A Cloudlet-Based Proximal Discovery Service for Machine-to-Machine Applications

Jonas Michel, Christine Julien,

The Center for Advanced Research in Software Engineering
Department of Electrical and Computer Engineering The University of Texas at Austin

TR-ARiSE-2013-002

ARiSE
Advanced Research in Software Engineering

A Cloudlet-Based Proximal Discovery Service for Machine-to-Machine Applications *

Jonas Michel and Christine Julien

Department of Electrical & Computer Engineering
The University of Texas at Austin
Austin, Texas USA
{jonasrmichel,c.julien}@mail.utexas.edu

Abstract. Many of today's applications attempt to connect mobile users with resources available in their immediate surroundings. Existing approaches for discovering available resources are either *centralized*, providing a single point of lookup somewhere in the cloud or *ad hoc*, requiring mobile devices to directly connect to other nearby devices. In this paper, we explore an approach based on *cloudlets*, marrying these two approaches to reflect both the proximity requirements of the applications and the dynamic nature of the resources. We present the design and implementation of a cloudlet-based proximal discovery service, solving key technical challenges along the way. We then use real world data traces to demonstrate, evaluate, and benchmark our service and compare it to a completely centralized approach. We find that, in supporting highly localized queries, our service outperforms the centralized approach without significantly affecting the quality of the discovery results.

Key words: location-based discovery, pervasive computing

1 Introduction

Pervasive computing demands the ability to discover locally available resources, whether data shared by nearby users or digital capabilities in the surroundings. More specifically, applications require a location-based discovery service that, when provided a user's or device's location, can "look up" nearby (digital) resources. In this paper, we focus specifically on the need to discover resources relevant in a particular user's space and time; we are interested in finding digital resources that are available in the user's *here and now*. Existing approaches fall into one of two general categories: (*i*) *centralized indexing* approaches that provide (at least abstractly) a single point of lookup for location-based discovery or (*ii*) *local broadcast* approaches that rely on ad hoc discovery among peers.

The dynamics of pervasive computing environments make the approaches from mobile ad hoc networks less desirable simply due to the high degree of

* This work was funded, in part, by the National Science Foundation, Grant #CNS-0844850 and a Google Research Award. The views and conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

churn in information. These dynamics force continuous updates to the distributed structures that overlay the mobile ad hoc network to support persistent discovery. On the other hand, as the density of available smart devices (both carried by users and embedded in environments) increases, completely centralized approaches fail to scale. The obvious next step is to take the best of both approaches, distributing the centralized index in a way that is reflective of both the proximity requirements and dynamic nature of location-based discovery. This is precisely the strategy employed by distributed spatial indexing (e.g., Peer-Tree [6] and P2PR-Tree [21]) in which the universe is decomposed into a hierarchy of regions; each node in a peer-to-peer (P2P) network is responsible for managing data concerning a region in the distributed index. These approaches assume that nodes are capable of arranging themselves in a connected network topology using ad hoc wireless communication [6] or a global addressing scheme [21]. We make no such assumptions—indeed, supporting the formation of such a P2P network is a motivating scenario of our work.

Our location-based discovery service creates *cloudlets* that are responsible for maintaining knowledge about the digital resources available in specific regions of space. A cloudlet is a trusted computing resource with good Internet connectivity that is available for use by physically nearby mobile devices over a wireless LAN [33]. Essentially, a cloudlet has the same responsibilities as the cloud—it hosts a service that performs the significant computation required by a mobile application while enabling the mobile device to act as a thin client with respect to the service. Cloudlets differ from the cloud in that they are inherently within close physical proximity to the mobile devices that utilize their resources, which reduces network latency and jitter and mitigates peak bandwidth demands [33]. Consequently, cloudlets service fewer users and keep data close to where it originates. Satyanarayanan et al. envision cloudlets to satisfy the requirements necessary to transform many areas of human activity through classes of mobile computing that "seamlessly augment users' cognitive abilities via compute-intensive capabilities" (e.g., speech recognition, natural language processing, computer vision, augmented reality, planning, decision making, etc.).

We leverage cloudlets to implement proximal discovery for pervasive computing applications that operate in densely populated and highly mobile physical spaces. A cloudlet-based approach for co-located peer discovery becomes seriously advantageous, perhaps even necessary, when applications need to operate in spaces with hundreds of thousands of devices per square mile (e.g., as in the Internet of Things [1]). Consider the following scenario on a university campus:

It's the first week of the semester and construction efforts on several university buildings are behind schedule, requiring paths and building entrances be temporarily blocked off as necessary by construction crews. In an effort help students navigate campus and discover nearby friends, interest groups, and special first-week events amongst these sporadic closures, the university has deployed a mobile application that leverages the campus' cloudlet infrastructure. The app enables student users to advertise themselves, their location, and student-run events. Additionally, special privileges have been given to construction teams' foremen who

may indicate pathways and building entrances as "closed" or "open" via the mobile app. As students and construction foremen move about campus, their devices periodically push this information to the cloudlet responsible for the region of the campus they are in via the university's wireless network. Ultimately, this changing and location-dependent information is displayed on the app's "live" campus map that dynamically updates as fresh data is received from nearby cloudlets.

One could also envision such an application to be used to deliver location-dependent messages. For example, if a student wished to form an impromptu study group, she could employ the cloudlet service to alert nearby classmates of her availability and location. Alternatively, the university may wish to use the application to deliver location-dependent emergency alerts or requests for ground-level information (e.g., pictures of a particular crowded area as described in [32]) if, for example, there were a dangerous suspect or a missing person.

A cloudlet-based proximal discovery service performs the same duties as would a location-based service that may exist in the cloud—the service provides extra knowledge about a user's physical surroundings. However, by harnessing a cloudlet infrastructure, data is kept geographically close to where it will most likely be relevant, rather than in a distant cloud under a third party's administration. Moreover, each cloudlet only serves tens to hundreds of users who experience crisp interaction with the cloudlet service due to cloudlets' physical proximity and one-hop network latency. Our example scenario also teases out a key technical challenge: as users move, they migrate between *administrative regions*, the physical regions managed by cloudlets. Devices that are on the *fringe* of two bordering administrative regions may fail to discover other devices beyond the region's boundary, though they may be physically nearby. Some means of synchronization between bordering cloudlets is therefore necessary.

We present the architecture, implementation, and API for a cloudlet-based proximal discovery service that leverages a physically dispersed infrastructure of cloudlets, each of which provides the service for a specific geographic region. The distributed nature of a cloudlet infrastructure mitigates the congestion and resource contention of a centralized cloud-based mechanism. Moreover, the physical and logical proximity of cloudlets to the clients they serve results in low one-hop network latency. These factors combined provide the support required by mobile pervasive applications targeting densely populated physical spaces. A key technical challenge in the design of a cloudlet-based proximal discovery service is the synchronization of information at the boundaries of bordering or overlapping cloudlet administrative regions. As users and devices move around, they inevitably migrate between cloudlets' administrative regions. An application running on a device within the fringe of one of these ambiguous spaces may request information that exists *beyond* the administrative boundary of its respective cloudlet's administrative region, though the geographic relevance of that information may be physically close. We evaluate our service's performance in terms of the system-level *cost* and the *quality* with which it performs proximal neighbor discovery under various bordering cloudlet synchronization strategies.

2 Related Work

Existing location-dependent resource discovery approaches can generally be classified into two categories: centralized methods, which rely on a single index to resolve spatial queries, and distributed methods, which can further be decomposed into infrastructure-dependent and infrastructure*less* (i.e., ad hoc) approaches.

Centralized Location-Dependent Discovery. Geographic information systems (GIS) have emerged over the last few decades out of advancements in both database management systems (DBMS) and positioning and tracking systems (e.g., GPS) (see [26] for an excellent survey of this work) and are now an industry standard. A GIS is a collection of software geared at efficiently performing a wide range of operations over geographic data, for example, resolving spatial queries, generating maps, detecting geographic patterns over time, etc. [11]. GIS extensions exist for most modern DBMS and are the enabling technology for location-based services (LBS), which integrate a mobile device's location with other (spatially-dependent) information [34].

Google Latitude[1], Foursquare[2], Facebook Places[3], and Follow Me [38] are cloud-based examples of LBS that enable users to share their location with friends. The NearMe wireless proximity server [17] compares clients' Wi-Fi fingerprints (observed access points and signal strengths) to compute their relative proximity. MoCA (Mobile Collaboration Architecture) [29] is a client-server middleware for developing and deploying context-aware applications; MoCA supports mobility by monitoring a mobile client's location and switching to the application proxy (an intermediary between a mobile client and the application server that exists at the edge of a wired network) closest to the user. Similarly, Hydra [31] facilitates mobile pervasive application development by providing mobile agents that follow a user about a pervasive environment and construct a virtual machine that meets a user's current needs based on her location, tasks, number of co-located people, etc.

These approaches employ a single point of lookup to store participants' locations and resolve queries for location-based resources. Ypodimatopoulos and Lippman point out that any system that centralizes user location information by definition compromises user privacy [38], which can have potentially dangerous implications (e.g., burglary[4] and personal information inference [10]). For this reason their Follow Me indoor location sharing service is intended to be implemented at the per-building level instead of at a global level. We further argue that a completely centralized approach fails to meet the requirements of pervasive applications that demand low latency and target densely populated physical spaces where hundreds of thousands of devices may be carried by mobile users (e.g., Body Area Networks [3]) and embedded in the environment (e.g., the Internet of Things [1] and Web of Things [12]). These types of applications are

[1] `http://latitude.google.com`
[2] `http://foursquare.com`
[3] `http://facebook.com/about/location`
[4] `http://pleaserobme.com`

more aptly supported by computing resources that are physically near participating devices and do not rely on a (potentially distant) single globally known resource to manage devices' location information. This flavor of approach both reduces network latency and keeps location availability as local as possible.

Distributed Location-Dependent Discovery. The efficient discovery of "nearby" peers is an active area of interest in peer-to-peer (P2P) applications, where nodes interact directly with one another in a decentralized and localized fashion. Zero Configuration Networking (zeroconf)[1] and the serverless messaging extensions of the Extensible Messaging and Presence Protocol[2] (XMPP) both enable automatic discovery of available services on a local area network (LAN). Friends Radar [19] uses XMPP messaging to support P2P location sharing. Likewise, Virtual Cloud [15] employs XMPP messages to facilitate forming on-demand mobile clouds comprising co-located mobile devices. While zeroconf discovery over a wide-area network (WAN) is possible, it requires advanced setup at each client. Our proximal discovery service makes no assumptions about the *type* of network clients are a part of, simply that they are reachable.

Still other P2P applications maintain network overlays and routing tables to leverage *logical* locality. pSense [35] enables discovery of virtually visible peers in position-based massively multiplayer online games (MMOGs) through localized multicast; this eliminates the need to propagate players' location updates to a global resource. Proximal peer discovery could be implemented on top of a decentralized routing and location infrastructure like Tapestry [39] or Chord [36]. These approaches could certainly be applied in scenarios where *physical* locality was the citizen of interest. However, overlay and routing table based approaches require knowledge of at least one peer already in the network who may act as an entrance point for the new peer. In the implementation we present here, we require advanced knowledge of cloudlets and their administrative regions. However, one could envision cloudlet resources existing at the edge of a wired network infrastructure (e.g., in physically dispersed wireless access points) [32], and a device would implicitly interact with the cloudlet resource hosted on the Wi-Fi access point it was currently connected to.

Purely ad hoc approaches are *inherently* localized in both space and time due to the attenuation of radio signals as they propagate. FlashLinQ [37] is a telecommunication technology that enables long-range (approximately two mile) P2P discovery and operates in a licensed 5MHz spectrum. Other existing pieces of work use short range communication to localize (and co-localize) mobile users. Virtual Compass [2], for example, constructs a two dimensional graph of nearby devices via periodic (Wi-Fi or Bluetooth) signal strength measurements. Clearly, ad hoc strategies are advantageous in densely populated spaces as they eliminate the need for a bottleneck centralized resource and keep device interaction entirely local. Nevertheless, the maintenance of accurate routing tables, network overlays, and distributed data structures becomes expensive in scenarios that exhibit heavy churn in formation and high degrees of node mobility.

[1] `http://zeroconf.org`
[2] `http://xmpp.orgextensions/xep-0174.html`

Distributed Spatial Indexes. Our work aims to combine characteristics from both extremes: we desire the reliability and simplicity of centralized LBS systems and the scalability of entirely distributed approaches. We adopt a similar approach to that of distributed spatial indexing techniques [6, 21], which leverage the hierarchical nature of a spatial index structure (e.g., an R-Tree [13]) to strategically distribute portions of its hierarchy among networked peers. Rather than distribute portions of a holistic data structure, our proximal discovery service distributes independent computing resources that each employ their own spatial index. Moreover, our discovery service is designed under the assumption that pervasive applications likely require localized device interaction; we dictate that a computing resource monitoring device location information for a spatial region be physically near or within that region. Therefore our discovery service is able to keep location data about proximal peers as local as possible.

3 Architecture & Implementation

Under the assumption that cloudlets will be deployed within an existing Internet infrastructure [33], we implement our proximal discovery service as a Web service intended to be hosted on a cloudlet computing resource and made available through a RESTful API over HTTP. Very generally, the REST (REpresentational State Transfer) style [8] is a design technique for implementing remote procedure calls over the Web. Typically, in REST architectures, clients initiate requests to servers, which process the requests and return a response. Unlike WS-* integration techniques (e.g., SOAP, WSDL, the WS-* stack) that focus on *functions* and use HTTP merely as a transport-level protocol for application-level functionality, REST's requests and responses focus on the transfer of *resources* and use HTTP constructs directly as an application-level protocol[1]. In other words, "RESTful" Web services adopt a *data*-centric (rather than an *operation*-centric) model where "everything" is treated as a resource accessible strictly via HTTP operations (GET, PUT, POST, DELETE).

A RESTful architecture is an ideal candidate for any cloudlet-based Web service for two reasons. First, REST has become a standardized design technique and is the basis for most Web 2.0 services (e.g., those provided by Google, Twitter, Facebook, Amazon, etc.). Therefore, a RESTful cloudlet-based service can seamlessly integrate into existing Internet infrastructures. Second, since REST integrates application-level functionality directly with HTTP, it does not require additional higher level constructs or middleware. REST's lightweight nature makes it ideal for resource-constrained pervasive computing devices [12]. In the particular case of our proximal discovery service, it is natural to translate our scenario's notion of nearby *devices* to that of the REST concept of *resources*.

The Proximal Discovery Service. Our proximal discovery service's architecture is shown in Fig. 1. The service has two components that reside on a cloudlet: a spatial index to keep track of resources' locations and a REST

[1] See [24] for a thorough comparison of WS-* and REST.

API that enables remote storage and retrieval of these resources via an Internet connection. A "resource" is anything that a particular application wishes to associate with a geographic location—e.g., a user's mobile device, a sensor embedded in the environment, a location-dependent message, or a mobile software agent that migrates between machines. We implement the spatial index using a PostgreSQL[1] database with PostGIS[2] geospatial database extensions. PostGIS employs an R-Tree [13] to intelligently index and efficiently query geospatial data. Our service's REST API is implemented in Node.js[3], which is a particularly good fit for our use case because of its small footprint and ability to efficiently handle many simultaneous connections and requests.

Our service exposes a minimalist REST API with the following resource "routes." We aim solely to facilitate discovery of physically nearby resources, leaving other application-specific functions to applications themselves.

`POST /devices` A client (mobile device, embedded sensor, software agent) issues this request to an instance of the discovery service to create or update the resource representing the client on the cloudlet.

`GET /neighbors` A client issues this request to query a cloudlet for resources within its vicinity. We parameterize the request with a *range* to limit the query's geographic search space. An application could further impose additional application-specific parameters. Upon receiving this request, the cloudlet responds with a list of resources whose locations are believed to be within the request's range of the client.

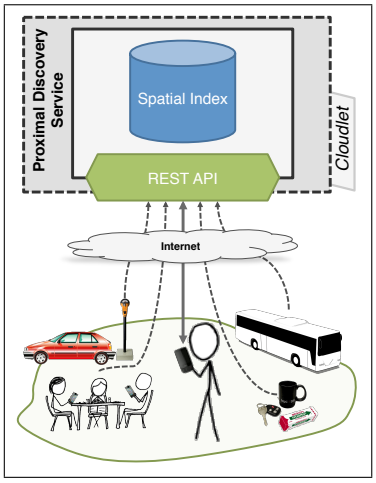


Fig. 1. The Proximal Discovery Service's Architecture. Dashed arrows are `POST /devices` requests; the solid arrow is a `GET /neighbors` request.

Our approach only provides an advantage over a centralized cloud-based LBS when many independent instances are distributed across a geographic region, where each instance assumes responsibility for a particular sub-region. However, important challenges arise when clients physically move between these sub-regions or issue queries that cross the boundaries of these sub-regions.

Distributed Cloudlet Synchronization. Our proximal discovery service supports pervasive computing applications in densely populated physical spaces. To fully utilize the potential of a cloudlet infrastructure and support large numbers of anticipated clients, many instances of our service must be deployed in a geographic region, with each instance responsible for a particular sub-region. These spaces are highly mobile—people and their devices move and inevitably

[1] `http://postgresql.org`
[2] `http://postgis.net`
[3] `http://nodejs.org`

migrate between adjacent cloudlet-administered sub-regions; our proximal discovery service must address the distributed synchronization of cloudlets governing these adjacent sub-regions.

Consider the scenario in Fig. 2. A space is decomposed into nine *administrative regions* (AR). One cloudlet exists in each AR and is responsible for responding to cloudlet-bound requests for that region. The solid dot represents a mobile client that has traveled along the dashed path over a period of time. As the client moves, it periodically issues a `POST /devices` with its current location to the cloudlet responsible for the AR it occupies. At the point in time shown, this client wishes to discover nearby resources within a range of r, so it issues a `GET /neighbors` query to $cloudlet_1$. However, the query's range extends beyond the borders of $cloudlet_1$'s AR and into those of $cloudlet_0$, $cloudlet_3$, and $cloudlet_4$. Without some means of synchronization, $cloudlet_1$ can only respond with the four resources in the shaded region of the query's target space (i.e., the resources within its AR). To facilitate this synchronization, our proximal discovery service implements a third resource route:

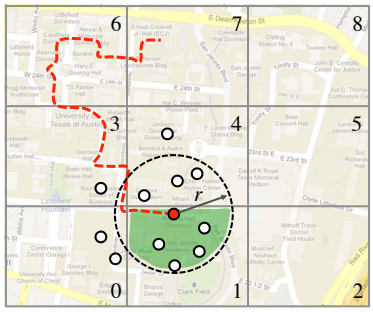


Fig. 2. A query for resources within a range of r from a mobile client.

`POST /fringes` A neighboring cloudlet issues this request to communicate knowledge of resources that exist at or near the border of an adjacent AR.

To both capture and quantify "at or near the border," we introduce the concept of a *fringe*, illustrated in Fig. 3. A fringe is a sliver of space of width δ on the interior of an AR's border. For simplicity, the ARs shown in Fig. 3(a) are rectangular, and therefore the computed fringes are also rectangular. In actuality, ARs can be of any shape and even overlapping. AR_i's fringe with AR_j consists simply of any point within AR_i that is within a specified distance (δ) of AR_j. Consider the more complex pair of ARs in Fig. 3(b). AR_A overlaps AR_B; any resource within the area marked X can report to either AR_A or AR_B (or both of them). The fringe that AR_B computes to create a digest for AR_A contains all of the resources it knows about located in Y *and* all of the resources it knows about located in X (because they may not have also reported their locations to A). Fringes can also overlap; consider the adjacency of AR_B and AR_C; the fringe of AR_B with respect to AR_C (not depicted) will include not only their overlapping area but also a sliver adjacent to this area that also overlaps Y.

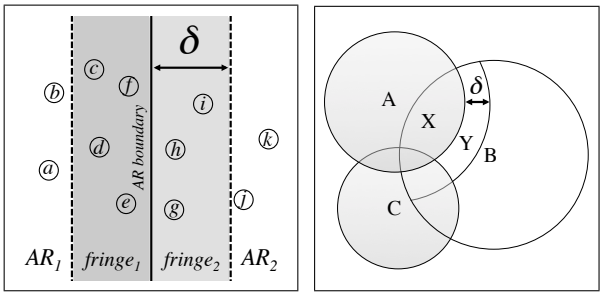


Fig. 3. Computing administrative region fringes.

Periodically, our service computes each of its fringes' *digest*, or a snapshot of the fringe's resources, which it sends to the appropriate adjacent cloudlet via a `POST /fringes`. In Fig. 3(a), for example, $cloudlet_1$ computes AR_1's fringe digest to be (c, d, e, f), which it `POST`s to $cloudlet_2$.

Our service entails three variable deployment parameters, each which must be tuned for a particular application's needs:

μ The number of administrative regions (and correspondingly, cloudlets) that a geographic region is decomposed into.
δ The width of administrative regions' fringes.
T The interval of time between fringe digest computations.

A particular combination of parameters, represented by the tuple (μ, δ, T), is a *synchronization strategy*. In Section 5 we evaluate our service's performance under various synchronization strategies in terms of the *quality* with which proximal discovery queries are satisfied and the system-level *cost* of the strategy.

Operating Assumptions. We assume clients can localize themselves and have a labeled map of cloudlets and their ARs. Localization may be too expensive a task for severely resource constrained devices (e.g., battery operated sensors); however, lightweight localization methods are currently a very active area of research [2, 4, 20, 25]. Alternatively, resource-constrained devices could communicate with cloudlets via less resource-constrained *gateways* [12]. We also assume that cloudlets are aware of neighboring cloudlets; in our implementation, clients and cloudlets each possess a copy of the same labeled map.

4 Application Examples

We next provide some concrete examples of mobile and pervasive machine-to-machine (M2M) applications that are enabled by our proximal discovery service, how they can be built with our service, and the inherent advantages of doing so.

Location Based Services. Perhaps the most obvious application of our service is for the types of LBS discussed earlier (e.g., location-based social networking services, location sharing services, friend finders, and spatial crowdsourcing [16]). Using our cloudlet-based service for localized discovery, such services could target small and heavily crowded regions (e.g., music festivals, parades, theme parks, university campuses, etc.) much more efficiently. Instead of shipping application requests to a (physically and logically) distant central cloud index, requests *and* users' location information would be localized to the geographic areas where they are inherently relevant, providing low-latency responses, lighter server-side loads, and privacy gains through the distributed cloudlets [38].

P2P Network Overlays and Routing. In Section 2 we discussed the fact that using a P2P routing and location infrastructure like Tapestry [39] or Chord [36] to implement proximal resource discovery requires new network participants to have advanced knowledge of at least one peer already in the P2P overlay. This crucial discovery step could be performed with our proximal discovery service, where the resource of interest is any peer already in the overlay.

The use of a cloudlet infrastructure, in this case, encourages and facilitates P2P interaction between devices that are *physically* near one another. Peer physical proximity is not a requirement for these systems but can be advantageous if the P2P overlay exists to store spatio-temporal events [40].

Mobile Ad Hoc and Opportunistic Networks. Our proximal discovery service could also be used to construct *virtual* mobile ad hoc network (MANETs) and opportunistic networks. Beyond the ability to implement MANET-style routing algorithms without a separate dedicated radio interface, virtualization of MANETs has added security benefits [14]. Virtual MANETs could be used to form on-demand mobile clouds [30], execute geographic routing [5], implement location-based publish-subscribe [7], or enable wireless sensor network (WSN) query-access mechanisms [9, 18, 22] across smartphones. The realization of cloudlet-based virtual MANETs for mobile computing could conceivably lead to a renaissance of techniques formerly developed for MANETs and WSNs.

5 Evaluation

We next benchmark the performance of our service in terms of its system-level cost and the quality with which the service performs proximal resource discovery under various combinations of (μ, δ, T) (number of administrative regions, fringe width, fringe digest computation period), in an effort to guide application developers in tuning our service's parameters to meet application requirements. We simulate mobile clients using two real-world and one simulated mobility trace data sets (Table 1). Our framework is implemented in Python.

Table 1. Mobility Trace Data Sets

Data Set	**Geographic Region**	**Description**
UT-Real	$640m^2$ UT Austin campus	24 hours of location information from 18 users of a mobile application deployed on the UT Austin campus in June 2013
UT-Sim	$640m^2$ UT Austin campus	6 hours of simulated location information for 200 nodes generated using MobiSim [23] and Levy-walk [28] mobility
Cabspotting [27]	$10km^2$ downtown San Francisco	6 hours of location information for ∼500 taxi cabs in downtown San Francisco, USA

We decompose a square geographic region into μ administrative regions (ARs), assign each AR one instance of our discovery service, and generate a labeled map of these regions and their ARs; the clients and the μ service instances each receive a copy of this map. Each simulated client moves about the region and issues a `POST /devices` with its current location at most once per minute to the cloudlet-hosted service responsible for the AR it occupies. Every T seconds, each of the μ instances of the discovery service computes the digest[1] for each

[1] To prevent "stale" resources, we restrict digests to contain only resources that issued a `POST /devices` within the last T seconds (i.e., since the last fringe digest).

of its fringes. The service instance then issues a `POST /fringes` containing the digest to the respective neighboring service instance. During each simulation, we perform periodic proximal resource discovery queries (i.e., cloudlet-bound `GET /neighbors` requests): at 10 randomly chosen simulation times we randomly select 25 simulated clients[1] and issue three queries (r =["near", "medium", "far"])[2]. For each data set, we also generate an *Oracle*, which is effectively a central (cloud) server to which every client posts its location once per minute.

Table 2 shows our evaluation parameters; in cases where we explored multiple values, the value in parentheses is the default. We first evaluate the *quality* of our service in terms of the mean number of *false positives* (resources *in* a cloudlet result and *not in* the corresponding *Oracle* result) and *false negatives* (resources *not in* a cloudlet result, but *in* the corresponding *Oracle* result) produced per query under various synchronization strategies. In both cases, we normalize the false positives and false negatives to the size of the *Oracle* result set. That is, a value of 0.1 for a normalized false negative score indicates that for every 10 items in the *Oracle* result, there was one result missing from the cloudlet result.

Table 2. Evaluation Parameters

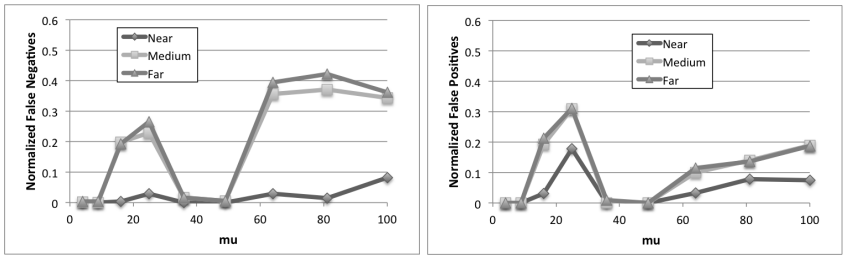
Parameter	**Value(s)**
μ: *number of cloudlets*	$4, 9, 16, (25), 36, 49, 64, 81, 100$
δ: *fringe width (fraction of width of an AR)*	$\frac{1}{10}, \frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, (\frac{1}{5}), \frac{1}{4}, \frac{1}{3}, \frac{1}{2}$
T: *digest update period (in seconds)*	$60, 300, (900), 1800, 3600, 7200, 18000$
t: *client location update period (in seconds)*	60[3]
u: *location update size (in bytes)*	80

Fig. 4 shows the false negative and false positive rates for varying the number of cloudlets for the *UT-Real* and *Cabspotting* traces; we omit the results for *UT-Sim* as they are very similar to the results for *Cabspotting*. Two observations are relatively consistent across all data sets. First, the errors show a slight upward trend as the number of cloudlets increases. As the cell sizes decrease, queries are increasingly likely to rely on the digest from neighboring cloudlets for correct information, and this information is more out of date than the local cloudlet's information. Second, the false positive and false negative rates for "near" queries are significantly better than for "medium" or "far." This demonstrates (and begins to benchmark) that the cloudlet-based approach is more suited for highly localized search and is not as well suited to searching for information that is located further afield. Consider a query in the *UT-Real* when μ is 64, where each cloudlet is responsible for a 10 x $10m^2$ area. A "medium" query, looking for resources within $50m$, may match resources that are not even located in an adjacent administrative region and will be impossible to find using our service.

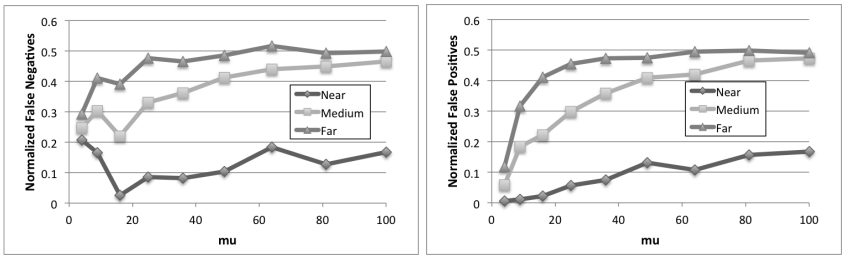
[1] Only 18 clients were available in the *UT-Real* data set.

[2] We define the ["near", "medium", "far"] ranges as $(25m, 50m, 75m)$ for the *UT-Real* and *UT-Sim* data sets and $(100m, 500m, 1000m)$ for the *Cabspotting* data set.

[3] The client update period was set to exactly 60 seconds for *UT-Sim*. For *UT-Real* and *Cabspotting*, the period was determined by the data set but was close 60 seconds.



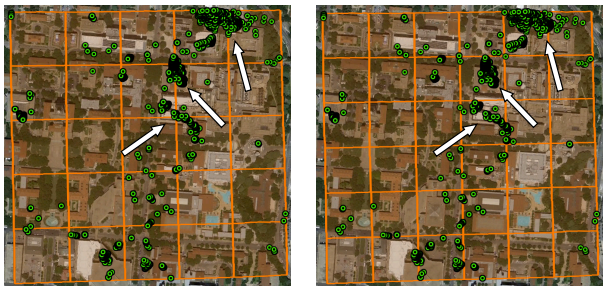
(a) *UT-Real*



(b) *Cabspotting*

Fig. 4. Impact of varying the number of cloudlets.

Fig. 4(a) also shows an interplay between the behavior of real users and the definition of cloudlets. For μ values of 36 and 49, the upward trend of the error rates is violated. The reason can be identified by examining the location traces of the users, shown in Fig. 5.



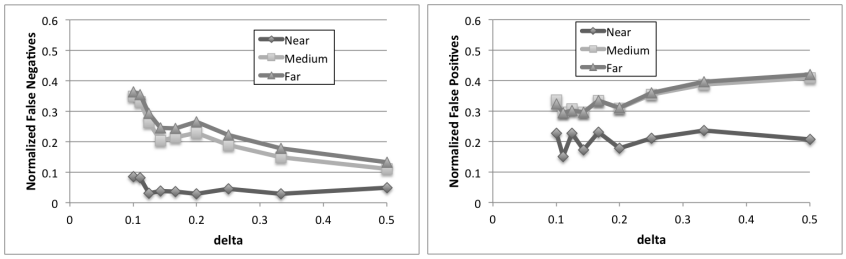
(a) $\mu = 25$ (b) $\mu = 36$

Fig. 5. A "poor" choice of μ may split clusters.

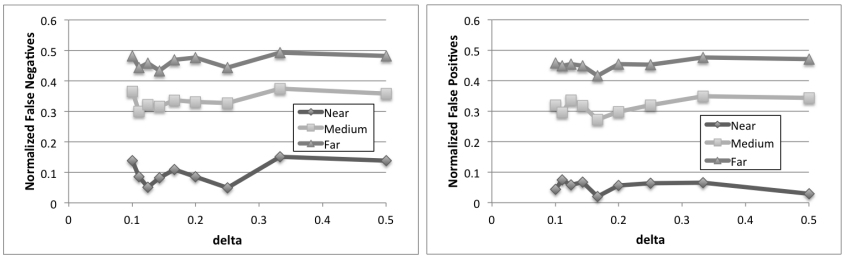
In the case of $\mu = 36$, the common areas where users cluster (in this case, three buildings on the university campus) happen to lie entirely within single administrative regions. In the case of $\mu = 25$, these clusters cross the boundaries of administrative regions. Users' queries therefore also often cross these boundaries, meaning that they increasingly rely on digests for correct query resolution. The conclusion from this observation is that defining administrative regions should account for user mobility patterns and should not create artificial boundaries to separate users within natural congregation areas.

We next examine the impact of varying the fringe width. Fig. 6 shows the false positive and false negative rates for the *UT-Real* and *Cabspotting* traces; we again omit the results for *UT-Sim*, which are again very similar to the results for *Cabspotting*. Except within the (small) *UT-Real* data set, there is little relationship between changing δ and false positives and false negatives. These figures again show the significant difference in quality for "near," "medium,"

and "far" queries. While the error (as measured by the false positive or false positive rate) is routinely below 10% for "near" queries, it grows up to nearly 50% for "far" queries. This again demonstrates that the cloudlet based approach is particularly suited to very local queries of the immediate surroundings.



(a) *UT-Real*



(b) *Cabspotting*

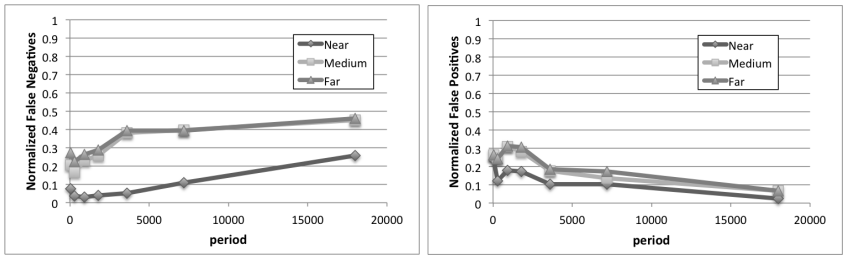
Fig. 6. Impact of varying the fringe width.

Finally, we examine the quality of our cloudlet based discovery service with respect to the update interval for fringe digests in Fig. 7. As T increases up to 50 minutes, quality degrades substantially. In the limit, the false positive rates also decrease; this is a result of the fact that the digests are simply not updated well, so they do not contain extra (irrelevant) information. However, we can also discern that it is acceptable for the update interval to be larger than the frequency with which the clients update their own cloudlets; specifically, any setting of T under 900 seconds has only a marginal impact on false negatives.

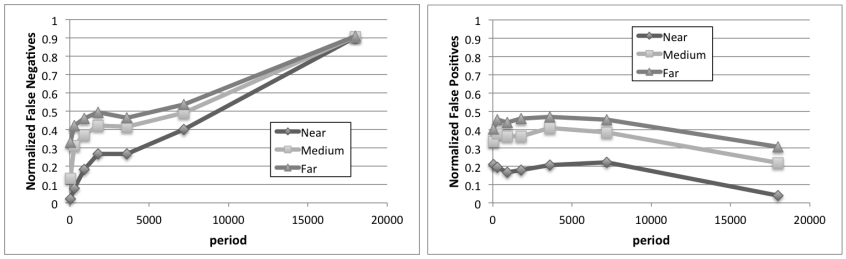
We also benchmark the system-level *cost* of a synchronization strategy in terms of the mean overhead (in KB) for sending both individual location updates to the local cloudlet and the cost of sending the fringes according to the update period. Because individual location updates are sent every minute, the cost, per minute, of sending individual location updates to the local cloudlet is computed as: $n \times 80\text{B} \times 1\text{ hop} \times \frac{1\text{KB}}{1024\text{B}}$, where n is the number of clients sending location updates, and 80 bytes is the size of a client location update (from Table 2). We assume that each client is located within one network hop of the cloudlet server. We compute the cost of sending the digests between cloudlet servers as:

$$4(\mu + \sqrt{\mu}) \times \overline{size_{digest}}\text{B} \times 1\text{ hop} \times \frac{1\text{KB}}{1024\text{B}}$$

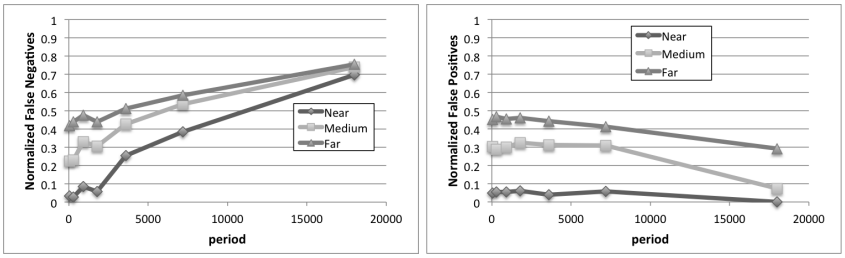
where $4(\mu + \sqrt{\mu})$ is the number of fringes in our square region and $\overline{size_{digest}}$ is measured during execution. We assume a single network hop between adjacent



(a) *UT-Real*



(b) *UT-Sim*



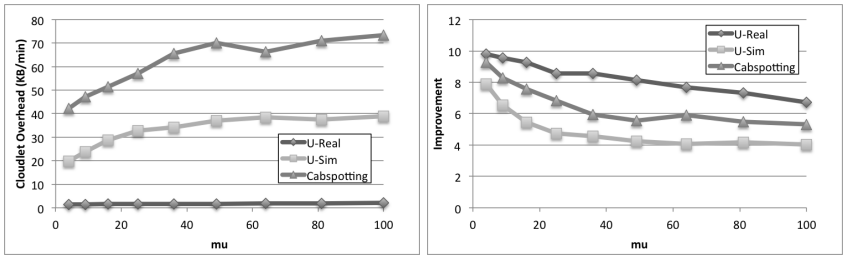
(c) *Cabspotting*

Fig. 7. Impact of varying the fringe digest update period.

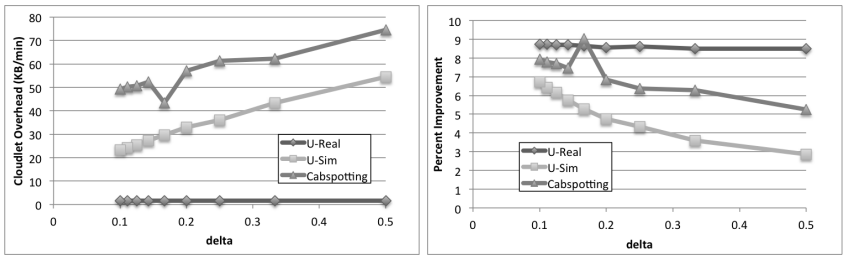
cloudlet servers. The total cost for the cloudlet approach is the sum of these two values. We compare this to the system-level cost of the centralized approach, computed as: $n \times 80\text{B} \times h\,\text{hop} \times \frac{1\text{KB}}{1024\text{B}}$, where the only cost is sending the clients' individual updates to the central server. In the following results, we used the (conservative) assumption of 10 hops to the central server[1]. Fig. 8 shows the results, which include both the absolute values of the overhead (measured in KB/min) for the cloudlet based approach and the improvement of the cloudlet based approach over the centralized approach.

Overhead increases with both increasing μ and increasing δ. As μ increases, there are more digests to be sent because there are more fringes. Note, though, that the overhead increases relatively slowly for increasing μ. Overhead increases with δ simply because, as the fringe size increases, there are more resources within the fringe, so the size of the digest grows. Interestingly, the overhead

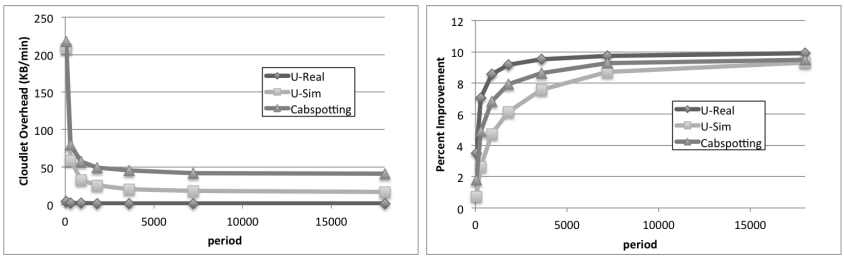
[1] The value of 10 hops is significantly below the values we measured using `traceroute`s to common cloud servers, which were routinely above 15 hops. The number of network hops is not the only (or necessarily "best") measure of cost, but it gives a reasonable measure of the relative costs of these two approaches.



(a) *Varying* μ



(b) *Varying* δ



(c) *Varying* T

Fig. 8. Cloudlet overhead and comparison to Cloud based approach.

falls off rapidly with increasing T; for all values of T greater than 60 seconds, we observed relatively low overheads (and therefore significant improvements in comparison to the centralized approach). The plots in Fig. 8 show one additional interesting phenomenon. The degree of improvement for the *Cabspotting* trace is consistently better than the degree of improvement for the *UT-Sim* data set. In real situations, resources tend to cluster in non-uniform ways, which the cloudlet-based approach is designed to be sensitive to. This is in contrast to the manufactured *UT-Sim* case in which the resources are uniformly distributed with no attention to how real users or resources would be distributed in a real space.

Our evaluation of the cloudlet based proximal resource discovery service has demonstrated the situations in which it can prove beneficial over a more traditional cloud based approach. The cloudlet based approach may not always be the ideal option (e.g., it does not have a high quality of query resolution for "far" queries, or queries about resources that are multiple hops away in the local network). For applications searching for very local resources (i.e., resources in the immediate environs), the cloudlet based proximal resource discovery pro-

vides high quality discovery at significantly decreased costs (with respect to the amount of data transmitted within the network in total). The use of digests to aid in discovering resources in neighboring cloudlets can be beneficial, especially when the administrative regions are defined not randomly, but with input about how they match with the real spaces that people inhabit.

6 Conclusion

The discovery of nearby digital resources is a definitive requirement of pervasive computing, characterized by location-based mobile applications in digitally-rich and highly dynamic physical spaces. As the density of both resources and users continues to grow in such operating spaces, it becomes far more practical, perhaps even crucial, to utilize proximally available computing resources rather than a distant cloud resource to facilitate the discovery of nearby users and resources. In this paper, we presented the design and implementation of a distributed location-based discovery service based on physically dispersed *cloudlets*, which provide the service for well-defined physical regions. Our approach takes advantage of the inherent proximity of clients, cloudlets, and relevant resources. We explored various deployment settings of our service and demonstrated that a cloudlet-based approach requires significantly less in-network traffic than a centralized cloud-based approach while still providing good "look-up" query quality. Our evaluation focused on cloudlets that were responsible for rectangular regions; our implementation does presuppose a particular shape of a cloudlet's administrative region (AR)—our service could operate over cloudlets with ARs of arbitrary shapes that even overlap. Immediately, we envision that our proximal discovery service could be used to supplement existing location-based discovery services and P2P overlay approaches in heavily populated small physical spaces. More generally, and more importantly, our service enables the realization of *virtual* MANETs, the full-scale creation of which could lead to a revival of interest in a whole body of MANET and WSN resource management techniques.

References

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Net.*, 54(15):2787–2805, 2010.

[2] N. Banerjee, S. Agarwal, P. Bahl, R. Chandra, A. Wolman, and M. Corner. Virtual compass: Relative positioning to sense mobile social interactions. In *Proc. of Pervasive*. 2010.

[3] M. Chen, S. Gonzalez, A. Vasilakos, H. Cao, and V. Leung. Body area networks: A survey. *Mobile Net. and App.*, 16(2):171–193, 2011.

[4] I. Constandache, X. Bao, M. Azizyan, and R. R. Choudhury. Did you see Bob?: human localization using mobile phones. In *Proc. of MobiCom*, 2010.

[5] S. Das, H. Pucha, and Y. Hu. Performance comparison of scalable location services for geographic ad hoc routing. In *Proc. of INFOCOM*, 2005.

[6] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of P2P*, 2003.
[7] L. Fiege, F. C. Gärtner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proc. of Middleware*, 2003.
[8] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.
[9] I. Galpin, C. Brenninkmeijer, A. Gray, F. Jabeen, A. Fernandes, and N. Paton. Snee: a query processor for wireless sensor networks. *Dist. and Parallel Databases*, 29:31–85, 2011.
[10] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez. Show me how you move and I will tell you who you are. In *Proc. of SPRINGL*, 2010.
[11] M. Goodchild. *Research Methods in Geography*, pages 376–391. Wiley-Blackwell, third edition, 2010.
[12] D. Guinard and T. Vlad. Towards the web of things: web mashups for embedded devices. In *Proc. of WWW*, 2009.
[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. of SIGMOD*, 1984.
[14] D. Huang, X. Zhang, M. Kang, and J. Luo. MobiCloud: Building secure cloud framework for mobile computing and communication. In *Proc. of SOSE*, 2010.
[15] G. Huerta-Canepa and D. Lee. A virtual cloud computing provider for mobile devices. In *Proc. of MCS*, 2010.
[16] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proc. of SIGSPATIAL*, 2012.
[17] J. Krumm and K. Hinckley. The NearMe wireless proximity server. In *Proc. of UbiComp*. 2004.
[18] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, 2003.
[19] R. Mayrhofer, C. Holzmann, and R. Koprivec. Friends radar: Towards a private P2P location sharing platform. In *Proc. of EUROCAST*. 2012.
[20] M. Minami, Y. Fukuju, K. Hirasawa, S. Yokoyama, M. Mizumachi, H. Morikawa, and T. Aoyama. DOLPHIN: A practical approach for implementing a fully distributed indoor ultrasonic positioning system. In *Proc. of UbiComp*. 2004.
[21] A. Mondal, Y. Lifu, and M. Kitsuregawa. P2PR-Tree: An R-Tree-based spatial index for peer-to-peer environments. In *Proc. of EDBT Workshops*. 2005.
[22] L. Mottola and G. Picco. Programming wireless sensor networks with logical neighborhoods. In *Proc. of InterSense*, 2006.
[23] S. M. Mousavi, H. Rabiee, M. Moshref, and A. Dabirmoghaddam. MobiSim: A framework for simulation of mobility models in mobile ad-hoc networks. In *Proc. of WIMOB*, 2007.
[24] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big" web services: making the right architectural decision. In *Proc. of*

WWW, 2008.
[25] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. BeepBeep: a high accuracy acoustic ranging system using cots mobile devices. In *Proc. of SenSys*, 2007.
[26] D. Peuquet. Making space for time: Issues in space-time data representation. *GeoInform.*, 5:11–32, 2001.
[27] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/mobility, 2009.
[28] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong. On the levy-walk nature of human mobility. *IEEE/ACM Trans. Netw.*, 19(3):630–643, jun 2011.
[29] V. Sacramento, M. Endler, H. Rubinsztejn, L. Lima, K. Goncalves, F. Nascimento, and G. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *Dist. Sys. Online, IEEE*, 5(10):1–14, 2004.
[30] F. A. Samimi, P. K. McKinley, and S. M. Sadjadi. Mobile service clouds: A self-managing infrastructure for autonomic mobile computing services. In *Proc. of SelfMan.* 2006.
[31] I. Satoh. Dynamic deployment of pervasive services. In *Proc. of ICPS*, 2005.
[32] M. Satyanarayanan. Mobile computing: the next decade. *SIGMOBILE Mob. Comput. Commun. Rev.*, 15(2):2–10, Aug. 2011.
[33] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.
[34] J. Schiller and A. Voisard. *Location-Based Services.* Morgan Kaufmann, 2004.
[35] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. Buchmann. pSense: Maintaining a dynamic localized peer-to-peer structure for position based multicast in games. In *Proc. of P2P*, 2008.
[36] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM*, 2001.
[37] X. Wu, S. Tavildar, S. Shakkottai, T. Richardson, J. Li, R. Laroia, and A. Jovicic. FlashLinQ: A synchronous distributed scheduler for peer-to-peer ad hoc networks. In *Proc. of Allerton*, 2010.
[38] P. Ypodimatopoulos and A. Lippman. Follow me: a web-based, location-sharing architecture for large, indoor environments. In *Proc. of WWW*, 2010.
[39] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2001.
[40] A. Ziotopoulos and G. de Veciana. P2P network for storage and query of a spatio-temporal flow of events. In *Proc. of PerCom Workshops*, 2011.