A Location Privacy Aware Friend Locator

Laurynas Šikšnys and Jeppe R. Thomsen

Aalborg University
Department of Computer Science
Selma Lagerløfs Vej 300, DK-9220 Aalborg Ø
Denmark

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 equiped with a new set of features, performs well compaired to existing solutions.

1 Introduction

Mobile devices with built-in geo-positioning capabilities are becoming cheaper and more popular [1]. 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 [2]. Thus several friend-locator services, like *iPoki*, *Google Latitude*, and *Fire Eagle* [1] 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 [3] 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

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

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 [4–6]. 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 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 [5] (Fig. 1a) or rectangular-shaped and not user-location-centered [6] (Fig. 1b). 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. 1c) 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. 1d) 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.

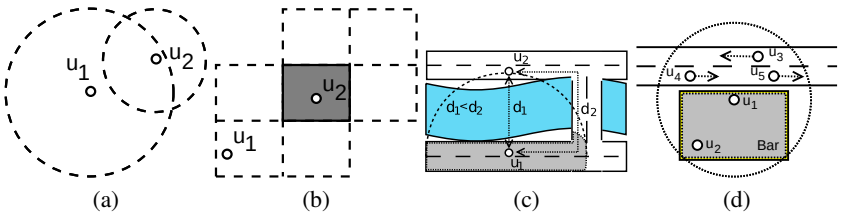


Fig. 1. Types of vicinities and proximity detection scenarios

The introduced shortcoming of the scenarios can be eliminated if instead of static-shape, dynamic-shape vicinities were used. In the first case (Fig. 1c), 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. 1d), if user u_1 could preselect vicinity region, matching the bar as it is his "area of inter-

est", 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 2 and then define our problem setting in Section 3. The VICINITYLOCATOR is presented in Section 4. In Section 5 We present 3 general attacks applicable to our solution. Section 6 presents extensive experimental results of our proposed approach.

2 Related Work

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

2.1 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 [7–10] 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 [11, 12] 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 can not 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.

2.2 Privacy-aware proximity detection methods

Anonymous User Tracking for Location-Based Community Services Ruppel et al. [4] 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. [13] 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. [5] 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 are visualized by Fig. 2a. 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 2b 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.

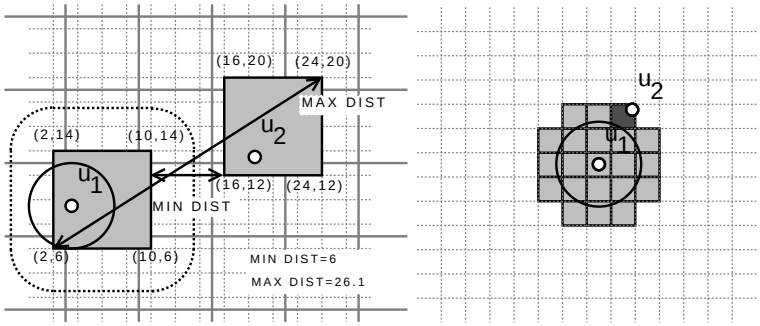


Fig. 2. Concepts used in Mascetti et al. [5] solution

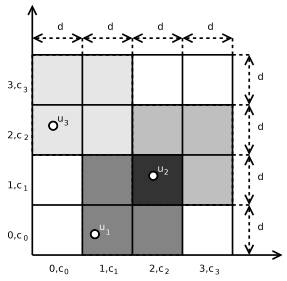
A Location Privacy Aware Friend Locator Šikšnys et al. [6] have developed a centralized privacy-aware proximity detection method, called FRIENDLOCATOR, which provides strong privacy guaranties and employs spacial grid-based technique to optimize communication cost. The whole space is divided into equal-sized 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 [6] visualized in Fig. 3a and 3b. 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. 3b. 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 ($\alpha^-$, 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 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.

$$\Gamma(e_1, e_2) = \big((e_1.\alpha^- = e_2.\alpha^-) \vee (e_1.\alpha^- = e_2.\alpha^+) \vee (e_1.\alpha^+ = e_2.\alpha^-)\big) \\ \wedge \big((e_1.\beta^- = e_2.\beta^-) \vee (e_1.\beta^- = e_2.\beta^+) \vee (e_1.\beta^+ = e_2.\beta^-)\big)\,. \tag{1}$$

They 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. 3a corresponds to users required level, then by evaluating Eq. 1 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)

Fig. 3. 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. 1c and 1d) 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 3a) and their intersections at required level triggers the proximity event.

2.3 Our contribution

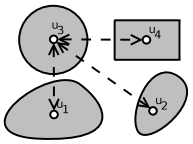
In this work we combine best features of the previously presented proximity detection approaches to build our solution, the VICINITYLOCATOR. The solution combines the ideas of encrypted coordinates, their blind evaluation at the server-side, and vicinity's representation by granules, where sizes can be changed dynamically. Our solution, unlike Mascetti et al. [5] 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.

3 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 contacts are visualized in Fig. 4a. 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. 4b.



(a)

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

(b)

Fig. 4. User locations, vicinities, and friend-ship relation example

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.

4 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.

4.1 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* [6], 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[2] specifies a divisions of the spatial domain into a number of non-overlapping regions, called granules[5]. 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 $id(g) \in \mathbb{N}$.

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

– 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. $L(l) < L(l-1)$.
– Every granule in level l is fully contained by some granule at level $l-1$.

Figure 5a visualizes a valid granularity list, where uniform grids are uses 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. 5b. 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.

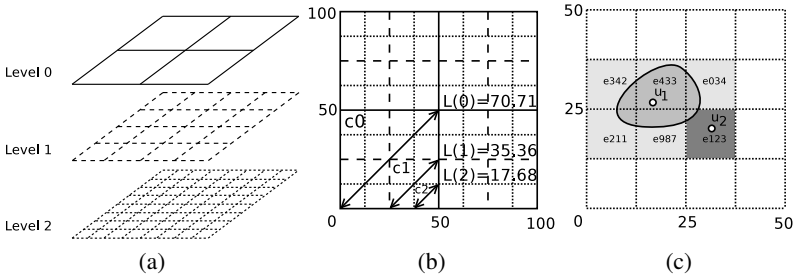


Fig. 5. The valid list of granularities and the behavior of the granularity-based classifier

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

Similarity to FriendLocator [6], 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. 5c) 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 1.

Lemma 1. *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 3.*

Proof. According the Section 3, 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 encrypted index of u_2's location granule $e123$, 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. $MaxDist(g) \leq L(l)$. 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 can not be found in $\mathbf{g_v}$ then $loc(v) \notin vic(u)$.

According to Lemma 1, 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.

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

4.2 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.

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

onInitialization A handler executed only once at a startup of the LS or a MD.

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. Client and server messages types

Algorithms 1 and 2 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. 1) 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. 6a and 6b, where user's u_1 vicinity and user's u_2 current location mapping into list of granularities (grids) are visualized at 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. 6a and 6b) 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 $0..GL(u) - 1$. 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 $\mathbf{onMessageReceived}$ 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

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.

mapLocToGranularity(Level number $level$)
- $g_l \leftarrow \{id(g) | g \in \Gamma(level) : loc(u) \in g\}$;
- $\mathbf{g_v} \leftarrow \{id(g) | \forall g \in \Gamma(level) : g \cap vic(u) \neq \emptyset\}$;
- **return** $(g_l, \mathbf{g_v})$;

pushAndSend()
- $(g_l, \mathbf{g_v}) \leftarrow mapLocToGranularity(|GS|)$;
- Push $\langle g_l, \mathbf{g_v} \rangle$ to stack GS;
- Send to LS $M_{el}(u, |CS| - 1, \Psi(g_l), \{\Psi(g) | \forall g \in \mathbf{g_v}\})$;

onLocationChange()
- $wasPopped \leftarrow false$
- **while** $|GS| > 0$ *and* $top(GS) \neq mapLocToGranularity(|CS| - 1)$ **do**
  - Pop from stack GS;
  - $wasPopped \leftarrow true$;
- **if** $wasPopped$ *or* $|GS| = 0$ **then**
  - **pushAndSend**()

onMessageReceived(Message M_{LevInc}, Level l)
- **while** $|GS| \leq l$ *and* $|GS| \leq L_{max}(u)$ **do**
  - **pushLocAndSend**()

onMessageReceived(Message M_{prox}, Friend v, Level$l$)
- Output "Friend ",v," with precision ", $L(l)$, " is inside our vicinity!";

Algorithm 1: The MD's event handlers in our proximity based service.

encrypted vicinity granules $\mathbf{g_v^*}$ 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 $\mathbf{g_v^*}$ 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 $\mathbf{g_v^*}$ 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. 6 example. The server finds that g_l^* of user u_2 lays in $\mathbf{g_v^*}$ of user u_1 at level 0 in Fig. 6a, 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

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.

onMessageReceived(Message M_{el}, User u, Level $l, g_l^*, \mathbf{g_v^*}$)
- **while** $|GL(u)| > 0$ *and* $|GL(u)| \text{¿ } l$ **do**
  - Pop from stack $GL(u)$;
- Push $\langle g_l^*, \mathbf{g_v^*} \rangle$ to stack $GL(u)$;
- **foreach** $v \in \mathbf{M}$ *such that* $v \neq u$ *and* $|GL(v)| > 0$ *and* $(u, v) \in \mathbf{F}$ **do**
  - $l_m \leftarrow min(|GL(u)| - 1, |GL(v)| - 1, L_{max}(u), L_{max}(v))$;
  - $vInU \leftarrow get(GL(v), l_m).g_l^* \in get(GL(u), l_m).\mathbf{g_v^*}$;
  - $uInV \leftarrow get(GL(u), l_m).g_l^* \in get(GL(v), l_m).\mathbf{g_v^*}$;
  - **if** $vInU = true$ *or* $uInV = true$ **then**
    - **if** l_m = $L_{max}(u)$ *or* l_m = $L_{max}(v)$ **then**
      - **if** $vInU$ *and* $v \notin \mathbf{P(u)}$ **then**
        - insert v into $\mathbf{P(u)}$;
        - send $M_{prox}(v, l_m)$ to MD u;
      - **if** $uInV$ *and* $u \notin \mathbf{P(v)}$ **then**
        - insert u into $\mathbf{P(v)}$;
        - send $M_{prox}(u, l_m)$ to MD v;
    - **else**
      - **if** $l_m = |GL(u)| - 1$ **then**
        - send $M_{LevInc}(l_m + 1)$ to MD u
      - **if** $l_m = |GL(v)| - 1$ **then**
        - send $M_{LevInc}(l_m + 1)$ to MD v
  - **if** $vInU = false$ **then**
    - remove v from $\mathbf{P(u)}$;
  - **if** $uInV = false$ **then**
    - remove u from $\mathbf{P(v)}$;

Algorithm 2: **onMessageReceived** event handler on the LS.

are higher than level 0 precision, the LS no longer founds g_l^* in $\mathbf{g_v^*}$ and then nothing happens until one of them starts moving. When u_1 sends his location data for level 0 in Fig. 6b, 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 $\mathbf{g_v^*}$ at level 1 in Fig. 6c 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. 6d, 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}.

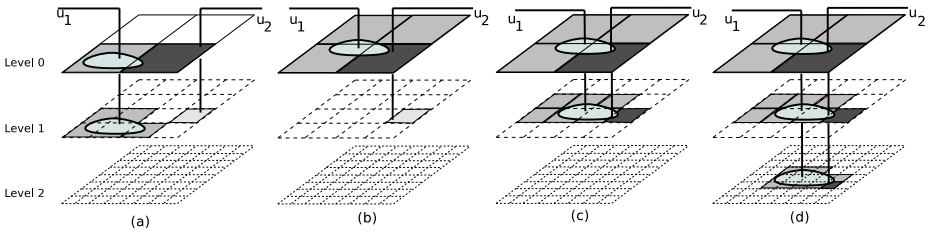


Fig. 6. Example of level changes and proxinity detection within VICINITYLOCATOR

Note that similarly to the FRIENDLOCATOR [6], 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.

4.3 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 some 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 4.3 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 $\mathbf{g_{vIns}}$ visualized unencrypted representation of sets g^*_{vDel} and g^*_{vIns}.

Note, that introduction of m_3 helps reducing communication only if $|m_3.\mathbf{g^*_{vDel}}|$ + $|m_3.\mathbf{g^*_{vIns}}| < |m_2.\mathbf{g^*_v}|$. The incremental updates impact on client communication is evaluated in Sec. 6.

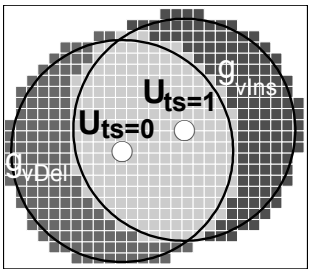


Fig. 7. Granules deletion and insertion sets

4.4 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 $\mathbf{g^*_v}$. This operation could be expensive in terms of computation especially if $\mathbf{g^*_v}$ stores encrypted granules of high granularity levels.

Linear granule search worst case performance $O(|\mathbf{g^*_v}|)$ can be improved up to $O(log(|\mathbf{g^*_v}|))$ if clients would be required to sort encrypted granules in $\mathbf{g^*_v}$ 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 $\mathbf{g^*_v}$ locally.

4.5 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 [6]. 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.

5 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.

5.1 Compromised Client

If an attacker gains control over a client he will, for each group (see 4.5) 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 can not 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 [3].

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.

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 notesing 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

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

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. [4]. 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}.

5.2 Compromised Server & Client

If the attacker has gained control over both the server and client, he has all info from 5.1 as well as all users encrypted center and vicinity granules, as well as their group memberships (see 4.5).

The attacker can do the same as in 5.1, 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 5.1, 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)

5.3 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 compair 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 compair granules from different intervals. If we furthermore assume that the server wont be informed when Ψ function is changed, then this attack becomes void.

6 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

[4] 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 devided into at $l+1$

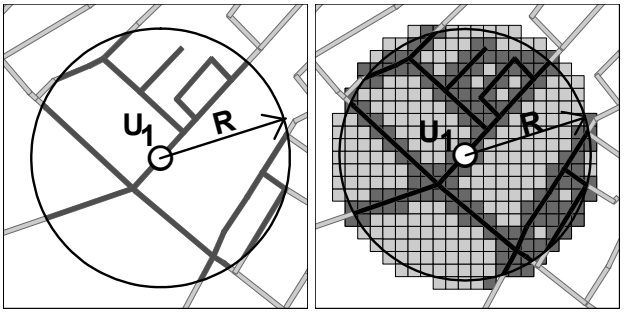
prototype of our solution, as well as the approach from [6] for comparison. For simplicity in comparing the two implementations, Γ contains granularities as a grid with uniform squares, where edge length $B(l)$ depends on level l. We set $B(l) = L_0 \cdot 2^{-l}$ where $B(l) = \frac{L(l)}{\sqrt{2}}$, l is level, and L_0 is the cell side length at level 0.

6.1 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 users vicinity.

User U_1 first calculate the intersection of road segment with his circular area of the interest, in proximity detection (see Fig. 8a). Afterwards U_1 rasterizes [5] the area of his vicinity (see Fig. 8b).

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.



Before rasterization. (a) After rasterization (b)

Fig. 8. U_1 Vicinity

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. 8b 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.

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

Data Generation The datasets used in our experiments are based upon the German city of Oldenburg. The data generator [14] 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). **TODO:** ***update precision of facts with formula from Oldenburg website***

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 / L_{max} for FRIENDLOCATOR and VICINITYLOCATOR respectively, is set to 6, giving a $\epsilon / B(L_{max})$ of 200 units [6]. In the VICINITYLOCATOR implementation, the vicinity region is circular, and the default radius is 500, corresponding to 260 meters.

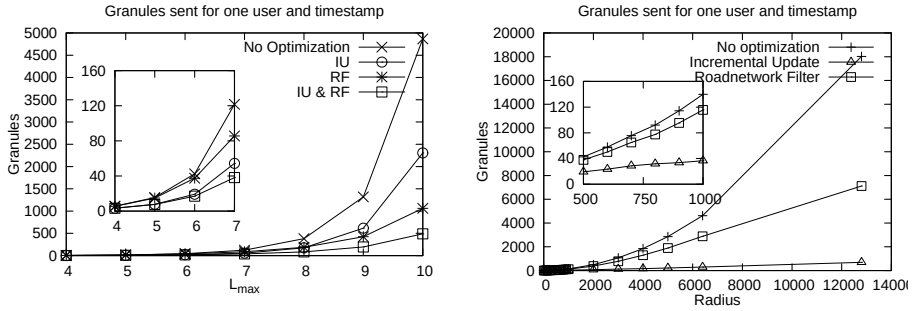
Experiments We will in the following focus mainly on the performance parameter of messages, since it is an important parameter in real world usage since users will have to pay for data when using VICINITYLOCATOR.

In Fig. 9a 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. At level 10 the IU update saves a user 50% of the granules he would have to send, and the RF technique saves a user almost 80%. While the effect of the two optimisation techneques by them self are really good, then it is only when we combine them that we really can eliminate all the duplicated granules sent to the server. When we combine IU and RF we send less than 10% of the granules sent by the unoptimized version of VICINITYLOCATOR.

In Fig. 9b 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. The test is plotted without any optimizations, with incremental update(IU), and with road network filter(RF). When using RF there is a significant reduction which, as expected larger when the radius increase 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.

We want to motivate what an optimal L_0 size might be. To this end we vary L_0 and L_{max}, keeping $B(L_{max})$ constant (See Fig. 10b). Since we vary L_{max}, but keep the size $B(L_{max})$ constant, the λ for both FRIENDLOCATOR and VICINITYLOCATOR will

[6] λ is $\epsilon * (2\sqrt{2} - 1)$ for FRIENDLOCATOR and $B(L_{max}) * \sqrt{2}$ for VICINITYLOCATOR



(a) (b)

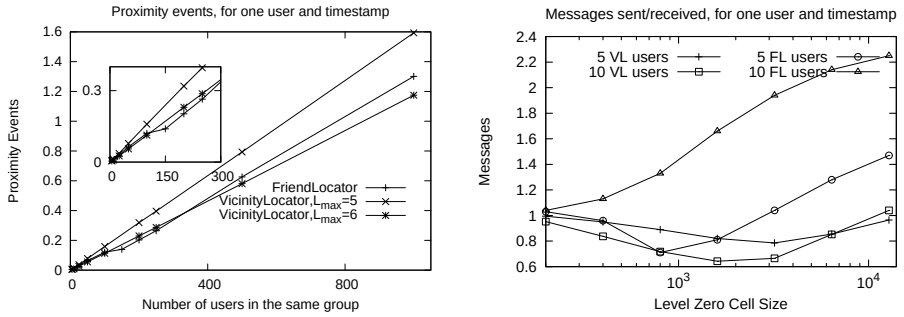
Fig. 9. (a) Effect of increasing L_{max} with and without the Incremental Update(IU) and Roadnetwork Filter(RF) optimizations. (b) The total amount of messages when increasing radius of vicinity.

be constant throughout the tests. We plot the graph for 5 and 10 users in the system, setting all users in one group for both test. This gives the effect that at smaller values of L_0 there will be fewer or no level shifts (but maybe more cell boundary crossings). We compair against FRIENDLOCATOR using ϵ and radius of 200. It is clear from Fig. 10b that the VICINITYLOCATOR performs much better in the amount of messages. The optimal L_0 is the lowest point on each of the four graphs in Fig. 10.

In Fig. 10a 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 [7] 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 compair VICINITYLOCATOR and FRIENDLOCATOR with a L_ϵ/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.

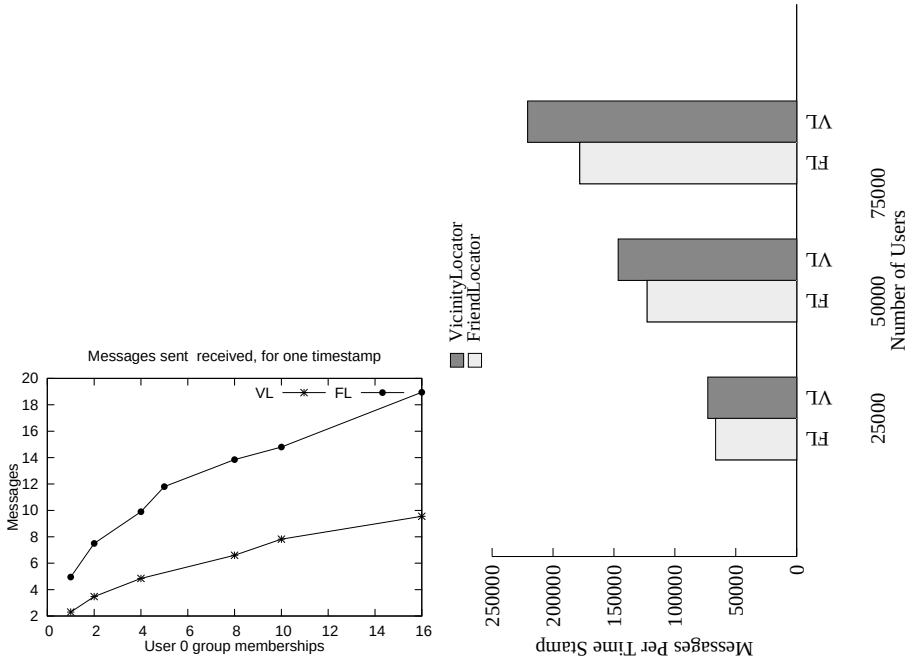
In Fig. 11 we show the effect on messages sent/received by $user_0$ for each time stamp, when we (i) keep the number of friends constant at 80 (ii) increase the number of groups $user_0$'s friends are partitioned into (iii) let $user_0$ be member of all groups. It is clearly seen that VICINITYLOCATOR 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.

[7] VICINITYLOCATOR does proximity detection by vicinity- and center cell overlap, while FRIENDLOCATOR detects proximity by searching overlap between 4 cells of every user.



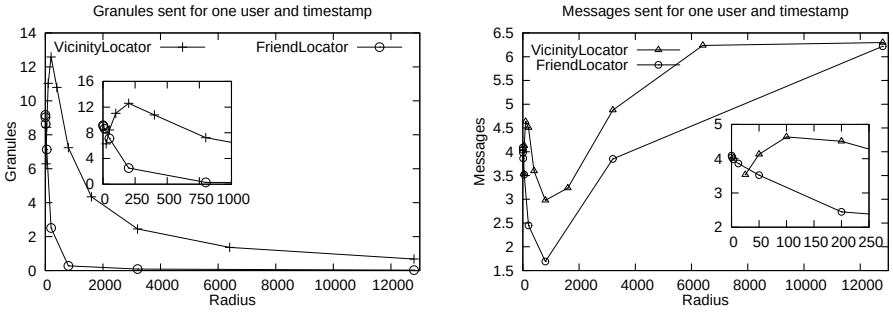
(a) (b)

Fig. 10. (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



(a) (b)

Fig. 11. (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.



(a) (b)

Fig. 12. Simulating the behaviour of FRIENDLOCATOR with VICINITYLOCATOR, the two approaches are compaired on the amount of messages (b) and granules (a) sent to server.

7 Conclusion

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

The client maps its location into a granule and, based on the shape of its vicinity, finds all granules contained in the clients vicinity. The client then encrypts its location- and vicinity- granules and send them to the server which checks for proximity by doing test for inclusion between u_1 location granule and u_2 set vicinity granules. The server does the test blind, without ever knowing anything about user locations.

We look at 3 interesting types of attacks which can be applied to VICINITYLOCATOR, and we show that VICINITYLOCATOR has numerous features, all helping to limit the effect of any attack.

Experimental results show that VICINITYLOCATOR performs very well when compared to FRIENDLOCATOR by continuously incurring lower cost in terms of messages. We show VICINITYLOCATOR is scalable to high number of users.

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 behaviour, giving lower commnication cost for the users.

References

1. Canalys.com: Gps smart phone shipments overtake pnds in emea (November 2008)
2. ABIresearch: Location-based mobile social networking will generate global revenues of $3.3 billion by 2013 (August 2008)
3. Heining, A.: Stalk your friends with google (February 2009)
4. Ruppel, P., Treu, G., Küpper, A., Linnhoff-Popien, C.: Anonymous User Tracking for Location-Based Community Services. In: LoCA. (2006) 116–133
5. Mascetti, S., Bettini, C., Freni, D., Wang, X.S., Jajodia, S.: Privacy-aware proximity based services. In: MDM, IEEE Computer Society (2009)

6. Šikšnys, L., Thomsen, J.R., Šaltenis, S., Yiu, M.L., Andersen, O.: A Location Privacy Aware Friend Locator. In: SSTD. (2009) 0–100
7. Gruteser, M., Grunwald, D.: Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In: USENIX MobiSys. (2003) 31–42
8. Mokbel, M.F., Chow, C.Y., Aref, W.G.: The New Casper: Query Processing for Location Services without Compromising Privacy. In: VLDB. (2006) 763–774
9. Duckham, M., Kulik, L.: A Formal Model of Obfuscation and Negotiation for Location Privacy. In: PERVASIVE. (2005) 152–170
10. Ardagna, C.A., Cremonini, M., Damiani, E., di Vimercati, S.D.C., Samarati, P.: Location Privacy Protection Through Obfuscation-Based Techniques. In: DBSec. (2007) 47–60
11. Khoshgozaran, A., Shahabi, C.: Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In: SSTD. (2007) 239–257
12. Ghinita, G., Kalnis, P., Khoshgozaran, A., Shahabi, C., Tan, K.L.: Private Queries in Location Based Services: Anonymizers are not Necessary. In: SIGMOD. (2008) 121–132
13. Liu, K., Giannella, C., Kargupta, H.: An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining. In: PKDD. (2006)
14. Brinkhoff, T.: A Framework for Generating Network-Based Moving Objects. GeoInformatica **6**(2) (2002) 153–180