Private and Flexible Proximity Detection in Mobile Social Networks

Laurynas Šikšnys* Jeppe Rishede Thomsen* Simonas Šaltenis* Man Lung Yiu†
* *Department of Computer Science*
Aalborg University DK-9220, Denmark
†*Department of Computing*
Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Abstract

A privacy-aware proximity detection service determines if two mobile users are close to each other without requiring them to disclose their exact locations. Existing proposals for such services provide weak privacy, give low accuracy guarantees, incur high communication costs, or lack flexibility in user preferences.

We address these shortcomings with a client-server solution for proximity detection, based on encrypted, multi-level partitions of the spatial domain. Our service notifies a user if any friend users enter the user's specified area of interest, called the vicinity region. This region, in contrast to related work, can be of any shape and can be flexibly changed on the fly. Encryption and blind evaluation on the server ensures strong privacy, while low communication costs are achieved by an adaptive location-update policy. Experimental results show that the flexible functionality of the proposed solution is provided with low communication cost.

1. Introduction

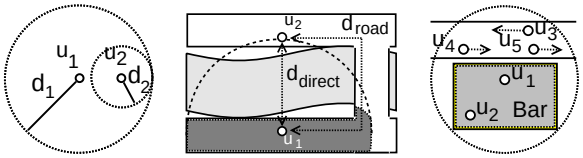
Mobile devices with geo-positioning capabilities are becoming cheaper and more popular [1]. By sharing their location information (e.g., via Wi-Fi, Bluetooth, or GPRS), mobile users can enjoy a variety of location-based services (LBSs). An interesting type of such services is a *friend-locator* service, which shows users their friends' locations on a map and/or helps identify nearby friends. Friend-locators together with other mobile social-networking services are predicted to become a multi-billion dollar industry over the next few years [2]. Several friend-locator services, like *iPoki*, *Google Latitude*, and *Fire Eagle*[1] are already available.

1. ipoki.com; google.com/latitude; fireeagle.yahoo.net

In existing friend-locator services, nearby friends can be notified automatically (as in Location Alerts of Google Latitude) or a user has to check an on-screen map for nearby friends manually. This works only if users agree to share their exact locations or at least obfuscated location regions (e.g., a city center). However, LBS users often require some level of privacy and may even feel threatened [3] if it is not provided. If 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, nearby-friend detection is not always possible in existing friend-locator products.

To address this problem, a number of privacy-aware proximity detection methods were proposed [4]–[7]. They allow two online 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. The proximity message is generated when friends approach each other closer than some predefined distance threshold d, which, in some approaches, can be defined individually for each user. For example, in Figure 1a, user u_2 is in u_1's proximity (defined by a distance threshold d_1), but user u_1 is not in u_2's proximity (defined by a distance threshold d_2).

This type of proximity definition enables proximity detection only in a non-constrained Euclidean space, e.g., a 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 defined as the shortest path distance (which is not always equal to the "crow-fly" distance), then the existing methods are not applicable. For example, if two users are located on different banks of the river (see Figure 1b) such that existing methods classify them being in proximity, the generated proximity notification might not be very useful, as it might be difficult for users to meet each other (shortest path distance



(a) (b) (c)

Figure 1. User vicinities and proximity detection scenarios

$d_{road} >$ direct distance d_{direct}).

Moreover, proximity defined by a distance threshold does not allow users to choose "areas of interest". This could be inconvenient in some cases, e.g., if a user uses the service to find friends in a bar (see Figure 1c), while his friends are traveling along some nearby road with no intention of entering the bar. In this example, user u_1 will get useless proximity notifications from friends u_3, u_4, and u_5. This scenario is likely if the user has many friends and the road is traffic-intensive.

To eliminate the shortcomings of the threshold-based proximity definition, we introduce the concept of *vicinity region*. Two users are said to be in proximity of each other if the vicinity region of one user contains the location of the other user. The vicinity region is an area around the user's location that defines his "scope of interest" and it can be understood as a parameter of a spatial range query over the locations of his friends. In addition, we allow the vicinity region to be changed dynamically. In Figure 1b, user u_1 would select a vicinity region (shown in dark gray) such that the road-network distance between every point of the region and u_1's location is less than some threshold. Similarly in Figure 1c, user u_1 would preselect a vicinity region matching the bar and only friend u_2 would be identified as being in proximity.

In practice, such vicinity regions coud be drawn on maps manually by the users or generated automatically based on the shapes of the nearest spatial objects.

Existing proximity detection methods only support static, circular-shaped, user-location-centered vicinities. The challenge is to develop a proximity detection method that supports both user location privacy and dynamic-shape vicinities.

The contribution of this paper is the design of a client-server location-privacy aware proximity detection service, the VICINITYLOCATOR. The proposed solution is based on both spatial cloaking and encryption, where the central server checks users proximity blindly knowing no spatial data. In contrast to some related work, no peer-to-peer communication between the users is necessary. The users are allowed to specify individually their dynamic vicinities of any shape and a combination of minimum location privacy and service accuracy. To reduce communication cost, a flexible location-update policy requires users to update their location data only when leaving automatically adjustable regions that shrink and expand depending on the distance of a user's closest friend.

The paper is organized as follows. The related work is reviewed in Section 2 followed by our problem setting in Section 3. The VICINITYLOCATOR and experimental results are presented in Sections 4 and 5 respectively.

2. Related work

A lot of research on proximity-detection among mobile users focuses on optimizing the communication and computation costs [8] ignoring the location privacy. In this section we focus only on privacy-related research. First, 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), e.g., gas stations. The goal is then to retrieve from the server the nearest POIs to the user, without revealing the user's 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 [9]–[12] techniques generalize the user's exact location q into a region Q', which is then used for querying the LBS server. The server returns all the results that are relevant to any point in Q'. Such techniques ensure that the user's position is known to the server not more precisely than the area of Q'. Alternatively, if the attacker knows the locations of all users but not the *identities* of the querying users, the identity of the querying user can be inferred only with some probability, if Q' is chosen to include the locations of multiple users.

The transformation approaches [13], [14] 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, which assume public data sets, cannot be directly applied for

proximity detection. However the concepts of spatial cloaking and transformation can be and are used in the existing privacy-aware proximity detection approaches.

2.2. Privacy-aware proximity detection

Ruppel et al. [6] develop a centralized solution that supports proximity detection with a certain level of privacy. It 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 the transformed locations. However, Liu et al. [15] points out that such distance-preserving mapping is not safe and an attacker can easily derive the secret mapping function.

Mascetti et al. present a centralized solution `Longitude` [4]. Here users apply spatial cloaking followed by modular transformation prior to sending their locations to the server. The applied transformation prevents the disclosure of location information, but introduces false proximity detections that do not occur in our proposed approach.

`Hide&Crypt`, presented in [5], is a privacy preserving solution which employs a filter-and-refine paradigm. First, users spatially cloak their locations and send them to the server. Then, the server computes a minimum and maximum distances (Figure 2a) between these cloaking regions. Depending on the specified thresholds and computed distances, the server classifies friends being in, not-in, or possibly-in proximity. The last case requires refinement.

In the peer-to-peer refinement step, first, the two users map their locations and vicinity regions to a spatial subdivision, e.g., a grid. Then, they use a protocol for set-inclusion checking to determine if the mapped location of one user lays inside the mapped vicinity of the other user. In Figure 2b, this corresponds to checking whether the ID of the dark cell (u_2's location) is in a set of the IDs of the gray cells (u_1's vicinity region). Using a *secure two-party computation* (SMC) protocol, this is performed without the users exposing to each other their mapped locations and vicinities. Note that, as in most proximity detection approaches, there is a trade-off between the proximity detection accuracy and the communication cost. The finer the grid used by the users is, the more accurately the proximity is detected, but at the cost of more communication.

Unlike our approach, `Hide&Crypt` does not completely hide locations of the users from the central server, which always knows their cloaked regions. If strong privacy is required, users are forced to perform

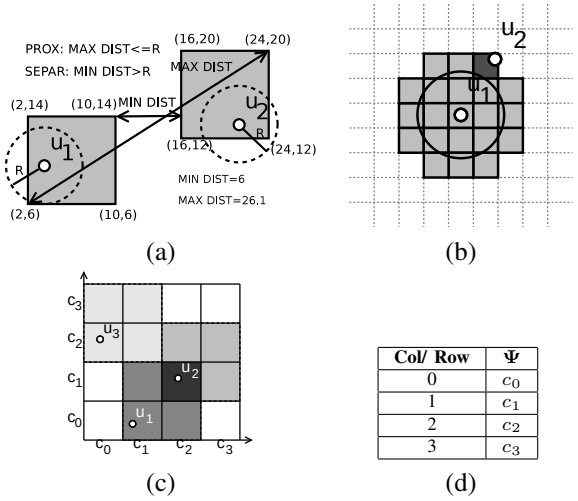


Figure 2. `Hide&Crypt` and FRIENDLOCATOR

user-to-user communication more frequently, thus significantly increasing the amount of client communication due to an expensive SMC protocol.

FRIENDLOCATOR by Šikšnys et al. [7] is a centralized, privacy-aware proximity detection method, which provides strong privacy guarantees and employs a grid-based, adaptive position-update policy to optimize communication cost. Users map their locations into four cells of a grid, and, prior to sending to the server, encrypt them using one-to-one encryption function shared among the users. For example, if the encryption function Ψ is defined as in Figure 2d, the encrypted coordinates of user u_1 in Figure 2c are $(x : [c1, c2], y : [c0, c1])$. By checking for matching encrypted coordinates among the groups of four cells, sent by different users, the server can detect that user u_1 is in proximity with u_2, but u_3 is not in proximity with anyone.

Similar to the approach presented in this paper, FRIENDLOCATOR employs an adaptive position-update policy, in order to reduce the communication cost. The policy works by tracking users in a sparse grid while they are far away from their friends. Only when a potential proximity is detected in a sparse grid, the users are asked to switch to a finer grid in a predefined list of grids.

The limitation of the FRIENDLOCATOR is a rather low, and uncontrollable, accuracy of the proximity detection. On a proximity notification, the actual distance between the two users can be in the range from d to $d + \lambda$, where d is the width/height of a grid cell. Here $\lambda = d(2\sqrt(2) - 1)$ is a service accuracy parameter stemming from the square shape of the grid cells.

Existing privacy-aware approaches use static

circular-shape vicinity so they cannot directly support the "river" and the "bar" proximity detection scenarios (see Figures 1b,c). However some solutions, like `Hide&Crypt` [5], might be adapted to support dynamic vicinities, though current limitations will persist and introduce new problems such as how to efficiently update user locations and vicinities.

The VICINITYLOCATOR presented in this paper combines the ideas of encrypted coordinates, their blind matching at the server-side, and the adaptive update-policy based on multi-level spatial subdivisions (such as a list of grids) to support the flexibly defined vicinity regions. Our solution, unlike Mascetti et al. [5] proposal, employs only centralized architecture, where the server knows no users' location data.

3. Problem definition

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

We assume a setting where a set of mobile users form a social network. Every user uses a mobile device (MD) with positioning and communication capabilities, allowing them to have access to a central location server (LS) and to determine their own locations. In the following, the terms *mobile device*, *user*, and *client* are used interchangeably. The set of all MDs is denoted by $\mathbf{M} \subset \mathbb{N}$. The social network containing the users from $\mathbf{M}$ is defined by the friendship relation $\mathbf{F}$, where $\{(u,v),(v,u)\} \subseteq \mathbf{F}$ if $u, v \in \mathbf{M}$ are friends.

At every time instant, every user $u \in \mathbf{M}$ defines its current $loc(u)$ and $vic(u)$. Here $loc(u) = (loc(u).x, loc(u).y)$ represents u's 2D location in Euclidean space and $vic(u)$ specifies u's dynamic *vicinity region*. The vicinity region is a region around the user's current location. It can have any shape and can consist of multiple, possibly disconnected, parts.

The privacy-aware proximity detection service notifies user $u \in \mathbf{M}$ if any of his friends $v \in \mathbf{M} | (u,v) \in \mathbf{F}$ enters u's vicinity region. More specifically, u's friends are classified to be in proximity or not by checking the 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 not in u's proximity;
3) if $distLV(loc(v), vic(u)) \leq \lambda$, the service can freely choose to classify v as being in u's proximity or not.

Here $distLV(l,v)$ denotes 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 the service accuracy parameter. It introduces a degree of freedom in the detection of location-to-vicinity intersection. Note that small values of λ correspond to higher accuracy. When the classification is complete, u is provided with a set of friends that are now classified as being in proximity while they were not in proximity before. Every user can adjust their desirable service accuracy level λ and the service must minimize communication cost for a given λ setting. Intuitively, more communication is needed for higher accuracy.

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

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

The following section details our proposed proximity detection service that meets these requirements.

4. Privacy-aware proximity detection

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

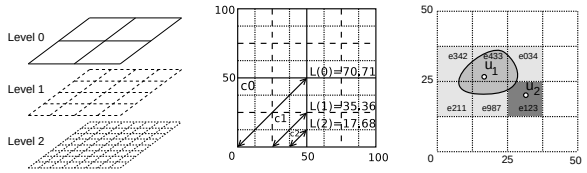
4.1. Proximity detection idea

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

Similarly to FRIENDLOCATOR [7], where all users in $\mathbf{M}$ share a list of grids, we let all users in $\mathbf{M}$ share a *list of granularities*, denoted by Γ. A single granularity $\Gamma(l)$ $(l = 0, 1, 2, ...)$ specifies a subdivision of the spatial domain into a number of non-overlapping regions, called granules [5]. The granularity's index $l \geq 0$ in Γ is termed *the level of granularity*. Every granularity $\Gamma(l)$ satisfies the following three properties:

- Every granule $g \in \Gamma(l)$ is identifiable by an index $id(g) \in \mathbb{N}$.
- 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 the maximum Euclidean distance between any two points in g.
- Granule size bound $L(l)$ at level $l > 0$ is always lower than at level $l-1$, i.e., $L(l) < L(l-1)$.
- Every granule at level $l > 0$ is fully contained by some granule at level $l-1$.

Figure 3a visualizes a valid granularity list, where uniform grids are used as granularities and grid cells are used as granules at levels 0 to 2. The "top-view"



(a) (b) (c)

Figure 3. The valid list of granularities and the behavior of the granularity-based classifier

projection of this list is provided in Figure 3b. Solid, dashed, and dotted lines depict the boundaries of cells at levels 0, 1, and 2 respectively. The figure also shows that maximum distances between any two points of cells $c0$, $c1$, and $c2$ at levels 0, 1, and 2 are respectively 70.71, 35.36, and 17.68. These values correspond to L values of levels 0, 1, and 2.

The list of granularities Γ is globally defined in the system and fixed for all clients. We let all users in $\mathbf{M}$ share an encryption function Ψ. Function $\Psi : \mathbb{N} \mapsto \mathbb{N}$ is a one-to-one mapping that is used to map the index $id(g)$ of some granule g to the 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 a peer-to-peer fashion or with a help of a trusted third-party server.

When Ψ is known by clients, but not by LS, each client can use Ψ to encrypt indices of granules that enclose their location and vicinity at some granularity. Encrypted representation of these indices can be compared on LS without the 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 (see Figure 3c). The user u_2 finds the granule g_l, containing his location, applies Ψ on its index $id(g_l)$ and sends the encrypted representation, $e123$, to LS. Similarly user u_1 finds all the granules intersecting his vicinity region $\mathbf{g_v}$, applies Ψ on their indices and sends their encrypted representations, $\{e342, e433, e034, e211, e987, e123\}$, to LS. If encrypted index of u_2's location granule, $e123$, can be found in the set of encrypted indices of user u_1 vicinity, then we can conclude that user u_2 is in u_1's vicinity. We term such location-vicinity intersection detection approach *granularity-based classifier*. Based on the knowledge about the list of granularities, the following lemma can be proven.

Lemma 4.1: The granularity-based classifier can be used to classify one user as being in another user's proximity or not with the accuracy parameter $\lambda = L(l)$, defined in Section 3.

Proof: According to Section 3, in order for user u_2 to be in u_1's proximity or not in proximity, respective conditions $distLV(loc(u_2), vic(u_1)) \leq \lambda$ and $loc(u_2) \notin vic(u_1)$ must hold. Because Ψ is a one-to-one mapping, if the encrypted index of u_2's location granule g_l is in the set of encrypted indices of u_1's vicinity-intersecting granules $\mathbf{g_v}$, we know that granule g_l is in a set of granules $\mathbf{g_v}$. This in turn means that granule g_l both encloses u_2's location $loc(u_2)$ and intersects with u_1's vicinity $vic(u_1)$. Utilizing properties of granularity at level l, we know that the maximum Euclidean distance between any two points within granule g_l is not more than $L(l)$, i.e., $MaxDist(g_l) \leq L(l)$. Thus, $distLV(loc(u_2), vic(u_1)) \leq L(l)$, $L(l) = \lambda$. Similarly, it can be proven that if g_l cannot be found in $\mathbf{g_v}$, then $loc(u_2) \notin vic(u_1)$. □

According to Lemma 4.1, the λ value depends on the function L and the granularity level l. Note that higher levels provide higher proximity detection accuracy, but the number of vicinity-intersecting granules increases causing higher client-server communication. Thus, we let every user $u \in \mathbf{M}$ select a constant $L_{max}(u)$ that controls the functionality of the system as follows:

- User u will never use granularities of higher than $L_{max}(u)$ levels, thus limiting his worst case communication cost.
- User u lets other users to detect him in a proximity with no higher than $\lambda = L(L_{max}(u))$ accuracy. Thus, $L_{max}(u)$ can be used to specify u's minimum location-privacy requirements with respect to u's peers.
- User u will be able to detect friends being in his proximity with no higher than $\lambda = L(L_{max}(u))$ accuracy.
- Every encrypted coordinate of the user that is sent to LS will have no higher than $L(L_{max}(u))$ resolution, i.e., if an attacker breaks the encrypted coordinate, then deciphered value will correspond to a cloaking region with maximum distance between two points no lower than $L(L_{max}(u))$.

We assume that users can freely select their L_{max} at runtime and upload it to LS.

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 MD and LS. In particular:

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

Table 1. Client and server message types

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 $\langle g_l, \mathbf{g_v} \rangle$, where positions on the stack correspond to levels. g_l is $loc(u)$ granule index at level l. $\mathbf{g_v}$ is a set of indices of granules intersecting $vic(u)$ at level l.
$L_{max}(u)$ - user specified highest granularity level.

1 ***mapLocationToGranularity***(Level number $level$)
2 $\quad g_l \leftarrow id(g) | g \in \Gamma(level) : loc(u) \in g$;
3 $\quad \mathbf{g_v} \leftarrow \{id(g) | \forall g \in \Gamma(level) : g \cap vic(u) \neq \emptyset\}$;
4 $\quad$ **return** $(g_l, \mathbf{g_v})$;
5 ***pushAndSend***()
6 $\quad (g_l, \mathbf{g_v}) \leftarrow mapLocationToGranularity(|GS|)$;
7 $\quad$ Push $\langle g_l, \mathbf{g_v} \rangle$ to stack GS;
8 $\quad$ Send to LS $M_{el}(u, |CS| - 1, \Psi(g_l), \{\Psi(g) | \forall g \in \mathbf{g_v}\})$;
9 ***onLocationChange***()
10 $\quad wasPopped \leftarrow false$
11 $\quad$ **while** $|GS| > 0$ *and* $top(GS) \neq mapLocationToGranularity(|CS| - 1)$ **do**
12 $\quad\quad$ Pop from stack GS;
13 $\quad\quad wasPopped \leftarrow true$;
14 $\quad$ **if** $wasPopped$ *or* $|GS| = 0$ **then**
15 $\quad\quad$ **pushAndSend**()
16 ***onMessageReceived***(M_{LevInc}, Level l)
17 $\quad$ **while** $|GS| \leq l$ *and* $|GS| \leq L_{max}(u)$ **do**
18 $\quad\quad$ **pushAndSend**()

Algorithm 1: MD's event handlers

- **onMessageReceived(**msg**,** arg**)** handler is executed on MD or LS each time a message of type msg with arguments arg is received. A summarizing list of employed messages with their arguments is presented in Table 1.

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

Algorithms 1 and 2 specify the behavior of MD and LS. They contain local data definitions, helper functions, and main event handlers.

A mobile device u remembers its last sensed geographical location $loc(u)$, and, for granularity levels $l = 0..(|GS| - 1)$, stores its location and vicinity mappings (indices of location and vicinity granules) in the stack GS (see Algorithm 1). Once MD changes its location, the **onLocationChange** handler is triggered. If user's location change invalidates current location or vicinity granules at levels $l = (|GS| - 1)..0$, MD pops respective elements from GS, reducing his current level $(|GS| - 1)$. This corresponds to zero or more u's switches from finer to coarser granularities. If the current level is reduced, **pushAndSend** is called, which computes the new location and vicinity mappings for level $|GS|$ (current + one level) and sends them to LS.

Client granularity level shifts are shown in Figures 4a and 4b, where user's u_1 vicinity and user's u_2 current location mapping into a list of granularities (grids) are visualized at consecutive time points. Note, that 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 (Figure 4a and 4b) because his location change invalidates his vicinity mappings at levels 1 and 0. In contrast, u_2's location change causes no changes of location mappings at levels 1 and 0, thus he stays at level 1.

For every user u, LS stores $GL(u)$, which is an encrypted version of the user's 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 the help of M_{el} messages. When LS receives this type of message, handler $\mathbf{onMessageReceived}$ is executed. It updates $GL(u)$ (lines 2–3 in Algorithm 2) and checks if any of u's friends entered his vicinity or if user u entered any of his friends' vicinities. This is checked by searching if encrypted location granule g_l^* can be found in the set of encrypted vicinity granules $\mathbf{g_v^*}$ for some friend v 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 $L_{max}(v)$ (line 6). If g_l^* is found in $\mathbf{g_v^*}$ but l_m is lower than $L_{max}(u)$ and $L_{max}(v)$, the proximity detection accuracy can still be increased, as L_{max} values specified by the users are not yet reached. Thus, LS sends M_{LevInc} to one or both users, asking them to increase their current levels (lines 17–21). Otherwise, if g_l^* is found in $\mathbf{g_v^*}$ and l_m is equal to $L_{max}(u)$ or $L_{max}(v)$, LS sends M_{prox} message, informing the user about a friend in his vicinity (lines 11–16).

To show the working of the algorithms, Figure 4 provides an example, where $L_{max}(u_1) = L_{max}(u_2) = 2$. In Figure 4a, the server finds that g_l^* of user u_2 lays in $\mathbf{g_v^*}$ of user u_1 at level 0 and, because

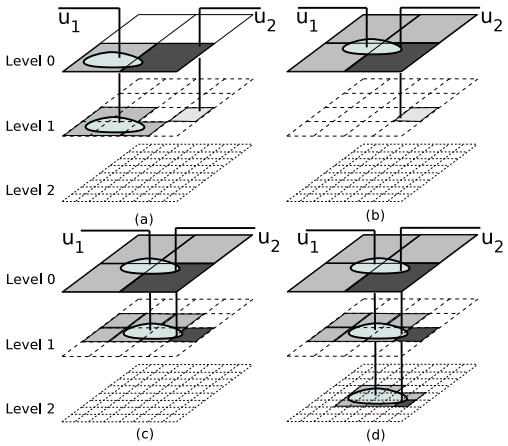


Figure 4. Example of level changes and proximity detection

$0 = l_m$ is lower than $L_{max}(u_1)$ or $L_{max}(u_2)$, it sends M_{LevInc} messages to both users asking them to increase their current levels. When they both deliver level 1 encrypted coordinates, LS no longer finds g_l^* in $\mathbf{g_v^*}$ and nothing happens until one of the users starts moving. When u_1 sends his location data for level 0 in Figure 4b, the l_m is set to 0 and because it is lower than $L_{max}(u_1)$ or $L_{max}(u_2)$ and user u_2 is at level 1 already, only user u_1 is asked to switch to level 1. When this is done and LS finds g_l^* in $\mathbf{g_v^*}$ also at level 1 in Figure 4c, it asks both users to increase their levels as $1 = l_m$ is still lower than $L_{max}(u_1)$ or $L_{max}(u_2)$. Finally, when two users deliver their encrypted location data to LS for level 2 in Figure 4d, the user u_1 is informed about u_2's proximity with message M_{prox}.

Note that, similarly to FRIENDLOCATOR [7], the presented algorithms implement a kind of adaptive region-based location update policy. The clients update their encrypted location data on LS only when location change triggers the change of location or vicinity granularity mappings at their current levels. If a user is far away from his friends, then he or she stays at a low-level granularity with large cells, which results in few location updates as the user moves. Only when the user approaches one of his friends is he asked to switch to higher levels with smaller granules. 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.

Next we review several optimizations that can further 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 represents a social network.
$L_{max}(u) \forall u \in \mathbf{M}$ - highest allowed 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. For level l the g_l^* is an encrypted index of granule containing u's location. The $\mathbf{g_v^*}$ is a set of encrypted indices of granules, that intersect u's vicinity.
$\mathbf{P(u)} \subseteq \mathbf{M}$ - a set of $u \in \mathbf{M}$'s currently proximate friends.

onMessageReceived$(M_{el}, \text{User } u, \text{Level } l, g_l^*, \mathbf{g_v^*})$
  while $|GL(u)| > 0$ *and* $|GL(u)| > l$ **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: LS event handler

4.3. Incremental update optimization

According to the presented algorithms, even if only few of vicinity region granules change due to the user's movement, the user's encrypted data must be updated on LS by sending an M_{el} message. Thus, in most cases, user's two consecutive M_{el} messages would contain duplicated encrypted vicinity region granules.

The communication can be reduced by enabling the so called *incremental updates* (IU). We introduce a new type of message M_{elUpd}. In addition to items u, l, g_l^* from M_{el}, it contains $\mathbf{g^*_{vDel}}$, $\mathbf{g^*_{vIns}}$, where

$\mathbf{g^*_{vDel}}$, $\mathbf{g^*_{vIns}}$ define encrypted granules that must be deleted and inserted on LS in order to update u's set of encrypted vicinity granules 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 a client may choose to 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 $m_3.\mathbf{g^*_{vIns}}$ = $m_2.\mathbf{g^*_v} \setminus m_1.\mathbf{g^*_v}$. Figure 5a visualizes locations, vicinities, and vicinity-intersecting granules of user u at two consequent time points 0 and 1. Darkened cells show $\mathbf{g_{vDel}}$ and $\mathbf{g_{vIns}}$ sets. Note, that introduction of m_3 helps reducing communication only if $|m_3.\mathbf{g^*_{vDel}}|$ + $|m_3.\mathbf{g^*_{vIns}}| < |m_2.\mathbf{g^*_v}|$.

Our presented client and server algorithms (See Algorithm 1, 2) can be easily modified to support incremental updates. As presented in Section 4.2, once a user goes from higher to lower levels (switches into coarser granularity), granules of active level are removed from the stacks GS and GL, without possibility to reuse them. The idea is to preserve these granules on both the client and the server such that it would be possible to update them with incremental updates.

The impact of incremental updates on client communication is evaluated in Section 5.

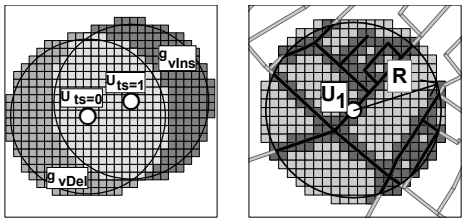
4.4. Road-network filtering of granules

The VICINITYLOCATOR's ability to handle user vicinities of arbitrary shapes can also be used to minimize the amount of granules needed to be sent to the server when users move on a road network. Based on this assumption, we propose a *road network filter* (RF) for circular vicinity regions defined by a Euclidean distance threshold.

When RF is enabled, users use the intersection of a location-centered circle with road segments as their vicinities. An example of such a vicinity mapped into granules is shown in Figure 5b as dark cells. Note that, for simplicity, we use Euclidean distance and do not consider road-network distance based vicinities as shown in Figure 1b.

4.5. Grouping of users

As described in Section 4, all users share a single encryption function Ψ. Security of Ψ directly influences the location privacy of all users in the system. An adversary knowing Ψ can easily decipher encrypted granules of a user's current location and his vicinity. It is difficult to ensure that function Ψ will stay secret in case of a high number of users of the service.



(a) (b)

Figure 5. IU and RF optimizations

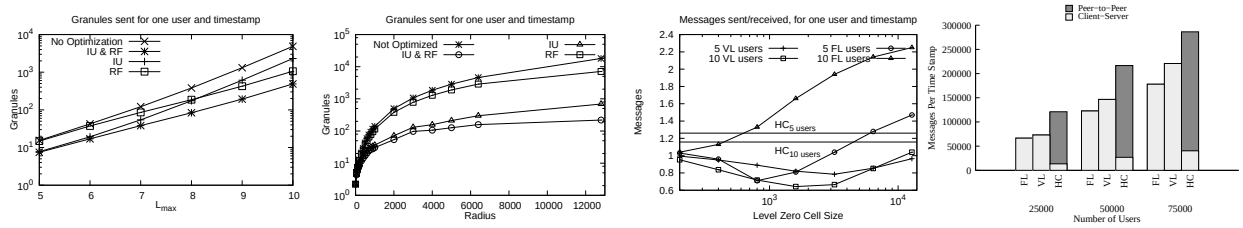
In order to limit the number of affected users in case of a leaked Ψ, *grouping of users* can be enforced. The idea is that all users in the system are grouped into possibly overlapping groups, so that each user is put into one or more groups. Both friends and non-friends can belong to the same group, 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 is compromised. Note also that, even in case of a leaked Ψ, users' minimum privacy requirements, specified by L_{max}, are still guaranteed.

To support user groups, the client and server algorithms should treat each group individually such that clients report their encrypted data for all groups that they are part of and the server independently analyzes encrypted data for every distinct group. We envision user grouping and distribution of the encryption functions as an automated process that is transparent to the users. Design of user grouping strategies is an interesting future research direction.

5. Experimental results

In this section, we present the results of performance experiments, demonstrating the efficiency and real-world applicability of the proposed algorithms. We have implemented a prototype of VICINITYLOCATOR, as well as `Hide&Crypt` [5], and FRIENDLOCATOR [7] for comparison. For simplicity in comparing the three implementations, Γ contains granularities as grids with uniform squares, where edge length $B(l)$ depends on level l. We set $B(l) = L_0 \cdot 2^{-l}$ where L_0 is the cell width at level 0. $L(l) = B(l)\sqrt{2}$ in this case.

Data generation. The generator [16] was used to generate data sets based on the German city of Oldenburg. The data sets cover an area of 26915×23572 $units^2$, corresponding to 14×12.26 km^2, and they contain location records for each user at each time stamp. The duration between two consecutive timestamps is 1



(a) (b) (c) (d)

Figure 6. (a) Effect of increasing L_{max}. (b) Granules sent when increasing radius of vicinity. (c) Messages sent when increasing L_0, keeping $B(L_{max})$ constant. (d) Server messages for one time stamp.

minute and the average speed of the users is 52 km/h (i.e. 1670 units per time stamp).

For road network filter (see Section 4.3), in order to realistically simulate real roads, the road edges are represented as polygons. We choose the width of polygons based on road types.

Experimental setting. `Hide&Crypt`, FRIENDLOCATOR, and VICINITYLOCATOR share a number of settings which we, unless otherwise stated, set to default values: the number of users is set to 50000; the number of timestamps is set to 40 (producing a workload of 2 million location records); users are partitioned into disjoint groups, each group containing 250 users. Within the same group, the friend relationships between users form a complete graph.

Default cell size L_0 is 12800 units and the maximum level allowed by users, denoted by L_ϵ and L_{max} for FRIENDLOCATOR and VICINITYLOCATOR respectively, is set to 6, yielding cell side lengths of 200 units at this level. In the VICINITYLOCATOR, the vicinity region is circular, and the default radius is 500, corresponding to 260 meters.

We consider two cases for `Hide&Crypt`: minimum privacy and total privacy. For min. privacy we set G^{sp} and G^u equal to 200 to mirror the same minimum user location privacy as in VICINITYLOCATOR & FRIENDLOCATOR. For maximum privacy we set $G^{sp} = 80000$ to force `Hide&Crypt` to detect proximity in peer-to-peer mode, revealing no spatial information to the server.

Experiments. We focus mainly on the performance parameter of transmitted messages and granules since they are important parameters in real world usage as users will have to pay for data when using the service.

In Figure 6a, we show the cost of increasing the accuracy of proximity detection. For all users we change L_{max} that consequently sets the accuracy ($\lambda = L(L_{max})$). The experiment was run with no optimizations, with incremental updates (IU), with road network filter (RF), and both the road network filter and the incremental updates (IU & RF). Note the logarithmic scale on the y-axis. At level 10, the IU and RF techniques respectively save a user 50% and 80% of the granules he would have to send with no optimization. When IU and RF are combined, less than 10% of the granules are sent compared to the unoptimized version of VICINITYLOCATOR.

Figure 6b shows the effect the increased radius of the user's vicinity has on the number of granules sent. 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 RF and IU combination performs best, dramatically reducing the granule count. The amount of messages does not change for either of the options.

In the next set of experiments we investigate the setting of an optimal cell width at level 0, the L_0. Note that changing L_0 gives the effect that there will be fewer or no level shifts (but maybe more cell boundary crossings) in order to have the same proximity detection accuracy. To this end we vary L_0 and L_{max}, keeping $B(L_{max}) = 200$ and user's vicinity radius equal to 200 throughout the tests (Figure 6c). 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. We run `Hide&Crypt` prototype with $\delta_A = 200$ to demonstrate what a user would pay in terms of messages, when a complete privacy or VICINITYLOCATOR-equivalent level of minimum privacy is required. The graph shows results of `Hide&Crypt` with minimum privacy only. When maximum privacy is required numbers for 5/10 users are 8.2/14.88 messages. The optimal L_0 for VICINITYLOCATOR or FRIENDLOCATOR is the lowest point on the graphs. The graphs show that, with the right settings, VICINITYLOCATOR is competitive with `Hide&Crypt` in terms of transmitted messages even when the privacy of `Hide&Crypt` is set at the

minimum (the two horizontal lines).

Figure 6d shows the total amount of messages sent by 25000, 50000, and 75000 users for one timestamp. Light bars correspond to client-server messages, and dark bars to peer-to-peer messages of `Hide&Crypt` with minimum privacy. It is clear that `Hide&Crypt`, even with minimum privacy (the communication is much higher when complete privacy is required), incurs a higher amount of messages compared to VICINITYLOCATOR or FRIENDLOCATOR due to expensive peer-to-peer communication. VICINITYLOCATOR incurs reasonable amount of communication considering the flexibility it offers. Also note that VICINITYLOCATOR knows no users' spatial data unlike `Hide&Crypt`, which knows the cloaking regions of all users.

6. Conclusion

In this paper we develop the VICINITYLOCATOR, a client-server solution for detecting proximity by checking for inclusion of one user's location inside another user's vicinity, offering users control over both location privacy and accuracy 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 testing for the inclusion of an encrypted location granule within a set of encrypted vicinity granules of a different user.

Experimental results show that VICINITYLOCATOR is communication-efficient for real world applications and it is scalable to a high number of users. Our presented optimization techniques work very well and, in some cases, cut the amount of transferred data by approximately 90%. VICINITYLOCATOR's features such as irregular shaped vicinities and adjustable accuracy does not introduce significant communication overhead when compared with the alternative approaches.

An interesting future research direction is to explore our approach in a road network, such that the user's vicinity is defined by all paths accessible from his current location within a given threshold of network distance.

References

[1] Canalys.com, "Gps smart phone shipments overtake pnds in emea," November 2008. [Online]. Available: http://www.canalys.com/pr/2008/r2008111.html

[2] ABIresearch, "Location-based mobile social networking will generate global revenues of $3.3 billion by 2013," August 2008. [Online]. Available: http://www.abiresearch.com/abiprdisplay.jsp?pressid=1204

[3] "Stalk your friends with google," 2009. [Online]. Available: http://features.csmonitor.com/innovation/2009/02/04/stalk-your-friends-with-google/

[4] S. Mascetti, C. Bettini, and D. Freni, "Longitude: Centralized privacy-preserving computation of users' proximity." in *Secure Data Management*, 2009, pp. 142–157.

[5] S. Mascetti, C. Bettini, D. Freni, X. S. Wang, and S. Jajodia, "Privacy-aware proximity based services," in *MDM*, 2009, pp. 31–40.

[6] P. Ruppel, G. Treu, A. Küpper, and C. Linnhoff-Popien, "Anonymous User Tracking for Location-Based Community Services," in *LoCA*, 2006, pp. 116–133.

[7] L. Šikšnys, J. R. Thomsen, S. Šaltenis, M. L. Yiu, and O. Andersen, "A Location Privacy Aware Friend Locator," in *SSTD*, 2009, pp. 405–410.

[8] A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler, "Buddy Tracking - Efficient Proximity Detection Among Mobile Friends," in *INFOCOM*, 2004, pp. 298–309.

[9] C. A. Ardagna, M. Cremonini, E. Damiani, S. D. C. di Vimercati, and P. Samarati, "Location Privacy Protection Through Obfuscation-Based Techniques," in *DBSec*, 2007, pp. 47–60.

[10] M. Duckham and L. Kulik, "A Formal Model of Obfuscation and Negotiation for Location Privacy," in *PERVASIVE*, 2005, pp. 152–170.

[11] M. Gruteser and D. Grunwald, "Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking," in *USENIX MobiSys*, 2003, pp. 31–42.

[12] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The New Casper: Query Processing for Location Services without Compromising Privacy," in *VLDB*, 2006, pp. 763–774.

[13] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan, "Private Queries in Location Based Services: Anonymizers are not Necessary," in *SIGMOD*, 2008, pp. 121–132.

[14] A. Khoshgozaran and C. Shahabi, "Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy," in *SSTD*, 2007, pp. 239–257.

[15] K. Liu, C. Giannella, and H. Kargupta, "An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining," in *PKDD*, 2006, pp. 297–308.

[16] T. Brinkhoff, "A Framework for Generating Network-Based Moving Objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.