$ and $L(l-1) \geq 2L(l)$.

The same holds in the other dimension, thus when $L(l-1) \geq 2L(l)$ no two user locations exist for which Γ holds at level l, but not at level $l-1$. $\square$ $\square$

Assume that $L(l) = g \cdot 2^{-l}$, where g is some level zero cell size. Then $L(l-1) = 2L(l)$ and the list of LMGs satisfies the conditions of Theorem 4, which means that, if a proximity between friends u_1 and u_2 is detected at level $L_\epsilon(u_1, u_2)$, it will also be detected at level $L_\epsilon(u_1, u_2) - 1$. This enables incremental proximity detection under which users stay at the grid of the lowest-possible level such that few grid-cell updates are necessary. Only when proximity at a low level is detected, the users are asked to switch to a higher level. The following section presents the algorithms that implement this approach.

1.6.3 The FriendLocator design

The design of FriendLocator is based on an incremental proximity detection idea and employs client-server architecture. It provides privacy for all user locations and offers users a choice of discrete proximity detection settings, i.e., discrete values of ϵ and λ.

As described in Section 1.6.2, all users in the system share the list of LMGs, defined by Ψ and L. The FriendLocator uses a trusted third-party server (TTS) as one possible alternative to achieve this. The TTS is accessible from all MTs and LS and it maintains and distributes L_ϵ, Ψ, and L to the appropriate parties. Note that TTS does not participate in friend locating, thus knows no user-location data.

We define FriendLocator by providing handler algorithms for different type of software events on mobile terminals and the location server. In particular,

onMessageReceived(*msg*, *arg*) is a handler executed on MT or LS each time one receives a message of type *msg* with arguments *arg* from the other party. A summary of message types with their arguments is presented in Table 1.2.

onLocationChange(l_{new}) is a handler executed on MT each time its positioning device reports a new geographical location l_{new}.

Algorithm 1 specifies a behavior of MT. It contains local data definitions, functions, and main software event handlers employed by a client.

On initialization, MT contacts TTS to get Ψ and L. The TTS replies with an M_{ttsC} message and MT stores the data in its memory.

Every MT locally stores a stack CS, which contains unencrypted LMG cell coordinates (k, m) for different levels, starting from level zero at the bottom of the stack and up to u's *current level*, $|CS| - 1$. When MT u gets a new location from its positioning device, the **onLocationChange** handler is fired. If u's location change triggers an LMG cell change at

Message	Args	Sender	Description
M_{el}	u, l, e	MT	MT with id equal to u sends this message to LS in order to report its encrypted values $e = (\alpha^-, \alpha^+, \beta^-, \beta^+)$ at level l.
M_{prox}	v	LS	MT is informed that his friend v is within proximity.
M_{LevInc}	l	LS	MT is asked to increase its level up to level l.
M_{ttsC}	L, Ψ	TTS	MT is provided functions L and Ψ when any of them changes on TTS.
M_{ttsS}	$\mathbf{M}$, L_ϵ	TTS	LS is provided a set of all users $\mathbf{M}$ and function L_ϵ when any of them changes on TTS. Note that L_ϵ encapsulates a social-network.

Table 1.2: FRIENDLOCATOR LBS message types

some levels starting from level $|CS|$ and lower, all such cells are removed from CS automatically decreasing u's current level. Then, function **pushLocAndSend** is called, which computes u's cell coordinates for level $|CS| + 1$, adds them to CS, and sends their encrypted version to LS.

If the location server asks MT to increase his current level to l (by sending an M_{LevInc} message), the corresponding handler executes **pushLocAndSend** multiple times to increase the level (lines 13 and 14). Handling of M_{prox} message is trivial.

Algorithm 2 provides local data definitions and software event handlers for LS. Note that function $get(S, i)$ gets the i-th element of stack S counting from its bottom.

The LS maintains a set of user identification numbers, $\mathbf{M}$, and a set of proximity levels L_ϵ for all pairs of friends within the social-network. In addition, for every $u \in \mathbf{M}$, a stack $SL(u)$ is maintained, containing 4-tuples of encrypted values for levels zero to u's current level. A set $\mathbf{P}$ is used to remember pairs of friends currently in proximity in order to avoid sending duplicated M_{prox} messages.

Once the location server gets from TTS an M_{ttsS} message and saves $\mathbf{M}$ and L_ϵ, it can start processing M_{el} messages from MTs. A message M_{el} with arguments u, l, and e is handled by, first, removing old 4-tuples of encrypted values of levels higher or equal than l from the stack $SL(u)$ (lines 2 and 3). The received e is then pushed to $SL(u)$, followed by searching for a proximity between u and some $v \in \mathbf{M}$ with the defined $L_\epsilon(u, v)$. Let l_m be the highest level commonly employed by users u and v, but not higher than $L_\epsilon(u, v)$. If Γ is true at level l_m, two cases are possible. If $l_m = L_\epsilon(u, v)$, then LS sends proximity messages to users u and v (lines 10 to 12). Otherwise one or both of the users are on level l_m that is too low, thus LS tells one or both users to switch to a common level $l_m + 1$ (lines 14 to 17).

To illustrate the working of the algorithms an example scenario is visualized in Figure 1.2. It shows the geographical locations of two friends u_1 and u_2, and their mappings into LMGs at

```
Data: $u \in \mathbf{M}$ - current user ID.
$CS$ - a stack of LMG cell coordinates $(k, m)$ at different levels, starting from level 0.
$l_{cur}$ - $u$'s current geographical location.
1  pushLocAndSend()
2      Push $CM(l_{cur}.x, l_{cur}.y, L(|CS|))$ result to stack $CS$;
3      Send to LS $M_{el}(u, |CS| - 1, E_{\Psi}(top(CS)))$;
4  onLocationChange(Location $l_{new}$)
5      $l_{cur} \leftarrow l_{new}$;
6      $wasPopped \leftarrow false$
7      while $|CS| > 0$ and $top(CS) \neq CM(l_{cur}.x, l_{cur}.y, L(|CS| - 1))$ do
8          Pop from stack $CS$;
9          $wasPopped \leftarrow true$;
10     if $wasPopped$ or $|CS| = 0$ then
11         pushLocAndSend(G)
12 onMessageReceived(Message $M_{LevInc}$, Level $l$)
13     while $|CS| \leq l$ do
14         pushLocAndSend(G)
15 onMessageReceived(Message $M_{prox}$, Friend $v$)
16     Output "Friend ",$v$, " is in proximity!";
```

Algorithm 1: The MT's event handlers in FRIENDLOCATOR.

4 different snapshots of time. Note that lower level grids are on top in the figure. Assume that u_1 and u_2 have agreed on $L_\epsilon(u_1, u_2) = 2$ and have already sent their encrypted values for levels 0 and 1 to LS. Figure 1.2a visualizes a scenario when LS detects a proximity at level 0, but not at level 1. As $L_\epsilon(u_1, u_2) > 1$, nothing happens until a location change. Figure 1.2b shows the case where both users have changed their geographical location. User u_2 did not go from one cell to another at his current level 1, thus it did not report a new 4-tuple of encrypted values. On contrary, user u_1 changed the cell not only on level 1, but also on level 0, thus it reports new 4-tuples of encrypted values for level $l_{cur}(u_1) = 0$. The LS detects a proximity between u_1 and u_2 at level 0 and commands u_1 to switch into a level 1, because $L_\epsilon(G, u_1, u_2) > 0$. Figure 1.2c shows user locations mapping into LMGs when u_1 have delivered a 4-tuple of encrypted values for his new current level 1. Again, LS detects proximity at level 1 and commands both users u_1 and u_2 to switch to level 2. When 4-tuples of encrypted values for level 2 are delivered from users u_1 and u_2 to LS, it detects the proximity at this level (see Figure 1.2d) and, because $2 = L_\epsilon(u_1, u_2)$, proximity notifications are sent to u_1 and u_2.

```
Data: M - a set of users; $SL(u) \forall u \in \mathbf{M}$ - a stack of user's $u$ 4-tuples of encrypted
      values for different levels of LMGs; $\mathbf{P} \subseteq \mathbf{M} \times \mathbf{M}$ - a set of currently proximate
      friend pairs.
1  onMessageReceived(Message $M_{el}$, User $u$, Level $l, e$)
2      while $|SL(u)| > 0$ and $|SL(u)| \geq l$ do
3          Pop from stack $SL(u)$;
4      Push $e$ to stack $SL(u)$;
5      foreach $v \in \mathbf{M}$ such that $v \neq u$ and $|SL(v)| > 0$ and $L_\epsilon(u, v)$ is defined do
6          $l_m \leftarrow min(|SL(u)| - 1, |SL(v)| - 1, L_\epsilon(u, v))$;
7          if $\Gamma(get(SL(u), l_m), get(SL(v), l_m))$ then
8              if $l_m = L_\epsilon(u, v)$ then
9                  if $\langle u, v \rangle \notin \mathbf{P}$ then
10                     insert the pair $\langle u, v \rangle$ into P;
11                     send $M_{prox}(v)$ to MT $u$ ;
12                     send $M_{prox}(u)$ to MT $v$;
13             else
14                 if $l_m = |SL(u)| - 1$ then
15                     send $M_{LevInc}(l_m + 1)$ to MT $u$
16                 if $l_m = |SL(v)| - 1$ then
17                     send $M_{LevInc}(l_m + 1)$ to MT $v$
18         else
19             remove the pair $\langle u, v \rangle$ from P;
```

Algorithm 2: The LS's event handlers in FRIENDLOCATOR.

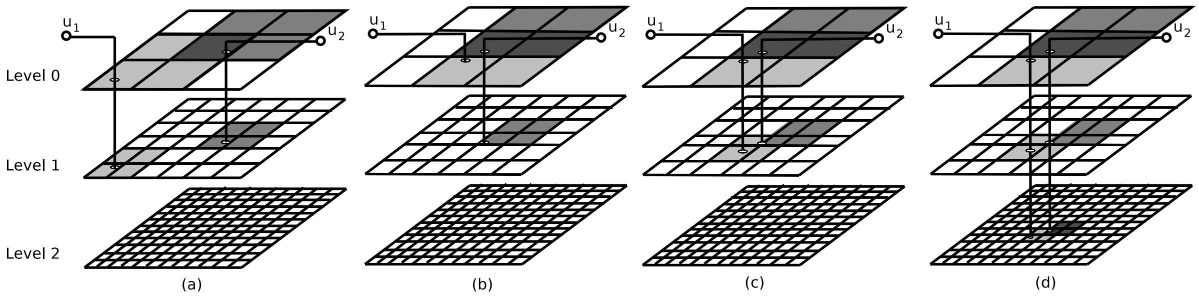


Figure 1.2: Proximity detection in the FRIENDLOCATOR

Note that the presented algorithms implement a kind of adaptive region-based update policy. If a user is far away from his friends, he or she stays at a low-level grid with big cells, which results in few cell-change updates as the user moves. Only when the user approaches one of the friends, is he asked to switch to higher levels with smaller grid cells. Thus, at a given time point, the user's current communication cost is not affected by the total number of his or her friends, but by the distance of the closest friend. The scalability of FriendLocator is explored experimentally in Section 1.8.

1.7 Attacks and Extensions

In this section, we address most significant types of privacy attacks, the *joint LS & MT attack* and the *history attack*. In order to prevent these attacks or reduce their effects, we offer two solutions, the *grouping of users* and the *dynamic transformation and encryption*.

1.7.1 Attacks

Joint LS & MT Attack. Section 1.6.1 explains how the presented proximity detection approach does not preserve user location privacy when the LS and an MT are collectively malicious. When an attacker of FriendLocator has access to LS and at least one MT, intercepted Ψ and L can be used to find current geographical regions, i.e., LMG grid cells, of every user in the system. The *grouping of users*, presented in Section 1.7.2, limits the number of users affected by such an attack.

LS History Attack. Consider two consecutive four-tuples of encrypted values received by the server: (c_3, c_2, c_4, c_6) followed by (c_2, c_7, c_6, c_5). Each four-tuple corresponds to four adjacent grid cells. Because the upper right corner of the first four grid cells has the same encrypted coordinates (c_2, c_6) as the lower left corner of the next four grid cells, the server can conclude that the user moved approximately north-east between the two timestamps. In this way, if the server observes the encrypted values for some time, quite an accurate approximation of the shape of the user's trajectory can be constructed, especially if the user stays at a high-level grid with small grid-cell size. If it is known that the users move in a road network, the obtained trajectory shape can be matched to a real road network, disclosing both the user's location history and his or her current location. The *dynamic transformation and encryption*, presented in Section 1.7.3, is a solution to minimize such vulnerability.

1.7.2 Grouping of Users

In order to limit the number of users that can be affected by a joint LS & MT attack, we group the users into, possibly overlapping, groups, so that each user is put into one or more groups. Both friends and non-friends can belong to the same group, but if two users are friends, they

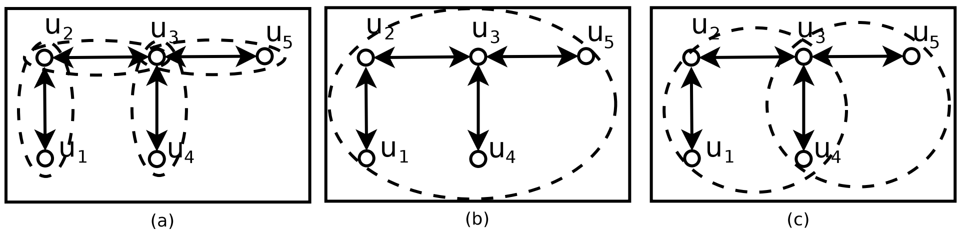


Figure 1.3: User grouping

must be in at least one common group. Each such group G is assigned a distinct pair of Ψ_G and L_G, thus, if the joint LS & MT attack is successful, it affects only the groups which contain the malicious MT. The location privacy of the users in these groups is compromised, but all other users are unaffected as they use different functions Ψ and L.

User-grouping support in our system can be achieved by letting the FriendLocator server treat each user group G as a new set of all users, $\mathbf{M}$. Each user u in the system, depending on how many groups he or she is a part of, will have to work with multiple Ψ_G and L_G.

Figures 1.3a and 1.3b visualize two user-grouping extremes. Here, user groups are depicted by ellipses and pairs of friends are represented by two circles connected with the arrows. In Figure 1.3a, each pair of friends in the system belongs to a distinct user group. In Figure 1.3b, all users belong to a single user group. If an attacker can access LS and the MT of u_1 in Figure 1.3a, he can discover LMG cells only of u_1 and u_2. In contrast, if the attacker has access to LS and some u in Figure 1.3b, he is able to discover LMG cells of any user in the system. If we consider communication efficiency, in FriendLocator with user-group support, a single user movement might cause cell crosses in LMGs of multiple user groups. Because such cell crosses result in updates to the server, the user grouping in Figure 1.3a might cause increased amount of client communications in comparison with Figure 1.3b. Summarizing, Figure 1.3a provides maximal privacy, however suffers from communication overhead. On the contrary, in Figure 1.3b, MT's computational and communication performance is much better, however there is much higher privacy-vulnerability risk. User groups enable a trade off between these extremes, as exemplified in Figure 1.3c. We evaluate the communication overhead when multiple user groups are used in Section 1.8.

In this paper, we do not consider how the user groups are created. This can be done automatically or manually by the users themselves. In the latter case, the users that have a certain level of mutual trust can be grouped to the same user group. Note that the trust extends only to sharing encryption functions. No location information (encrypted or non-encrypted) is ever exchanged directly between the users.

1.7.3 Dynamic Transformation and Encryption

The history attack can be made ineffective by periodically modifying the grid function CM (see Equation 1.1). For example, every few minutes, all grids can be rotated by an angle that is random but common to all users. The new $CM(x, y, d, \theta)$ would get an additional argument, angle θ. This angle would then be periodically re-distributed to the clients by TTS. Alternatively, to remove this redistribution cost, common pseudo-random number generator could be used by all clients. Note that such a solution adds to communication costs as every change of CM requires re-sending the current encrypted grid-coordinates of every user (and resetting the current grid level to zero).

1.8 Experimental Study

We have implemented the FRIENDLOCATOR, which consists of our proposed proximity detection solution (in Section 1.6) coupled with the support of user groups (see Section 1.7.2). For the sake of comparison, we also implemented a Baseline privacy-aware proximity detection solution, which will be presented in Section 1.8.1. Both the FRIENDLOCATOR and Baseline are implemented in the C # language.

In this section, we mainly study the communication cost (i.e., the number of messages) of the solutions with respect to various parameters. Observe that this performance metric is machine-independent. Section 1.8.2 discusses the experimental setting and Section 1.8.3 presents our experimental results.

1.8.1 Competitor Solution

We consider a competitor solution called Baseline , which also operates on the client-server architecture. It offers the users location privacy via a grid-based spatial obfuscation technique (see Figure 1.4a), where the parameter d denotes the side length of a grid cell. A user u_i computes the cell c_i that contains his location and then sends the bounding rectangle of c_i to the server. The user does not need to send any further messages to the server until it moves into another cell.

The filter and refinement paradigm is applied for detecting the proximity among friend pairs. Suppose that the users u_i and u_j are friends. Let their current cells be c_i and c_j respectively. The server computes the minimum distance $mindist(c_i, c_j)$ and the maximum distance $maxdist(c_i, c_j)$ between the cells c_i and c_j [14]. An example is illustrated in Figure 1.4a. Then, the server compares those distances with the proximity detection distance ϵ, as follows.

1. If $maxdist(c_i, c_j) \leq \epsilon$, then proximity is detected.
2. If $mindist(c_i, c_j) > \epsilon$, then no proximity is detected.
3. If $mindist(c_i, c_j) \leq \epsilon < maxdist(c_i, c_j)$, then the server invokes the refinement step — requesting the users u_i and u_j to detect proximity themselves by using the peer-to-peer Strip algorithm [2].

Observe that, in Baseline , the value of d can be varied independently of ϵ without affecting the correctness of proximity detection. This value represents a trade-off between the privacy and the communication cost. A large d offers the user a high degree of privacy but the users may need to participate in the refinement step frequently, incurring high communication cost. On the other hand, a small d helps reducing the total refinement cost of the users, but it only provides the user a small amount of privacy.

It is worth noticing that the privacy notion offered by Baseline is weaker than the one offered by FriendLocator. For Baseline , the server knows the cell c_i where the user u_i is located. In contrast, FriendLocator employs encrypted coordinates, making it hard for the server to derive the possible spatial region containing the user.

1.8.2 Experimental Setting

Workload Generation and Problem Parameters. The network-based generator [4] is used to produce a workload of users moving on the road network of the city of Oldenburg. The area of the map is 26915*23572 units^2, corresponding to 14.00*12.26 km^2. Observe that 1 unit is equivalent to 0.52 meters. A location record is generated for each user at each time stamp, whereas the duration of two consecutive time stamps is 1 minute. The average speed of the users is 52 km/h (i.e., 1670 units per time stamp).

Unless otherwise stated, the number of users is set to 50000 and the number of time stamps is set to 40. Thus, the workload consists of 2 million location records. We partition the set of users into disjoint groups, where each group contains 250 users by default. Within the same group, the friend relationships between users form a complete graph and there are no friendships between the users of different groups. The proximity detection distance ϵ is set to 200 units by default.

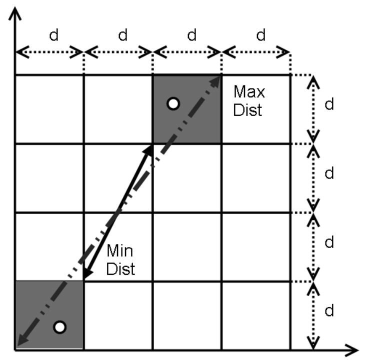
Setting of System Parameters. For the Baseline method, the default value of d is set to 200 units, in order to offer sufficient privacy protection to the users.

For the FriendLocator method, we set its LMG function $L(l) = g \cdot 2^{-l}$, where l indicates the level and g denotes the side length of the cell at the lowest level (i.e., level 0). The default value of g is 12800 units. The proximity level L_ϵ is set to 6, meaning that the side length of a cell at the highest level equals to ϵ (i.e., 200 units).

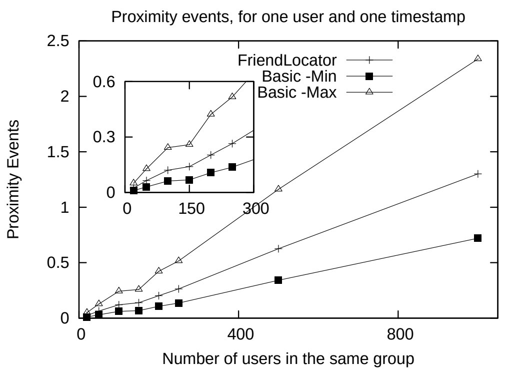
1.8.3 Experiments

In the following experiments, unless otherwise stated, we report the number of messages (sent and received) per user per time stamp.

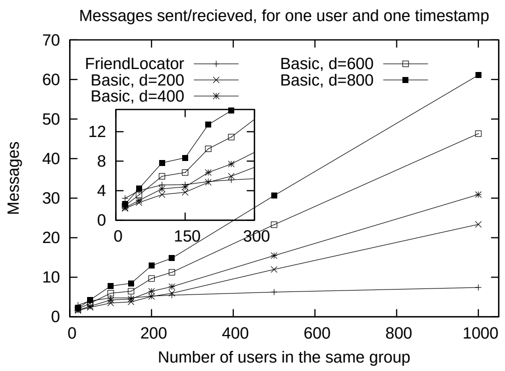
In the first experiment, we compare the number of proximity events generated by FRIENDLOCATOR and Baseline methods. In order to experimentally demonstrate Theorem 3, we run two instances of the Baseline : (i) *Base-Min* fixes its proximity detection distance ϵ to $L(L_\epsilon) = 200$, and (ii) *Base-Max* fixes its ϵ value to $2\sqrt{2} \cdot L(L_\epsilon) = 2\sqrt{2} \cdot 200$. Figure 1.4b plots the average number of proximity events per user per time stamp, by varying the number of users per group. The number of proximity events in FRIENDLOCATOR stays between those obtained from the *Base-Min* and *Base-Max*, confirming with the result of Theorem 3.



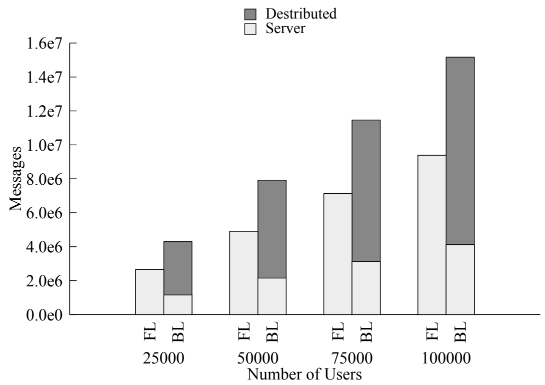
(a) concept of Baseline



(b) prox. events per user, varying group size



(c) cost per user, varying group size



(d) cost of the system, varying number of users

Figure 1.4: Effect of the number of users per group, and the total number of users

We then investigate the communication cost (i.e., the number of messages) per user per time stamp in the FriendLocator and Baseline methods. We consider instances of Baseline with the value of d ranging from 200 to 800, representing different degrees of privacy protection. Figure 1.4c shows the cost with respect to various numbers of users per group. The cost of Baseline becomes high in case the users require a high amount of privacy (say, $d = 800$). For very small group sizes, FriendLocator is slightly less efficient than Baseline . As the group size increases, the cost of FriendLocator increases very slowly and it outperforms the Baseline by far. The cost of Baseline increases linearly because the cost of the refinement step is linear to the number of users in a group.

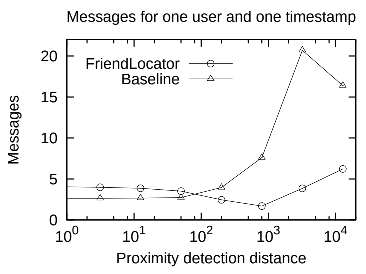
Next, we study the effect of the total number of users on the communication cost, in terms of the distributed cost among the users and the centralized cost at the server. Figure 1.4d shows the total number of messages during 40 time stamps as a function of the total number of users in the system, with the number of users per group fixed to 250. Clearly, FriendLocator incurs substantially lower total cost than Baseline .

We proceed to study the impact of the proximity detection distance ϵ on the cost per user per time stamp (see Figure 1.5a). Both Baseline and FriendLocator have similar performance at small ϵ (below 10). As ϵ increases, Baseline invokes the refinement step frequently so its cost rises rapidly. At extreme ϵ values (above 10000), most of the pairs are within proximity so the frequency and cost of executing the refinement step in Baseline are reduced. Observe that the cost of FriendLocator is robust to different values of ϵ, and its cost rises slowly when ϵ increases.

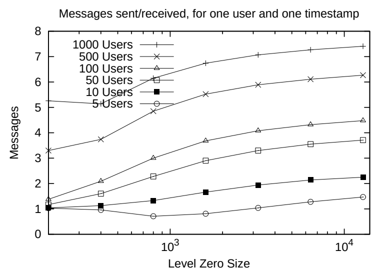
Recall from Section 1.8.2 that the setup of FriendLocator is configured by two parameters: the base level cell extent g and the proximity level L_ϵ. Now, we fix the proximity detection distance ϵ to 200, while varying the values of g and L_ϵ such that $L(L_\epsilon) = 200$. Figure 1.5b shows the cost per user per time stamp as a function of the base-level cell size g. The value of g is increased from 200 to 12800, while L_ϵ is reduced from 6 down to 0 respectively. At low values of g, the lowest-level grid is dense so the users rarely change their levels. However, a user needs to report a new encrypted tuple frequently as he or she travels across adjacent cells. On the other hand, at large values of g, the lowest-level grid is sparse. Thus, it is more likely for two friends to fall in the same or adjacent cells in the base level, causing them to switch to a denser level and increase the cost. The graphs show that the performance of FriendLocator is robust with respect to various values of g. The cost at $g = 12800$ is only at most two times of the cost at $g = 200$. Remember, that high number of levels (equivalent to large value of g in this experiment) gives high flexibility for the users when choosing proximity distance. Another nice feature of FriendLocator is that, when the number of users within the same group increases from 5 to 1000, the cost per user per time stamp increases only five times, although the number of friendships grows exponentially.

Note that the user groups are disjoint in all previous experiments. We proceed to investigate how the overlapping among user groups affects the communication cost. In this experiment, a specific user $u_\star$ has 80 friends. They are assigned to a set of overlapping groups such that: (i) all groups have the same size, and (ii) any different groups share only the common user $u_\star$ but nobody else.

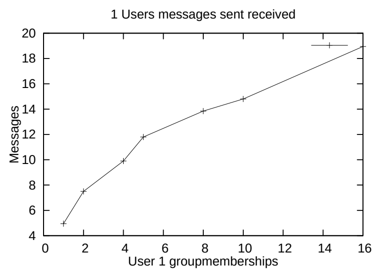
Figure 1.5c plots the communication cost for the user $u_\star$ per time stamp. When $u_\star$ is a member of many groups, each group has a small number of users and $u_\star$ spends less cost per group. Thus, the communication cost of $u_\star$ grows sub-linearly with the number of groups that he belongs to.



(a) varying ϵ



(b) varying base-level cell size



(c) effect of overlapping groups

Figure 1.5: Effect of various parameters on the communication cost per user

1.9 Conclusion

In this paper we develop the FRIENDLOCATOR, a client-server solution for detecting the proximity among friend pairs, while offering them location privacy. The client maps a user's location into a grid cell, converts it into an encrypted tuple, and sends it to the server. Based on the encrypted tuples received from the users, the server determines the proximity between them blindly, without knowing their actual locations. Furthermore, we propose a multi-grid approach for optimizing the communication cost of our solution, and also study its resilience against two interesting types of attacks. Experimental results suggest that FRIENDLOCATOR incurs low communication performance, it is scalable to a large number of users, and its performance is robust with respect to various parameters.

In the future, we plan to extend the proposed solution for proximity detection of moving users on a road network, in which the distance between two users is constrained by the shortest path distance between them.

Part III

VicinityLocator
-
Flexible Proximity Detection In Mobile Social Networks

Abstract

The privacy-aware proximity detection service determines if two users, e.g., friends, vehicles, etc., are close to each other without requiring them to disclose their exact locations. Existing proposals of such service provide weak privacy, low precision guarantees, or lack of flexibility for user settings.

In this work, we combine the best features of existing proposals and build our client-server solution for proximity detection based on encrypted partitions of the spatial domain. Our service notifies a user if any pre-selected users enters his specified "area of interest", called a vicinity region. Unlike in other proposals, our solution supports irregular-shaped, dynamically-changeable vicinity regions, which enables various proximity detection scenarios that are not possible with existing solutions.

It also offers strong user location privacy where the server evaluates proximity queries blindly without manipulating spatial the data of any user. Experimental results show that our solution, being equipped with a new set of features, performs well compared to existing solutions.

1.10 Introduction

Mobile devices with built-in geo-positioning capabilities are becoming cheaper and more popular [5]. Disclosing their location information (e.g., via Wi-Fi, Bluetooth, or GPRS), mobile users can enjoy a variety of location-based services (LBSs). One type of such services is a *friend-locator* service, which shows users their friends' locations on a map and/or helps identify nearby friends. Friend-locator together with other mobile social-networking services are predicted to become a multi-billion dollar industry over the next few years [1]. Thus several friend-locator services, like *iPoki*, *Google Latitude*, and *Fire Eagle* [3] are now available on the Internet.

In existing friend-locator services, the detection of nearby friends can be done only manually by a user, e.g., by periodically checking a map on the mobile device screen. This works only if the user's friends agree to share their exact locations or at least obfuscated location regions (e.g., downtown area). However, LBS users often require some level of privacy and may even feel threatened [9] if it is not provided. If all user's friends require complete privacy, they have to disable their location sharing, thus also preventing the user from finding his or her nearby friends. Consequently, due to poor location-privacy support, the nearby-friend detection is not always possible in existing friend-locator products.

Nearby friends detection can be enabled to privacy-concerned users utilizing existing privacy-aware proximity detection methods [15, 12, 16]. They allow two users to determine if they are close to each other without requiring them to disclose their exact locations to a service provider

[3]http://www.ipoki.com; http://www.google.com/latitude; http://fireeagle.yahoo.net

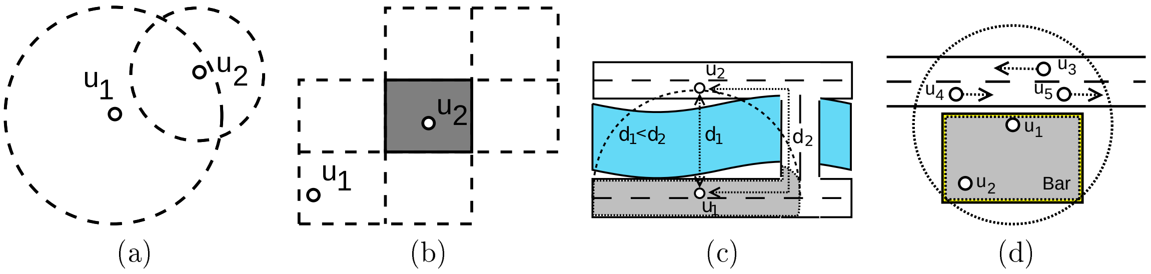


Figure 1.6: Types of vicinities and proximity detection scenarios

or other friends. They track all users in the real-time and generate notifications if any two friends becomes close to each other.

Most of existing methods assume that two users are close to each other if a so-called *vicinity region* of one user either contains the location or intersects the vicinity region of other user. The vicinity regions enclose users' locations and can be understood as parameters of a spatial range query over exact or region-enclosed user locations. Existing proximity detection methods only support static shape vicinities that are circular-shaped and user-location-centered [12] (Fig. 1.6a) or rectangular-shaped and not user-location-centered [16] (Fig. 1.6b). Both types of vicinities enable proximity detection only in non-constrained Euclidean space, e.g. football field with no obstacles, where on proximity notification users can walk in a straight line to each other. However if the distance between two users is constrained by the shortest path distance, that is not always equal to crow-fly distance, then the existing methods are not applicable. For example, if two users are located on different banks of the river (Fig. 1.6c) such that existing methods classify them being in proximity, then generated proximity notification might not be very useful for users. It might be complicated for users to meet each other, because the distance (d_2) of shortest path following the road-network could be much higher than the crow-fly distance (d_1). Moreover, fixed-shape vicinity based services do not allow users to choose "areas of interest". This could be inconvenient in some cases, e.g. if the user uses the service to find friends in some bar (Fig. 1.6d) and his friends are traveling along some nearby road with no plans entering the bar, then the user is flooded with meaningless proximity notifications. This scenario is very likely if the user has many friends and the road is traffic-intensive.

The introduced shortcoming of the scenarios can be eliminated if instead of static-shape, dynamic-shape vicinities were used. In the first case (Fig. 1.6c), if user u_1 at current time moment could select vicinity region such that distance between every point of the region and u_1 location following the road-network is less than some threshold, then user u_2 would not be detected in u_1 proximity. Similarly in the second case (Fig. 1.6d), if user u_1 could preselect vicinity region, matching the bar as it is his "area of interest", then only user u_2 would be identified by the system. The challenge is to develop the proximity detection method that supports both user location privacy and dynamic-shape vicinities.

To address the challenge, we combine ideas from existing solutions and develop a client-server, location-privacy aware proximity detection service, the VicinityLocator. The proposed solution is based on both spatial cloaking and encryption. Here groups of users share an encryption function and some space-partitioning, e.g. a grid, mesh, or Voronoi diagram, that are used to compute encrypted representations of user locations and dynamic vicinities. Encrypted representations are sent to the central server, which, based on the values received, compute proximity between users blindly without knowing any spatial data of the users. Users can individually specify their dynamic vicinities, minimum location privacy requirements and service precision settings. Our VicinityLocator employs a flexible location-update policy, forcing users to update their location data only when leaving some automatically adjustable regions, that shrink and expand depending on the distance of a users closest friend.

The paper is organized as follows. We briefly review related work in Section 1.11 and then define our problem setting in Section 1.12. The VicinityLocator is presented in Section 1.13. In Section 1.14 we present 3 general attacks applicable to our solution. Section 1.15 presents extensive experimental results of our proposed approach.

1.11 Related Work

In this section we review general location privacy preserving techniques followed by the relevant work on location privacy in proximity detection services.

1.11.1 General Location Privacy Techniques

In the most common setting assumed in location-privacy research, an LBS server maintains a public set of points-of-interest (POI), such as gas stations. The goal is then to retrieve from the server the nearest POIs to the user, without revealing the user's private location q to the server. Many location privacy solutions exist for this setting and they can be broadly classified into two categories: spatial cloaking and transformation.

Spatial cloaking [8, 13, 6, 3] is applied to generalize the user's exact location q into a region Q', which is then used for querying the server. The region Q' is then sent to the LBS server, which returns all the results that are relevant to any point in Q'. Such technique ensures that even if the attacker knows locations of all users, the identity of the querying user can be inferred only with some probability.

The transformation approaches [10, 7] map the user's location q and all POIs to a transformed space, in which the LBS server evaluates queries blindly without knowing how to decode the corresponding real locations of the users.

In contrast, in the proximity detection problem, the users' locations are both query locations and points-of-interest that must be kept secret. Thus the existing spatial cloaking and transformation techniques that assume public datasets cannot be directly applied for the proximity detection. However the concepts of spatial cloaking and transformation can be and are used in the existing privacy-aware proximity detection methods.

1.11.2 Privacy-aware proximity detection methods

Anonymous User Tracking for Location-Based Community Services

Ruppel et al. [15] develop a centralized solution that supports proximity detection and provides the users a certain level of privacy. It first applies a distance-preserving mapping (a rotation followed by a translation) to convert the user's location q into a transformed location q'. Then, a centralized proximity detection method is applied to detect the proximity among those transformed locations. However, Liu et al. [11] points out that such distance-preserving mapping is not safe and the attacker can easily derive the mapping function and compute the users' original locations.

Privacy-Aware Proximity Based Services

Mascetti et al. [12] present a privacy preserving solution which employ the filter-and-refine paradigm.

All users collectively agree on privacy settings and individually specify radii of their circular-shaped vicinities. Note that only this type of vicinity is supported. The privacy settings are specified by two - coarse and fine - spatial-domain subdivisions, so called granularities. Each of them contains discrete number of non-overlapping regions, called granules. The coarse and the fine spatial granularities represents users minimum privacy requirements, for the central server and any other users respectively, with each granule being a minimum uncertain region.

A user maps his location into some granule g_c of the coarse granularity and then constructs the cloaking region by merging g_c intersecting granules of the fine granularity. The cloaking regions of every user are sent to the server. When the server receives the cloaked region of some user u_1, it performs a rough proximity detection between u_1 and his friends. For all friends u_2 of $u_1$the server first computes a minimum and maximum distances between cloaking regions of u_1 and u_2. Then, depending on the specified thresholds and computed distances, the server classifies users being in, not-in, or possibly-in proximity. For the first two outcomes the server immediately report the proximity status to the users, however if users are classified as being possibly-in proximity, then it forces them to perform user-to-user communication to refine their proximity status. The introduced concepts required by server-side proximity detection

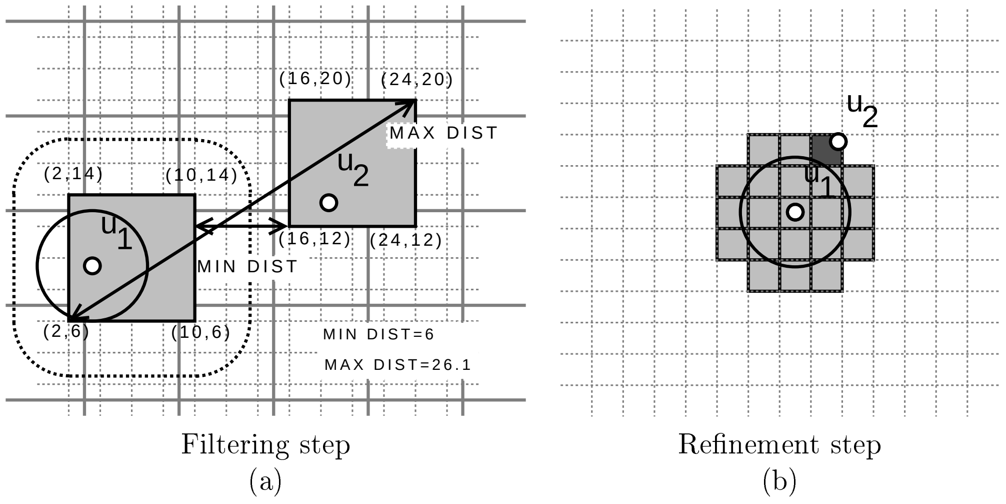


Figure 1.7: Concepts used in Mascetti et al. [12] solution

are visualized by Fig. 1.7a. It visualizes coarse and fine granularities (two overlapping grids in the background), u_1 and u_2 cloaking regions, minimum and maximum distances between them, user u_1 true vicinity (solid-line circle), and u_1's vicinity, seen from the server perspective (dashed-line region).

In the refinement step, first two users map their locations and vicinity regions into the fine granularity where granules usually are much smaller than in the coarse granularity. Later they check if one user location enclosing granule lays inside the set of granules, intersecting the other user vicinity. Depending on the result of the granule-inclusion checking, either proximity or separation is detected. Note that due to utilized secure two-party computation protocol, the set-inclusion checking is performed without need to reveal one user's location and vicinity granules to another user. Figure 1.7b visualizes a scenario, where user u_2 current location enclosing granule (dark rectangle) intersect with u_1 vicinity region. In this case, proximity between u_1 and u_2 will be detected.

Unlike in our approach, their proposal does not completely hide user locations from the central server as it always knows their cloaked regions. If the strong privacy is required users are forced to perform user-to-user communication more frequently thus significantly increasing amount of client communication due to expensive secure two-party computation protocol. Also, in case of strong privacy, the amount of false-positives in the proximity detection are introduced with no specified distance guarantees for the users.

A Location Privacy Aware Friend Locator

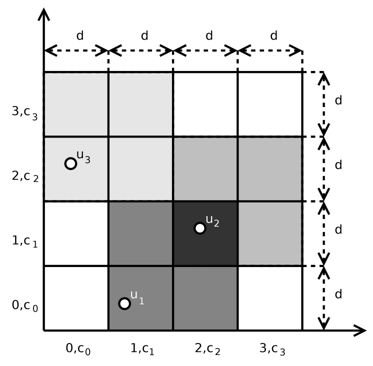
Šikšnys et al. [16] have developed a centralized privacy-aware proximity detection method, called FRIENDLOCATOR, which provides strong privacy guaranties and employs spacial grid-based technique to optimize communication cost. The whole space is divided into equal-sized grid cells, such that sizes can be changed for each user individually at the runtime. A group of users map their locations into the grid and, prior to sending these mappings to the server, encrypts them with a shared secret encryption function. This process can be explained by the example [16] visualized in Fig. 1.8a and 1.8b. Here some users u_1, u_2, and u_3 share a grid, where each row and column has encrypted values c_0 .. c_3 assigned according to Ψ, shown in Fig. 1.8b. Users u_1, u_2, and u_3 map their locations into grid cells (1,0), (2,1), and (0,2) and utilizing Ψ construct encrypted coordinates (c_1,c_2,c_0,c_1), (c_2,c_3,c_1,c_2), and (c_0,c_1,c_2,c_3) respectively. Here encrypted coordinates contains 4 integers $(\alpha^-, \alpha^+, \beta^-, \beta^+)$, where (α^-, α^+) and (β^-, β^+) are encrypted values of two adjacent columns k and $k+1$ and two adjacent rows m and $m+1$, where k and m are the column and row number of cell containing user location.

The central server receives users' encrypted coordinates, performs their matching, and in case of a match informs the pair of users. Here some encrypted coordinates e_1 and e_2 match if Eq. 1.4 holds, and it is then known that the distance between two users is lower than the grid cell sizes dependent constant. Users, unlike the server, know how to compute the constant.

$$\begin{aligned}\Gamma(e_1, e_2) \;=\; & \big((e_1.\alpha^- = e_2.\alpha^-) \vee (e_1.\alpha^- = e_2.\alpha^+) \vee (e_1.\alpha^+ = e_2.\alpha^-)\big) \\ & \wedge \big((e_1.\beta^- = e_2.\beta^-) \vee (e_1.\beta^- = e_2.\beta^+) \vee (e_1.\beta^+ = e_2.\beta^-)\big)\,. \end{aligned} \tag{1.4}$$

Users fix some constants on the server specifying how many consecutive encrypted location matches must be found at the so called *list of grids* in order to detect users as being in proximity. Here the *list of grids* defines multiple grids with constantly decreasing cell sizes, where every grid is identifiable by its level number. After each encrypted coordinate match, the server checks if required level is reached. If so, then users are informed about their proximity, otherwise the server sends a message to one or both of the users asking them to use a finer grid, i.e., increase their current level, for the next matching iteration. For examples, if we assume that the grid in Fig. 1.8a corresponds to users required level, then by evaluating Eq. 1.4 we can deduce that users u_1 and u_2 are in proximity and u_3 in separation with others.

Users can shift from finer to coarser grids as they move. Once some user change his location, he automatically switches into coarsest possible grid, where user's location change caused the cell cross. Note that depending on user movement length and cell sizes of user's current level grid, the movement may or may not trigger user location update.



(a)

Col/ Row	Ψ
0	c_0 (5)
1	c_1 (9)
2	c_2 (1)
3	c_3 (7)

(b)

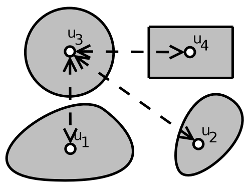
Figure 1.8: Example of proximity detection in the FriendLocator

The limitation of the FriendLocator is its low, and uncontrollable, precision of the proximity detection. On the proximity notification the actual distance between two users can be any in the range from ϵ to $\epsilon+\lambda$, where ϵ is required proximity distance and $\lambda = \epsilon(2\sqrt(2)-1)$ is the precision parameter. Note, that in most other proximity detection approaches, unlike in FriendLocator, the parameter λ can be chosen freely by users. High and unchangeable λ values might be unacceptable in some applications especially if high values of ϵ are used.

Moreover, the FriendLocator is not suitable for the "river" and the "bar" proximity detection scenarios (See explanation of Fig. 1.6c and 1.6d) as it does not support dynamic-shape vicinities. It only mimics rectangular-shaped and non-centered user-location vicinities, where regions are constructed from 4 adjacent grid cells (See Fig 1.8a) and their intersections at required level triggers the proximity event.

1.11.3 Our contribution

In this work we combine best features of the previously presented proximity detection approaches to build our solution, the VicinityLocator. The solution combines the ideas of encrypted coordinates, their blind evaluation at the server-side, and vicinity's representation by granules, where sizes can be changed dynamically. Our solution, unlike Mascetti et al. [12] proposal, employs only centralized architecture, where the server knows no spatial data of users. It allows users individually select preferable proximity detection precision and supports changing over time, irregular-shaped vicinities. These are unsupported features in existing privacy-aware proximity detection solutions. Our proposal is designed for 2D environment, but it can without much change be applied to n-dimensions.



(a)

$$\mathbf{F} = \left\{ \begin{array}{l} (u_1, u_3), (u_3, u_1), (u_3, u_2), \\ (u_2, u_3), (u_3, u_4), (u_4, u_3) \end{array} \right\}$$

(b)

Figure 1.9: User locations, vicinities, and friend-ship relation example

1.12 Problem Definition

In this section we introduce relevant notations, formally define privacy requirements and behavior of our proximity based service.

We assume a setting where a set of users form a social network and all of them carries a mobile device(MD) with positioning and communication capabilities. All MDs are online and have access to central location server (LS). We use the terms *mobile device*, *user*, and *client* interchangeably and denote the set of all users or MDs by $\mathbf{M} \subset \mathbb{N}$. The users forming the social network are defined by the friend-ship relation $\mathbf{F}$, where $\{(u, v), (v, u)\} \in \mathbf{F}$ if $u, v \in \mathbf{M}$ are friends.

Let us assume a 2D scenario, where users from $\mathbf{M}$ can freely move in Euclidean space and every user $u \in \mathbf{M}$ at the current time moment defines $loc(u)$ and $vic(u)$. Here $loc(u) = (loc(u).x, loc(u).y)$ represents u's 2D location and $vic(u)$ specifies u's dynamic *vicinity region*. The vicinity region is a single- (like circle, rectangle, etc.) or multi-parted (composition of more than one circle, polygon, etc.) region around a user location and it can be understood as an infinite set of spatial points that changes over the time. Introduced concepts are visualized in Fig. 1.9a. Here arrows, small circles, and filled regions represent users' friend-ships, locations, and vicinities. A corresponding friend-ship relation $\mathbf{F}$ is given in Fig. 1.9b.

The privacy-aware proximity based service notifies user $u \in \mathbf{M}$ if any of his friends $v \in \mathbf{M} | (u, v) \in \mathbf{F}$ enters his vicinity region. More specifically, first all of u's friends are classified to be in proximity or separation by checking following conditions:

1. if $loc(v) \in vic(u)$ user v is in u's proximity;
2. if $distLV(loc(v), vic(u)) > \lambda$, user v is in u's separation;
3. if $distLV(loc(v), vic(u)) \leq \lambda$, the service can freely choose to classify v as being in u's proximity or separation.

Here $distLV(l, v)$ denote a shortest Euclidean distance between location l and vicinity region v. If l is inside v then $distLV(l, v) = 0$.The $\lambda \geq 0$ is a service precision parameter and introduces a degree of freedom in the detection of location-to-vicinity intersection. Note, that small values of λ corresponds to higher precision. When the classification is complete, u is provided with a set of proximate friends that now are classified as being in proximity while they were not in proximity (were in separation) before. Every user has to have ability to tweak their desirable service precision level λ and the service has to possibly minimize amount of client communication depending on its λ setting.

In addition to that, for every user $u \in \mathbf{M}$ the service has to satisfy following location privacy requirements:

- The exact location of u is not disclosed to any party (e.g., any other user or the LS).
- User u allows nobody else but his friends to see him in their vicinities.

The following section details our proposed proximity based service, that meet these requirements.

1.13 Our privacy-aware proximity based service

In this section we introduce base concepts employed in our proximity detection service, followed by client, server algorithms and examples of behavior.

1.13.1 Proximity detection idea

This section describes how the LS can locate users in their friend's vicinity without disclosing their locations and vicinities.

Similarly to *Incremental Proximity Detection Approach* [16], where all users in $\mathbf{M}$ share a list of grids, here we let all users in $\mathbf{M}$ share a *list of granularities.* The list of granularities, denoted by Γ, contain finite or infinite number of granularities $\Gamma(l)|l = 0, 1, 2, ...$ A single granularity[4] specifies a divisions of the spatial domain into a number of non-overlapping regions, called granules[12]. The granularity's index $l \geq 0$ in the Γ is termed *the level of granularity.* Every granularity $\Gamma(l)$ at levels $l = 0, 1, 2, ...$ satisfies following three properties:

- Every granule $g \in \Gamma(l)$ is identifiable by an index, say $id(g) \in \mathbb{N}$.
- Every granule $g \in \Gamma(l)$ has a bounded, not higher than $L(l)$, size, i.e. $\forall g \in \Gamma(l), MaxDist(g) \leq L(l)$, where $MaxDist(g)$ is maximal Euclidean distance between any two points of region g.
- Granule sizes bound $L(l)$ in level l is always lower than in level $l-1$, i.e. $L(l) < L(l-1)$.

[4] Note, that a grid is a special case of granularity

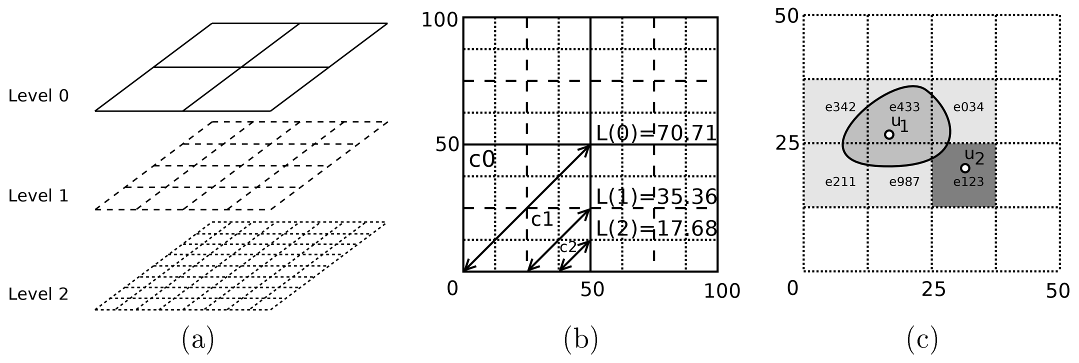


Figure 1.10: The valid list of granularities and the behavior of the granularity-based classifier

- Every granule in level l is fully contained by some granule at level $l-1$.

Figure 1.10a visualizes a valid granularity list, where uniform grids are used as granularities and grid cells are used as granules in levels 0 to 2. Note, that the figure shows only subsets of all available cells for every grid. The "top-view" projection of this list is provided in Fig. 1.10b. Solid, dashed and dotted lines depict boundaries of cells at levels 0, 1, 2 respectively. Every cell in levels $l = 0, 1, 2$ should be identifiable and not higher than $L(l)$ size. For example, maximal distances between any two points of some cells $c0$,$c1$, $c2$ at levels 0,1,2 respectively are 70.71, 35.36, 17.68 and they correspond to levels' L values in the uniform grids case. Moreover, cells of lower level grids fully contain cells of higher level grids.

Let us assume, that a list of granularities Γ is globally defined in the system and thus fixed on all clients.

Similarity to FriendLocator [16], we let all users in $\mathbf{M}$ also share an encryption function Ψ. $\Psi : \mathbb{N} \mapsto \mathbb{N}$ is one-to-one function that is used to map index $id(g)$ of some granule g to corresponding encrypted representation. In practice Ψ can be implemented as a keyed secure hash function (e.g. SHA-2) such that it is computationally infeasible for the attacker to break. A key of the hash function can be distributed among clients of $\mathbf{M}$ in peer-to-peer fashion or with help of some trusted third-party server.

When Ψ is known by clients, but not by the LS, each client utilizing Ψ can encrypt indices of granules that enclose their location and vicinity at some granularity. Encrypted representation of these indices can be compared on the LS without need to disclose indices of granules and consequently the vicinity or location of any user. In particular, assume that two friends u_1, u_2 use granularity of some level l (Fig. 1.10c) and the user u_2 finds his location containing granule g_l, applies Ψ on its index $id(g_l)$ and sends encrypted representation, $e123$, to the LS. Similarly user u_1 finds all his vicinity intersecting granules $\mathbf{g_v}$, applies Ψ on their indices and send their encrypted representations, $\{e342, e433, e034, e211, e987, e123\}$, to the LS. If encrypted index of u_2's location granule, $e123$, can be found in encrypted indices set of user u_1 vicinity then we can conclude that user u_2 is in u_1's vicinity with some precision. Let us call such user-to-vicinity intersection detection approach by *granularity-based classifier.* More over utilizing knowledge about each granularity in the list we can derive Lemma 5.

Lemma 5. *The granularity-based classifier can be used to classify the user u_2 as being in u_1's proximity or separation with precision parameter setting $\lambda = L(l)$, defined in Section 1.12.*

Proof. According the Section 1.12, in order for user u_2 to be in u_1's proximity or separation, conditions $distLV(loc(u_2), vic(u_1)) \leq \lambda$, and $loc(v) \notin vic(u)$ must hold. If the encrypted index of u_2's location granule, like $e123$, can be found in encrypted indices set of user u_1 vicinity, due to Ψ is one-to-one mapping we can conclude that u_2 location containing granule g_l is in u_1's vicinity intersecting granules set $\mathbf{g_v}$. Then we know that granule g_l both encloses u_2 location $loc(u_2)$ and intersects with u_1 vicinity $vic(u_1)$. Utilizing properties of granularity at level l, we know that maximal Euclidean distance between any two points within granule g_l is lower-equal than $L(l)$, i.e. $MaxDist(g_l) \leq L(l)$. Thus the shortest Euclidean distance between location $loc(u_2)$ and the vicinity region $vic(u_1)$ cannot be higher than $L(l)$, i.e. $distLV(loc(u_2), vic(u_1)) \leq \lambda$. Similarly we can prove that if g_l cannot be found in $\mathbf{g_v}$ then $loc(v) \notin vic(u)$. $\square$

According to Lemma 5, the λ value depends on granule sizes bound function L and the granularity level l. We can observe that higher levels provide higher proximity detection precision such that $\lim_{l\to\infty} \lambda(l) = \lim_{l\to\infty} L(l) = 0$. However as we go to higher levels the number of vicinity intersecting granules increases causing higher client communication. Thus we let for every user $u \in \mathbf{M}$ to select a constant $L_{max}(u)$ that has following meanings:

- User u will never use granularities of higher than $L_{max}(u)$ levels thus limiting his worts case communication.
- User u lets other users to detect him in a proximity with no higher than $\lambda = L(L_{max}(u))$ precision.
- User u will be able to detect friends being in his proximity with no higher than $\lambda = L(L_{max}(u))$ precision.
- Every encrypted coordinate of user that is sent to LS will have no higher than $L(L_{max}(u))$ resolution, i.e., if the attacker brakes the encrypted coordinate, then deciphered value will correspond to cloaking region with maximum distance between two points no lower than $L(L_{max}(u))$.

Users can freely select such L_{max} at runtime and upload it to the LS. Note that this does not violate user location privacy as it does not reveal any spatial information.

Message Type	Args	Sender	Description
M_{el}	u, l, g_l^*, $\mathbf{g}_\mathbf{v}^*$	MD	MD with id equal to u sends this type of message to the LS in order to report his encrypted location g_l^* and the vicinity $\mathbf{g}_\mathbf{v}^*$ for level l.
M_{prox}	v, l	LS	The LS sends this type of message to a MD to inform that his friend v is within his vicinity at granularity level l.
M_{LevInc}	l	LS	The LS sends this type of message to some MD to make it increase its level up to level l.

Table 1.3: Client and server messages types

The introduced granularity-based classifier is integrated into our proximity detection service which is defined using client and server algorithms in the following section.

1.13.2 Client and Server algorithms

We define our proximity based service by providing handler algorithms for different type of software events on a MD and the LS, in particular:

onMessageReceived(*msg*, *arg*) A handler is executed on a MD or the LS each time one receives a message of type *msg* with arguments *arg* from other party. A summarizing list of employed messages with their arguments is presented in Table 1.3.

onLocationChange() A handler executed on a MD, each time its geographical location changes.

Algorithms 3 and 4 specify the behavior of the MD and the LS. They contains local data definitions, functions and software event handlers.

A MD u remembers his last positioning unit reported geographical location $loc(u)$, and for granularity levels $l = 0..|GS| - 1$ (see Alg. 3) locally stores his location and vicinity mappings, i.e., indices of location and vicinity granules, in the stack GS. Once MD changes its location, the **onLocationChange** handler is triggered. Then if user's location change invalidates current location or vicinity granules at levels $l = |GS| - 1..0$, the MD removes respective elements from GS, thus reducing his employed current level $|GS| - 1$. Note, that this corresponds to zero or more u's switches from finer to coarser grids. At least one current level reduction is always followed **pushAndSend** call, which computes new location and vicinity mappings for $|GS|$ (current + one level) and sends them to the LS.

Client granularity level shifts can be visualized by Fig. 1.11a and 1.11b, where user's u_1 vicinity and user's u_2 current location mapping into list of granularities (grids) are visualized at

```
Data: $u \in \mathbf{M}$ - current user ID.
$loc(u)$ - user's current location.
$vic(u)$ - user's vicinity region.
$GS$ - Stack of 2-tuples $\langle g_l, \mathbf{g_v} \rangle$. Each 2-tuple correspond to a level $l$. $g_l$ is $loc(u)$ granule
index at level $l$. $\mathbf{g_v}$ is a set of granule indices, where each granule intersects $u$'s vicinity
and has granularity of level $l$.
$L_{max}(u)$ - user specified highest granularity level.
1  mapLocToGranularity(Level number $level$)
2      $g_l \leftarrow \{id(g) | g \in \Gamma(level) : loc(u) \in g\}$ ;
3      $\mathbf{g_v} \leftarrow \{id(g) | \forall g \in \Gamma(level) : g \cap vic(u) \neq \emptyset\}$;
4      return $(g_l, \mathbf{g_v})$;
5  pushAndSend()
6      $(g_l, \mathbf{g_v}) \leftarrow mapLocToGranularity(|GS|)$;
7      Push $\langle g_l, \mathbf{g_v} \rangle$ to stack $GS$;
8      Send to LS $M_{el}(u, |CS| - 1, \Psi(g_l), \{\Psi(g) | \forall g \in \mathbf{g_v}\})$;
9  onLocationChange()
10     $wasPopped \leftarrow false$
11     while $|GS| > 0$ and $top(GS) \neq mapLocToGranularity(|CS| - 1)$ do
12         Pop from stack $GS$;
13         $wasPopped \leftarrow true$;
14     if $wasPopped$ or $|GS| = 0$ then
15         pushAndSend()
16 onMessageReceived(Message $M_{LevInc}$, Level $l$)
17     while $|GS| \leq l$ and $|GS| \leq L_{max}(u)$ do
18         pushAndSend()
19 onMessageReceived(Message $M_{prox}$, Friend $v$, Level$l$)
20     Output "Friend ",$v$," with precision ", $L(l)$, " is inside our vicinity!";
```

Algorithm 3: The MD's event handlers in our proximity based service.

consecutive time snapshots. Note, that due to simplicity u_1's current location and u_2's vicinity mappings are not shown. User u_1 changes his location and shifts from level 1 to level 0 (Fig. 1.11a and 1.11b) because his location change invalidates his vicinity mappings at levels 1 and 0. In contrary, u_2 location change causes no location mapping changes in levels 1 and 0, thus he stays in level 1.

For every user u the LS locally stores $GL(u)$, which is an encrypted alternative for GS. It contains encrypted representations of u's location and vicinity granule indices for levels

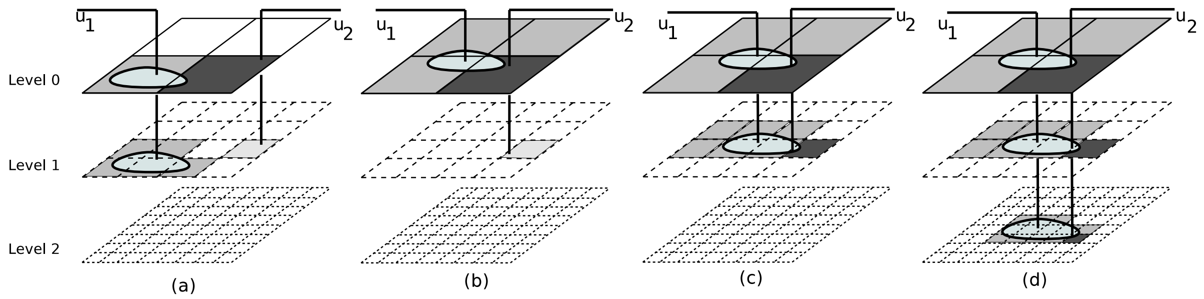


Figure 1.11: Example of level changes and proximity detection within VicinityLocator

$0..GL(u)-1$. The u's stack GS is synchronized with $GL(u)$ with help of M_{el} message. When the LS received this type of message, then the handler **onMessageReceived** is executed. It first updates the $GL(u)$ and later checks if any of u's friends entered its vicinity or if user u entered his friends vicinities. This is checked by searching if encrypted location granule g_l^* can be found in the set of encrypted vicinity granules $\mathbf{g}_\mathbf{v}^*$ for some friend f at some level l_m. The level l_m is the highest level, available in $GL(u)$ and $GL(v)$ that does not exceed $L_{max}(u)$ and $L_{max}(v)$. If g_l^* is found in the $\mathbf{g}_\mathbf{v}^*$ but the l_m is lower than $L_{max}(u)$ and $L_{max}(v)$, it means than the proximity detection precision can be still be increased as users specified L_{max} values are not yet reached, thus the LS sends M_{LevInc} to one or both users, asking them to increase they current levels. Otherwise, if g_l^* is found in the $\mathbf{g}_\mathbf{v}^*$ and l_m is equal to $L_{max}(u)$ or $L_{max}(v)$ then the LS sends M_{prox} message, informing a user about a presence of friend in his vicinity.

Let us assume that $L_{max}(u_1) = L_{max}(u_2) = 2$ in Fig. 1.11 example. The server finds that g_l^* of user u_2 lays in $\mathbf{g}_\mathbf{v}^*$ of user u_1 at level 0 in Fig. 1.11a, thus due to $0 = l_m$ is lower than $L_{max}(u_1)$ or $L_{max}(u_2)$ it sent M_{LevInc} messages to both users asking them to increase their current levels. When they both deliver level 1 encrypted coordinates, that are higher than level 0 precision, the LS no longer founds g_l^* in $\mathbf{g}_\mathbf{v}^*$ and then nothing happens until one of them starts moving. When u_1 sends his location data for level 0 in Fig. 1.11b, the l_m is set to 0 and due to it is lower than $L_{max}(u_1)$ or $L_{max}(u_2)$ and user u_2 is at level 1 already, only the user u_1 is asked to switch to level 1. Similarly, when the LS finds g_l^* in $\mathbf{g}_\mathbf{v}^*$ at level 1 in Fig. 1.11c it asks both users increase their levels as $1 = l_m$ is still lower than $L_{max}(u_1)$ or $L_{max}(u_2)$. When two users deliver their encrypted location data to the LS for level 2 in Fig. 1.11d, the $l_m = 2$ is equal to $L_{max}(u_1)$ and $L_{max}(u_u)$, then the user u_1 is informed about u_2 proximity with message M_{prox}.

Note that similarly to the FriendLocator [16], the presented algorithms implement a kind of adaptive region-based up-date policy. The clients updates their encrypted location data on the LS only when location change triggers the change of location or vicinity granularity mappings, i.e. user location and vicinity enclosing granules, at their current levels. And if some user is far away from his friends, then he or she stays at a low-level granularity with large cells, which results in few encrypted location data updates as the user moves. Only when the user approaches one of the friends, is he asked to switch to higher levels with smaller granules. Thus, at a given time point, the users current communication cost is not affected by the total number of his or her friends, but by the distance of the closest friend.

Next we review several optimizations possibility that can help reduce client communication and server computation costs, followed by a technique to minimize privacy leakage if encryption function Ψ is intercepted by an adversary.

```
Data: $\mathbf{M}$ - a set of users;
$\mathbf{F}$ - a friendship relation that contains pairs of friends and thus represents the social network.
$L_{max}(u) \forall u \in \mathbf{M}$ - highest granularity level specified by user $u$.
$GL(u) \forall u \in \mathbf{M}$ - a stack of 2-tuples $\langle g_l^*, \mathbf{g_v^*} \rangle$, where every 2-tuple corresponds to level $l$.
$g_l^*$ is an encrypted index of granule containing $u$ current location at level $l$. $\mathbf{g_v^*}$ is a set of encrypted granule indices, where each granule intersects $u$'s vicinity and has granularity of level $l$.
$\mathbf{P(u)} \subseteq \mathbf{M}$ - a set of $u \in \mathbf{M}$'s currently proximate friends.
1  onMessageReceived(Message $M_{el}$, User $u$, Level $l, g_l^*, \mathbf{g_v^*}$)
2    while $|GL(u)| > 0$ and $|GL(u)| > l$ do
3      Pop from stack $GL(u)$;
4    Push $\langle g_l^*, \mathbf{g_v^*} \rangle$ to stack $GL(u)$;
5    foreach $v \in \mathbf{M}$ such that $v \neq u$ and $|GL(v)| > 0$ and $(u, v) \in \mathbf{F}$ do
6      $l_m \leftarrow min(|GL(u)| - 1,\ |GL(v)| - 1,\ L_{max}(u),\ L_{max}(v))$;
7      $vInU \leftarrow get(GL(v), l_m).g_l^* \in get(GL(u), l_m).\mathbf{g_v^*}$;
8      $uInV \leftarrow get(GL(u), l_m).g_l^* \in get(GL(v), l_m).\mathbf{g_v^*}$;
9      if $vInU = true$ or $uInV = true$ then
10       if $l_m = L_{max}(u)$ or $l_m = L_{max}(v)$ then
11         if $vInU$ and $v \notin \mathbf{P(u)}$ then
12           insert $v$ into $\mathbf{P(u)}$;
13           send $M_{prox}(v, l_m)$ to MD $u$;
14         if $uInV$ and $u \notin \mathbf{P(v)}$ then
15           insert $u$ into $\mathbf{P(v)}$;
16           send $M_{prox}(u, l_m)$ to MD $v$;
17       else
18         if $l_m = |GL(u)| - 1$ then
19           send $M_{LevInc}(l_m + 1)$ to MD $u$
20         if $l_m = |GL(v)| - 1$ then
21           send $M_{LevInc}(l_m + 1)$ to MD $v$
22     if $vInU = false$ then
23       remove $v$ from $\mathbf{P(u)}$;
24     if $uInV = false$ then
25       remove $u$ from $\mathbf{P(v)}$;
```
Algorithm 4: **onMessageReceived** event handler on the LS.

1.13.3 Incremental update optimization

A client in the VicinityLocator service constantly report encrypted location data as he moves. The data consist of encrypted indices of user current location and vicinity enclosing granules of some granularity level. According the protocol even if one (or more) location and a vicinity enclosing granule changes due to user movement, user's respective encrypted data must be updated on the LS by sending a M_{el} message. Thus, in most cases user's two M_{el} messages of consequent time steps would contain duplicated encrypted granules.

Clients communication can be reduced by enabling so called *incremental updates*(IU). On user location change, instead of sending M_{el}, the client may send new type of message, say M_{elUpd} containing items u, l, g_l^*, $\mathbf{g_{vDel}^*}$, $\mathbf{g_{vIns}^*}$. New items $\mathbf{g_{vDel}^*}$, $\mathbf{g_{vIns}^*}$ define encrypted granules that must be deleted and inserted on the LS in order to fully update user u encrypted data for level l. More precisely, if m_1 and m_2 are two consequent messages of type M_{el} such that $m_1.u{=}m_2.u$ and $m_1.l = m_2.l$ then the client may send a message m_3 of type M_{elUpd} instead of m_2, where $m_3.\mathbf{g_{vDel}^*} = m_1.\mathbf{g_v^*} \setminus m_2.\mathbf{g_v^*}$ and $m_3.\mathbf{g_{vIns}^*} = m_2.\mathbf{g_v^*} \setminus m_1.\mathbf{g_v^*}$. For example, if some user wants to update his encrypted granules for time step 1 while encrypted data for time step 0 is already on the server, it is enough for him to send a message m_3, containing sets $\mathbf{g_{vDel}^*}$ and $\mathbf{g_{vIns}^*}$. Figure 1.12 visualizes locations, vicinities, and vicinity-intersecting granules of a user at two consequent time steps 0 and 1. Darkened sets of cells $\mathbf{g_{vDel}}$ and $\mathbf{g_{vIns}}$ visualize unencrypted representation of sets g_{vDel}^* and g_{vIns}^*. Note, that introduction of m_3 helps reducing communication only if $|m_3.\mathbf{g_{vDel}^*}| + |m_3.\mathbf{g_{vIns}^*}| < |m_2.\mathbf{g_v^*}|$.

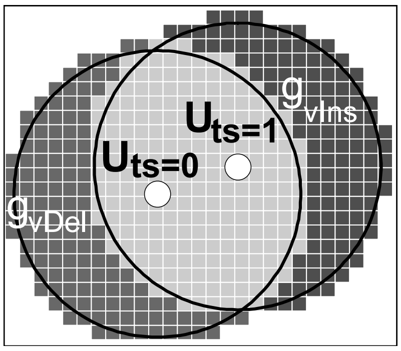


Figure 1.12: Granules deletion and insertion sets

Our presented client and server algorithms (See Alg. 3, 4) can be easily modified to support incremental updates. In the current implementation once a user goes from higher to lower levels (switches into coarser granularity) granules of active level are being removed from stacks on client and server (GS and GL) without possibility to reuse them. The idea is to preserve these granules on both client and server such that it would be possible to compute sets $\mathbf{g_{vDel}}$, $\mathbf{g_{vIns}}$,

$\mathbf{g}^*_{\mathbf{vDel}}$, and $\mathbf{g}^*_{\mathbf{vIns}}$.

The incremental updates impact on client communication is evaluated in Sec. 1.15.

1.13.4 Server computation optimization

Our VicinityLocator implementation checks if user's u_1 location lays inside a vicinity of user u_2 by performing u_1's encrypted location granule g^*_l search in the u_2 encrypted vicinity granules set $\mathbf{g}^*_{\mathbf{v}}$. This operation could be expensive in terms of computation especially if $\mathbf{g}^*_{\mathbf{v}}$ stores encrypted granules of high granularity levels.

Linear granule search worst case performance $O(|\mathbf{g}^*_{\mathbf{v}}|)$ can be improved up to $O(log(|\mathbf{g}^*_{\mathbf{v}}|))$ if clients would be required to sort encrypted granules in $\mathbf{g}^*_{\mathbf{v}}$ prior sending them to the LS. Then a binary search algorithm can be used on the LS to locate encrypted granule in an encrypted vicinity. As an alternative, the server may build the B-tree or hash table on values of $\mathbf{g}^*_{\mathbf{v}}$ locally.

1.13.5 Grouping of Users

Currently all users in $\mathbf{M}$ share a single encryption function Ψ. Security of Ψ directly influence location privacy of all users in the system. An adversary knowing Ψ can easily decipher encrypted granules of user current location and his vicinity. It is difficult to ensure that the function Ψ will stay secret in case of a high number of users of the VicinityLocator service.

In order to limit affected users in case of leaked Ψ, a so called *grouping of users* can be enforced. The friend-grouping is introduced in the long-paper version of the FriendLocator [16]. The idea is that all users in the system are grouped into possibly overlapping groups, so that each user is put into one or more groups. Both friends and non-friends can belong to the same group, but if two users are friends, they must be in at least one common group. Each such group G is assigned a distinct Ψ_G function and it is used by all members of G. Then if such Ψ_G is leaked, only the location privacy of the users in group G are compromised.

Our presented algorithms of VicinityLocator can be easily modified to support friend groups. The client and the server should treat each group individually such that client report his encrypted location data for all groups that he part of and the server analyzes encrypted data of one group users at the time. In this paper, we do not consider how these groups are created. This can be done automatically or manually by the users themselves.

1.14 Vulnerabilities & Points of Attack

We here address a few of the possible vulnerabilities of the VicinityLocator approach. We assume correct behavior of both the server and clients, and thus exclude attacks where an attacker may want to modify either server or client to e.g. trigger a “spamming” behavior. The attackers goal will for each attack be to compromise the privacy of the largest possible number of clients in the VicinityLocator system.

1.14.1 Compromised Client

If an attacker gains control over a client he will, for each group (see 1.13.5) the client is member of, have the Ψ function used to calculate granules at the client.

If the client itself is compromised by an attacker, the Ψ function is not much help to him, since he cannot do much else than encode granules sent to the server, but if one imagines that the attacker only temporarily gains control, then he can use the Ψ function to "Clone" the original client. This problem is however easily made void by changing the Ψ function regularly.

By using the *battleship* method the attacker can guess the location of other users with same group membership as the compromised client [5].

One way an attacker may "play battleship" in order to find the location of other users (with same group membership) would be to send a false vicinity covering the area he is interested in finding other users, the attacker will then be notified by the server if any other user is within his false vicinity. The attacker then continues to cut the vicinity in half, doing a binary search until he has found the granules of all users within the larger area he initially sent his false vicinity for at the start[6].

By using groups this attack is already very limited, since each client is assumed to have far fewer friends then the overall amount of users in the VicinityLocator system. Furthermore it is worth noticing that the attacker can never get an actual location of a user, since all he can get a matching granule which corresponds to a spacial area and not a point. There is also a build in limit on the amount of precision the attacker can achieve because each users L_{max} is a limit on the precision that any user will reveal.

If we limit the attackers goal to only focus on a single friend, then using the binary search method described will enable the attacker to track the single friend with the amount of vicinity splits he have to do in worst case being: $\Theta(Log(\frac{B(cg)}{B(max)}))$ where $B(cg)$ is the size of attackers current granule and $B(max)$ is the granule size at the maximum precision attacker can get, either by setting his own L_{max} or reaching the friends L_{max}.

[5] In the game of *battleships* two opponents take turn to guess the location of the others battleships placed at secret locations in a grid.

[6] The amount of granules needed to be searched in the worst case is $\frac{c^{l+1}-1}{c-1}$ where l is the number of levels needed to be traversed, and c the number of granules each granule at level l is divided into at $l+1$

1.14.2 Compromised Server & Client

If the attacker has gained control over both the server and client, he has all info from 1.14.1 as well as all users encrypted center and vicinity granules, as well as their group memberships (see 1.13.5).

The attacker can do the same as in 1.14.1, only now the attacker can skip the *battleship* step and decode obfuscated location (center granules) of friends directly, making it actually feasible to track all friends, this however is still only valid for the groups that the compromised client is member of.

This attack has the same limitations as 1.14.1, except that since the attacker now skip the *battleship* stage, it is feasible for the attacker to track many users (as long as they are in the same group as the compromised client)

1.14.3 Frequency

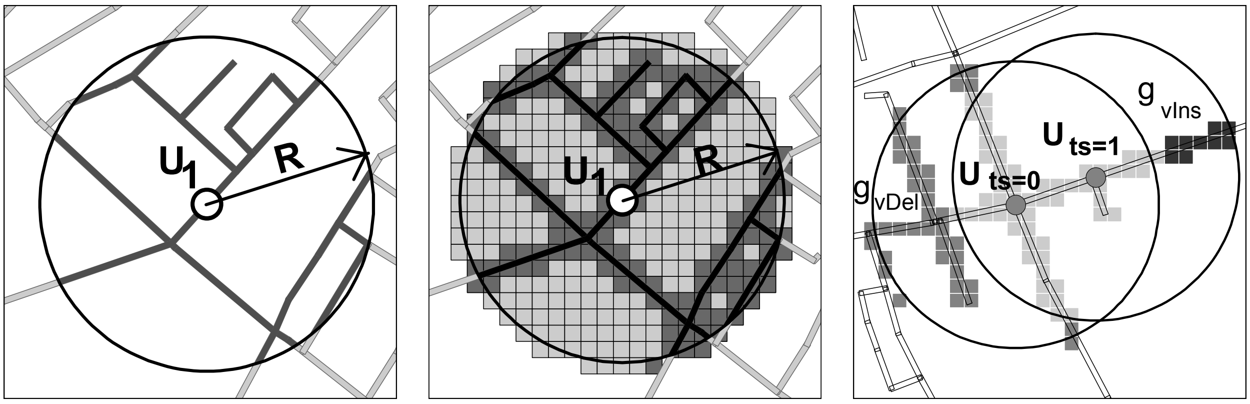
In this attack the server is compromised, and thus the attacker knows all users encrypted center and granules, stored on the server, as well as their group memberships.

The attacker can compare the frequency of users with same center and vicinity granules, the attacker can then see if many users have the same granules, and reason about the actual location (e.g. if attacker knows the national soccer team is playing, and he can see many users suddenly all sharing granules). The attacker can possibly collect the data over time and maybe make this attack more efficient by looking for frequency in locations over time, e.g. if there is a central place most people must pass during the day (city center/a bridge etc.), the attacker can then use historical information to identify which granules correspond to this location.

There is a simple solution to thwart the effectiveness of this attack, and that is to change the Ψ function as some interval, making it impossible for the attacker to compare granules from different intervals. If we furthermore assume that the server would not be informed when Ψ function is changed, then this attack becomes void.

1.15 Experimental Results

We here present performance tests to support our claims that our solution is efficient and applicable in a real world scenario. To support our claims we have implemented a prototype of our solution, as well as the approach from [16] for comparison. For simplicity in comparing the two implementations, Γ contains granularities as a grid with uniform squares, where edge length $B(l)$ depends on level l. We set $B(l) = L_0 \cdot 2^{-l}$ where $B(l) = \frac{L(l)}{\sqrt{2}}$, l is level, and L_0 is the cell side length at level 0.



Vicinity before rasterization

(a)

Vicinity after rasterization

(b)

RN- and IU-based vicinity rasterization

(c)

Figure 1.13:

1.15.1 Filtering and Unrestricted Vicinities

In the VicinityLocator prototype we have implemented a road network filter(RF) which minimizes the amount of granules needed to be sent to server, based on the road segments which intersects with the granules calculated for a user's vicinity.

User U_1 first calculate the intersection of road segment with his circular area of the interest, in proximity detection (see Fig. 1.13a). Afterwards U_1 rasterizes [7] the area of his vicinity (see Fig. 1.13b). The granules intersecting with road segments have been darkened to show that it is only these cells which will be sent to the server after U_1 has run the road network filter on his rasterized vicinity. To make the road network filter as realistic as possible, especially in dense grids at high levels, we have put in buffers around all edges in the Oldenburg files. The Oldenburg edges have two categories for road types, and when using the road network filter we have 2 different road widths to simulate small and large roads.

The idea of the road network filter is closely tied to the VicinityLocators ability to handle user vicinities of arbitrary shapes. In fact, the road network filter is a specific way to take advantage of this ability. It can clearly be seen in Fig. 1.13b that when the road network filter has been run U_1s vicinity is no longer circular, in fact it is not even solid anymore. It is this flexibility we take advantage of when using the road network filter.

When road network filter is used together with incremental updates (See sec. 1.13.3) the sets of granules, needed to be sent to server by user u, are visualized in Fig. 1.13c. Darkened sets of cells $\mathbf{g_{vDel}}$ and $\mathbf{g_{vIns}}$ correspond to u's vicinity cells at time step 0 that must be respectively

[7]By rasterizing we are here referring to the process of converting an area into uniform cells of a predefined size and shape

deleted and inserted in order to update cells of rasterized vicinity at time step 1.

Data Generation

The datasets used in our experiments are based upon the German city of Oldenburg. The data generator [4] gives allowance for controlling the number of users, their speed and each users number of consecutive position points. The area of Oldenburg is $26915 * 23572$ $units^2$, corresponding $14 * 12.26$ km^2. A location record is generated for each user at each time stamp, and the duration between two consecutive timestamps is 1 minute. The average speed of the users is 52 km/h (i.e. 1670 units per time stamp).

Test System Parameters

Both FriendLocator and VicinityLocator shares a number of settings which we, unless otherwise stated, set to default values for the experiments. The number of users is set to 50000 and the number of timestamps is set to 40, thus producing a workload of 2 million location records. We partition the set into disjoint groups, where each group contains 250 users by default. Within the same group, the friend relationships between users form a complete graph.

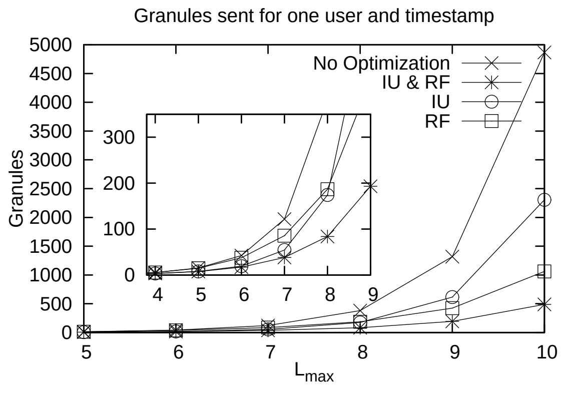
The default cell size of L_0 is 12800 units and the maximum level allowed by users, denoted L_ϵ and L_{max} for FriendLocator and VicinityLocator respectively, is set to 6, giving cell sizes ϵ and $B(L_{max})$ of 200 units [8]. In the VicinityLocator implementation, the vicinity region is circular, and the default radius is 500, corresponding to 260 meters.

Experiments

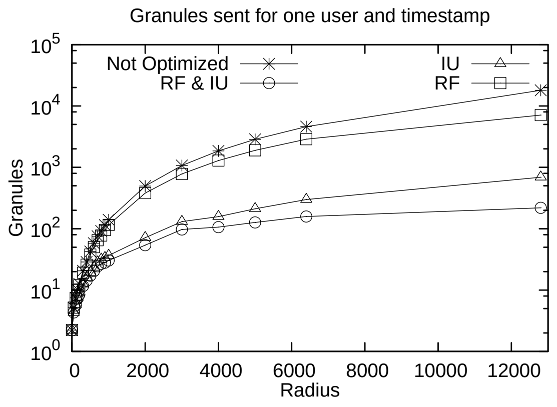
We will in the following focus mainly on the performance parameter of messages, since it is an important parameter in real world usage since users will have to pay for data when using VicinityLocator.

In Fig. 1.14a we show the cost of increasing the precision of proximity detection. We increase L_{max}, and thereby number of possible levels users can shift into. The experiment was run without any optimizations, with incremental updates(IU), with road network filter(RF), and with jointly applied road network filter and incremental updates (IU & RF). At level 10 the IU technique saves a user 50% of the granules he would have to send, and the RF technique saves a user almost 80%. The effect of the two optimization techniques is really good. When we combine IU and RF we send less than 10% of the granules sent by the unoptimized version of VicinityLocator.

[8] λ is $\epsilon * (2\sqrt{2} - 1)$ for FriendLocator and $B(L_{max}) * \sqrt{2}$ for VicinityLocator



(a)



(b)

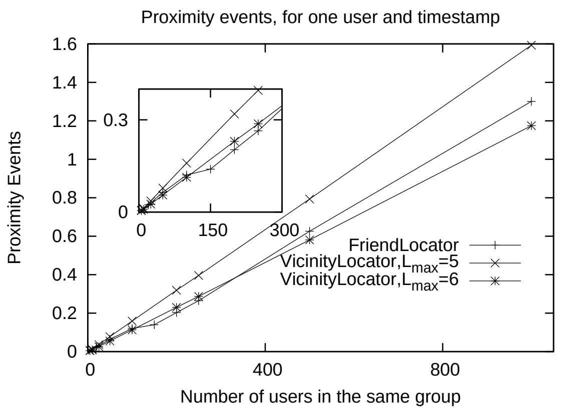
Figure 1.14: (a) Effect of increasing L_{max} with and without the Incremental Update(IU) and Roadnetwork Filter(RF) optimizations. (b) The total amount of messages when increasing radius of vicinity.

In Fig. 1.14b we show the effect on granules sent, when increasing the radius of users vicinity. This test is done with 2000 users split into 8 groups of 250 each, over 40 timestamps. When using RF there is a significant reduction which, as expected, is larger when the radius increases. This is because there are more road segments to work on. The IU optimization does however perform extremely well, giving a linear increase in granules, as oppose to the quadratic increase without any optimization. If RF and IU were to be used simultaneously the granule count would be lowered more. It is important to remember that the total amount of messages does not change for either of the three options.

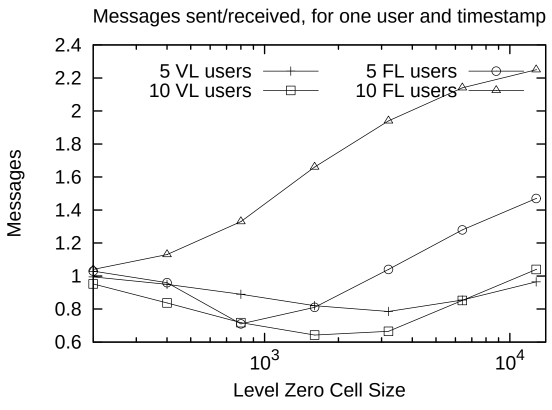
In Fig. 1.15a the amount of proximity events generated when increasing the size of groups are measured for FriendLocator and VicinityLocator. Because of the difference in way FriendLocator and VicinityLocator do proximity detection [9] VicinityLocator would have to set L_{max} to 0.37 level below L_ϵ of FriendLocator, and since levels are discrete values this is not possible. To make the comparison fair we therefore compare VicinityLocator and FriendLocator with L_ϵ and L_{max} of 6, as well as VicinityLocator for $L_{max} = 5$. When we do this we get a precision of VicinityLocator that is higher and lower than FriendLocator, for $L_{max} = 6$ and 5 respectively. We can see that the number of proximity events in FriendLocator is bound by the two test of VicinityLocator just as we would expect.

We want to motivate what an optimal cell side length L_0 at level 0 might be. Note that changing of L_0 gives the effect that there will be fewer or no level shifts (but maybe more cell

[9] VicinityLocator does proximity detection by vicinity- and center cell overlap, while FriendLocator detects proximity by searching overlap between 4 cells of every user.



(a)



(b)

Figure 1.15: (a) Change in number of proximity event when increasing number of users in a group. (b)The total messages sent by 2000 users, when decreasing level zero, keeping $B(L_{max})$ constant. Users have a radius of 200 (104 meters), the results are compared with FRIENDLOCATOR results

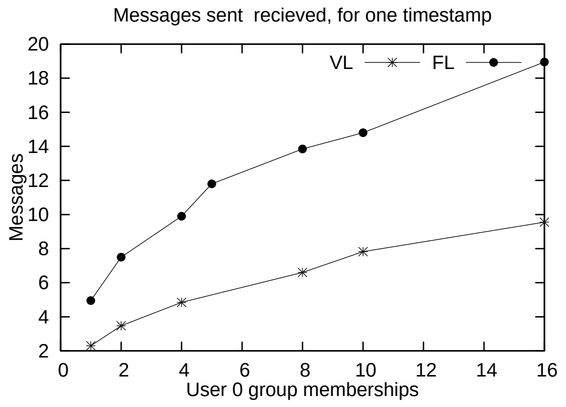
boundary crossings) in order to deliver same proximity detection precision. To this end we vary L_0 and L_{max}, keeping $B(L_{max}) = 200$ and users' vicinity radius equal to 200 throughout the tests (See Fig. 1.15b). We compare against FRIENDLOCATOR with equivalent precision, where setting of ϵ is equal to 200. We plot the graph for 5 and 10 users in the system, setting all users in one group for each test. It is clear from Fig. 1.15b that the VICINITYLOCATOR performs better in the amount of messages. The optimal L_0 is the lowest point on each of the four graphs in Fig. 1.15b.

In Fig. 1.16a we show the effect on messages sent/received by $user_0$ for each time stamp, when we (i) keep the number of friends constant at 80 (ii) increase the number of groups $user_0$'s friends are partitioned into (iii) let $user_0$ be member of all groups. It is clearly seen that VICINITYLOCATOR is almost consistently performing at 50% less messages for all partitions of $user_0$'s friends. We can see that partitioning the friends into more group raises the message cost, but it also heightens security and lets the user organize his friends, which may let the user save communication in the end, since the user may not want to update his location for all groups at all times e.g. he may only want to send updates to his "work"-group when he is at his job.

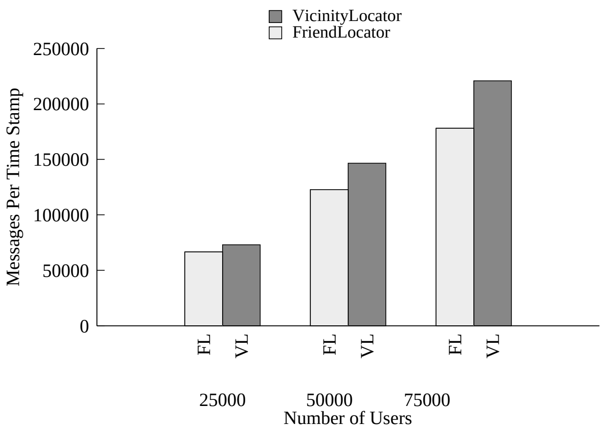
Figure 1.16b show the total amount of messages sent for one timestamp for 25000, 50000, and 75000 users with the number of users per group fixed at 250. It is clear from the graph that VICINITYLOCATOR incurs a higher amount of communication than FRIENDLOCATOR. But when looking the number of messages per user (See Table 1.4), then there is only a 0.6 message difference. Furthermore, the test show that the amount of communication is very

User count	Messages	
	FL	VL
25000	2.69	2.92
50000	2.46	2.93
75000	2.38	2.94

Table 1.4: Messages per User from Fig. 1.16b



(a)

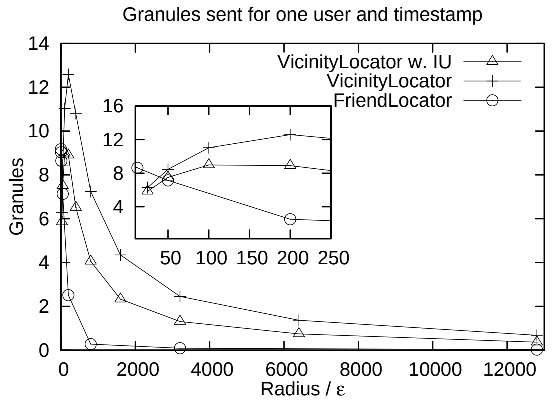


(b)

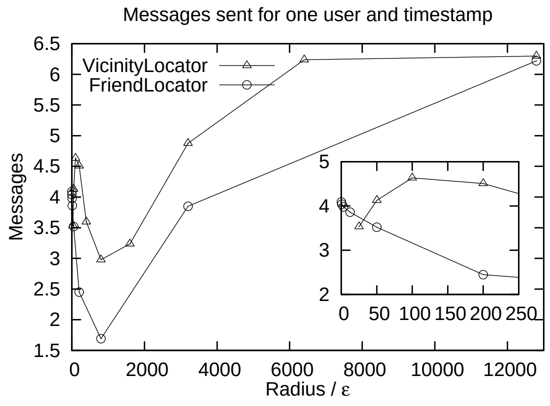
Figure 1.16: (a) The message cost for a user when increasing number of group memberships. (b) The total number of server messages for one time stamp for varying amounts of users.

stable with VicinityLocator, while the FriendLocator communication is actually falling.

In Fig. 1.17 we are trying to make a fair comparison between FriendLocator and VicinityLocator by forcing VicinityLocator to simulate the dependence between precision and privacy, which is one of the main problems in the FriendLocator. We do this simulation of the FriendLocator behavior by setting the $B(L_{max}) = \epsilon$ and $L_{max} = L_\epsilon$. It is clear from Fig 1.17a and 1.17b that VicinityLocator sends both more messages and granules than FriendLocator. When increasing the vicinity-radius or ϵ, the difference in messages is however not very large.



(a)



(b)

Figure 1.17: Simulating the behavior of FRIENDLOCATOR with VICINITYLOCATOR, the two approaches are compared on the amount of messages (b) and granules (a) sent to server.

1.16 Conclusion

In this paper we develop the VICINITYLOCATOR, a client-server solution for detecting proximity by inclusion of one user's location inside another user's vicinity, while offering users control over both location privacy and precision of proximity detection.

The client maps its location into a granule and finds all granules contained in his vicinity, which can be shaped arbitrarily. The client then encrypts its location- and vicinity- granules and sends them to the server which checks for proximity by checking for inclusion of u_1's location granule in the set of u_2's vicinity granules. The server does the test blindly, without ever knowing anything about user locations.

We look at three interesting types of attacks which can be applied to VICINITYLOCATOR, and we show that VICINITYLOCATOR has numerous features helping to limit the effect of any attack.

Experimental results showed that VICINITYLOCATOR performs adequate for real world application and it is scalable to high number of users. Our presented optimization techniques worked very well and in some cases cut the amount of data transferred by approximately 90%. Experiments showed, that VICINITYLOCATOR' features such as irregular shaped vicinities and adjustable precision do not give significant overhead in terms of communication when compared with the FRIENDLOCATOR. Note, that the FRIENDLOCATOR has none of the mentioned features.

In the future, we plan to extend the proposed solution for proximity detection to support dynamically changing shape and size of granules, adapting to user behavior, giving lower communication cost for the users.

Part IV

Conclusion

During the semester we have developed two distinct client-server based privacy preserving frameworks for notifying users in a proximity. They provide strong location privacy to users and offer control over various parameters such as proximity distances and precision.

The VicinityLocator introduces new features compared to other friend detection solutions and offers allot of flexibility, however it is slightly more expensive in terms of communication. The FriendLocator provides a different method than VicinityLocator for users to specify their desirable proximity distances. In the FriendLocator users pair-wise agree on their proximity distances, where as in the VicinityLocator every user in the system selects their "area-of-interest" individually. The FriendLocator is more limited in terms of control over precision parameters, but performs better in typical scenarios.

We have shown that both VicinityLocator and FriendLocator are resilient against general types of attack, and they both suggest methods to limit any privacy leak which would occur. The performance studies from both the VicinityLocator and FriendLocator paper suggest that the amount of communication our solutions require does not exceeds a limit we would assume users are willing to pay for their privacy. Also, experiments with prototypes show that they are scalable for large number of users and applicable in real world systems.

A possible topic for further studies would be to look at different criteria for location updates. Currently both the FriendLocator and VicinityLocator implement region based update policy, where users update their location once they cross cell boundaries. In other existing user tracking solutions more advanced update policies such as velocity-vector-based or roadnetwork-based policy help significantly cut down communication cost without affecting quality of service. Thus it would be interesting to see how any of those policies can be integrated into our solutions. Another interesting research direction would be our solution adaptation for some existing cloud computing infrastructure, such as Google AppEngine, Windows Azure, Amazon EC2, etc.

Part V

Appendix

Appendix A

A Location Privacy Aware Friend Locator (6 Page Version)

Abstract

A location-based service called friend-locator notifies a user if the user is geographically close to any of the user's friends. Services of this kind are getting increasingly popular due to the penetration of GPS in mobile phones, but existing commercial friend-locator services require users to trade their location privacy for quality of service, limiting the attractiveness of the services. The challenge is to develop a communication-efficient solution such that (i) it detects proximity between a user and the user's friends, (ii) any other party is not allowed to infer the location of the user, and (iii) users have flexible choices of their proximity detection distances. To address this challenge, we develop a client-server solution for proximity detection based on an encrypted, grid-based mapping of locations. Experimental results show that our solution is indeed efficient and scalable to a large number of users.

Introduction

Mobile devices with geo-positioning capabilities are becoming cheaper and more popular. Consequently users start using *friend-locator* services (e.g., Google Latitude, FireEagle) for seeing their friends' locations on a map and identifying nearby friends.

In existing services, the detection of nearby friends is performed manually by the user, e.g., by periodically examining a map on the mobile device. This works only if the user's friends agree to share either exact or obfuscated location. However, LBS users usually demand certain level of privacy and may even feel insecure if it is not provided [9]. Due to the poor support for location privacy in existing friend-locator products, it is sometimes not possible to detect nearby friends if location privacy is desired. The challenge is to design a communication-efficient friend-locator LBS that preserves the user's location privacy and yet enables automatic detection of nearby friends.

To address the challenge, we develop a client-server, location-privacy aware friend-locator LBS, called the FriendLocator. It first employs a grid structure for cloaking the user's location into a grid cell and then converts it into an encrypted tuple before it is sent to the server. Having received the encrypted tuples from the users, the server can only detect proximity among them, but it is unable to deduce their actual locations. In addition, users are prevented from knowing the exact locations of their friends. To optimize the communication cost, the FriendLocator employs a flexible region-based location-update policy where regions shrink or expand depending on the distance of a user from his or her closest friend.

The rest of the paper is organized as follows. We briefly review related work in Section A and then define our problem setting in Section A. The FriendLocator is presented in Section A. Section A presents experimental results of our proposal and Section A concludes the paper.

Related Work

In this section, we review relevant work on location privacy and proximity detection.

Location privacy

Most of the existing location privacy solutions employ the *spatial cloaking* technique, which generalizes the user's exact location q into a region Q' used for querying the server [8]. Alternative approaches [10, 18, 7] have also been studied recently. However, all these solutions focus on range/kNN queries and assume that the dataset is public (e.g., shops, cinemas). In contrast, in the proximity detection problem, the users' locations are both queries and data points that must be kept secret.

Proximity detection

Given a set of mobile users and a distance threshold ϵ, the problem of proximity detection is to continuously report all events of mobile users being within the distance ϵ of each other. Most existing solutions (e.g., [2]) focus on optimizing the communication and computation costs, rather than location privacy.

Recent solutions were proposed [15, 12] to address location privacy in proximity detection. Ruppel et al. [15] develop a centralized solution that applies a *distance-preserving mapping* (i.e., a rotation followed by a translation) to convert the user's location q into a transformed location q'. Unfortunately, Liu et al. [11] point out that distance-preserving mapping can be easily attacked. Mascetti et al. [12] employ a server and apply the filter-and-refine paradigm in their secure two-party computation solution. However, it lacks distance guarantees for the proximity events detected by the server, and leads to low accuracy when strong privacy is required. Unlike our approach, the central server in their proposal knows that a user is always located within his or her cloaked region.

Our solution is fundamentally different from the previous solutions [15, 12] because we employ encrypted coordinates to achieve strong privacy and yet the server can blindly detect proximity among the encrypted coordinates.

Problem Definition

In this section we introduce relevant notations and formally define the problems of proximity detection and its privacy-aware version.

In our setting, a large number of mobile-device users form a social network. These mobile devices (MD) have positioning capabilities and they can communicate with a central location server (LS). We use the terms *mobile devices* and *users* interchangeably and denote the set of all MDs (and their users) in the system by $\mathbf{M} \subset \mathbb{N}$.

The friend-locator LBS notifies two users $u, v \in \mathbf{M} | u \neq v$ if u and v are friends and the proximity between u and v is detected. Given the distance thresholds ϵ and λ, the proximity and separation of two users u and v are defined as follows [2]:

1. If $dist(u, v) \leq \epsilon$, then the users u and v are in proximity;
2. If $dist(u, v) \geq \epsilon + \lambda$, then the users u and v are in separation;
3. If $\epsilon < dist(u, v) < \epsilon + \lambda$, then the service can freely choose to classify users u and v as being either in proximity or in separation.

Here, $dist(u, v)$ denotes the Euclidean distance between the users u and v. The parameter ϵ is called the *proximity distance*, and it is agreed/selected by u and v. The parameter $\lambda \geq 0$ is a service precision parameter and it introduces a degree of freedom in the service. As different pairs of friends may want to choose different proximity distances, we use $\epsilon(u, v)$ to denote the proximity distance for the pair of users $u, v \in \mathbf{M}$. For simplicity we assume mutual friendships, i.e., if v is a friend of u, then u is a friend of v, and we let the proximity distance to be symmetric, i.e., $\epsilon(u, v) = \epsilon(v, u)$ for all friends $u, v \in \mathbf{M}$.

A proximity notification must be delivered to MDs when proximity is detected. Any subsequent proximity notification is only sent after separation have been detected.

The friend-locator LBS must be efficient in terms of mobile client communication and provide the following privacy guarantees for each user $u \in \mathbf{M}$: (i) The exact location of u is never disclosed to other users or the central server. (ii) User u only permits friends to detect proximity with him.

Proposed Solution

In this section we propose a novel, incremental proximity detection solution based on encrypted grids. It is designed for the client-server architecture, it is efficient in terms of communication, and it satisfies user location-privacy requirements (see Sec. A).

Grid-based encryption

Let us consider three parties: two friends, u_1 and $u_2 \in \mathbf{M}$, and the location server (LS). Both users can send and receive messages to and from LS. User u_1 is interested in being informed by LS when user u_2 is within proximity and vice versa.

Assume that users u_1 and u_2 share a *list of grids*, where a grid index within the list is termed *level*. Grids at all levels are coordinate-axis aligned and their cell sizes, i.e., width and height, at levels $l = 0, 1, 2, ...$ are fixed and equal to $L(l)$. We let $L(l) = g \cdot 2^{-l}$, where g is some level zero cell size. Then sizes of cells gradually decrease going from lower to higher levels, level zero cells being the largest.

Each column (row) of each of these grids is assigned a unique *encryption number*. A grid within the list, together with encryption numbers, constitutes a Location Mapping Grid (LMG). Each user generates such a list of LMGs utilizing two shared private functions L and ψ, where $\Psi : \mathbb{N} \mapsto \mathbb{N}$ is a one-to-one encryption function (e.g., AES) mapping a column/row number to an encryption number.

Incremental proximity detection

Assume that users u_1 and u_2 use an LMG of some level l. Whenever a user moves into a new cell of LMG, the following steps are taken:
(i) The user maps the current location (x, y) into an LMG cell $(k,m)=(\lfloor x/L(l)\rfloor, \lfloor y/L(l)\rfloor)$.
(ii)The user computes an encrypted tuple $e = (l,\alpha^-,\alpha^+,\beta^-,\beta^+)$ by applying $E_\Psi(l, k, m) = (l, \Psi(k), \Psi(k+1), \Psi(m), \Psi(m+1))$, where (α^-,α^+) and (β^-,β^+) are encrypted values of adjacent columns k and $k + 1$ and adjacent rows m and $m + 1$ respectively.
(iii) The user sends the encrypted tuple e to LS.

Since u_1 and u_2 use the same list of LMG, with the same encryption-number assignments for each column and row, the LS can detect proximity between them by checking if the following function is true:

$$\Gamma(e_1, e_2) = (e_1.l = e_2.l) \wedge ((e_1.\alpha^- = e_2.\alpha^-) \vee (e_1.\alpha^- = e_2.\alpha^+) \vee (e_1.\alpha^+ = e_2.\alpha^-))$$
$$\wedge ((e_1.\beta^- = e_2.\beta^-) \vee (e_1.\beta^- = e_2.\beta^+) \vee (e_1.\beta^+ = e_2.\beta^-)).$$

Parameters e_1 and e_2 are encrypted tuples delivered from users u_1 and u_2 respectively. Note that since Ψ is a one-to-one mapping, Γ is evaluated to *true* if and only if k_{u_1} or $k_{u_1}+1$ matches k_{u_2} or $k_{u_2}+1$ and m_{u_1} or $m_{u_1}+1$ matches m_{u_2} or $m_{u_2}+1$, where (k_{u_1}, m_{u_1}) and (k_{u_2}, m_{u_2}) are LMG cells of users u_1 and u_2 respectively.

In the extended version of this paper we prove that an LMG at level l can be used to detect proximity with the following settings $\epsilon = L(l)$, $\lambda = L(l) \cdot (2\sqrt{2} - 1)$, i.e., Γ is always *true* when $dist(u_1, u_2) \leq L(l)$ and always *false* when $dist(u_1, u_2) \geq L(l) \cdot 2\sqrt{2}$. Every two friends $u_1, u_2 \in \mathbf{M}$ choose an LMG level, called *proximity level* $L_\epsilon(u_1, u_2)$ that corresponds best to their proximity detection settings. Then our approach forces every user to stay at the lowest-possible level such that few grid-cell updates are necessary. Only when proximity between friends $u_1, u_2 \in \mathbf{M}$ is detected at a low level, are they asked to switch to a higher level. This repeats until required level $L_\epsilon(u_1, u_2)$ is reached or it is determined that users are not in proximity.

Figure A.1 illustrates the approach. It shows the geographical locations of two friends u_1 and u_2, and their mappings into LMGs at 4 snapshots in time. Note that lower level grids are on top in the figure. Assume that u_1 and u_2 have agreed on $L_\epsilon(u_1, u_2) = 2$ and have already sent their encrypted tuples, for levels 0 and 1 to LS. Figure A.1a visualizes when LS detects a proximity at level 0, but not at level 1. As $L_\epsilon(u_1, u_2) > 0$, nothing happens until a location change. In Figure A.1b both users have changed their geographical location. User u_2 did not go from one cell to another at his current level 1, thus he did not report a new encrypted tuple. User u_1 however, changed cells at both level 1 and level 0, he therefore sends a new encrypted tuple for level 0. The LS detects a proximity between u_1 and u_2 at level 0 and asks u_1 to switch to level 1, because $L_\epsilon(u_1, u_2) > 0$. Figure A.1c shows user LMG mapping when u_1 has delivered new encrypted tuple for level 1. Again, LS detects proximity at level 1 and commands both users u_1 and u_2 to switch to level 2. When both encrypted tuples for level 2 are delivered to LS, it detects the proximity at this level (see Figure A.1d) and, because $2 = L_\epsilon$, proximity notifications are sent to u_1 and u_2.

Note that the presented algorithms implement an adaptive region-based update policy. If a user is far away from his friends, then he stays at a low-level grid with large cells, resulting in few updates for the user's future movement. Only when the user approaches one of his friends, he is asked to switch to higher levels with smaller grid cells. Thus, at a given time moment, the user's current communication cost is not affected by the total number of his friends, but by the distance to his closest friend.

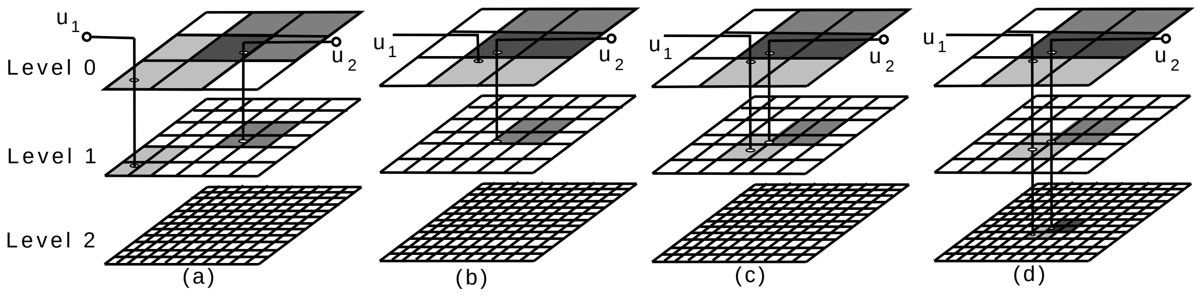


Figure A.1: Two-user proximity detection in the FRIENDLOCATOR

Experimental Study

The proposed FRIENDLOCATOR and a competitor solution, called Baseline , were implemented in C#. In this section, we study their communication cost in terms of messages received by the clients and the server. The network-based generator [4] is used to generate a workload of users moving on the road network of the German city Oldenburg. A location record is generated for each user at each timestamp.

Competitor Solution

The Baseline employs the filter-and-refine paradigm for proximity detection among friend pairs. Each user cloaks its location by using a uniform grid, and sends its cell to the server. Filtering is performed at the LS, which calculates the *min* and *max* distances [14] between the cells c_i and c_j of the users u_i and u_j. The LS then checks the following conditions:

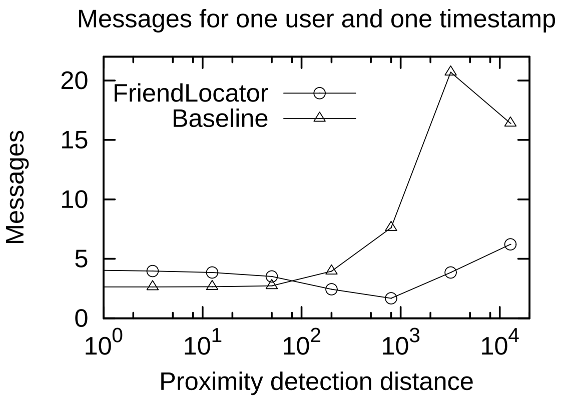
1. If $maxdist(c_i, c_j) \leq \epsilon$, then LS detects a proximity.
2. If $mindist(c_i, c_j) > \epsilon$, then LS detects no proximity.
3. If $mindist(c_i, c_j) \leq \epsilon < maxdist(c_i, c_j)$, then users u_i and u_j invoke the peer-to-peer Strips algorithm [2] for the refinement step.

The resulting communication cost is lower than Strips due to the use of a centralized (untrusted) server. Observe that, the Baseline does not use encrypted tuples as in our FRIENDLOCATOR solution, so it offers a weaker notion of privacy.

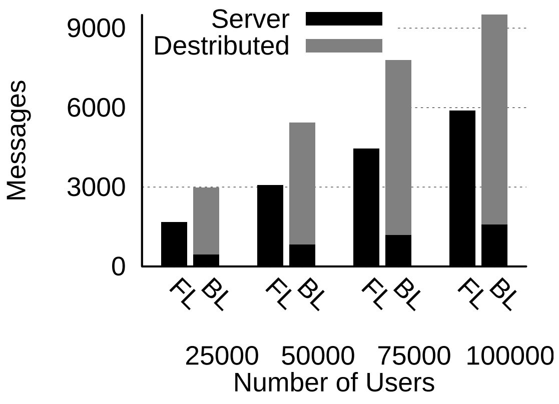
Experiments

We first study the impact of the proximity detection distance ϵ on the cost per user per timestamp (Fig. A.2a). Both Baseline and FRIENDLOCATOR have similar performance at small ϵ

(below 10). As ϵ increases, Baseline invokes the refinement step frequently so its cost rises rapidly. At extreme ϵ values (above 10000), most of the pairs are within proximity so the frequency and cost of executing the refinement step in Baseline are reduced. Observe that the cost of FriendLocator is robust to different values of ϵ, and its cost rises slowly when ϵ increases. Figure A.2b shows the total number of messages during 40 timestamps as a function of the total number of users in the system. Clearly, FriendLocator incurs substantially lower total cost than Baseline . In Fig. A.2b the distributed messages represent peer-to-peer messages.



(a) Client message cost vs. ϵ



(b) System message cost for each timestamp

Figure A.2: Effect of various parameters on the communication cost

Conclusion

In this paper we develop the FriendLocator, a client-server solution for detecting proximity among friend pairs while offering them location privacy. The client maps a user's location into a grid cell, converts it into an encrypted tuple, and sends it to the server. Based on the encrypted tuples received from the users, the server determines the proximity between them blindly, without knowing their actual locations. Experimental results suggest that FriendLocator incurs low communication cost and it is scalable to a large number of users.

In the future, we plan to extend the proposed solution for privacy-aware proximity detection among moving users on a road network, in which the distance between two users is constrained by the shortest path distance between them.

Bibliography

[1] ABIresearch. Location-based mobile social networking will generate global revenues of $3.3 billion by 2013, August 2008.

[2] Arnon Amir, Alon Efrat, Jussi Myllymaki, Lingeshwaran Palaniappan, and Kevin Wampler. Buddy Tracking - Efficient Proximity Detection Among Mobile Friends. In *INFOCOM*, 2004.

[3] Claudio Agostino Ardagna, Marco Cremonini, Ernesto Damiani, Sabrina De Capitani di Vimercati, and Pierangela Samarati. Location Privacy Protection Through Obfuscation-Based Techniques. In *DBSec*, pages 47–60, 2007.

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

[5] Canalys.com. Gps smart phone shipments overtake pnds in emea, November 2008.

[6] Matt Duckham and Lars Kulik. A Formal Model of Obfuscation and Negotiation for Location Privacy. In *PERVASIVE*, pages 152–170, 2005.

[7] Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD*, pages 121–132, 2008.

[8] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *USENIX MobiSys*, pages 31–42, 2003.

[9] Andrew Heining. Stalk your friends with google, February 2009.

[10] Ali Khoshgozaran and Cyrus Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, pages 239–257, 2007.

[11] Kun Liu, Chris Giannella, and Hillol Kargupta. An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining. In *PKDD*, 2006.

[12] Sergio Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, and Sushil Jajodia. Privacy-aware proximity based services. In *MDM*. IEEE Computer Society, 2009.

[13] Mohamed F. Mokbel, Chi-Yin Chow, and Walid G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*, pages 763–774, 2006.

[14] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest Neighbor Queries. In *SIGMOD*, pages 71–79, 1995.

[15] Peter Ruppel, Georg Treu, Axel Küpper, and Claudia Linnhoff-Popien. Anonymous User Tracking for Location-Based Community Services. In *LoCA*, pages 116–133, 2006.

[16] Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen. A Location Privacy Aware Friend Locator. In *SSTD*, pages 0–100, 2009.

[17] Zhengdao Xu and Hans-Arno Jacobsen. Adaptive Location Constraint Processing. In *SIGMOD*, pages 581–592, 2007.

[18] Man Lung Yiu, Christian S. Jensen, Xuegang Huang, and Hua Lu. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*, pages 366–375, 2008.