A Location Privacy Aware Friend Locator

Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen

Department of Computer Science, Aalborg University
DK-9220 Aalborg, Denmark

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

1 Introduction

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

In existing services, the detection of nearby friends is performed manually by the user, e.g., by periodically examining a map on the mobile device. This works only if the user's friends agree to share either exact or obfuscated location. However, LBS users usually demand certain level of privacy and may even feel insecure if it is not provided [5]. Due to the poor support for location privacy in existing friend-locator products, it is sometimes not possible to detect nearby friends if location privacy is desired. The challenge is to design a communication-efficient friend-locator LBS that preserves the user's location privacy and yet enables automatic detection of nearby friends.

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

The rest of the paper is organized as follows. We briefly review related work in Section 2 and then define our problem setting in Section 3. The `FriendLocator`

is presented in Section 4. Section 5 presents experimental results of our proposal and Section 6 concludes the paper.

2 Related Work

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

Location privacy. Most of the existing location privacy solutions employ the *spatial cloaking* technique, which generalizes the user's exact location q into a region Q' used for querying the server [4]. Alternative approaches [6, 11, 3] have also been studied recently. However, all these solutions focus on range/kNN queries and assume that the dataset is public (e.g., shops, cinemas). In contrast, in the proximity detection problem, the users' locations are both queries and data points that must be kept secret.

Proximity detection. Given a set of mobile users and a distance threshold ϵ, the problem of proximity detection is to continuously report all events of mobile users being within the distance ϵ of each other. Most existing solutions (e.g., [1]) focus on optimizing the communication and computation costs, rather than location privacy.

Recent solutions were proposed [10, 8] to address location privacy in proximity detection. Ruppel et al. [10] develop a centralized solution that applies a *distance-preserving mapping* (i.e., a rotation followed by a translation) to convert the user's location q into a transformed location q'. Unfortunately, Liu et al. [7] point out that distance-preserving mapping can be easily attacked. Mascetti et al. [8] employ a server and apply the filter-and-refine paradigm in their secure two-party computation solution. However, it lacks distance guarantees for the proximity events detected by the server, and leads to low accuracy when strong privacy is required. Unlike our approach, the central server in their proposal knows that a user is always located within his or her cloaked region.

Our solution is fundamentally different from the previous solutions [10, 8] because we employ encrypted coordinates to achieve strong privacy and yet the server can blindly detect proximity among the encrypted coordinates.

3 Problem Definition

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

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

The friend-locator LBS notifies two users $u, v \in \mathbf{M} | u \neq v$ if u and v are friends and the proximity between u and v is detected. Given the distance thresholds ϵ and λ, the proximity and separation of two users u and v are defined as follows [1]:

1. If $dist(u, v) \leq \epsilon$, then the users u and v are in proximity;
2. If $dist(u, v) \geq \epsilon + \lambda$, then the users u and v are in separation;

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

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

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

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

4 Proposed Solution

In this section we propose a novel, incremental proximity detection solution based on encrypted grids. It is designed for the client-server architecture, it is efficient in terms of communication, and it satisfies user location-privacy requirements (see Sec. 3).

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

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

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

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

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

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

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

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

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

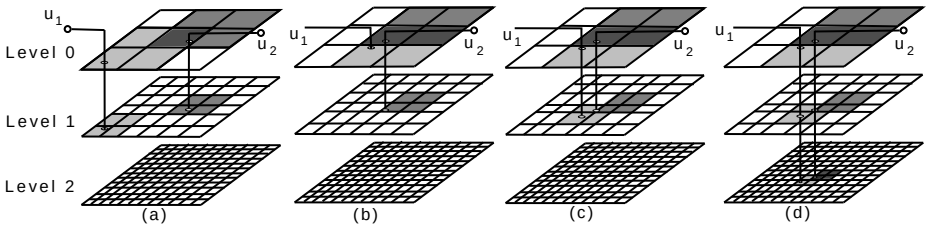


Fig. 1. Two-user proximity detection in the `FriendLocator`

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

5 Experimental Study

The proposed `FriendLocator` and a competitor solution, called `Baseline`, were implemented in C#. In this section, we study their communication cost in terms of messages received by the clients and the server. The network-based generator [2] is used to generate a workload of users moving on the road network of the German city Oldenburg. A location record is generated for each user at each timestamp.

Competitor Solution. The `Baseline` employs the filter-and-refine paradigm for proximity detection among friend pairs. Each user cloaks its location by using a uniform grid, and sends its cell to the server. Filtering is performed at the LS, which calculates the *min* and *max* distances [9] between the cells c_i and c_j of the users u_i and u_j. The LS then checks the following conditions:

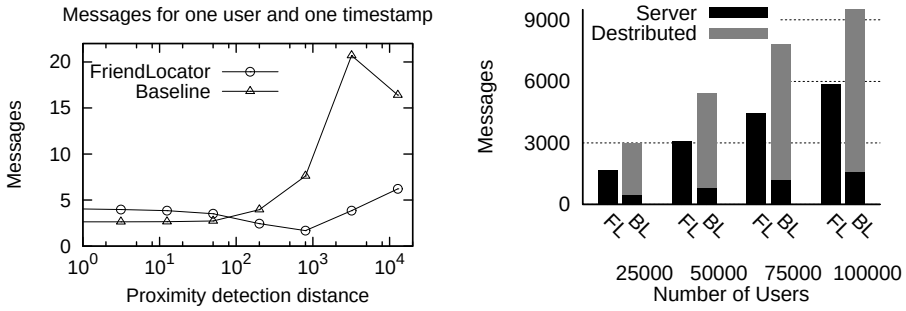
1. If $maxdist(c_i, c_j) \leq \epsilon$, then LS detects a proximity.
2. If $mindist(c_i, c_j) > \epsilon$, then LS detects no proximity.
3. If $mindist(c_i, c_j) \leq \epsilon < maxdist(c_i, c_j)$, then users u_i and u_j invoke the peer-to-peer Strips algorithm [1] for the refinement step.

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

Experiments. We first study the impact of the proximity detection distance ϵ on the cost per user per timestamp (Fig. 2a). Both `Baseline` and `FriendLocator` have similar performance at small ϵ (below 10). As ϵ increases, `Baseline` invokes the refinement step frequently so its cost rises rapidly. At extreme ϵ values (above 10000), most of the pairs are within proximity so the frequency and cost of executing the refinement step in `Baseline` are reduced. Observe that the cost of `FriendLocator` is robust to different values of ϵ, and its cost rises slowly when ϵ increases. Figure 2b shows the total number of messages during 40 timestamps as a function of the total number of users in the system. Clearly, `FriendLocator` incurs substantially lower total cost than `Baseline`. In Fig. 2b the distributed messages represent peer-to-peer messages.

6 Conclusion

In this paper we develop the `FriendLocator`, a client-server solution for detecting proximity among friend pairs while offering them location privacy. The client maps a user's location into a grid cell, converts it into an encrypted tuple, and sends it to the



(a) Client message cost vs. ϵ (b) System message cost for each timestamp

Fig. 2. Effect of various parameters on the communication cost

server. Based on the encrypted tuples received from the users, the server determines the proximity between them blindly, without knowing their actual locations. Experimental results suggest that `FriendLocator` incurs low communication cost and it is scalable to a large number of users.

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

References

1. A. Amir, A. Efrat, J. Myllymaki, L. Palaniappan, and K. Wampler. Buddy Tracking - Efficient Proximity Detection Among Mobile Friends. In *INFOCOM*, 2004.
2. T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.
3. 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.
4. M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *USENIX MobiSys*, 2003.
5. A. Heining. Stalk your friends with google, February 2009.
6. A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, 2007.
7. K. Liu, C. Giannella, and H. Kargupta. An Attacker's View of Distance Preserving Maps for Privacy Preserving Data Mining. In *PKDD*, 2006.
8. S. Mascetti, C. Bettini, D. Freni, X. Wang, and S. Jajodia. Privacy-Aware Proximity Based Services. In *MDM*, 2009, to appear.
9. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest Neighbor Queries. In *SIGMOD*, 1995.
10. P. Ruppel, G. Treu, A. Küpper, and C. Linnhoff-Popien. Anonymous User Tracking for Location-Based Community Services. In *LoCA*, 2006.
11. M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*, 2008.