Wireless Netw
DOI 10.1007/s11276-014-0690-5

TRack others if you can: localized proximity detection for mobile networks

Chi Zhang · Jun Luo

Abstract For a set of mobile users with designated friendship relations, it is a recurring issue to keep track of whether some friends appear in the vicinity of a given user. While both distributed and centralized solutions for proximity detection have been proposed, the cost metrics for evaluating these proposals are always based on counting the *number* of message (e.g., query or update) exchanges. However, as mobile users often rely on wireless networks to maintain their connectivity, the cost incurred by any message passing is strongly affected by the distance between the sender and receiver. In this paper, we propose TRack Others if You can (TROY) as a novel distributed solution for proximity detection. Extending the principle of spatial tessellations, TROY incurs only localized message exchanges and is thus superior to existing proposals in terms of more realistic cost metrics that take into account the *actual energy consumption* of message passing. Moreover, our spatial tessellations inspired analytical framework allows for a meaningful comparison with an existing work. Finally, we use extensive experiments to validate the efficiency of TROY.

Keywords Proximity detection · Mobile networks · Energy efficiency · Spatial tessellation

C. Zhang · J. Luo (✉)
School of Computer Engineering, Nanyang Technological University, Singapore, Singapore
e-mail: junluo@ntu.edu.sg

C. Zhang
e-mail: czhang8@ntu.edu.sg

1 Introduction

With the proliferation of location positioning technologies (e.g., GPS) and mobile communication devices, proximity detection has emerged as an important type of location-based online services. On one hand, geo-social networking applications and social discovery platforms, e.g. Blendr (blendr.com) and Grindr (grindr.com), allow users to browse and possibly meet other users who are nearby to their present locations. Also, a traveling businessman attends a large conference in a city and would like to be alerted and possibly meet other nearby attendee of the conference [1]. In these applications, social groups are formed by enrollment and the tracked users of each user comprise all the other enrolled users in the service. On the other hand, instead of tracking all the users within proximity, some friend-locator services, such as Google Latitude, track only users' friends within proximity, where social groups are formed by friend relationships. Precisely, we are aiming to tackle the following problem in our paper:

Problem description given a set U of mobile users and a social network $G_S(U, E)$ representing either the enrollment or the friendship relations among them, *proximity detection* [1] aims to alert an user u_i if u_j, where $(u_i, u_j) \in E$, enters a circle centered at the location of u_i with a radius R_i. Here R_i is the *proximity threshold* for u_i.

In mobile applications, communication cost is the most important optimization goal, due to the limited bandwidth and battery power on the users' mobile devices [16]. In fact, battery drainage due to communications has becomes a primary issue for the contemporary mobile devices, as communications entail far more energy than computations. Since mobile devices always rely on wireless communications to maintain connectivity among them, the real communication cost (i.e., the incurred *energy consumption*) depends on both the number of message exchanges and the *distance* between the sender to the receiver.

As the proximity detection problem requires the track of all nearby users for every user in a mobile application, it is challenging to design communication-efficient solutions. A naive solution may require that each user reports his location to his friends in a distributed setting, or to a centralized server in a centralized setting, periodically (e.g., per second). However, such a solution would incur a tremendous communication cost.

The importance of the proximity detection applications and the challenging nature of the problem have motivated the development of dedicated methods for it. A few safe region based centralized solutions [20, 24] have been proposed to optimize the communications cost between mobile users and the server, assuming that the locations of moving users can be represented as linear functions of time. However, these centralized solutions have two main drawbacks: First, in many real situations it is difficult to derive a linear function of time to represent the future locations of human. Second, for proximity detection applications that need to handle a huge number of mobile users, these centralized approaches can suffer from dramatic performance degradation in terms of server load and network bandwidth.

Another problem of the existing proposals [1, 20, 24] is that they focus only on reducing the *number* of message exchanges while ignoring the fact that the incurred *energy consumption* of sending a message is strongly correlated with the *distance* from the sender to the receiver [17]. A realistic communication cost metric that considers both number of messages and distance should be used to guide and evaluate the proposed algorithm for proximity detection.

In this paper, we tackle the proximity detection problem in a peer-to-peer mobile network following the setting of [1], i.e., there exists no centralized server in the network. Compared with the centralized setting, the peer-to-peer mode has the following benefits for proximity detection.

- *Flexibility* Unlike its centralized counterpart, a peer-to-peer proximity detection algorithm can be easily implemented by a group of mobile devices without the need for setting up centralized supporting infrastructure.
- *Robustness* As a peer-to-peer solution does not rely on a centralized server that is prone to single point of failure, it can be highly robust and resilient to device failures or user churning.
- *Energy efficiency* While a centralized solution would requires user devices to communicate with a central server, the peer-to-peer setting enables us to design a solution in which users communicate only with nearby users, reducing the average energy consumption per communication.

Our proposed solution, TRack Others if You can (TROY), extends the idea of spatial tessellations [15] to "code" the proximity information. The resulting cell-based distributed data structure enables TROY to fully confine the message exchanges among only the closest users, and it also substantially reduces the data maintenance cost for individual mobile users. All these significantly improve the efficiency of proximity detection.

The challenge in designing TROY resides in the maintenance of cells in a peer-to-peer mobile setting, so that a cell always contains only the user that generates it and the message exchanges for proximity detection can still be confined among only the closest users under mobility. To achieve this, we propose a novel communication-efficient solution to cell maintenance. Under the TROY framework, we first design a basic algorithm for the setting where each user needs to track all the other users in the system (i.e., the social graph G_S as a complete graph). Then we generalize the algorithm for arbitrary social graph G_S.

We also propose an analytical framework to theoretically compare the communication cost of the proposed method and the existing distributed proximity detection method. As the communication distances need to be taken into account, we introduce a cost model based on spatial tessellations. Essentially, we model the mobile users as a spatial point process, and hence the distances between nodes can be theoretically characterized. As a result, we may derive cost metrics to better represent the energy consumption of operating TROY and the competing solution [1].

The contributions of this paper are summarized as follows:

- We propose a novel distributed solution, TROY, for energy efficient proximity detection.
- We develop a novel way of modeling of the communication cost measured by a realistic metric. Our theoretical analysis shows the advantage of TROY over a competing solution [1].
- We perform extensive experiments to compare TROY with existing alternatives, and the experimental results again demonstrate the superiority of TROY over both [1] and [24].

In the rest of the paper, we first briefly survey the literature in Sect. 2. Then we present the basic version of TROY (for complete G_S) and its extension for arbitrary G_S in Sects. 3 and 4, respectively. These are followed by theoretical analysis in Sect. 5 and simulations in Sect. 6 We finally conclude our paper in Sect. 7.

2 Related work and preliminary

We review the existing work on continuous spatial query processing in Sect. 2.1 and proximity detection in Sect. 2.2 Moreover, Sect. 2.2 also serves as the preliminary for our algorithm design and evaluation.

2.1 Continuous spatial query processing

The problem of proximity detection is related to continuous spatial query processing on moving objects, which mainly focuses on the *continuous k nearest neighbor* (*k*-NN) query and *continuous range query*. The *k*-NN query determines the nearest *k* objects to a given query point among all the moving objects. This is a very different type of queries from the proximity query. The continuous range query returns the moving objects that are currently located inside the query region: techniques developed for it could be adapted for the proximity detection problem, although they may not scale to the proximity detection problem where each object corresponds to a *continuous* range query. Note that the query can be either static or moving. We divide continuous spatial query processing methods into centralized methods and distributed methods.

Centralized continuous spatial query processing Many server-side methods have been proposed for continuous spatial query processing. For example, SINA [11], SEA-CNN [22], CPM [13] and the grid-based algorithm [25] for *k*-NN queries and SINA [11] for range queries. In such methods, the server refreshes the result periodically (every a specified time unit) to maintain the results of spatial queries based on the location updates of moving objects. They typically employ grid indexes to index moving objects at the server side for efficient processing of moving queries. While the aforementioned proposals do not make assumption on the future locations of moving objects, another type of methods represents the locations of moving points as functions of time, and predict the query result based on the moving functions. When the motion function of an object changes, it updates to the server and the server recomputes query results associated with the object. Examples for this latter type include [9] for continuous *k*-NN and spatial join queries, and [27] for continuous processing of intersection join between two sets of moving rectangles.

The methods discussed above focus on the computational efficiency issue at the server but disregard the communication cost between users and the server. There also exist centralized algorithms [6, 12] that take into account the communication cost for processing range or *k*-NN queries. They utilize the concept of *safe region*, which is a set of points where the query point can move without changing the query answer. For example, Mouratidis et al. [12] propose an algorithm for optimizing the communication between users and the server for continuous *k*-NN queries. In this method, every object in the answer is assigned a safe region where it can move without making the result change, and only objects that may influence some query result need to communicate their updated locations, thus reducing the number of location updates from the moving objects.

The constraint detection problem [23] is closely related to the proximity detection problem. A specified constraint is satisfied if a set of *n* moving objects are within a circle with a predefined diameter. The proposed solution [23] aims to optimize the computation in the server by an adaptive location constraint indexing approach, which adaptively merges and splits the cells of a grid index. This work does not take communications cost into consideration.

In general, these centralized methods, beside a potential high energy consumption, cannot be used for the proximity detection problem in our setting without centralized servers.

Distributed continuous spatial query processing In contrast with the employment of a single centralized server, several studies (e.g., [8, 21]) introduce a distributed server infrastructure to partition the entire service region into a set of service zones and cooperatively handle requests of continuous range queries.

Furthermore, MQM [4] and MobiEyes [5] assume a distributed environment, where the mobile hosts have the computing capability to process continuous queries. MQM [4] processes continuous static range queries on moving objects while MobiEyes [5] is developed for continuous moving range queries. Both approaches distribute the job of processing queries to mobile nodes of the system, the scalability in terms of server load can be much better than any centralized processing based solution. The setting of MobiEyes [5] goes closer to ours for proximity detection. However, a centralized server is still required to work as a mediator coordinating the query processing: each moving object monitors the queries it can affect and communicate location updates to the centralized server.

As the aforementioned approaches still require centralized servers, they do not apply to the proximity detection problem in our setting.

2.2 Dedicated solutions for proximity detection

Solutions specific to proximity detection can be either centralized [20, 24] or distributed [1]. We briefly explain their ideas, as well as relevant parameters that will be used in analysis and simulations for comparing them with TROY.

Tracking based solutions Aiming to reduce the number of location updates from users to the server, Treu et al. [20] adapt the safe region (also called dead reckoning) mechanism for proximity detection. Assuming that each user updates the server with not only the velocity but also thresholds for both direction and speed, the proposed algorithm [20] assigns ring sector as the safe region where a user may locate since the previous update. This sector is time-variant: it grows larger in time, indicating an uncertainty as an increasing function of time (Fig. 1 right). As a position update is generated only if the user moves out of the safe region, such a mechanism ensures the correctness of proximity detection while reducing the number of updates.

Based again on the safe region approach, Yiu et al. [24] build a simply cost model to analyze the *fixed-radius mobile detection* (FMD, a conventional safe region approach where the region is a disk with fixed radius λ around a user, see the left figure in Fig. 1) in the context of proximity detection. They also propose two self-tuning policies to allow individual users to enlarge or reduce its safe region, named *reactive mobile detection* (RMD) and *cost-based mobile detection* (CMD), respectively.

Irrespective of how much communication cost can be reduced by making location update "smarter", a tracking based proximity detection is bounded to generate redundant updates, simply because tracking is far beyond proximity awareness. We will later show that, under some common scenarios, a tracking-based proximity detection approach continuously generates updates while our proposal incurs zero communication costs.

Tracking-free proximity detection In a mobile peer-to-peer setting, it is not always possible to have a centralized database. As maintaining location information in such a distributed setting is highly complicated, efficient proximity detection needs to get rid of its reliance on location tracking. Motivated by the fact that proximity awareness depends only on mutual distances that contain less information than user locations, the Strips algorithm is proposed by Amir et al. [1] as a distributed proximity detection mechanism. The idea is straightforward: each user maintains a *proximity indicator* for each of its friends. This indicator, as illustrated in Fig. 2, is a strip of width $R + \varepsilon$ centering around the perpendicular bisector of the line segment determined by a friend and the user, assuming a uniform proximity threshold R. The extra width ε is added such that the proximity threshold would not be compromised by the transmission delay when two users happen to be on the boundary of the strip. Once a user enters a strip, an update procedure is triggered such that the user and its corresponding friend exchange messages to verify their mutual distance. If the distance is larger than R, they rebuild the proximity indicator; otherwise a proximity notification is generated.

There are two drawbacks for the Strips algorithm. First, users need to communicate with all of their friends to build and maintain the set of strips, leading to a large cost. Second, the correlations among these strips are not fully utilized. In fact, the strips are *correlated* in the sense that a strip closer to a user implies to some extend farther ones. In other words, if the user does not enter a closer trip, it will not enter farther ones. As the Strips algorithm treats each strip independently, many redundant updates can be generated, resulting in a high communication cost.

3 TROY basic: proximity detection for a complete social graph

In this section, we first explain the rationale behind TROY. Then we introduce the local cell algorithm (LCA) as a basic version of TROY. In order to simplify the basic design, we assume that the social graph is complete (which represents the case that G_S is determined by the enrollment of certain mobile applications, see Sect. 1); a general TROY presented in Sect. 4 will relax this assumption. We first present LCA assuming a uniform proximity threshold R, and then we explain how to extend LCA to handle the case where each user u_i has a distinctive R_i in Sect. 3.3.

3.1 Principles of TROY

We first use an example to demonstrate how the redundantly maintained strips (which we have discussed in Sect.

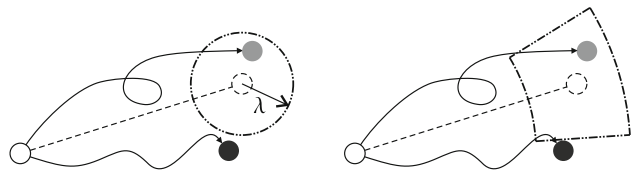

Fig. 1 Convention safe region (*left*) and the ring sector based safe region (*right*). The areas indicated by *dash-dot lines/curves* are the safe regions. *Opened circle* Original location, *filled gray circle* actual location that triggers no update, *open dotted circle* estimated location, *filled block circle* actual location that triggers an update

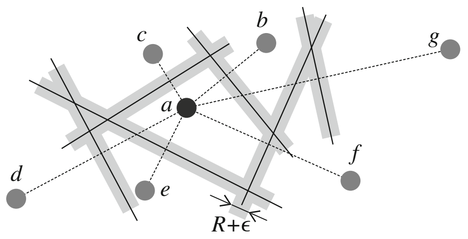

Fig. 2 The strips generated by user a to indicate its proximity with respect to its friends $\{b, c, d, e, f, g\}$

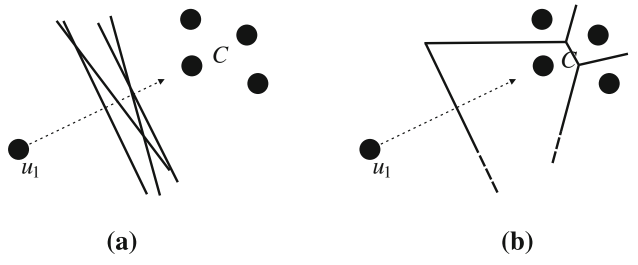

Fig. 3 Whereas the Strips algorithm needs to repetitively update $|C|$ strips (**a**), TROY only updates one strip until u_1 joins C (**b**). To simplify the exposition, we only draw the bisectors though they should be strips with a width $R + \varepsilon$. **a** The strips algorithm, **b** TROY

2.2) lead to unnecessary energy cost and to motivate our cell-based design. Let us consider a user u_1 moves towards a clique of friend C, a scenario shown in Fig. 3. Note that in Fig. 3 we only draw the edges as line segments though they are actually strips with a width $R + \varepsilon$ (see Sect. 2.2). We assume that the distance d from u_1 to C is much larger than the diameter of C's convex hull. As the Strips algorithm maintains a strip for each friend of u_1, it keeps update in total $|C|$ strips when moving, as shown in Fig. 3(a).

In contrast, if we require each user to maintain only the closest strips in TROY, u_1 needs to update only one strip (the one between u_1 and its closest friend) in Fig. 3(b) when u_1 enters the strip. Consequently, the number of message exchanges incurred by Strips is $|C|$ times larger than that by TROY. In fact, TROY takes advantage of the correlation among strips (see Sect. 2.2) and allows the closest friend to "shadow" others. Here, by "shadow", we mean other users in clique C are temporarily hidden from u_1 by the user closest to u_1. Obviously, the users that are shadowed from u_1 are not in proximity with u_1. Note that this mechanism also reduces the average communication distance, as to be analyzed in Sect. 5.

Interestingly, the combination of all the closest strips actually forms a **cell** containing the user itself (see Fig. 4(a) for a better illustration of the cell). In particular, when the strips are initialized to be centered around the bisectors between pairs of users, the resulting cells are Voronoi cells.

3.2 Local cell algorithm (LCA)

We proceed to present the local cell algorithm (LCA), which establishes and maintains a cell for each user in TROY. In its initialization phase, LCA requires each user $u_i \in U$, with location ℓ_i, to execute a localized algorithm to compute the Voronoi cell V_i. This procedure is straightforward: the user simply contacts the close-by users to establish Voronoi edges. The algorithm terminates when the cell is closed and does not contain any other users. We

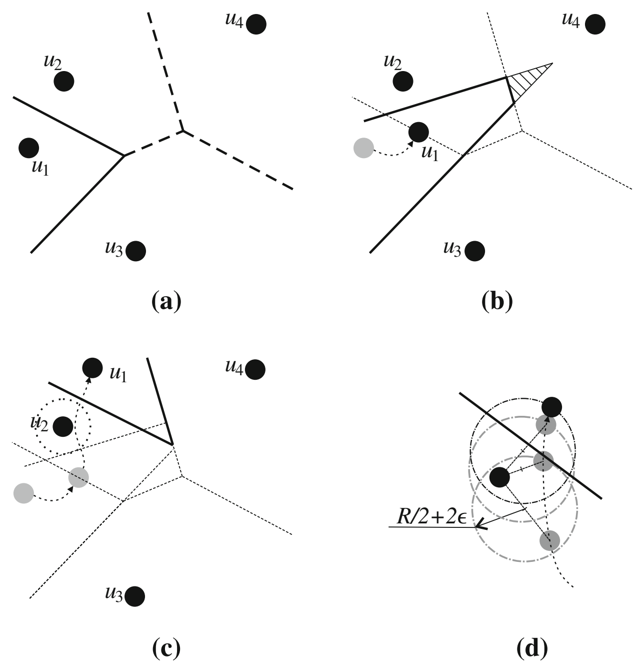

Fig. 4 Illustrating LCV operations. To simplify the exposition, we only draw the edges as line segments though they are actually strips with a width $R + \varepsilon$. **a** Original cells, **b** Cell adjust: edge merging, **c** Cell adjust: edge deleting, **d** Tracking in proximity

refer the readers to [2] for an implementation of this procedure. The outcome, the cell edges and their corresponding user locations, is kept in a local database. For a user u_i, one database entry concerning its adjacent user u_j is as follows:

(user) id	(user) location	edge
u_j	ℓ_j	$(\mathbf{a}_{ij}, b_{ij})$

where $\mathbf{a}_{ij} = \ell_j - \ell_i$ is the normal (vector) of the edge between u_i and u_j, and $b_{ij} = \frac{1}{2}\|\ell_i - \ell_j\|_2 + \mathbf{a}_{ij}^T \ell_i$ gives the initial position of the edge; hence the edge can be represented by $\mathbf{a}_{ij}^T \mathbf{x} = b_{ij}$. There is also an entry for u_i itself, in which the third field is NULL. Note that all the entries jointly form V_i, the cell for u_i. Similar to the Strips algorithm (see Sect. 2.2), we also deem each edge not as a line segment but a strip of width $R + \varepsilon$ centered around the line segment. We hereafter use the terms *edge* and *strip* interchangeably.

Though the cells initially generated are Voronoi cells built using existing algorithms, it is not cost-effective to maintain the Voronoi cells in the presence of user mobilities in our distributed setting.[1] Instead of maintaining

[1] Existing techniques [10, 26] on the dynamic maintenance of Voronoi cells are centralized, and hence do not apply to our case.

Voronoi cells, we propose a communication cost efficient solution to maintain cells for individual users such that for each user, the cell of the user does not contain other users, thus guaranteeing the correctness of proximity detection. The main idea of our proposal is that each user only maintains its own cell; it adjusts the cell boundaries upon hitting some edge(s), aiming at guaranteeing no other users fall into its own cell. Specifically, our *Cell Adjust* algorithm is outlined in Algorithm 1.

Algorithm 1: LCA Cell Adjust

```
Input: User $u_i$ and the local database $u_i.DB$ of $u_i$
Output: Adjusted database $u_i.DB$
1  upon HITEDGE($u_j$)
2  $init \leftarrow$ true
3  SENDMSG($u_i.DB$) to $u_j$
4  upon RECVMSG($u_j.DB$)
5  if $u_j \in u_i.DB.id$ then
6      compute $(\mathbf{a}_{ij}, b_{ij})$ and update $u_j$'s entry in $u_i.DB$
7      forall the $edge \in u_j.DB.edge$ do
8          choose the ones that minimize the area of $V_i$
9          update $u_i.DB$ with the corresponding new entries
10     foreach $entry \in u_i.DB$ do
11         if $entry.edge \notin V_i$ then
12             delete it from $u_i.DB$
13     if $init$ = false then
14         SENDMSG($u_i.DB$) to $u_j$
15     else
16         $init \leftarrow$ false
```

If a user u_i hits the edge separating it from u_j (meaning it enters the strip implied by that edge), a round of message exchange will take place between u_i and u_j, involving a notification from u_i to u_j (lines 1–3) and a reply message from u_j to u_i (lines 13–14). In each message, the whole database maintained by sender is attached (lines 3 and 14).[2] Upon receiving the message, u_i (resp. u_j) updates its own database (hence the maintained cell) by:

a-1 updating the edge (line 6) with respect to u_j (resp. u_i),
a-2 choosing an entry from u_j's (resp. u_i's) database such that the area of V_i (resp. V_j) is minimized by adding the edge into the existing cell boundaries and merge this entry into u_i's (resp. u_j's) database (lines 7–9), and
a-3 deleting edges (lines 10–12) that have been excluded from the current V_i (resp. V_j).

These updates allow both u_i and u_j to get aware of other users (hence the corresponding strips) that were shadowed from them. This is possible as the cell adjacency is *transitive*: user may get aware of other users whose cells are not adjacent through those adjacent friends. It is possible that no entry in u_j's (resp. u_i's) database may reduce the area of V_i (resp. V_j), then in that case step a-2 leads to no change in u_i's (resp. u_j's) database. We illustrate these database updates (with respect to u_1) in Fig. 4(b)–(d) demonstrate what happen after u_1 enters the strip between itself and u_2. In Fig. 4(b), an edge between u_2 and u_4 (originally in u_2's database) is merged with u_1's database (due to a-2). Consequently, the hatched area is excluded from u_1's cell. In Fig. 4(c), the edge between u_1 and u_3 is deleted from u_1's database as it has been excluded from u_1's new cell. When u_1 and u_2 are in proximity (a dotted circle of radius R in Fig. 4(c) roughly marks such a case), a special tracking process is invoked (which will be discussed later), as shown in Fig. 4(d).

Note that a cell may not be a Voronoi cell after even just one adjustment. This enables LCA to have a very low message complexity: one message exchange compared with on average six exchanges if a Voronoi cell were reconstructed upon each adjustment [2]. Nevertheless, this does not affect the correctness of LCA, as shown by the following proposition.

Proposition 1 *For an arbitrary user u_i, LCA maintains V_i so that u_i is the only user located inside V_i.*

Proof The proof is by contradiction. Assume there is a u_k that is also located inside V_i. This is possible only if u_k has passed across one of the edges of V_i. As the edges either belong to u_i or some close-by user u_j that causes the adjust of V_i, either u_i or u_j should have been notified by u_k before. A contradiction.

It is straightforward to see that the worst-case time complexity of the Cell Adjust algorithm is $\mathcal{O}(m)$, where $m \ll |U|$ is the maximum number of entries (edges of a cell) in each user's database. The average-case complexity is only $\mathcal{O}(1)$, because the average number of entries is bounded by a constant [15].

For the cases shown in Fig. 4(b), (c), u_4 is no aware of the cell adjust of u_1. Therefore, in u_4's database, it is u_2 that is adjacent to it. Consequently, if u_4 hits the edge between itself and u_2, it will initiate an message exchange with u_2. Fortunately, as wireless communication is omni-directional, u_1 can "overhear" the exchange and hence reply to u_4 in addition to u_2 if it stays in between u_2 and u_4 (i.e., u_1 has an entry about u_4 in its database). Though overheard messages are usually dropped due to address mismatch in a common wireless network, LCA verifies every overheard message against the cell database. An address matching is identified and a reply message is sent if any database entry matches the source or destination address of a message (line 5). Note that this procedure does not introduce any extra communication cost.

When a user u_i moves into the proximity D_j of user u_j, a special track mechanism is invoked to let them track each

[2] As each user only maintains its own cell, the database has a very small size and can hence be contained in one network packet. Therefore, no extra cost is incurred.

other, until they are again separated further than R. We borrow this local tracking algorithm from [1]: a disk D_{ij} of radius $R/2 + 2\varepsilon$, centered at at the midpoint between the two user locations, is created. If either user leaves the disk, the distance between them is re-measured. If the distance is smaller than $R + 2\varepsilon$, a new disk is created; otherwise a new strip is created. This procedure is shown by Fig. 4(d); it guarantees that, as far as a new strip is not created, two users are still in proximity to each other.

Given that the energy consumption of a wireless transmission grows in power law of the distance between two users, LCA is obviously much more cost-efficient than the existing proposals, as a user needs only to communicate with its adjacent users, while other proposals either need to update to a (possibly) far away server [24] or exchange messages with several far away users [1]. We will give a concrete demonstration of this advantage in Sect. 5.

3.3 Algorithm extensions and optimizations

For LCA described above, we assume that all users have an identical proximity threshold R. In fact, it is rather straightforward to extend the basic LCA to handle user specific proximity thresholds (i.e., a distinct R_i for u_i). The extension works as follows: when two users u_i and u_j exchange messages to build a strip between them, the width of the strip is set as $R_{ij} + \varepsilon$ where $R_{ij} = \max\{R_i, R_j\}$. For proximity tracking, the disk D_{ij} is set to have a radius of $R_{ij}/2 + 2\varepsilon$. It is easy to show that this extension guarantees the correctness of LCA under non-identical proximity thresholds. Of course, this procedure is biased towards the users with larger thresholds, as their friends (with smaller thresholds) are paying more cost than they need to maintain the correctness of the proximity detection system.

Observing the case with u_2 shown in Fig. 4 indicates that LCA may introduce certain unnecessary overhead to a user who is static or rarely moves. When u_1 passes by u_2, the latter's database experience several drastic changes. For example, the strip between u_2 and u_3 and that between u_2 and u_4 are removed due to the "shadowing" of u_1 during some time periods, but they are added back to u_2's database after u_1 moves away. If the algorithm could somewhat predict such "back-and-forth" behaviors, some database operations can be saved.

Here we propose a simple optimization to LCA, aiming at suppressing the unnecessary database operations. We first set a global speed threshold σ such that any user whose speed is below this value is considered as quasi-static. We also add a fourth field $\mathbf{v}_j$ into a database entry concerning u_i: it indicates the velocity of the user u_i at the time the entry is added into a database. Now, if u_j needs to delete an existing edge from its database due to the intervention of u_i (or vice versa), it actually perform the deletion only if $\|\mathbf{v}_i - \mathbf{v}_j\|_2 \leq \sigma$; otherwise the edge remains in the database. The outcome is that u_j's cell may temporarily contain both u_i and u_j. However, such a situation happens only when these edges are "shadowed" by a fast (relative to u_j) moving user u_i, whose effects on V_j can be transient, so deleting these edges is not necessary.

4 TROY reloaded: coping with general social graphs

The LCA algorithm and its extensions involved in the basic TROY framework work on a complete social graph; they apply to social groups formed by enrollment. For applications where the social graph G_S is not complete, LCA could be in trouble. As the cell constructed for each user is meant to "separate" it from its friends, a cell edge, by default, does not exist for two geographically close users that do not have a friendship relation. This means that a user's database may not contain information about other close-by users, which may compromise the algorithm's correctness.

We illustrate this with an example. Consider that two geographically separated user sets contain respectively u_i and u_j, but u_i and u_j are surrounded by their respective friends in each set. Moreover, u_i and u_j are mutual friends, and the link between them in G_S is the only link between these two sets. As a result, u_i and u_j are hidden from each other by their respective friends; they might never get to learn the proximity of each other, simply because LCA relies on the transitivity of cell adjacency (in other words, a user may get aware of other friends whose cells are not adjacent through those adjacent friends) that always holds only for a complete social graph. We illustrate such a case in Fig. 5(a), (b). It is clear from the figures that, given an initial setting shown in (a), the two friends u_1 and u_4 will not be aware of each other later even if they are actually in the proximity (b). In the following, we first describe a naive solution, Underlay LCA (U-LCA), and then present our main proposal, Overlay LCA (O-LCA).

4.1 Underlay LCA (U-LCA): a naive solution

The idea for U-LCA is rather straightforward. Though the social graph G_S is not complete, we may consider an extended complete social graph G'_S that shares the same vertex set with G_S when detecting proximity among users. However, whether a proximity notification is generated still depends on the original graph G_S. By decoupling the proximity detection database from the social relation database, U-LCA allows proximity detection to execute in a non-discriminating manner at a lower protocol layer. More specifically, LCA works at a lower layer, maintaining

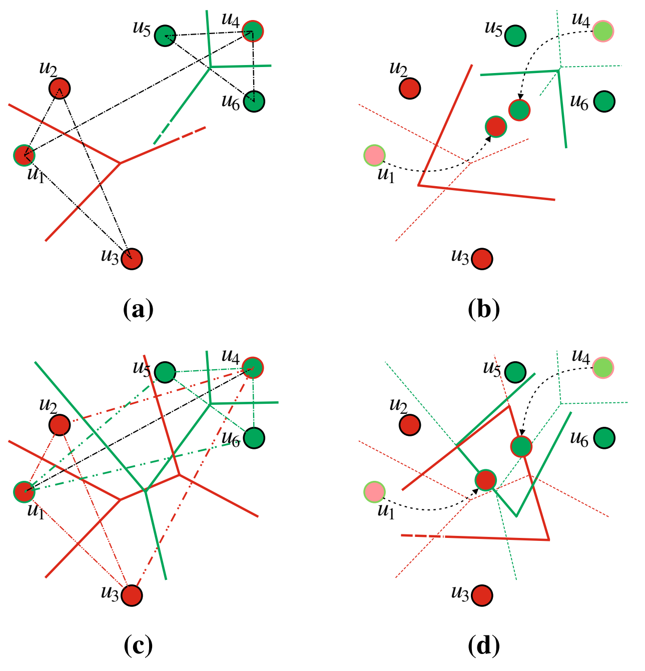

Fig. 5 Whereas the *solid thick lines* represent cell edges (*strips*) and *colors* are used to distinguish two different cliques, the *dash-dot line* represent the edges in E_S and the *dash-double-dot lines* represent the edges the extended subgraphs. **a** Original cells, **b** detection failure, **c** original cells after partition, **d** detection success

a database as if every user is every others' friend. In other words, u_i maintains a cell V'_i with respect to G'_S. In case that a proximity alert is fired due to a user u_i entering a strip separating it from u_j, a database representing G_S at a higher layer of u_i and u_j is checked. An actual proximity notification to u_i or u_j is generated only if there is a match in the database, i.e., $(u_i, u_j) \in E_S$.

Although U-LCA appears to incur some extra maintenance cost as it is based on G'_S, its actually communication (energy) cost may still be lower than the Strips algorithm. The reason is twofold: On one hand, the degree of a real social graph G_S, $\Delta(G_S)$, is often much larger than the number of edges of a cell generated by $\{\ell_l\}_{u_l \in U}$ (6 on average [18]), U-LCA maintains fewer entries for each user. On the other hand, the cost in updating each edge (or strip) is lower for U-LCA due to the shorter distance required for each message exchange.

4.2 Overlay LCA (O-LCA)

While U-LCA globally "complete" G_S to handle the incompleteness in G_S, we may locally complete G_S to further reduce the overhead. As this approach requires only one database (similar to the basic LCA), we term it Overlay-LCA (or O-LCA) to differentiate it from U-LCA.

We hereby define *k-hop friendship* as the relation between two users whose shortest path in G_S has a length k. And we denote by *local k-hop partition* the procedure that, given a user $u_i \in U$, for each friend u_j of u_i, we extract from G_S a subgraph and extend it to a *complete* graph $G_S^{i,j}$, where the vertex set of $G_S^{i,j}$ contain u_i, u_j, and all u_j's l-hop friends ($1 \leq l < k$); the generated graph $G_S^{i,j}$ is called u_i's local k-hop partition with respect to u_j. For example, after a local 2-hop partition for the case in Fig. 5(a) with $i = 1$ and $j = 4$, u_1 joins the user set $\{u_4, u_5, u_6\}$ to form a complete graph $G_S^{1,4}$ with respect to u_4, which is highlighted by green color in Fig. 5(c); conversely, $G_S^{4,1}$ is highlighted by red color for the case with $i = 4$ and $j = 1$. Based on a k-hop partition, O-LCA maintains in every user's database several tables, each corresponding to a cell V_i^j for $G_S^{i,j}$. The basic idea is that u_i uses the cell V_i^j to keep track of the proximity of u_j, rather than a strip used in [1]. For instance, Fig. 5(d) shows that u_1 will not lose track of u_4 by using V_1^4 (green), as the graph $G_S^{1,4}$ is complete. The difference in database entry between O-LCA and LCA is that the value k is stored as an extra field for each entry in O-LCA. When two users are detected as in proximity by O-LCA, a notification is generated only if they have a 1-hop friendship. A redundancy check is applied before a message exchange is initiated, such that only one message is sent if the strip entered by a user belongs to multiple cells.

The following proposition shows the correctness of O-LCA under any local k-hop completion for $k \geq 2$.

Proposition 2 *Let $\{G_S^{i,j}\}$ be the partition and extension of G_S through a local k-hop partition with $k \geq 2$. After user u_i updates its database and hence the cell edges according to O-LCA, no other u_i's friend could possible be located in V_i^j.*

Proof The proof is again by contradiction. Assume $k = 2$ (the weakest case) and some u_j enters V_i^j without notifying u_i. However, V_i^j is maintained by O-LCA on top of $G_S^{i,j}$, a complete subgraph. According to Proposition 1, u_j cannot enter V_i^j without notifying u_i. A contradiction.

The worst-case complexity of performing cell adjusts is $\mathcal{O}(\hat{m}^2)$, where $\hat{m}$ denotes the maximum number of friends a user may have, because $\hat{m}$ tables may be updated with each containing up to $\hat{m}$ entries. Fortunately, the average-case complexity is much lower, which is $\mathcal{O}(\bar{m})$, where $\bar{m}$ denotes the average number of friends a user may have, as the average number of entries is bounded by a constant [15]. We also have the following observations:

- If G_S is a complete graph, then $V_i^j = V_i$, for all u_j such that $(u_i, u_j) \in E_S$. Therefore, O-LCA falls back to the basic LCA.

- If k is set to the diameter of G_S, then local completions all become global; hence O-LCA becomes equivalent to U-LCA.
- If G_S has its *clustering coefficient* (the ratio between the number of closed triplets and that of connected vertex triples [14]) equal to one, O-LCA with $k = 2$ is equivalent to the basic LCA.

As social networks often have a high clustering coefficient [19], the last observation implies that the energy efficiency of O-LCA can be close to that of the basic LCA. The reason is that, though each user u_i needs to maintain several cells, the edges of these cells are very likely to coincide if the clustering coefficient of G_S is large. We will give a qualitative analysis in Sect. 5.3. In practice, we set $k = 2$ and obviously the number of entries of a local database of a user is bounded by the number of 2-hop friends of the user.

5 Analysis

In this section, we analytically compare TROY with the Strips algorithm [1]. As the analytical framework used by the proposal in [24] is significantly different, we can make the comparison with [24] through only experimental study. The metric we use to compare them is the energy consumption of communications (or communication cost) demanded by a proximity detection system. The energy cost spent to maintain databases is ignored as is it negligible compared with the communication cost. We model TROY in terms of the basic LCA in Sects. 5.1 and 5.2, but we discuss the performance of O-LCA using the high clustering coefficient feature of social graphs in Sect. 5.3. The comparisons between TROY and the Strips algorithm are made using two different user distributions: regular distribution and arbitrary distribution.

5.1 Regular user distribution on a grid

We first study the case where users are regularly distributed on a square grid of two rows, as shown in Fig. 6. Let d be the distance between two closest users.

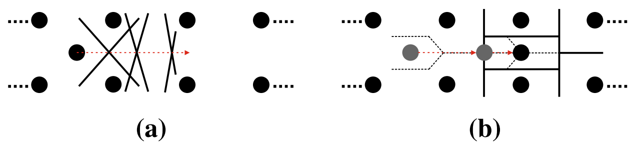

Fig. 6 Strips incurs an increasing number of strip updating per column (**a**), while TROY only updates the two neighboring users (on the forward direction) twice for each column (**b**). **a** The Strips algorithm, **b** TROY

We assume a user, starting from the center of a square, moves horizontally from left to right in Fig. 6. Let c be the number of columns the moving user passes. Figure 6(a) shows that the number of strips being updated grows superlinearly in c, so does the communication distance for each update. Using the result derived in [1] that the number of updates incurred by moving towards another user at distance x is $\Omega(\log x)$, we can obtain that the number of times of updating strips as

$$\sum_{i=1}^{c} \Omega(\log id) = \Omega(c \log c)$$

The total communication cost depends on the function relation between the communication cost and communication distance for one update. According to [17], the communication cost is proportional to the communication distance raised to power η, where $\eta = 2{-}6$ is the *path loss exponent*. In this paper, we take a conservative value $\eta = 2$, which actually favors Strips. Consequently, the total communication costs can be represented as

$$\sum_{i=1}^{c} \Omega(i^2 d^2) = \Omega(c^3 d^2)$$

On the contrary, TROY only updates $4c$ strips after passing c columns, and the communication distance for each update can be two alternating constants $\frac{\sqrt{2}}{2} d$ and $\frac{1}{2} d$ given the special geometry of the grid, as illustrated by Fig. 6(b). We also numerically compute both the number of updates and total communication cost in such a grid setting, and we plot the curves of these two quantities in Fig. 7. We observe that TROY incurs fewer updates and less communication cost than Strips.

5.2 Arbitrary user distribution

We analyze the chance of updating strips of the two algorithms in a more general case, where users are distributed according to a homogeneous Poisson point process in R^2 with intense measure µ. The main difference between TROY and Strip is that, TROY only maintains the Voronoi-like edges, and the Strips takes care of each strip between every user pair, including the Voronoi-like edges. In order to show this difference, we first introduce the *number of disturbed users* (*NDU*) as a metric: suppose at one point of time, only one user u is moving while others being static. *NDU* is the number of users whose strip with u can potentially be updated. Based on the principle of TROY, we can partition the 2D space using cell tessellation generated by all the users. If u takes a random walk within its own cell, we have $NDU_{\text{Strips}} = NDU_{\text{TROY}} = 0$. If the random walk may cover an area A larger than u's cell, $NDU_{\text{TROY}} = \mu|A|$ while $NDU_{\text{Strips}} = \mu|\mathcal{F}(A)|$, where $\mathcal{F}(A)$ denotes the *fundamental domain* associated with A, i.e.

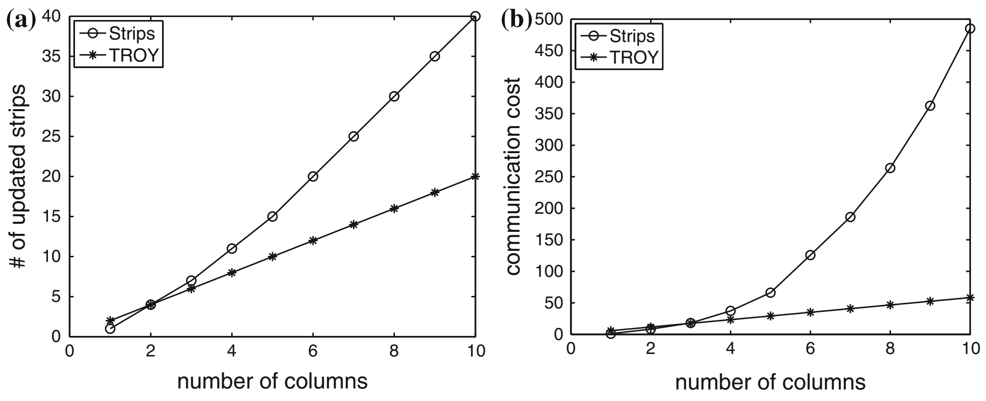

Fig. 7 Numerical comparisons between Strips and TROY. **a** # of updated strips. **b** Total communication cost

$$\mathcal{F}(A) = \bigcup_{\mathbf{x}\in A} D(\mathbf{x}, \|\mathbf{x}\|_2)$$

with the origin set at the current location of u and $D(\mathbf{x}, y)$ denotes a circle centered at $\mathbf{x}$ of radius y. For any convex area A, it can be shown that $|\mathcal{F}(A)|$ is four times of $|A|$ (see Fig. 8), so the $NDU_{\text{Strips}} = 4 \cdot NDU_{\text{TROY}}$.

In terms of the communication cost, the advantage of TROY becomes more conspicuous. it is easy to see that the average distance from u to a user in $\mathcal{F}(A)$ is twice of that in A. Further counting the fact that the communication cost per update is proportional to communication distance raised to power 2, the total communication cost of Strips can be 16 times of that of TROY.

5.3 Inferring the performance of O-LCA

As the performance of O-LCA depends strongly on the actual social graph G_S and the user distribution, it is rather hard to evaluate it quantitatively. Fortunately, according to the observation we made (in Sect. 4.2) in terms of clustering coefficient of G_S, we may qualitatively infer extra overhead brought by O-LCA as compared with the basic LCA.

Assume we take $k = 2$ of O-LCA, a most commonly used O-LCA setting. Denote by γ_S the clustering coefficient of G_S. According to the algorithm description for O-LCA, each user has to maintain cells within a set of subgraphs $\{G_S^{i,j} | (u_i, u_j) \in E_S\}$, which appears to be incurring a large communication cost. However, the cost can still be lower than the Strips algorithm for two reasons. On one hand, γ_S, by its definition, also indicates the percentage of overlap between any two subgraph G_S^{ij1} and G_S^{ij2} in their vertex sets. In the extreme case where $\gamma_S = 1$, all the subgraphs coincide and hence O-LCA degenerates to the basic LCA (as we observed in Sect. 4.2). Therefore, given a reasonable value of the clustering coefficient (according to [19], it may be as high as 0.588 for company director network), the extra overhead in terms of message number introduced by O-LCA can still be comparable to the basic LCA. On the other hand, with relatively small γ_S, the number of messages incurred by O-LCA can sometimes be larger than the Strips algorithm (as we will show in Sect. 6), the communication distance for each message exchange can still be shorter for O-LCA. As a result, O-LCA may well have an advantage in the total communication cost, due to the power law relation between (communication) energy consumption and distance.

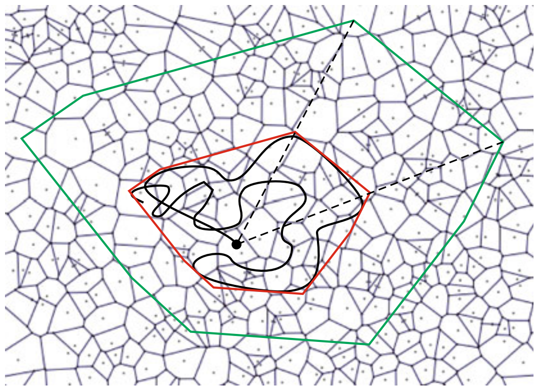

Fig. 8 Comparing strips and TROY with respect to *NDU*. The inner (*red*) polygon represents *A* (which contains the random walk trajectory of *u*, while the outer (*green*) polygon represents $\mathcal{F}(A)$ (Color figure online)

6 Experimental study

In this section, we experimentally evaluate the communication cost of the proposed system TROY, and empirically compare it with Strips [1], the-state-of-art proximity detection algorithm on peer-to-peer mobile networks. We also report the experimental results of the best centralized algorithm, namely reactive mobile detection (RMD; [24]), for reference purpose. Note that RMD works on a centralized setting. We consider the communication energy consumption as the most important metric for proximity detection applications, as elaborated in the introduction, but we also report the total number of messages incurred by each algorithm.

6.1 Experiment setting

Mobility model To simulate user mobility, we use two types of mobility model. The first model is produced by the network-based moving object data generator [3], the parameters we can control are the road model and the maximum moving speed $V_{\max}$. we generate the movement of users on the Oldenburg road network (also used in in [24]). We will call this mobility model Oldenburg. The other model is the Random Waypoint (RW) model [7] commonly used in mobile network simulations. In the RW model, each user node randomly chooses to move or to stay, and a time period is randomly set for either case. If a node decides to move, a velocity is chosen randomly, i.e., a speed is chosen uniformly between a minimum and a maximum values (represented by $V_{\min}$ and $V_{\max}$ and a direction is chosen uniformly in $[0, 2\pi)$ This behavior repeats at the end of each time period. To be consistent with the Oldenburg model, we set $V_{\min} = 0$ RW model. The user movements in both mobility models are constrained in a square region of $1{,}000^2$ meters

User graph To model the relationships among users for the two types of applications discussed in the introduction, we consider two kinds of social relationship model, namely complete graph model and real-life social network model from SNAP (snap.stanford.edu) shown in Table 1. Additionally, we generate a series of artificial social network models using the SNAP graph library for scalability evaluation by setting the average number of friends per user as 100. The complete graph model is used to represent the enrollment-based proximity detection application (such as Blendr and Grindr mentioned in Sect. 1), while the social network model is for applications like Google Latitude. We randomly map the vertex sets of individual user graphs to the movement region.

Setting of RMD RMD is a self-tuning safe region algorithm proposed by Yiu et al. [24]. which work in a centralized setting. It is similar to FMD (see Sect. 2.2), except that the safe region radius λ decreases by dividing $\alpha > 1$ when distance between two users need to be confirmed, and increases by multiplying α when a user gets out of the uers's safe region. RMD periodically measures user locations (and performs proximity detection) every ΔT. In our experiments, we set $\alpha = 2$, $\lambda = 20$ m initially with a range [2, 100] m, and $\Delta T = 1$s. Although another two algorithms, FMD and CMD, are proposed in [24], RMD performs slightly better than CMD and much better than FMD. as it is shown in [24]. Therefore, we only report results of RMD. The location of the server for RMD is set at the center of the square region, which would minimize the average transmission distance between users and the server.

Metric As we mentioned in Sect. 5, we assume the energy consumption for sending a message is proportional to square of the transmission distance (i.e., $\eta = 2$). Again, this is a conservative value that actually favors RMD and Strips, as a real value of η can be up to 6 according to [17]. Without loss of generality and also for ease of exposition, we take the square of transmission distance as the energy consumption in our simulations.

A list of parameters used in our simulations, along with their default values, are also shown in Table 2. We preform simulations for 500 s. Here we deliberately take a much longer simulation period than the 100 s adopted in [24] for the stationarity (hence validity) of our simulations; we refer to "Appendix" for details.

6.2 Experiments on complete graph

We evaluate the communication cost of all methods on a complete social graph G_S, where every pair of users are friends. The results on both mobility models are given in Figs. 9 and 10, respectively.

Figures 9(a) and 10(a) show that TROY outperforms both RMD and Strips by orders of magnitude in terms of the communication cost (note the logarithm scale is used for y-axis in all figures). On one hand, TROY incurs only local (hence short distance) communications, and thus it may significantly reduce the communication cost per transmission compared with the centralized scheme RMD that often requires long distance communications. On the other hand, TROY beats Strips due to the shadowing effect of the cellbased structure, which is consistent with our analysis in Sect. 5.

Figures 9(b) and 10(b) shows the number of messages since the number of messages is used in previous work,

Table 1 Social relationship graph statistics

	Nodes #	Edges #	Clustering coefficient
Wiki-vote	7,115	103,689	0.1255
ca-AstroPh	18,772	396,160	0.318
cit-HepPh	34,546	421,578	0.1457
soc-sign-Slashdot090221	82,168	948,464	0.02411

Table 2 List of experimental parameters

Symbol	Description
N	Number of users in network
$R = 12$ m	Sensitive range for users
$V_{\max} = 10$ m/s	Maximum speed of users

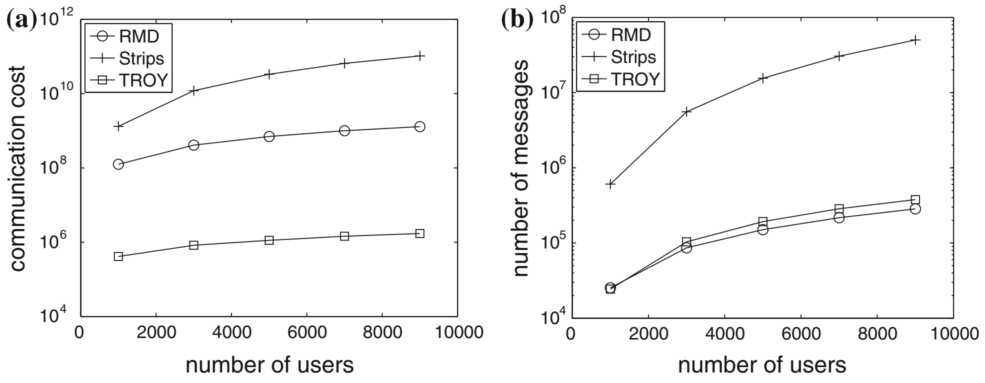

Fig. 9 A complete social graph over the Oldenburg model. **a** Communication cost. **b** Number of message

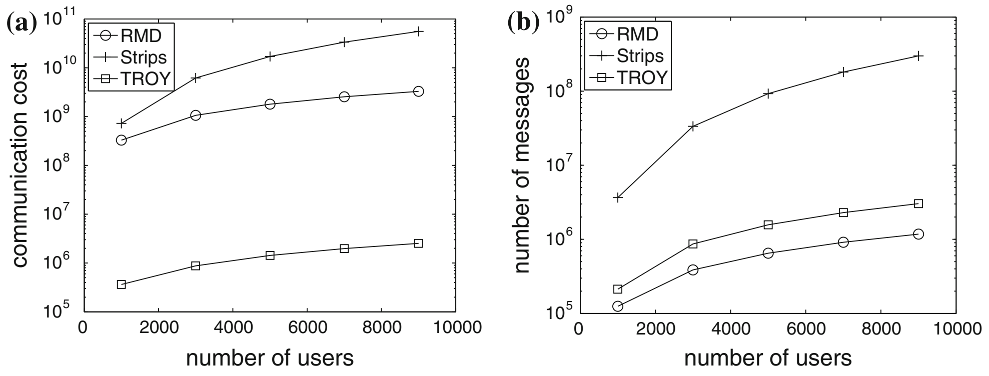

Fig. 10 A complete social graph over the RW model. **a** Communication cost. **b** Number of message

although the metric cannot reflect the communication cost. It is interesting to see that RMD has an advantage over both Strips and TROY in terms of the number of messages, which will be observed in the other experiments as well, although the metric incorporating the distance factor in communication cost used in Figs. 9(a) and 10(a) report contrary findings. *This demonstrates that the number of message is not a good metric to evaluate the system performance*, as it fails to properly indicate the actually incurred energy consumption of a proximity detection scheme.

We also observe from Figs. 9(a) and 10(a) that the communication cost of all the three algorithms grows with the number of users. Moreover, the communication cost of Strips is more sensitive to the number of users than both RMD and TROY. In a complete graph G_S, users are friends of each other, so increasing the network size is equivalent to increasing the number of friends for all users. This naturally leads to the growth of both the communication cost and the number of messages. For Strips, each user needs to maintain a strip between the user and every friend of the user. This means that each user may potentially need to communicate with all friends, making Strips more sensitive to the network size. In contrast, TROY needs only to take care of the closeby friends, while RMD is designed to be relatively independent of the network size, which makes them less sensitive to the network size.

6.3 Experiments on general social graphs

We evaluate the communication cost for the scenarios where the social graphs are determined by the friendship relations among users. Since LCA, the basic version of TROY, does not work in these cases, we compare O-LCA for $k = 2$ with RMD and Strips.

The evaluation results on the four real-life social graphs (shown in Table 1) using the Oldenburg model and the RW model are given in Figs. 11 and 12, respectively. We observe that TROY excels in the communication cost for all the graphs under both mobility models. This is due to TROY's ability of confining message exchanges between close-by users. We also observe that RMD performs relatively worse on the RW model than the Oldenburg model while TROY and Strips are relatively stable. This might be because in RWmodel the users' movement is less predictable and thus a user is more likely to move beyond the user's safe region. If we compare the results on the general social graph and those on complete graph, we find that TROY does pay a price to be extended to support general social graphs: the advantage of TROY over Strips decreases compared with the cases with a complete social graph, due to the extra overhead introduced by O-LCA (compared with the basic LCA) to handle arbitrary social graphs.

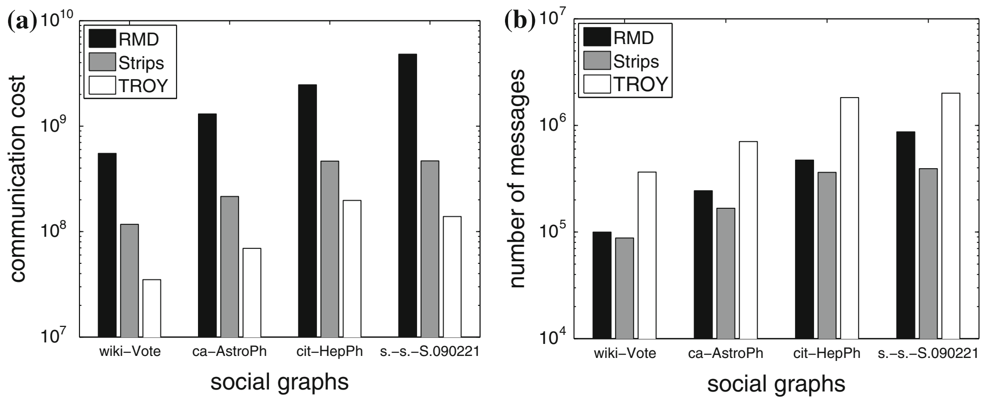

Fig. 11 Real social graphs over the Oldenburg model. **a** Communication cost. **b** Number of message

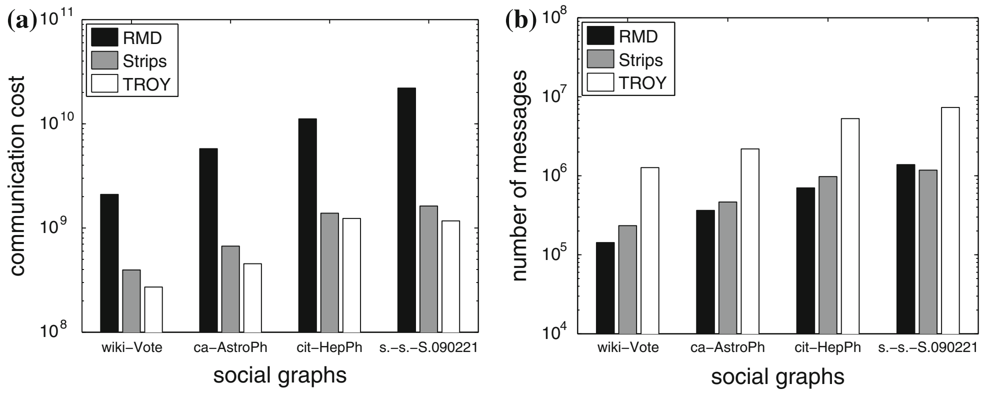

Fig. 12 Real social graphs over the RW model. **a** Communication cost. **b** Number of message

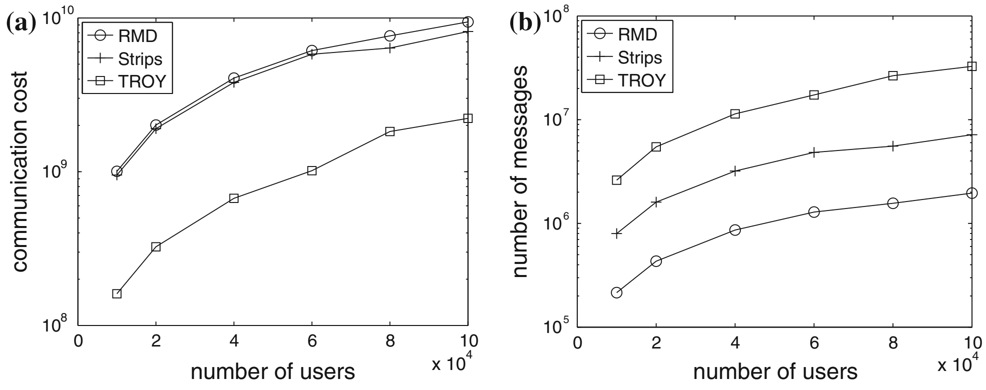

Fig. 13 Artificial social graphs over the Oldenburg model. **a** Communication cost. **b** Number of message

To study the scalability of the different methods with the number of users in the social graph model, we adopt six sets of artificial social graphs generated by the SNAP graph library with node sizes ranging from 10,000 to 100,000. The results are shown in Figs. 13 and 14. It can be observed that TROY exhibits a similar scalability to the other algorithms. In other words, the communication cost of TROY grows at the same pace as the other two algorithms with the increase of network size.

For all the cases, both the communication cost and the number of messages increase with the network size, similar to what we have observed for the complete social graph.

6.4 Influence of sensitive range *R*

In this experiment, we evaluate the effect of the sensitive range *R*, a crucial parameter of a proximity detection service, on the communication cost. Due to the space limit, we choose to present only four sets of results in Figs. 15, 16, 17, and 18, on two types of graph, namely a complete social graph and the Astro-Physics social graph (ca-AstroPh), with the two mobility models.

We observe that the sensitive range R has different impacts on the three algorithms under comparison: TROY has a largely decreasing cost in R and Strips behaves inversely, while RMD is insensitive to R. The insensitivity of RMD to R stems from the fact that RMD adopts essentially a tracking (rather than proximity detection) mechanism and is thus largely independent of R. The increasing cost of Strips in R is mainly due to the "thicker" strips resulting for larger R. Although TROY (O-LCA) is affected similarly by R, the thicker strips have another effect on it. Roughly speaking, the chance of having coincident cell edges (recall O-LCA maintains several cells) gets larger, making O-LCA closer to

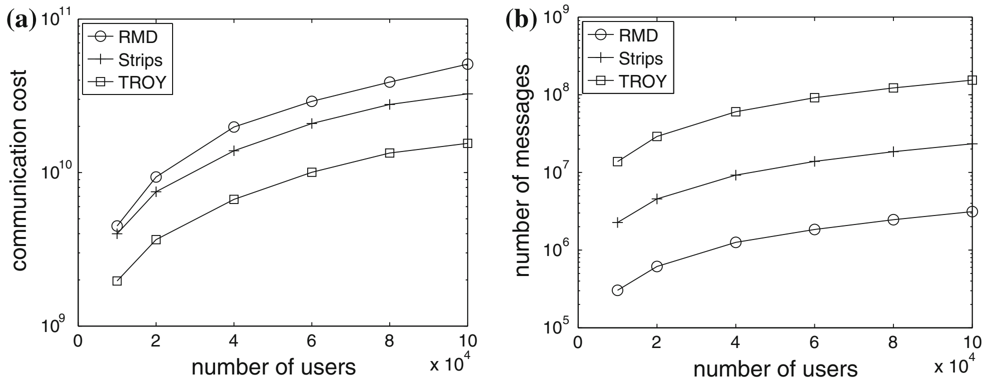

Fig. 14 Artificial social graphs over the RW model. **a** Communication cost. **b** Number of message

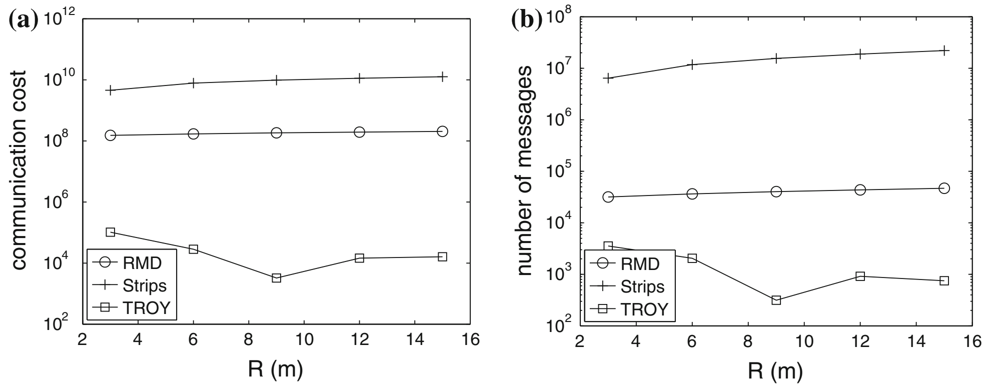

Fig. 15 A complete social graph over the Oldenburg model with a varying sensitive range. We set $N = 10{,}000$. **a** Communication cost. **b** Number of message

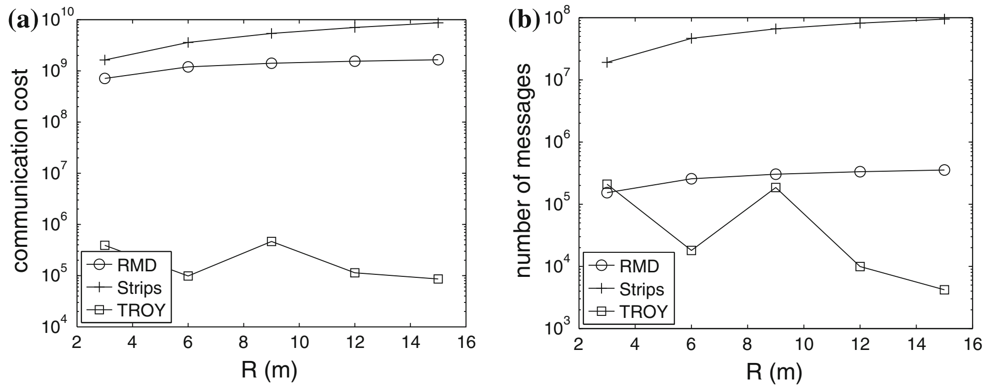

Fig. 16 A complete social graph over the RW model with a varying sensitive range. We set $N = 10{,}000$. **a** Communication cost. **b** Number of message

the basic LCA and hence gaining better performance. The interaction of these two effects leads to the non-monotonic behavior of TROY

6.5 Influence of the maximum speed

In this experiment, we study the influence of the maximum moving speed of users on the communication cost of proximity detection. We choose to report four sets of results, obtained on two types of graph, namely a complete social graph and the Astro-Physics social graph, with the two mobility models (as in Sect. 6.4)

Figures 19, 20, 21 and 22 show the results. We observe that, while both TROY and Strips have increasing cost as the maximum speed increases, the cost of RMD is not sensitive to the speed. Though this seems to suggest that RMD is robust to user mobility, we have to be careful with this interpretation due to the following reasons.

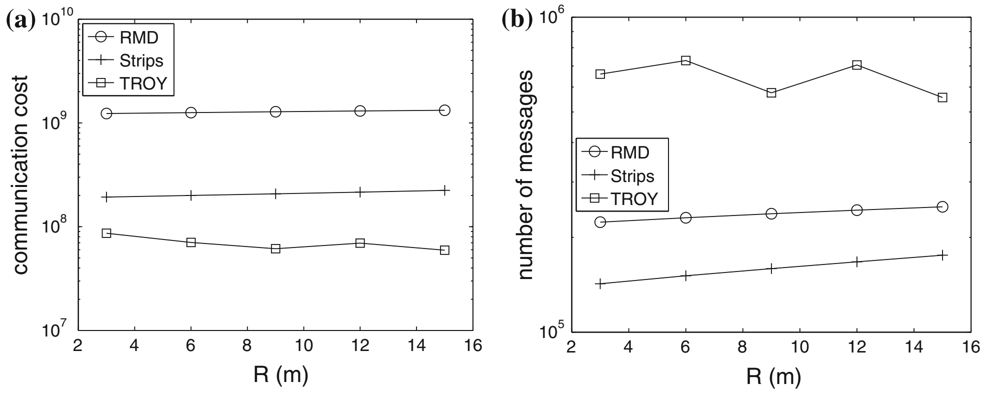

Fig. 17 Astro-physics social graph over the Oldenburg model with a varying sensitive range. **a** Communication cost. **b** Number of message

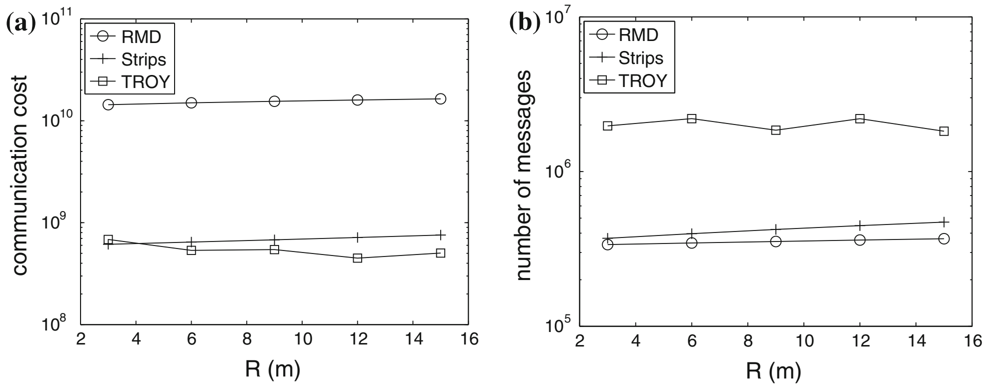

Fig. 18 Astro-Physics social graph over the RW model with a varying sensitive range. **a** Communication cost. **b** Number of message

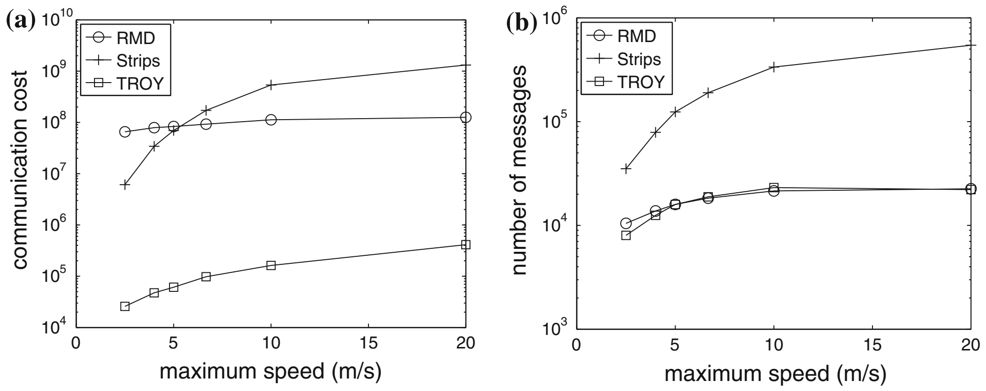

Fig. 19 A complete social graph over the Oldenburg model with a varying maximum speed. **a** Communication cost. **b** Number of message

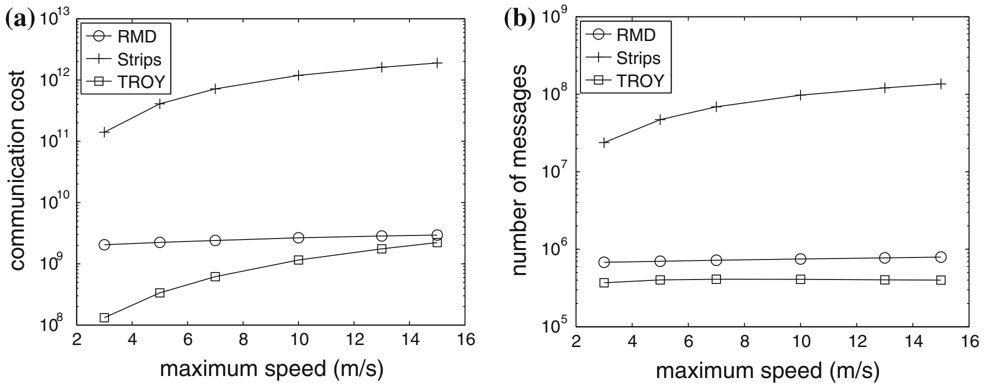

Fig. 20 A complete social graph over the RW model with a varying maximum speed. **a** Communication cost. **b** Number of message

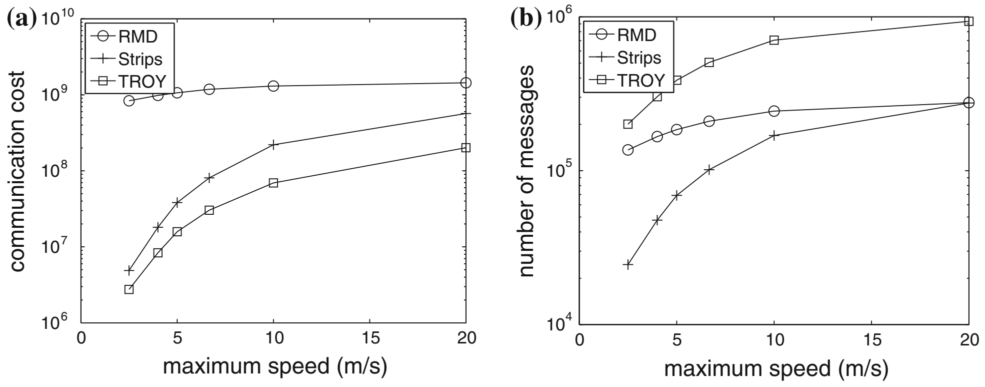

Fig. 21 Astro-physics social graph over the Oldenburg model with a varying maximum speed. **a** Communication cost. **b** Number of message

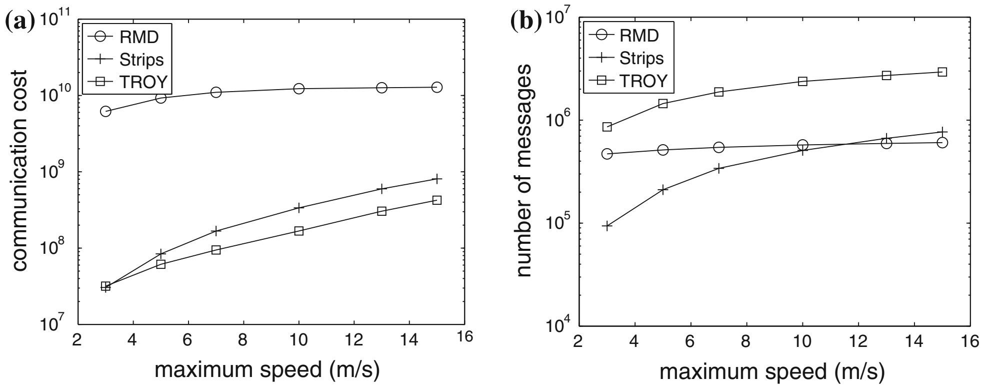

Fig. 22 Astro-physics social graph over the RW model with a varying maximum speed. **a** Communication cost. **b** Number of message

The advantage of RMD (a safe region based approach) lies in its (presumable) ability of predicting the future location of a user. This ability is not affected by the speed, but it becomes mostly prominent when movements always follow lines, which happen to be case for both mobility models that we adopt. Actually, this is the feature for almost all mobility models, as there is virtually no way to emulate the micro-movements of human users. Therefore, the observed advantage of RMD in robustness may most probably be an artifact caused by the mobility model. This artifact exists for all the previous experimental results, but it becomes more conspicuous now due to the increased cost of both Strips and TROY.

7 Conclusion

Proximity detection is an important application in mobile social networks. Aims at reducing the number of messages, the existing proximity detection algorithms are developed based on a problematic assumption that the communication cost is only determined by the number of

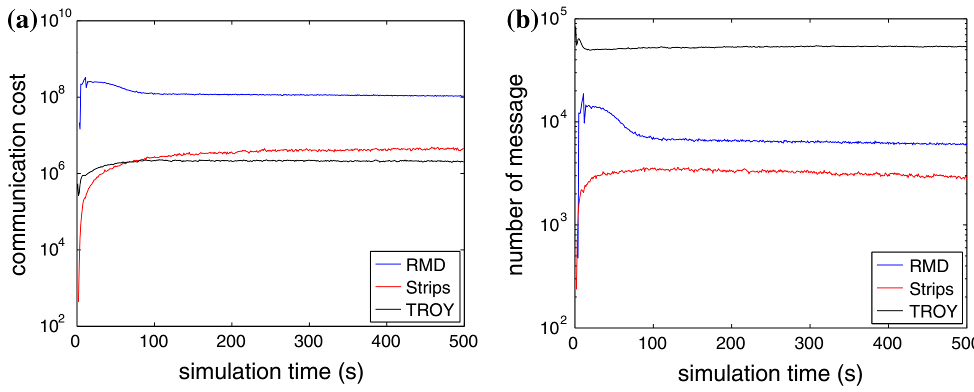

Fig. 23 Communication cost and message number change with time. **a** Communication cost. **b** Number of message

message exchanges. However, it is a well known fact that the communication cost incurred by any message transmission is strongly affected by the distance between the sender and receiver, as mobile users often rely on wireless networks to maintain their connectivity. Motivated by this, we propose TROY as a new proximity detection approach for peer-to-peer mobile networks. TROY maintains cells dedicated for individual users and incurs only localized message exchanges, which makes TROY superior to existing proposals in terms of more realistic communication cost metric that takes into account the distance of message transmission. We propose an analytical framework to compare TROY with Strips, the state-of-the-art distributed proximity detection algorithm. We also conduct extensive experiments on both complete graph (for applications where social groups are formed by enrollment) and general social graphs (for applications where social groups are formed by friend relationships) over two mobility models to study the communication cost of TROY. Experimental results demonstrate the superiority of TROY over Strips, as well as the state-of-the-art centralized algorithm RMD.

Appendix: Simulation stationarity under mobility

As our simulations all rely on some form of randomized mobility model, we need to be careful in choose a simulation period to guarantee the validity of our simulations. Roughly speaking, as there is no guarantee that the initial states of the randomly moving nodes are stationary, the results obtained from a very short simulation period may not be representative or can even be misleading. Here we use the Slashdot09 social graph over the Oldenburg model to illustrate the changes of the two metrics in time. Obviously, it takes about 100 s to make a mobile network enter its steady state. Therefore, the simulation results reported in [24] may well be measured during the transient state of the networks given their 100 s simulation time, and their validity is highly questionable. In our simulation, we set the simulation period to 500 s and only start to measure the cost after the first 100 s (Fig. 23).

References

1. Amir, A., Efrat, A., Myllymaki, J., Palaniappan, L., & Wampler, K. (2004). Buddy tracking—efficient proximity detection among mobile users. In *Proceedings of the 23rd IEEE INFOCOM*.
2. Bash, B., & Desnoyers, P. (2007). Exact distributed Voronoi cell computation in sensor networks. In *Proceedings of the 6th ACM/IEEE IPSN*.
3. Brinkhoff, T., & Str, O. (2002). A framework for generating network-based moving objects. *Geoinformatica, 6*, 202.
4. Cai, Y., Hua, K. A., & Cao, G. (2004). Processing range-monitoring queries on heterogeneous mobile objects. In *Mobile data management, MDM*, pp. 27–38.
5. Gedik, B., & Liu, L. (2006). Mobieyes: A distributed location monitoring service using moving location queries. *IEEE Transactions on Mobile Computing, 5*, 1384–1402.
6. Hu, H., Xu, J., & Lee, D. L. (2005). A generic framework for monitoring continuous spatial queries over moving objects. In *Proceedings of the ACM SIGMOD*, pp. 479–490.
7. Hyytia, E., Lassila, P., & Virtamo, J. (2006). Spatial node distribution of the random waypoint mobility model with applications. *IEEE Transactions on Mobile Computing, 5*(6), 680–694.
8. Ilarri, S., Mena, E., & Illarramendi, A. (2006). Location-dependent queries in mobile contexts: Distributed processing using mobile agents. *IEEE Transactions on Mobile Computing, 5*(8), 1029–1043.
9. Iwerks, G. S., Samet, H., & Smith, K. P. (2006). Maintenance of k-nn and spatial join queries on continuously moving points. *ACM Transactions on Database System, 31*, 485–536.
10. Kolahdouzan, M., & Shahabi, C. (2004). Voronoi-based K nearest neighbor search for spatial network databases. In *Proceedings of the 30th VLDB*.
11. Mokbel, M. F., Xiong, X., & Aref, W. G. (2004). Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceedings of the ACM SIGMOD*, pp. 623–634.
12. Mouratidis, K., Papadias, D., Bakiras, S., & Tao, Y. (2005). A threshold-based algorithm for continuous monitoring of k nearest neighbors. *IEEE Transaction on Knowledge and Data Engineering, 17*, 1451–1464.
13. Mouratidis, K., Papadias, D., & Hadjieleftheriou, M. (2005). Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *Proceedings of the ACM SIGMOD*, pp. 634–645.

14. Newman, M. (2003). Structure and function of complex networks. *SIAM Reviews, 45*(2), 167–256.
15. Okabe, A., Boots, B., Sugihara, K., & Chui, S. (2000). *Spatial tessellations: Concepts and applications of voronoi diagrams*, 2 ed. Chichester: Wiley.
16. Rahmati, A., & Zhong, L. (2007). Context-for-wireless: Context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 7th ACM/USENIX MobiSys*.
17. Rappaport, T. (2002). *Wireless communications: Principles and practice*, 2 ed. Upper Saddle River: Prentice-Hall Inc.
18. Stoyan, D., Kendall, W., & Mecke, J. (1995). *Stochasitc geormetry and its applications*, 2nd ed. Chichester: Wiley.
19. Strogatz, S. (2001). Exploring complex networks. *Nature, 420*, 268–276.
20. Treu, G., Wilder, T., & Küpper, A. (2006). Efficient proximity detection among mobile targets with dead reckoning. In *Proceedings of the 4th ACM MobiWAC*.
21. Wang, H., Zimmermann, R., & shinn Ku, W. (2006). Distributed continuous range query processing on moving objects. In *DEXA*, pp. 655–665.
22. Xiong, X., Mokbel, M. F., & Aref, W. G. (2005). Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proceedings of the 21st international conference on data engineering, ICDE '05*, pp. 643–654.
23. Xu, Z., & Jacobsen, A. (2007). Adaptive location constraint processing. In *Proceedings of the ACM SIGMOD*, pp. 581–592.
24. Yiu, M.-L., Hou, H., Šaltenis, S., & Tzoumas, K. (2010). Efficient proximity detection among mobile users via self-tuning policies. In *Proceedings of the 36th VLDB*.
25. Yu, X., Pu, K. Q., & Koudas, N. (2005). Monitoring k-nearest neighbor queries over moving objects. In *Proceedings of the 21st ICDE*, pp. 631–642.
26. Zhang, J., Zhu, M., Papadias, D., Tao, Y., & Lee, D.-L. (2003). Location-based spatial queries. In *Proceedings of the 30th ACM SIGMOD*.
27. Zhang, R., Lin, D., Ramamohanarao, K., & Bertino, E. (2008). Continuous intersection joins over moving objects. In *Proceedings of the 24th ICDE*, pp. 863–872.

Chi Zhang received his B.S. degree from Zhejiang University, China, in 2011. He is currently a Ph.D. student at the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests are embedded systems, social networks, and wireless sensor networks.

Jun Luo received his B.S. and M.S. degrees in Electrical Engineering from Tsinghua University, China, and the Ph.D. degree in Computer Science from EPFL (Swiss Federal Institute of Technology in Lausanne), Lausanne, Switzerland. From 2006 to 2008, he has worked as a post-doctoral research fellow in the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. In 2008, he joined the faculty of the School of Computer Engineering, Nanyang Technological University in Singapore, where he is currently an assistant professor. His research interests include wireless networking, mobile and pervasive computing, distributed systems, multimedia protocols, network modeling and performance analysis, applied operations research, as well as network security. More information can be found at http://www3.ntu.edu.sg/home/junluo.