Proximity Interactions with Crowdcast

Marios Constantinides, George Constantinou, Andreas Panteli, Theophilos Phokas,
Georgios Chatzimilioudis and Demetrios Zeinalipour-Yazti
{mconst02, gconst02, apante01, tphoka01}@cs.ucy.ac.cy, gchatzim@gmail.com, dzeina@cs.ucy.ac.cy
Dept. of Computer Science, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus

***Abstract*—We demonstrate a suite of proximity-based applications, called *Crowdcast*, that is build on top of our powerful *Proximity* framework. *Proximity*, efficiently connects you to your closest neighbors at all times, regardless of where you are and how far your closest neighbors are. Such a service, realizes our special operator that solves the *Continuous All* k*-Nearest Neighbor (*CAkNN*)* problem efficiently. *Proximity* does not require any additional infrastructure or specialized hardware and its efficiency is mainly achieved due to the smart *search space sharing* technique we devise.**

The *Crowdcast* application suite demonstrates how the *Proximity* data management algorithmics can give rise to novel proximity-based services. During the conference, we will allow attendees to use the *Crowdcast* applications throughout the venue site. They will be able to: (i) post text or vocal messages on a neighborhood pinup wall, which will be visible to their k nearest neighbors. For instance, an attendee might initiate a discussion with other attendees of the same session to clarify issues about the presentation without disturbing; (ii) extend their view or their hearing on the conference activities using the cameras and/or microphones of their neighbors; (iii) post local tasks in their neighborhood as part of organizing an activity, etc.

I. Introduction

We showcase a novel framework that extends the sensing capability of smartphones by allowing them to identify their geographically closest neighboring nodes, at all times, coined *Proximity*[1]. We extend the problem of computing the Nearest Neighbors for every user in the system (*ANN* query) to computing the k Nearest Neighbors for every user *Continuously* (*CAkNN* query).

Applications of the neighborhood "sensing" capability generate unique opportunistic data that can unfold the full potential of crowdsourcing, helping this new problem-solving model to fully penetrated the mobile workforce. Location-dependent crowdsourcing applications can further benefit from adding the temporal dimension to location data in order to exploit trajectory-related information. Similarly, they can benefit from inter-relations between location data, e.g., proximity information. It is essential to optimize and extend location-based search and similarity services.

In addition, classical location-based applications would allow somebody to send out SOS beacons to its geographically closest neighbors when in a life-threatening situation. Such a futuristic application could enhance public emergency services like *E9-1-1*[1]and *NG9-1-1*[2]

[1]Federal Communications Commission - Enhanced 911, Jan 2011, http://www.fcc.gov/pshs/services/911-services/enhanced911/

[2]Department of Transportation: Intelligent Transportation Systems New Generation 911, Jan 2011, http://www.its.dot.gov/NG911/

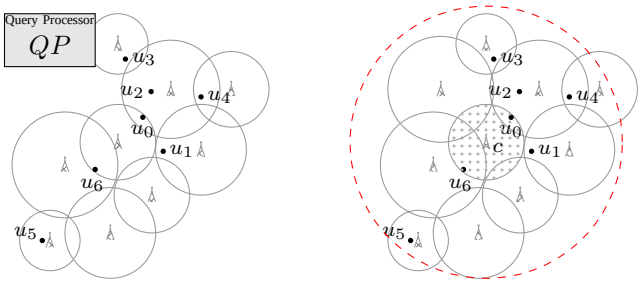


Fig. 1. (a) An instance of a smartphone network, mobile users and the centralized server. The 2 nearest neighbors of each user are: $u_0 \rightarrow \{u_1, u_2\}, u_1 \rightarrow \{u_0, u_2\}, u_2 \rightarrow \{u_3, u_0\}, u_3 \rightarrow \{u_2, u_0\}, u_4 \rightarrow \{u_2, u_3\}, u_6 \rightarrow \{u_0, u_1\}$. (b) The *search space* of the dotted area c is the area inside the dashed circle. Any user inside this search space is a *kNN* candidate for any user inside c.

Consider a set of smartphone users moving in the plane of a geographic region (Figure 1(a)). Let such an area be covered by a set of *Network Connectivity Points* (*NCP*) (e.g., cellular towers found in cellular networks, WiFi access points found in wireless 802.11 networks etc.) Each *NCP* inherently creates the notion of a *cell*[3]. A mobile user u is serviced at any given time point by one *NCP*, but is also aware of the other *NCP*s in its vicinity (e.g., cell-ids of other cell towers, or MAC addresses of WiFi hot-spots in the area).

To illustrate our abstraction, consider the example network shown in Figure 1(a), where we want to provide a micro-blogging channel between each user u and its $k = 2$ nearest neighbors. Each user concurrently requires a different answer-set to a globally executed query, as shown in the caption of the figure. Notice that each *NCP* has its own communication range and that the answer-sets are not limited within the *NCP* of the querying user. Additionally, there might be areas with dense user population and others with sparse user population. Consequently, finding the k nearest neighbors of some arbitrary user u could very likely involve a complex iterative deepening into neighboring *NCP*s. Figure 1(b) shows the search space, constructed by *Proximity*, satisfying the query for all the users inside cell c.

No previous work tackles the problem of continuous all k-nearest neighbor (*CAkNN*) queries, except our recent work in [1] that we aim to demonstrate through this paper. Previous

[3]Without loss of generality, let the cell be represented by a circular area with an arbitrary radius. Using other geometric shapes (e.g., hexagons, Voronoi polygons, grid-rectangles, etc.) for space partitioning would be equally applicable.

work on spatial services includes snapshot retrieval of the k-nearest neighbors (*kNN* and all-*kNN*) [2], [4] and continuous retrieval of k-nearest neighbors for a single user (continuous-*kNN*) [5], [3]. The former techniques require super-linear time for their tree-structure build-up phase and in order to answer our *CAkNN* queries they would need to be updated or re-built in every timestep, which is inefficient. The latter techniques are mostly efficient when users are mildly mobile and in order to answer our *CAkNN* queries they would need to run an instance for every user, which would calculate a new search space for every user.

We utilize a novel algorithm, called *Proximity* [1], to answer *all* k nearest neighbor queries continuously. The *Proximity* algorithm groups users of the same cell and uses the same search space for each group (*search space sharing*). It covers the complete space in a batch process by iterating over all user locations just once, making only a minimal number of comparisons between them. *Proximity* exploits a novel data structure for dividing the search space per *NCP* and enabling search space sharing among the mobile users within each *NCP*. The characteristics of the *Proximity* framework include robustness to high mobility patterns, as it is stateless and has a fast construction time. Furthermore, *Proximity* is robust to skewed distributions of users, as its space division technique depends solely on the distribution and communication range of the *NCP*s.

We start out by presenting the high-level algorithmics behind our *Proximity* framework, we then present our *Crowdcast* suite that realizes this framework and finally present our demonstration plan that will support both interactive scenarios.

II. Internal Algorithms

A. Preliminaries

Assume that there is some centralized (or cloud-like) service, denoted as QP (Query Processor) (see Figure 1(a)), which is accessible by all users in user set U. Allow each user u to report its positional information to QP regularly. These updates have the form *r*=$\{u, loc(u), ncp(u), ncp_{vic}(u)\}$, where $loc(u)$ is the location of user u [4], $ncp(u)$ is the *NCP* user u is registered to and $ncp_{vic}(u)$ is a list of *NCP*s in the vicinity of u.

The problem we consider in this work is how to efficiently compute the k nearest neighbors of all smartphones that are connected to the network, at all times. In order to better illustrate our definition, consider Figure 1(b), where we plot a timestep snapshot of 7 users $u_0 - u_6$ moving in an arbitrary geographic region. The result for this *timestep* to a $k = 2$ query would be *kNN*$(u_0) = \{u_1, u_2\}$, *kNN*$(u_1) = \{u_0, u_2\}$, *kNN*$(u_2) = \{u_3, u_0\}$, *kNN*$(u_3) = \{u_2, u_0\}$, *kNN*$(u_4) = \{u_2, u_0\}$, *kNN*$(u_6) = \{u_0, u_1\}$.

Search space sharing is achieved when the same search space is used by multiple users and it guarantees the correct *kNN* solution for all of them. The common search space S_c for the users U_c inside cell c would be defined as the union of the individual search spaces of every user in U_c. *Proximity* efficiently builds S_c with the assistance of complementary data structures as described in [1]. In Figure 1(b), the search space constructed by our framework for users u_0 and u_6 is the big dashed circle.

[4]The location of a user can be determined either by fine-grain means (e.g., AGPS) or by coarse-grain means (e.g., fingerprint-based geolocation Google Geolocation API (Jan 2011, http://code.google.com/apis/gears/api_geolocation.html).

B. The Proximity Framework

The *Proximity* framework [1] is designed in such a way that it is: i) *Stateless*, in order to cope with transient user populations and high mobility patterns, which complicate the retrieval of the continuous *kNN* answer-set. In particular, we solve the *CAkNN* problem for every timestep separately without using any previous computation or data; ii) *Parameter-free*, in order to be invariant to parameters that are network-specific (such as cell size, capacity, etc.) and specific to the user-distribution, iii) *Fast and scalable*, in order to allow massive deployment of the proposed framework.

For every timestep *Proximity* works in two phases: In the first phase we construct a specialized datastructure, called k^+-heap, per *NCP* using the location information reported from the users. In the second phase, the k nearest neighbors for each user are determined by scanning the respective k^+-heap and the results are reported back to the users. Specifically, at each timestep the server QP initializes our k^+-heap for every *NCP* in the network. The user location reports are gathered and inserted into the k^+-heap of every *NCP*. The k^+-heaps are updated with every insertion to contain only the mathematical *kNN* candidates. After all location reports have been received and inserted, each *NCP* has its search space stored inside its associated k^+-heap. After the build phase, each user scans the k^+-heap of its *NCP* to find its k nearest neighbors.

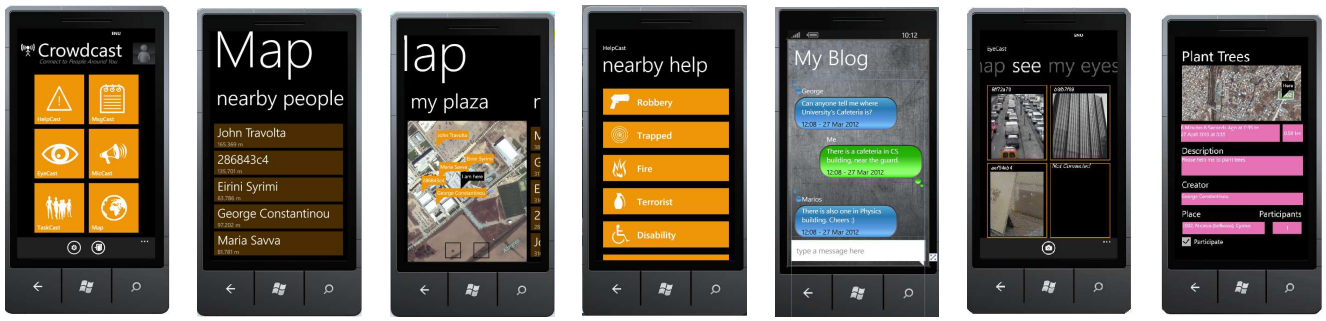
The efficiency of *Proximity* is mainly achieved due to a novel smart search space sharing technique. *Proximity* groups users of the same cell and uses the same search space for each group (*search space sharing*). Note that the search space includes all candidate kNN users that can reside in other near-by or even far-away cells. Using a novel data structure it builds the complete search space in a batch process by iterating over all user locations just once, performing minimal number of comparisons. *Proximity*'s efficiency in search time is independent of k, scales with the number of users in realistic traffic scenarios and outperforms its competitors by at least an order of magnitude.

C. Running Example

We will illustrate a hypothetical execution of our algorithm on the nodes of Figure 1.

Assume that the server QP has initiated a k^+-heap for every *NCP* and receives the user reports $R = r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_x$. Every report is inserted into every k^+-heap. For simplicity we will only follow the operation for the k^+-heap of *NCP* c.

After all reports are inserted into the k^+-heaps, the first phase of the *Proximity* Algorithm is completed and the search spaces are ready. For the second phase, the server scans a



(a) (b) (c) (d) (e) (f) (g)

Fig. 2. Screenshots from **Crowdcast**, an example application implementing the *Proximity* framework. *Proximity* efficiently connects you to your closest neighbors at all times, regardless of where you are and how far they are. (b) Those neighbors can be shown in a list or on a map. On top of functionality a whole suit of applications have been developed: (c) *Helpcast* to send out SOS beacons or disseminate natural disaster warnings, (d) *Msgcast* to post local micro-blogging messages, (e) *Eyecast* to extend the view on the urban environment using the cameras of one's neighbors, (f) *Miccast* to post local vocal messages and warnings, (g) *Taskcast* to post local tasks in your neighborhood as part of local crowdsourcing or organizing a charity event, etc.

single k^+-heap for each user. The server can scan the k^+-heap of any *NCP* that covers a user u to get the k neighbors of u, e.g. the *NCP* that actually services the user $ncp(u)$.

In our example in Figure 1(b), at the end of the build phase the k^+-heap of c includes users $\{u_6, u_0, u_1, u_2, u_3, u_4, u_5\}$. This is the common search space S_c for all users $\{u_0, u_6\}$ of c, which guarantees to include their exact k nearest neighbors.

III. Crowdcast Suite

The *Crowdcast*[5] implementation of the *Proximity* framework is developed in a generic way such that complementary services can be integrated in a seamless manner. *Proximity* efficiently connects you to your closest neighbors at all times, regardless of where you are and how far they are. Those neighbors can be shown in a list or on a map. On top of functionality a whole suit of applications have been developed: (i) *Helpcast*, to send out SOS beacons or disseminate natural disaster warnings; (ii) *Msgcast*, to post local micro-blogging messages; (iii) *Eyecast*, to extend the view on the urban environment using the cameras of one's neighbors; (iv) *Miccast*, to post local vocal messages and warnings; (v) *Taskcast*, to post local tasks in your neighborhood as part of local crowdsourcing or organizing a charity event, etc. The server has the overall picture of the user's whereabouts and can compute the k nearest neighbors for each user.

Apart from our *Crowdcast* implementation, the *Proximity* framework can be used as a core function for many outdoor location-based services. Mobile phones can be instrumental in community sharing and in vitalizing the economies of developing countries. Consider for example the African continent, where the rapid expansion of mobile telecom infrastructureis nowadays providing pervasive public utilities. Services such as M-Pesa[6], a mobile payment infrastructure of Kenyan-origin, is facilitating monetary transactions between individuals and boosting economic growth. Similarly, Jana[7] (formerly txtEagle) is bringing mobile crowd-sourcing services to approximately 2.1 billion people in the African continent (e.g., people fill out questionnaires that relate to blood-banks in hospitals, translate documents, etc. in exchange for mobile airtime).

[5]Crowdcast, Available at: http://www.zegathem.com/
[6]M-Pesa. Available at: http://www.safaricom.co.ke/index.php?id=250
[7]Jana. Available at: http://jana.com/

IV. Demo Plan

During the demonstration participants will be able to log in to the *Crowdcast* suite by creating their own user account or using preset accounts. Participants will be able to use a set of smartphones, which we will hand them during the demonstration, or use their own Windows smartphones to install and connect to the *Crowdcast* suite. The users will be asked to move around the venue and they will be able to see how their k nearest neighbors change as they move, using the neighborhood map. They users will be able to choose the values for k and see how the *Proximity* framework is able to identify the k nearest neighbors quickly and efficiently.

Further, the users will be asked to use any of the *Crowdcast* applications to interact with their neighbors. This might inspire the participants and open the way for discussion regarding future uses of the *Proximity* framework and the integration of vital applications into the *Crowdcast* suite.

Our goals in this scenario are twofold; give the participants a better understanding of how efficiency is achieved through search space sharing in our *Proximity* algorithm and illustrate how to this framework can be used in various real-world applications.

Acknowledgements: This work was supported by the fourth author's Startup Grant funded by University of Cyprus, MTN Cyprus and Bionic Cyprus.

References

[1] G. Chatzimilioudis, D. Zeinalipour-Yazti, W.-C. Lee, and D. M. Dikaikos. Continuous all k-nearest neighbor querying in smartphone networks. MDM '12.
[2] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:226–232, 1983.
[3] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. SIGMOD '05.
[4] P. M. Vaidya. An O(n log n) algorithm for the all nearest neighbors problem. *Discrete Computational Geometry*, 4:101–115, 1989.
[5] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. ICDE '05.