GeoServ: A Distributed Urban Sensing Platform

Conference Paper · June 2011

DOI: 10.1109/CCGrid.2011.10 · Source: IEEE Xplore

CITATIONS
18

READS
162

3 authors:

Jong Hoon (Joey) Ahnn
Target
17 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Uichin Lee
Korea Advanced Institute of Science and Technology
227 PUBLICATIONS **8,447** CITATIONS
SEE PROFILE

Hyun Jin Moon
University of California, Los Angeles
39 PUBLICATIONS **1,234** CITATIONS
SEE PROFILE

UNIVERSITY OF CALIFORNIA
Los Angeles

GeoServ: A Distributed Urban Sensing Platform

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Jong Hoon Ahnn

2011

The thesis of Jong Hoon Ahnn is approved.

Deborah Estrin

Songwu Lu

Mario Gerla

Miodrag Potkonjak, Committee Chair

University of California, Los Angeles

2011

Dedicated to Jesus Christ my Lord, my parents, my wife, my baby, and my friends.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Miodrag Potkonjak who is truly a passionate innovator in computer science research, and I have been very fortunate to work with and learn from him. He taught me how research is done and especially what makes an in uential research. Miodrag's endless energy and devotion for his students was also remarkable.

During the course of this dissertation, I have also been very fortunate to work with Uichin Lee at KAIST KSE in Prof. Gerla's group and Hyun Jin Moon at NEC Lab. Their deep knowl edge of sensor networking and data management were instrumental in designing a distributed urban sensing platform called *GeoServ*.

During the graduate study at UCLA, I am deeply indebted to my wife Yeon Sun. Always, her love, care, and trust has been the source of my strength and courage. I am glad that the greatest gifts of God, our first baby, is to be born in June, 2011. I have been very fortunate to have four parents who supported me throughout my graduate study. My parents in Korea have always supported me financially, mentally and spritually.

Many people helped to make my years at UCLA a very enjoyable experience. I especially thank the Korean folks at UCLA CS. I also want to thank folks at GSC and ANC members who continouly helped discipline me to live in the present of God.

VITA

1978	Born, Gumi, South Korea.
1997–2004	B.S, Electrical and Computer Engineering, Hanyang University, South Korea.
1999–2001	Assistant Researcher, FINDTECH, South Korea. Internet Video Indexing (i-VIBS) Solution and Home Video Editor (i-CAM) Solution.
2001–2003	Software Engineer, MOBILEONE COMMUNICATIONS, South Korea. Net-Game Pack Solution and Mobile Game Development.
2003–2004	Software Engineer, MGAME, South Korea. Online Bomberman Game Development.
2005–2006	Researcher, ANY Corporation, South Korea. TFT LCD Monitor for G.E and Siemens Medical Solutions and Digital Video Recorder (DVR) Solution.
2006–2007	M.Eng, Electrical and Computer Engineering, Cornell University, USA.
2007–2008	Visiting Scientist, Computer Science, Cornell University, USA. Live Objects Project, sponsored by Prof. Ken Birman.
2008–present	Ph.D, Computer Science, UCLA, USA.

2010–2010	Visiting Student Researcher,USC Information Sciences Institute, USA. Resource Allocation for Cloud Computing Project, sponsored by InfoSys.
2011–present	Adjunct Researcher, Knowledge Service Engineering, Korea Advanced Institute of Science and Technology (KAIST), South Korea. GeoServ: A Distributed Urban Sensing Platform Project, sponsored by Prof.. Uichin Lee.
2011–present	Adjunct Staff, RAND Corporation, USA. Mobile Support for Healthcare Project.

PUBLICATIONS

K. Ostrowski, K. Birman, D. Dolev, J. H. Ahnn. *Programming with Live Distributed Objects.* European Conference on Object-Oriented Programming, 2008.

J. H. Ahnn, K. Birman, K. Ostrowski, and R. V. Renesse. *Demo Proposal - Using Live Distributed Objects in Office Automation.* Middleware, 2008.

J. H. Ahnn, K. Birman, K. Ostrowski, and R. V. Renesse. *Using Live Distribued Objectsin Office Automation.* Middleware, 2008.

J. H. Ahnn, U. Lee, and H. J. Moon. *GeoServ: A Distributed Urban Sensing Platform.* To appear in CCGRID, 2011.

J. H. Ahnn, and M. Potkonjak. *What to Read? With Whom to Work? Where to Publish? - Scientific Techniques for Organizing and Conducting Engineering Research.* To appear in MSE, 2011.

J. H. Ahnn, H. Shi, L. Tang, T. Faber, and J. Mirkovic. *A Knowledge-based Genetic Algorithm for Cloud Resource Allocation.* ICDCS in submission, 2011.

ABSTRACT OF THE THESIS

GeoServ: A Distributed Urban Sensing Platform

by

Jong Hoon Ahnn

Master of Science in Computer Science

University of California, Los Angeles, 2011

Professor Miodrag Potkonjak, Chair

Urban sensing where mobile users continuously gather, process, and share *location-sensitive sensor data* (e.g., street images, road condition, traffic flow) is emerging as a new network paradigm of sensor information sharing in urban environments. The key enablers are smartphones (e.g., iPhones and Android phones) equipped with on-board sensors (e.g., cameras, accelerometer, compass, GPS) and various wireless devices (e.g., WiFi and 2/3G). The goal of this paper is to design a scalable sensor networking platform where millions of users on the move can participate in urban sensing and share *location-aware information* using always-on cellular data connections. We propose a two-tier sensor networking platform called *GeoServ* where mobile users publish/access sensor data via an Internet-based distributed P2P overlay network. The main contribution of this paper is two-fold: a location-aware sensor data retrieval scheme which supports geographic range queries, and a location-aware publish/subscribe scheme which enables efficient multicast routing over a group of subscribed users. We prove that GeoServ protocols preserve locality, and we validate their performance via extensive simulations.

CHAPTER 1

Introduction

The rising popularity of smartphones with onboard sensors (e.g., GPS, compass, accelerometer) and *always-on* mobile Internet connections via 2/3G (and 4G/LTE) has led to using smartphones as a platform for large-scale urban sensing (or participatory sensing) [21, 20]. Mobile users can perform location-aware micro-blogging by publishing/accessing brief micro-media updates such as photos or audio/video clips [7], measure personalized estimates of environmental impact and exposure (e.g., PIER [21]), profile road/driving conditions (e.g., potholes and honking [20]), and share live traffic information (e.g., Google Maps [10]).

Recent reports estimated that the number of smartphone users will catch and surpass the number of feature phone users in the U.S. by 2011, reaching more than 150 million users [34]. This means that millions of smartphone users on the move will be able to participate in urban sensing, which would be comparable to supporting large-scale web services [2]. For instance, 10 million mobile users could generate sensor data at the rate of 1KB/s per user (e.g., GPS, accelerometer, WiFi scanning data) and also send queries, requiring networking systems with a sheer amount of bandwidth ($>$80Gbps), storage space($>$36TB/hr), and computational power. Thus, there is a need for *scalable sensor networking systems* that can facilitate information sharing among millions of mobile users via always-on 2/3G connections.

One promising design option would be using a mobile-to-mobile overlay network of 2/3G users. Recently, Rybicki et al. [31, 32] proposed PeerTIS where mobile phones

on the road form a distributed hash table (DHT) to realize scalable information sharing in vehicular environments (e.g., congestion notification). However, mobile-to-mobile networking is not practical for several reasons. Most importantly, P2P connections between mobile devices are typically hampered by network address translation (NAT), a commonly used technique in the mobile operator's domain to better utilize limited IP address blocks and to provide secure Internet connectivity [16]; for P2P we need additional services such as session initiation protocol (SIP) or P2P proxy servers [18]. Moreover, P2P protocol operations such as routing and searching may require quite a few message exchanges over mobile nodes, which results in intolerable delays given that 2/3G cellular networks typically have a large round trip delay (i.e., several hundred milliseconds [25]). Also this causes significant resource consumption (e.g., battery, processing power, and bandwidth), which is a serious problem for resource-limited smartphones.

Therefore, it behooves us to consider using the Internet servers for large-scale urban sensing. Existing Internet-based systems are mostly based on centralized multi-tier architecture where sensor data are stored on the centralized back-end database servers (either directly or through web servers) [36, 23, 29]. Semi-hierarchical architecture was also proposed in the literature [8, 1] where each organization maintains database servers for its own stationary sensors, and information access is realized using a global naming service. While such centralized approaches could provide scalable services by provisioning more servers and bandwidth at the data centers, a viable alternative for participatory sensing is to leverage users' participation by allowing them to share their computing resources (e.g., desktops and home gateways) via a distributed P2P network, and to deliver those services in a comparable quality.

For this reason, we consider a two-tier sensor networking architecture for large-scale participatory sensing: i.e., Internet-based fixed servers form a distributed P2P

sensor networking overlay, through which mobile users can publish/access sensor data. For those who opt in to services, Internet-based P2P services are installed on their desktops/laptops, or on always-on micro servers such as home gateways and set-top boxes (called nanodatacenter nodes) [22].[1]

1.1 Applications with Location Awareness

Given that urban sensing apps are mostly *location-sensitive*, we horizontally partition sensor data based on geographic coordinates across Internet-based P2P overlay nodes. Moreover, since mobile users continually publish location-sensitive data, they are associated with overlay nodes that are responsible for the area where they are currently residing to minimize routing overhead. Both Internet servers and mobile clients have map data of the associated areas, e.g., Tigermap [35]. Consider the traffic information system. A mobile user located at location (X, Y) will be associated with an overlay node responsible for the current area. The mobile user generates traffic related information (e.g., GPS samples and speeds) and publishes that data to the overlay node—sensor data are "geographically partitioned" along overlay nodes. Representative queries include finding the average speed of nearby areas, and notifying traffic jam along one's driving route. Thus, we envision the following key overlay services:

- *Location-aware Sensor Data Retrieval Service* where geographic range queries (over the geographically partitioned sensor data stored in the overlay nodes) are efficiently handled; e.g., apps fetch GPS readings originated from a set of road segments to calculate the average speeds in that area.

- *Location-aware Publish/Subscribe Service* where mobile users or sensing apps

[1] We assume that the resulting overlay network is quite stable like Skype, where the frequency of node join/leave is less than 5% within 30 minutes [11].

can create/join a group and efficiently share location-sensitive data; e.g., mobile users subscribe to congestion information along one's driving route, and when congestion happens in a certain area, apps notify an event to all the subscribed users.

Given that structured P2P overlays or distributed hash tables (DHTs) generally provide better routing/searching performance with much lower overhead when compared with structureless P2P overlays (e.g., Gnutella), we consider DHTs as an underlying routing mechanism for data retrieval. However, conventional DHT-based storage systems (e.g., CFS [6] and PAST [30]) use consistent hashing which breaks up content locality, making it hard to support location-aware services. In our scenarios, for instance, two data items which are close in key space, e.g., one item at location $(1, 1)$ and the other item at location $(1, 2)$ may be far apart in DHT key space, requiring two individual unicast messages to retrieve them (no *geographic* locality).

To preserve content locality, several solutions have been proposed so far, such as SkipNet [13], D2 [24], Mercury [3]. These DHTs, however, are optimized for single attribute queries and cannot efficiently handle multi-attribute queries (2D geographic operations in our case). For instance, Mercury creates a distributed hash table for each attribute, and data (or pointers) are sent to *all tables* such that a query can be resolved only using a single table. For a given X coordinate, an overlay node is responsible for keeping the entire Y coordinate space, which makes the load balancing and storage management very difficult. Moreover, it is not clear how to support *location-aware publish/subscribe services* using these proposals — so far existing DHT-based publish/subscribe protocols [38, 27, 4] use consistent hashing, failing to preserve content locality.

1.2 Contributions

In this paper, we propose *GeoServ*, a scalable sensor networking platform for large-scale participatory sensing on the move. GeoServ linearizes 2D geographic space into fixed size grids (say 100m$\times$100m grids) with the Hilbert space filing curve (HSFC) [15] and use this grid ID space as DHT key space to preserve content (geographic) locality. Location-sensitive sensor data are geographically partitioned across overlay nodes using the grid ID space, and mobile clients can publish/access data through the overlay network. Based on this, we significantly extend Symphony DHT [19] and propose GeoTable, a location-aware data retrieval service over HSFC space which preserves geographic locality of 2D range queries, which is analogous to geocasting (or geographic routing) in wireless mobile ad hoc networks. Moreover, we propose *GeoPS* which supports "location-aware" publish/subscribe services such as sharing traffic information with a group of mobile users. The following are the main contributions of the paper:

- We propose *GeoTable*, a location-aware sensor data retrieval method. It reduces routing latency and handle load imbalances induced by skewed distribution of mobile users (e.g., large cities vs. rural areas). We provide a proof that GeoTable can preserve geographic locality.

- We propose *GeoPS*, a location-aware sensor data publish/subscribe method. GeoPS builds and manages a content distribution tree to the subscription group members, which preserves the content locality. We provide a proof for the efficiency guarantee of GeoPS.

- We evaluate effectiveness of GeoTable and GeoPS through extensive evaluation study.

The rest of the paper is organized as follows. We present the GeoServ system overview in Chapter 2. Location-aware routing and publish/subscribe of GeoServ are described in Chapter 3 and Chapter 4, respectively. We then discuss our evaluation method and results in Chapter 5. We present the related work in Chapter 6 and then conclude in Chapter 8.

CHAPTER 2

GeoServ Overview

This section first summarizes location-aware services that can be supported on GeoServ and discusses GeoServ system architecture.

2.1 Location-aware Services on the Road

Drivers may want to know various events or conditions on the road to make an informed decision; e.g., avoid driving on the roads with bad pavement conditions. The following are service examples:

- *Street-level Traffic Flow Information* Vehicles as sensors collect GPS measurements and can share data using wireless connectivity. This mobile sensor approach greatly extends coverage (over traditional fixed sensors), thus enabling *street-level traffic flow estimation*.

- *Vehicular safety warning services*: Non-time critical safety warning messages can be timely delivered over 2/3G networks (due to large RTT). This warning service can be treated as a *virtual electronic sign* on the road. Mobile users can detect safety related information using their smartphones either through manual blogging or automated filtering.

- *Ride quality monitoring*: Municipalities have been profiling roads using expensive profiling devices mounted on the vehicles that use GPS, accelerometer/laser

sensors [40]. Researchers have recently considered using less expensive commodity sensors and smartphones (e.g., pothole detection [39, 20]).

2.2 System Architecture

GeoServ is a two-tier sensor networking platform which exploits the P2P Internet infrastructure. We assume that users who opt in to GeoServ services install GeoServ software in both PCs/laptops and smartphones. Internet servers installed on PCs/laptops provide a distributed P2P sensor storage over the Internet, through which mobile clients can publish/access location-sensitive sensor data (see Figure 2.1). Since most sensor data is generated on the roads (and most queries are location sensitive), we assume that the primary search key (or key space) is geographic location. This is also true for most large-scale web applications [41, 42] which only support single key/table DB operations. For efficient data management, we divide the physical area into smaller grids where the grid size is a system parameter (say, 100m$\times$100m). The sensor data is then horizontally partitioned based on grid points, each of which is maintained by a corresponding Internet server. We assume that both Internet servers and mobile clients have map data of the associated grids, e.g., Tigermap [35]. We exploit the computation power of mobile nodes to reduce upload traffic whenever that is possible. Mobile users carry raw sensor data, and the processed data (e.g., average reading, image thumbnails) will be published to the P2P sensor storage.

Given this system model, the design requirements can be summarized as follows. First, apps should be able to seamlessly access the sensor storage that is horizontally partitioned based on geographic locations. This can be supported using specialized P2P routing protocols such as Key-Based Routing (KBR) [43] via Distributed Hash Table (DHT) [26, 19, 3], assuming geographic location-based key space. For instance, a traffic information service application may use geographic multicast routing to ac-

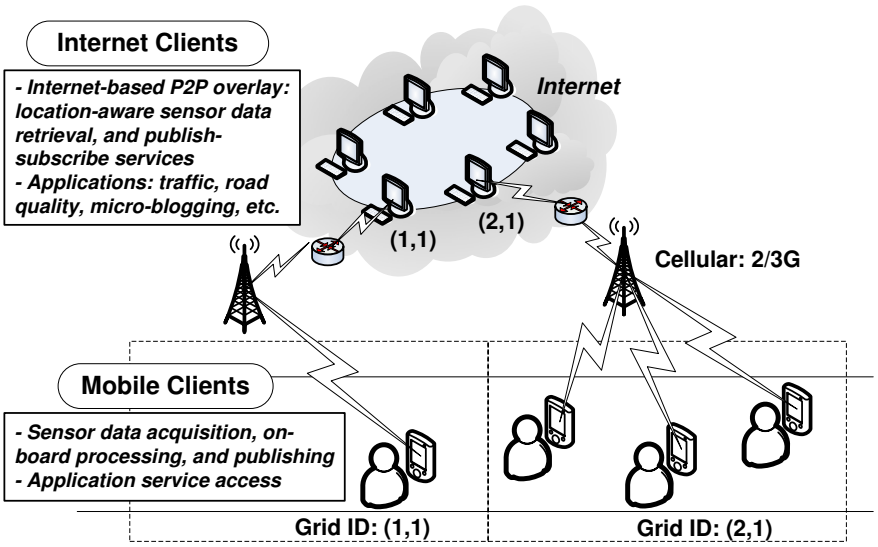

Figure 2.1: GeoServ, a two-tier sensor networking architecture

cess information around its neighboring regions. Second, heterogeneous distribution of road topology causes skewed distribution of vehicles (e.g., a big city vs. a remote village). As a result, the sensor data traffic is highly skewed. The system should balance the system load, by dynamically provisioning more nodes in the congested regions. Third, apps may require publish/subscribe services such as sharing traffic information with a group of mobile users. The routing layer must provide an efficient publish/subscribe routing method. Finally, since mobile users continually publish location-sensitive data, WiFi-like mobile user association based on one's current location is a must. If a mobile user does not directly contact the storage server which manages one's current location, there will be redundant data transfer, incurred by geographic routing. Since vehicles are highly mobile, the system must efficiently support dynamic handoffs from one geographic region to another.

GeoServ addresses the above requirements. GeoTable provides Key-Based Routing (KBR) whose key space is geographic location [43]. Apps running on Internet servers and mobile clients communicate one another via GeoTable. GeoServ provides

an efficient publish/subscribe routing method called GeoPS based on the geographic routing via GeoTable. Other components of GeoServ includes (1) GeoServDB which manages sensor data of the grid space which the server currently is in charge of and supports a remote database access protocol like Open Database Connectivity (ODBC) over GeoTable; and (2) GeoServMobile which provides control and transparent access of sensing resources and supports various application filters that process raw sensor data based on application demands and publish sensor data to GeoServDB over GeoTable. In this paper, we mainly illustrate the key components, namely GeoTable and GeoPS and leave the discussion of other components due to the page limit.

CHAPTER 3

Location-aware Sensor Data Retrieval Service

We illustrate the Hilbert space filling curve, review routing semantics, present a detailed routing mechanism and its improvement techniques (e.g., delay and load balancing) and prove that the Hilbert curve-based approach preserves content (geographic) locality.

3.1 Hilbert Space Filling Curve

In GeoServ, we divide the geographic area of interest into fixed size grids (say $R \times R$), and there are total $2^M \times 2^M$ grids where M is the smallest exponent that covers the entire area. For example, assuming that the size of the contiguous U.S. is approximated as $3000km \times 3000km$, it can be represented using $2^{13} \times 2^{13}$ fixed grids where R is given as $1km$. Given this 2D grid space, we use the Hilbert space filling curve, a linear mapping function where successive points are nearest neighbors in the 2D grid. The construction of the Hilbert curve is recursively defined. The basic mapping in Figure 3.1(a) is replicated in four quadrants. The lower left quadrant is rotated clockwise 90 degrees, the lower right quadrant is rotated anti-clockwise 90 degrees, and the sense (i.e., direction of traversal) of both lower quadrants is reversed. The two upper quadrants have no rotation and no change of sense (see Figure 3.1(b)). Thanks to the recursive construction above, the linear ID along the curve for any given grid point (x, y) can be easily calculated. The linear coordinate is augmented with two

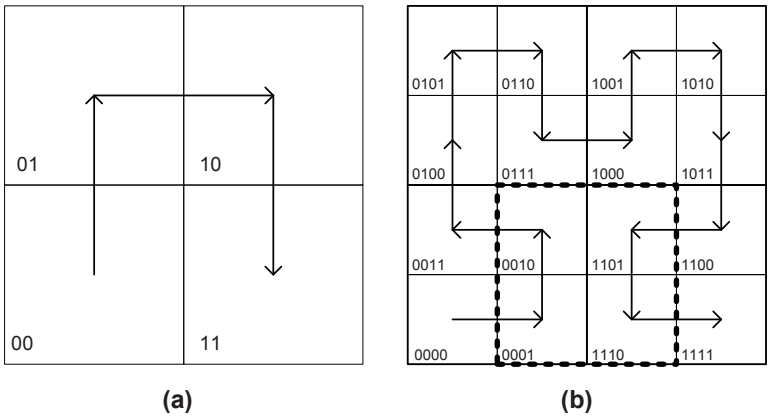

Figure 3.1: Recursive construction of the Hilbert curve

bits at a time for each recursion; i.e., the most significant bits (MSBs) of the x and y give the 2 MSBs of the resulting linear coordinate, along with the rotation and sense to be applied to the rest of the computation. Readers can find the detail algorithms and comparisons with other linearization schemes in [15].

3.2 Routing Semantics

The above definition enables GeoTable to map a 2D grid coordinate (x, y) to a D-bit numeric address on the Hilbert curve. Location-aware applications running on top of GeoServ (or mobile users) can access sensor data generated from a remote region which can be a grid point, or multiple contiguous grid points denoted using a line/curve segment or a generic polygon formed by a set of line segments; e.g., apps want to fetch GPS readings originated from a set of road segments to calculate the average speeds in that area. Depending on how many overlay nodes are deployed and the size of a queried region, the region could be covered by a single overlay node, or by multiple overlay nodes. Thus, this routing strategy can be treated as *geocasting* (which is widely used in wireless mobile ad hoc networks) because destination nodes are implicitly set

by specifying a target region — query packets are delivered to a group of overlay nodes that cover the region.

3.3 Geocasting To a Single Grid Point

Since there is only a single overlay node that covers a given grid point, this can be seen as geographic *unicast* routing of a query packet. The unicast routing exactly follows the routing policy Symphony DHT [19] that uses Kleinberg's Small World phenomenon. For completeness, we present Symphony DHT. In Symphony, a node joins the network by picking up a random ID (equivalent to a numeric grid ID on the Hilbert curve). Every node maintains two short links to one's 1-hop neighbors and $k \geq 1$ long distance links. Long distance links are constructed as follows. Consider a node whose ID is n and is responsible for the range $[\ell, r]$. Let I denote the space of D-bit Hilbert curve, $[0, 2^D)$. For each link, a node draws a number $x \in I$ based on the harmonic probability distribution function: $p_n(x) = 1/(n \log x)$ if $x \in [2^D/n, 2^D)$. Kleinberg showed that such a construction allows us to greedily route packets to a random node (i.e., in each hop, follow a long link that is closest to the destination) in $O(\log^2 n)$ hops on average [19]. Figure 3.2 shows an example. Readers can find the details of join/leave functions in [19].

3.4 Geocasting To Multiple Grid Points

The current GeoTable prototype supports simple rectangular area-based addressing as $\{(x1, y1), (x2, y2)\}$ that denotes lower left and upper right corners, respectively. Our system can be extended to support more complex shapes using polygons, defined by a set of line segments. For a given rectangular area, nodes first translate the area to find a set of *ordered* segments on the Hilbert curve where a segment is composed of

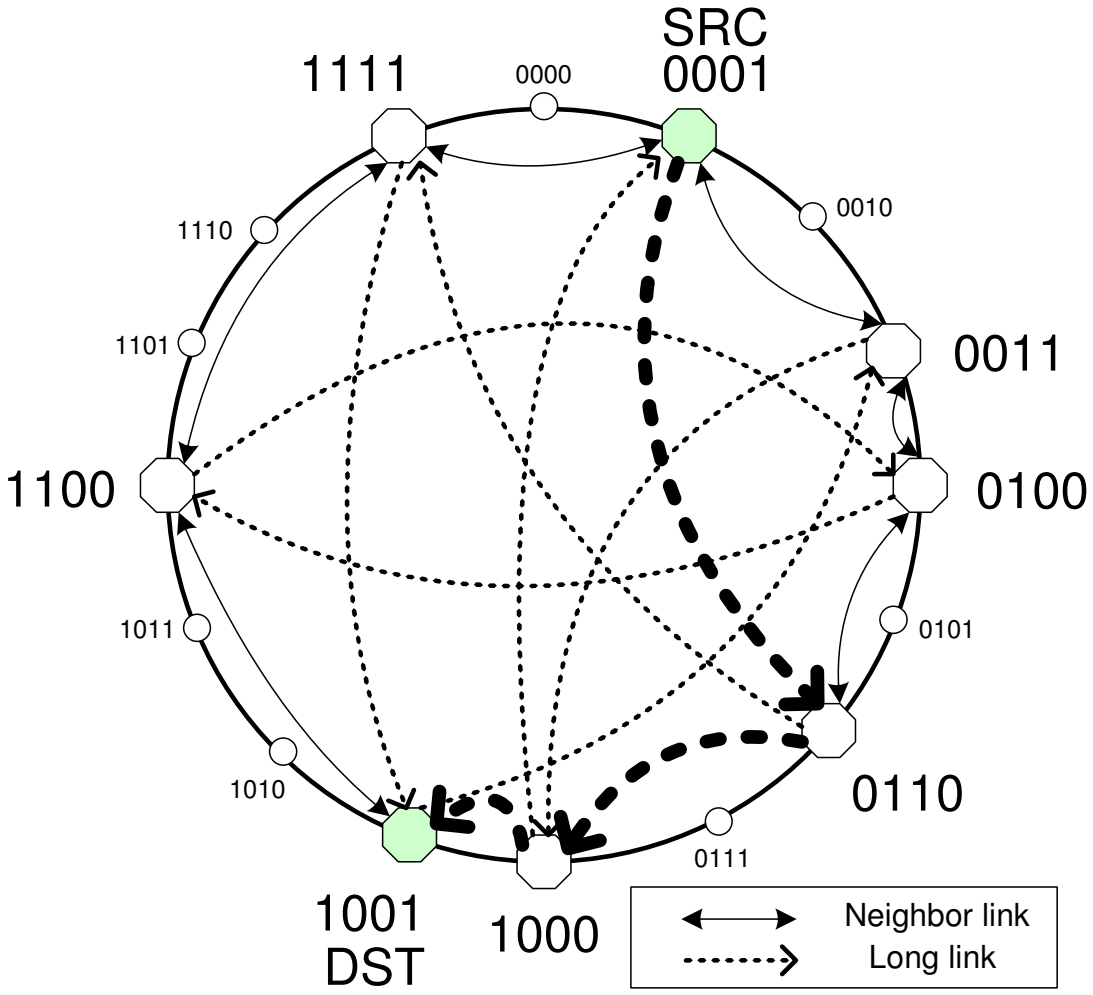

Figure 3.2: Illustration of unicast routing: Each node has neighbor links and one long link to a random location. Source located at 0001 sends a packet to the destination node located at 1001. It uses a long link to 0110 followed by neighbor links to 1000 and 1001 sequentially (thick dotted lines).

contiguous grid points. Consider a rectangular area in thick dotted lines in Figure 3.1(b). Recall that the Hilbert curve looses some of data locality (50% to be precise as the curve connects only two of its neighbors). Thus, it requires a set of segments to cover a rectangular area. In Figure 3.3, we have two segments, namely $\{[0001 - 0010], [1101 - 1110]\}$. Given this, geocasting is straightforward. For a given ordered list, a packet is first routed to the head of the first segment (e.g., 0001) using the aforementioned unicast routing scheme. By following the neighbor links, the first segment is scanned. Since an overlay node typically covers a span of key space, this is simply local scanning. After this, the query packet will be forwarded to the head of the next segment and another scan will be performed. This process repeats until we cover all the segments in the list.

We show that the expected routing cost of geocasting depends on the size of the target area. The following theorem shows that once a query is routed to the target area at the cost of $O(\log^2 n)$, and it can be resolved locally with a set of nodes that covers the queried area (which is defined as the least Hilbert curve mapping area in the theorem), meaning that routing is done only using a much smaller set of nodes (instead of using all nodes n).

Theorem 1 *The expected path length of geocasting to a square area-based range query of size $\sqrt{s} \times \sqrt{s}$ grids is upper bounded by $\Theta(\log^2 n + \sqrt{s} \log^2 S + s)$ where n is the number of overlay nodes, and S is the number of overlay nodes in the least Hilbert curve mapping area that contains the queried area.*

Proof 1 *We consider the worst case scenario of placing the square grids at the center of the geographic area. In Figure 3.1, for instance we place 2×2 square grids in the area: $\{0010, 1101, 1000, 0111\}$. We divide the region into four identical sub-regions whose size is $\sqrt{s}/2 \times \sqrt{s}/2$. In each sub-region, we can find a set of ordered segments*

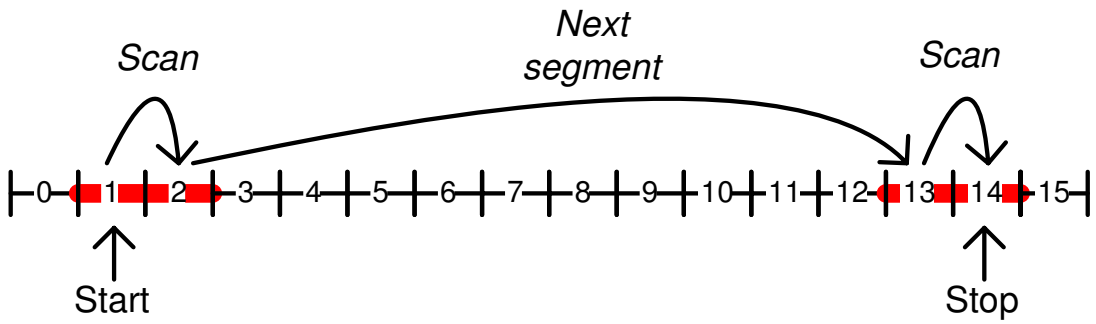

Figure 3.3: Ordered segments for geocasting in Figure 3.1(b). Query resolution is performed sequentially; e.g., the query packet is forwarded to the first segment which is then scanned; after this, it is forwarded to the next segment for scanning.

as in Figure 3.3. The request will be first routed to the first segment in the bottom left sub-region at the cost of $\Theta(\log^2 n)$ *[19].*

For a given sub-region, we need to prove that a request packet does not leave the local area that contains the sub-region. In GeoServ, the entire area is divided into $2^M \times 2^M$ *grids where the size of a grid is given as* $R \times R$*. The construction property of the Hilbert curve shows the following. Basic mapping in Figure 3.1(a) whose size is* 2×2 *grids is replicated in four quadrants, generating* $2^2 \times 2^2$ *grids. The resulted mapping is recursively replicated and will generate* $2^M \times 2^M$ *grids. For a given sub-region, we can find the least size mapping of* $2^i \times 2^i$ *that contains the region: i.e.,* $K = \arg\min_i\{2^i \times 2^i \geq s/4\}$*. Assuming that overlay nodes are uniformly distributed, we can find the expected number of overlay nodes in that area:* $S = \frac{2^K R \cdot 2^K R}{2^M R \cdot 2^M R} \cdot n = 2^{K-M} \cdot 2^{K-M} \cdot n$*. Routing is localized within these overlay nodes (that are responsible for the least size mapping area). Hence, the routing cost is upper bounded by* $\Theta(\log^2 S)$*.*

Now, let L *denote the number of contiguous segments within the sub-region. We need a look-up for each segment within which we can deliver the packet by following 1-hop neighbor links. Thus, the expected path length of geocasting is upper bounded by*

$\Theta(L \log^2 S + s - L)$; i.e., L unicasts and $s - L$ hop-by-hop packet transfers (scanning). The upper bound of the number of segments L can be found as follows. Let $s_i = \{b_i, e_i\}$ denote the i-th segment that ranges from b_i to e_i in Hilbert Space. By the construction of the Hilbert curve, b_i and e_i for any i can only be located at the boundary of the square area. The circumference of the sub-region is $2\sqrt{s}$, and there could be at most $\sqrt{s}$ segments by definition.[1] *The number of segments is simply upper bounded by $\Theta(\sqrt{s})$. Therefore, the expected path length within a sub-region is $O(\sqrt{s} \log^2 S + s)$. Since we only have a constant number of sub-regions (here four regions), the overall routing cost is simply bounded by the cost of unicast to each sub-region and the routing cost within a sub-region: i.e., $\Theta(\log^2 n + \sqrt{s} \log^2 S + s)$.*

3.5 Concurrent Geocasting

Since we have to scan segment by segment sequentially, geocasting may result in too much delay. Individual geocasting to each segment can reduce the delay, but the overall routing overhead is significantly increased (by a factor of the number of segments). There is a simple, but effective solution that can achieve the comparable result with minimal extra overhead. The idea is to utilize how the Hilbert curve is constructed; i.e., for each recursion, the linear coordinate is augmented with two bits at a time as shown in Figure 3.1 (e.g., 01 to [0100–0111]). This construction process can be represented using a tree in linear time [15] (see Figure 3.4). This tree allows us to perform binary search (with a fan-out of four). Any level is sub-space of its parent level. The goal is to follow the tree structure for routing. Upon finding the longest prefix where there are segments of interest (i.e., a region that covers those segments), we select the starting grid point in that region (say, grid 0001 in Figure 3.4) and tunnel a packet toward that location. Here, by tunneling we mean the packet is encapsulated

[1] *Readers can find the exact number of segments for $s = \Theta(1)$ in [15].*

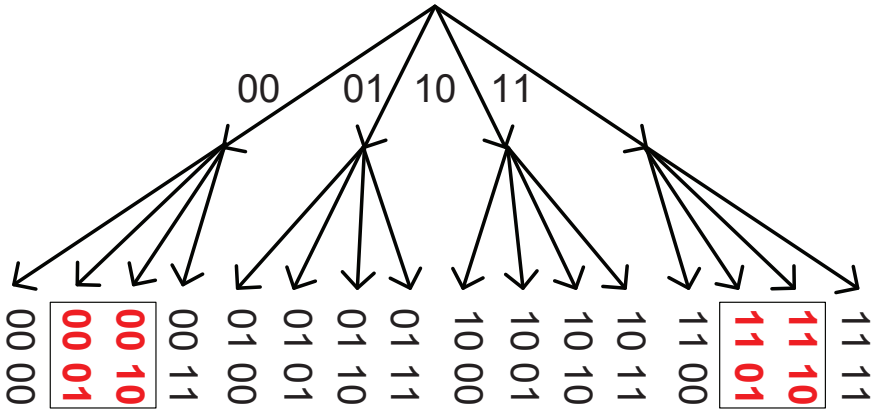

Figure 3.4: Tree representation of Hilbert curve construction

in the new packet destined to a new location. Once the packet reaches that region, it is forwarded to its sub-levels (via tunneling toward the starting grid point in each sub-level region). In Figure 3.4, we have two grid points, namely 0001 and 1101 that are the starting grid points of sub-regions, and the query packet is individually routed it to the head of each segment (instead of sequential scanning). Compared to individual goecasting to each segment, concurrent geocasting sends a query packet to a region once, and then it will be recursively forwarded to sub-regions in a localized fashion, which significantly reduces the overall routing overhead (as illustrated in Theorem 1).

3.6 Dynamic Load Balancing

Due to skewed distribution of mobile users (e.g., highly dense urban cities vs. low density rural areas), we should balance loads by placing more overlay nodes in those dense areas. Overlay nodes located in the less loaded regions can be dynamically moved to the overloaded regions (known as leave-join method [9]). GeoTable uses Mercury's load balancing mechanism to preserve locality of content retrieval [3]. This load balancing algorithm is simple, fully distributed, and converges quickly. Mercury

uses the approximate *node distribution histogram* over the key space (i.e., for a given key range, how many nodes are there). Based on this, Mercury periodically re-arranges routing tables to preserve the logarithmic DHT operations. The key idea is to arrange the routing table based on *node-based distance* instead of key space. In GeoTable, when a long link is drawn, we use this histogram to permit biased selection from the dense regions (instead of uniform distribution used in the original Symphony).

CHAPTER 4

Location-aware Publish/Subscribe Service

We have discussed geocasting in the previous section where a one-shot query is routed from an application to the region of data sources. In this section, we present the support for *subscription queries* of multiple users who are interested in data updates on a target region: e.g. traffic information on the commute route. We propose GeoPS, a *publish/subscribe service* where the data updates on a region are *published* to all users who have *subscribed* to that region. This section details GeoPS's locality-preserving multicast tree construction and management methods and their performance bounds via mathematical proofs.

4.1 GeoPS Overview

Given that majority of data consumers of location-sensitive data will be located near the area where the data are generated (e.g., traffic information on the commute route), the key design issue is to build a multicast tree a multicat tree that exploits the geographic locality of the group members.[1] Our approach called GeoPS is inspired by hierarchical geographic location services (HGLS) in mobile ad hoc networks such as GLS [17] and HIGH-GRADE [37] where the entire area is recursively divided into a hierarchy of smaller grids, and mobile users' current locations are efficiently tracked

[1]Note that if all subscribers are originated from a single region, we can easily implement the service using geocasting. In this section, we focus on more general scenarios where subscribers are from a set of non-contiguous regions.

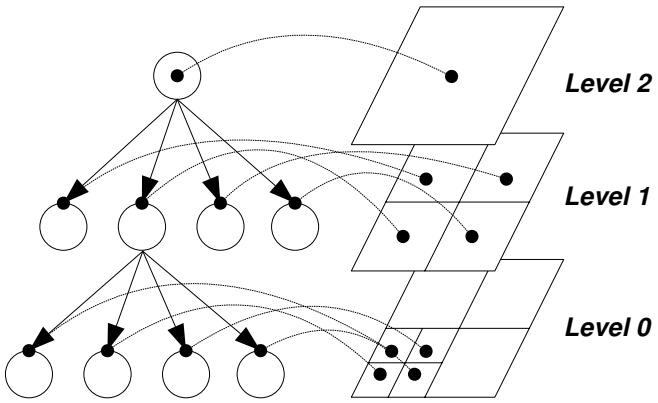

Figure 4.1: Illustration of a hierarchical geographic location service

under the geographic hierarchy. The key idea of GeoPS is to build a multicast tree over this geographic hierarchy and to use our geocasting algorithm over the tree to preserve geographic locality.[2] This is a major departure from existing DHT-based multicast solutions (e.g., Bayeux, Scribe) that *destroy locality* using consistent hashing and randomly distribute geographically correlated subscribers across the entire key space.

4.2 Review of HGLS

In mobile ad hoc networks, a location service keeps track of mobile nodes' current locations and lets mobile nodes to query the current location of an arbitrary node (e.g., to use it for geographic routing). In HGLS, a geographic hierarchy tree is constructed by recursively dividing the entire area into a hierarchy of smaller grids. Figure 4.1 shows an example where the root of a tree covers the entire network area (level 2), and each of its children covers a sub-region whose size is one fourth of the network area (level 1). For each level i, nodes have a pair of common hash functions $h_{i,x}(id)$ and $h_{i,y}(id)$ that map a node ID to a geographic coordinate (x, y) at level i. For a given

[2]We can easily prove that multicast routing is localized as we use our geocasting algorithm presented in the previous section (using similar proofs used in Theorem 1 and in Yu et al. [37]).

node whose ID is ℓ, one node located around the location $(h_{i,x}(\ell), h_{i,y}(\ell))$ is chosen as node ℓ's *location server* at level i. The node ℓ publishes its current location to the leaf region (level 0 area where the node is currently located), and all its upper level location servers along the *single path* of the geographic hierarchy tree are initialized as rendezvous points. Note that up-to-date location information is stored locally (at level 0 servers where the node is currently located), and rendezvous points are updated only when the node crosses the level boundary.

Given this, any node can send a location query for the node ℓ as follows. The query is first routed to location servers around $(h_{0,x}(\ell), h_{0,y}(\ell))$ in the level 0 area where the querying node is located. If the level 0 location servers do not have the information, the query is routed to the level 1 location servers for node ℓ that are located around $(h_{1,x}(\ell), h_{1,y}(\ell))$. The process is repeated until it finds the location servers at level i that have the path information (i.e., rendezvous point). The query then traverses down the hierarchy to find the exact location available at the level 0 location servers. In Figure 4.2, node A's current location is stored in node L0:000, and we have two rendezvous points at Level 1 (L1:00) and Level 2 (L2:0). Node D can find node A's location as follows. It queries node D's Level 0 sever (L0:033), but it fails to find the information. It tries Level 1 server (L1:03), fails, and finally finds a rendezvous point at Level 2 (L2:0). By following the links along the rendezvous points, we can find node A's current location at node A's Level 0 server (L0:000).

4.3 Multicast Tree Construction

In GeoPS, each group has a unique group ID which is the hash of the group's textual name concatenated with random string, e.g., hash("congestion at grid x, y + !?*2@"). This group ID is used for building a multicast tree per group, similar to node ID in HGLS. For a given $groupID$, we construct a multicast tree rooted at the rendezvous

point in level M (top level) using HGLS-like geographic partitioning as follows. Recall that the geographic area is divided into $2^M \times 2^M$ fixed grids where each grid is given as $R \times R$. At each hierarchy level i, we have a rendezvous point located at $(h_{i,x}(groupID), h_{i,y}(groupID))$. This location is mapped to Hilbert curve space, and the overlay node with node ID closest to this mapped address is selected as a rendezvous point in the overlay network.

When a node joins, the join request message propagates to upper levels starting from level 0 (where the node is currently located), and at each level, a node stores subscription information in the routing table for $groupID$. Note that routing to a rendezvous point is done via geocasting (with a single grid point) described in the previous section. When the message finds that there is an existing subscription entry for a given $groupID$, the rendezvous points in its upper levels were already initialized by other group members (a subscription entry of the group is already present). Thus, the message stops there, and the child node is simply added to the table (i.e., a direct path to the child). In Figure 4.2, when mobile user A joins, the subscription message is installed at L0:000, L1:00, and L2:0 sequentially. We repeat the same process when user B, C, D join, and Figure 4.2 shows the resulting multicast tree (dark gray nodes have the subscription entry). Now, when a new mobile user N joins, its subscription message will be installed at L0:003, and it will then be forwarded to L1:00. This level 1 node finds that there is an existing subscription entry set by mobile user A, and the subscription message stops propagating.

The leave process is similar to the join process. When a mobile node gracefully leaves the system, it sends a leave message to upper levels to remove the subscription information. In each level, if there is no more subscription entry for a given group, the message is sent to the upper levels sequentially.

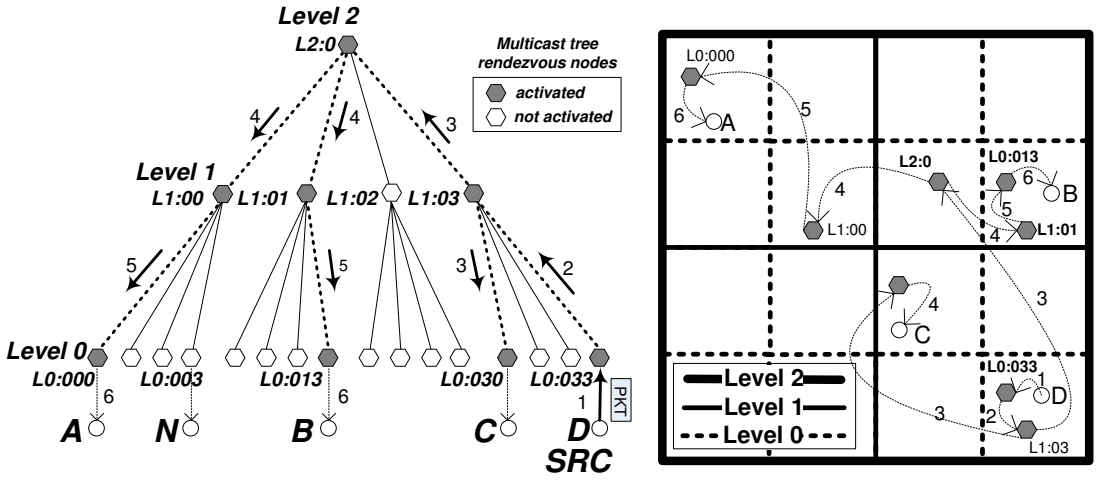

Figure 4.2: Subscription-based multicast example: D (source) and A, B, C (members)

4.4 Mobility Handling

A mobile client's subscription needs to be updated (to upper layers) whenever the client crosses the level boundary (via explicit leave and join). When there is a single subscriber for a given group, and this client crosses level m boundary, all rendezvous points at and below level $m+1$ need to be updated. In Figure 4.2, when mobile client C moves to the adjacent grid on the left (crossing level 1 boundary), rendezvous points at level 0, 1, 2 are updated; and when mobile client D moves to the adjacent grid upward (crossing level 0 boundary), those at level 0, 1 are updated. Interestingly, given that an overlay node typically keeps a fraction of grid space, one possible optimization would be not notifying updates as long as a mobile client is associated with the same overlay node.

4.5 Data Update Publish

A source can send a message along the tree starting from the leaf node (Level 0) and traversing toward the upper levels. When there is a matching subscription in an

intermediate node, it sends the message to each child in the subscription entry from which the packet starts traversing down the tree. Figure 4.2 shows an example. We have four members (mobile clients): A, B, C, and D. Source D sends the packet to L1:03 (step 1). L1:03 sends it to both L0:030 and L2:0 (step 2 and step 3). After this, L2:0 sends it to L1:00 and L1:01 (step 4). L1:00 and L1:01 send it to L0:000 and L0:013 respectively (step 5). They deliver the packet to A and B (step 6).

4.6 Minimum Depth Configuration

In practice, the number of overlay nodes is much less than the total number of grids (i.e., entire key space). Thus, the lowest depth should be configured as L_{M-K} rather than naught (where M is the maximum level, and K is the depth of a multicast tree) such that there is at least one overlay node in that region; otherwise, we are storing redundant rendezvous points (in sub-trees below the lowest level) to the same overlay node. Note that we can configure the depth in a distributed manner by utilizing the node distribution histogram and network size estimates from GeoTable's load balancing. The following theorem shows that we have $K = O(\log n)$ under uniform node distribution.

Theorem 2 *The depth of a multicast tree is bounded by $O(\log n)$ under uniform node distribution where n is the number of overlay nodes.*

Proof 2 *Assuming that overlay nodes are uniformly distributed, the expected number of overlay nodes in the lowest level is given as $\Theta(\frac{2^{M-K}R \cdot 2^{M-K}R}{2^M R \cdot 2^M R} \cdot n) = \Theta(\frac{n}{2^{2K}})$. Now, we want to assure that the lowest level is covered by at least a constant number overlay nodes, i.e., $\Omega(1)$. By considering this equality condition, we have $K = O(\log n)$.*

Note that the lowest level configuration has a positive impact under high mobility scenarios. If all $M+1$ levels are used for building a multicast tree, a mobile user's subscription needs to be frequently updated (to upper layers) whenever the node crosses the level boundary. When the lowest level is configured as L_{M-K}, this frequent update problem can be easily mitigated.

4.7 Load Lalancing

Some regions could be highly populated with mobile clients because dynamic load balancing in GeoTable re-organizes overlay nodes such that those regions are served by more number of nodes. In GeoPS, this can be easily handled by hierarchically re-partitioning the highly populated area (say R more levels result in the minimum level of L_{M-K-R}), which can be done by utilizing the node distribution histogram and network size estimates from GeoTable's load balancing.

4.8 Membership Management Cost

The following theorem proves that membership subscription can be performed efficiently. The cost of membership updates due to mobility has the same bound and can be proved similarly.

Theorem 3 *The routing cost of membership subscription is bounded by* $\Theta(\log^3 n)$ *under uniform node distribution where* n *is the number of overlay nodes.*

Proof 3 *The lowest level is given as* L_{M-K}*. The worst case happens when there are no initialized rendezvous points along the* single path *of the tree with depth* K*. This is also true when we add a new group. In Theorem 1, we show that the routing cost with a subset of nodes in a given area is bounded by* $O(\log^2 S)$ *where* S *is the number of*

overlay nodes in that area. Hence, routing the join message from level $i-1$ *to level* i *takes* $O(\log^2 \frac{n}{2^{2i}}) = O(\log^2 n - i^2)$ *hops where* $i = O(\log n)$ *as shown in the above theorem. There are* $K+1$ *levels. The total routing cost is simply* $(K+1)\log^2 n - \sum_0^K i^2 = \Theta(K \log^2 n - K^3/3)$. *Since we have* $K = O(\log n)$*, the overall cost is bounded by* $\Theta(\log^3 n)$.

CHAPTER 5

Evaluation

5.1 Simulation Setup

We implement an event-driven discrete-time simulator where each overlay hop takes a unit time. For the sake of a large network simulation, our simulator does not model any queueing delay at intermediate nodes or packet loss on links. The simulator is implemented in C# and supports dynamic node generation/join/leave, load balancing, and publish/subscribe features. For system evaluation, we consider a large-scale participatory vehicular sensing scenario where mobile users in the cars participate in vehicular sensing projects (e.g., traffic information sharing and road condition monitoring). For realistic mobility generation, we use VanetMobiSim that simulates macro- and micro-mobility patterns in urban environments [12]. Macro-mobility deals with road topology/structure and traffic signs (stop signs, traffic lights, speed limits), and micro-mobility models the speed and acceleration of each vehicle. For mobile scenarios, we use this mobility trace for the duration of 120s. We use the network area size of 12800m$\times$12800m. Two types of road topologies are used: Manhattan and Westwood. In the Manhattan topology, the grid size is set to 50m$\times$50m (i.e., 256$\times$256). The Westwood topology from Tigermap (TGR06037, Los Angeles) represents the area in the vicinity of the UCLA campus. We discretize the network area into grids of size 50m$\times$50m for the Hilbert curve-based linearization, resulting 256$\times$256 grids. Geographic range queries are made by specifying a square area (e.g., 4$\times$4 grids). Each

mobile node reports sensor data to its associated overlay node every second. The size of data is set to 128 Bytes (e.g., GPS sample, timestamp, accelerometer samples). We assume that each node knows its accurate geographic coordinate and thus can dynamically change their associated overlay node without any errors (e.g., no bouncing at the boundary). In GeoTable, the number of long links is set to five, as recommended in Symphony DHT [19]. Unless otherwise mentioned, for each configuration we report the average value of 30 runs.

5.2 Simulation Results

5.2.1 Location-aware Data Retrieval

GeoTable preserves geographic locality due to the construction of the Hilbert curve, and packet forwarding happens within the area of interest (usually, small fraction of the entire key space). To clearly show this, we place a querying node at a grid point $(0, 0)$ and measure the hop count of a remote query with the square area of size 4×4, by varying the location of the query's left-lower corner grid from (0,0) to (252,252). The number of overlay nodes is set to 1000. In Figure 5.1, we plot average hop counts over the 256×256 grids. For clarity, we present the average hop count within 16×16 grids in the figure. The figure shows that as distance from $(0, 0)$ increases, the hop counts increases (getting brighter). The reason why it shows non-uniform colors is that some degree of locality is lost after linearization, and long links are randomly assigned. In general, locality is preserved at the higher level thanks to the recursive construction property of the Hilbert curve; the average hop count increases as we move clockwise as in Figure 3.1.

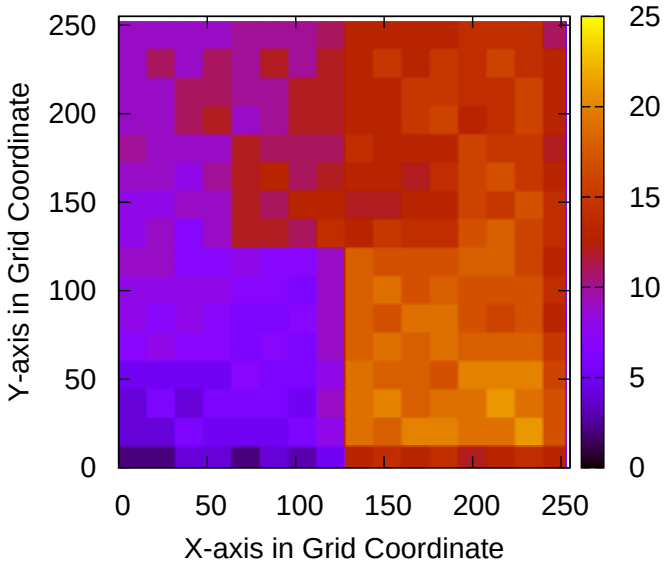

Figure 5.1: Locality of geocasting: routing cost from (0, 0) to (X, Y) in grid space

5.2.2 Impact of Query Region Sizes and Concurrent Geocasting

We show the sensitivity of routing cost with different query region sizes: 2×2 and 6×6. The querying node is located at grid (0, 0), and it sends queries with different area sizes. We vary the distance between the querying node and the query region from 0 to 255. We compare the performance of various schemes: conventional DHT with consistent hashing (CH), sequential geocasting (SG), and concurrent geocasting (CG). In CH, each grid is randomly mapped into the key space; 2×2 and 6×6 queries require 4 and 36 unicasts respectively.

Figure 5.2 shows that as the grid distance increases, the average hop count of SG and CG increases due to locality (and CG is better than SG), whereas that of CH does not change with the distance due to lack of locality. One caveat is that the delay improvement of CG against SG comes at the cost of more packet forwarding, yet that is far more efficient than issuing individual unicast to each segment. Figure 5.3 plots the relative overhead of concurrent geocasting (in log scale), showing that the total

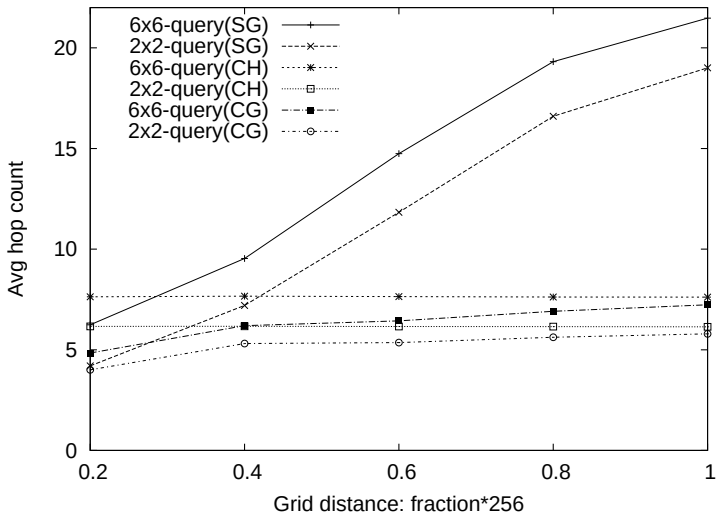

Figure 5.2: Average hop counts with different region sizes located at varying grid distances (SG: sequential geocasting, CG: concurrent geocasting, CH: consistent hashing-based DHT)

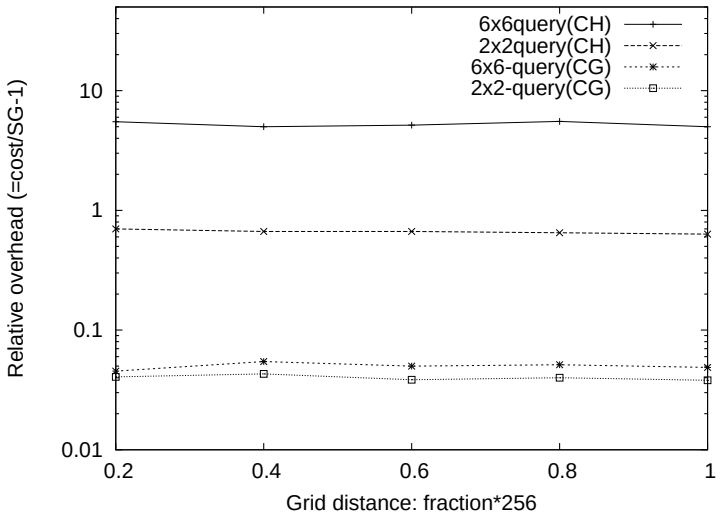

Figure 5.3: Relative overhead against sequential geocasting at varying grid distances (log scale in Y-axis)

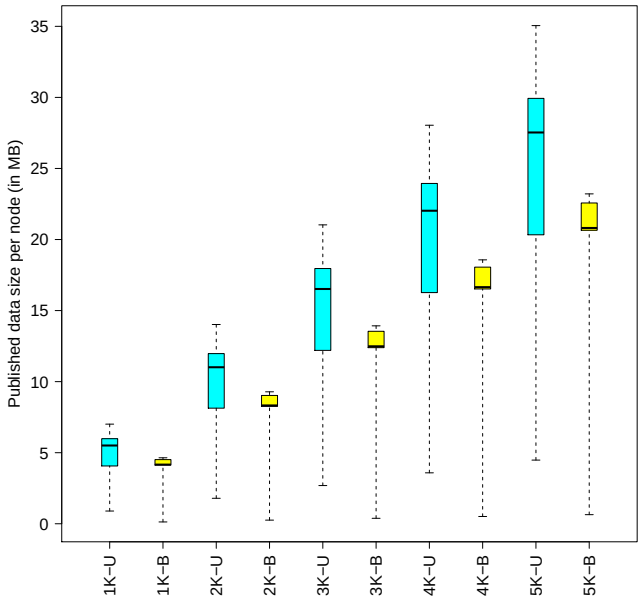

Figure 5.4: Total published data size per overlay node with different numbers of mobile clients (a boxplot shows min, 25%, median, 75%, and max; U: unbalanced, B: balanced, 1K: 1000)

routing cost does not largely deviate from that of sequential geocasting (as the target region size is much smaller than the entire map).

5.2.3 Load Balancing

Mobile clients publish sensor data to the overlay nodes. We study how heterogeneous distribution of mobile clients influences the overall load imbalance. We use the Los Angeles map to extract road topology information. The area size is 12,800m$\times$12,800m, centered at the UCLA campus. The northern parts of the area are residential area (low road density), whereas the southern parts are commercial districts (high road density). We use the grid size of 50m$\times$50m, and the area is composed of 256$\times$256 grids. We simulate different numbers of mobile clients from 1000 to 5000 with a gap of 1000

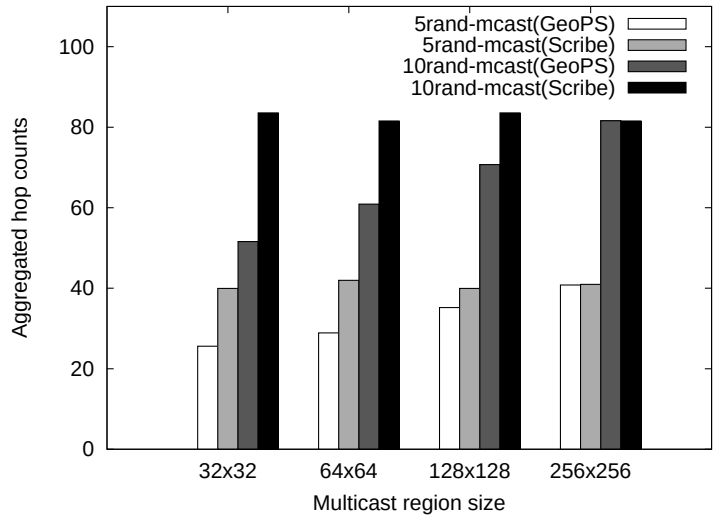

Figure 5.5: Subscription-based multicast routing comparison: GeoPS vs. Scribe

nodes. The number of overlay nodes is fixed to 1000 nodes. We measure the total published data size per node and draw a boxplot in Figure 5.4. In the figure, we show the case with load balancing and load unbalancing (denoted as B and U on the x-axis, respectively).. As expected, the total data size increases linearly, as the number of mobile clients increases (proportional to the number of mobile clients as we assumed data generation at constant rate). The case without load balancing shows much higher variation (in box plots) as opposed to the case with load balancing. There are still minor variations in the case with load balancing. This is because the system load of GeoTable becomes balanced after several iterations of leave/join-based load balancing operations.

5.2.4 Location-aware Publish/Subscribe Service

To show the geographic locality of our subscription-based multicast routing, we increase the width of the region in which all the multicast receivers lie. The region size ranges from 32×32 to 256×256. We vary the origin of the region from (0, 0) to the

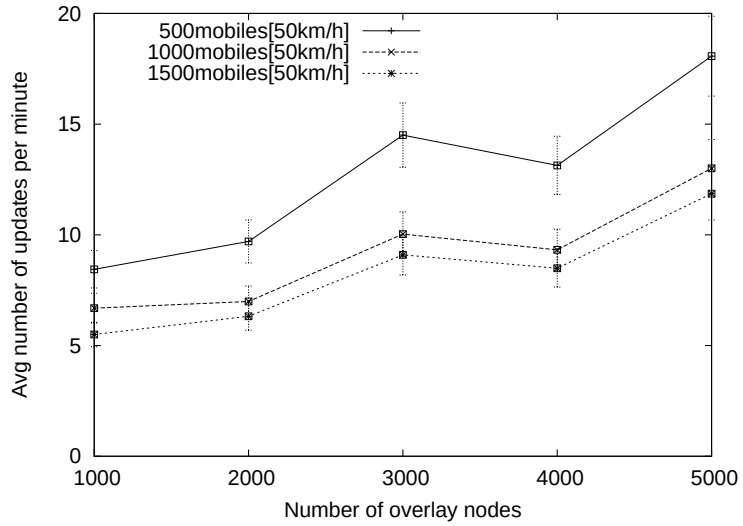

Figure 5.6: Subscription update frequency per node (per minute): Manhattan grids with different number of mobile and overlay nodes)

maximum allowable, e.g., for 32×32, it is (224, 224), and report the average hop count. We compare the performance of GeoPS with Scribe multicast routing protocol [4]. Recall that Scribe destroys the locality by using consistent hashing. We randomly choose 5 or 10 random grids within the region, and each grid is assigned with a subscriber (5 or 10 subscribers). We measure the aggregated number of hop counts to deliver a packet to all the multicast receivers. Figure 5.5 clearly shows that our multicast routing exploits the locality of receivers. As the area size (where the subscribers lie) increases, geographic locality among subscribers disappears, and accordingly, the cost of GeoPS increases, converging to that of Scribe in the case of 256×256.

5.2.5 Subscription Update Frequency with Mobility

We use different numbers of mobile clients, namely 500, 1000, and 1500, moving on Manhattan grids of size 256×256. We set the nodes' maximum speed to 50km/h. The number of overlay nodes is varied from 1000 to 5000. Figure 5.6 shows the average

number of subscription updates per node (per minute). The update frequency increases with the number of overlay nodes. As the number of overlay nodes increase, the average key space per node decreases, causing more updates. Note that because an overlay node typically keeps a fraction of grid space, a mobile client does not necessarily notify its update as long as it is associated with the same overlay node. The figure also shows that as the number of mobile nodes increases, the update rate decreases. This is because the more the number of vehicles, the lower the maximum vehicle speed in urban environments.

CHAPTER 6

Related Work

6.1 Internet-based Sensor Data Sharing

Internet-based approaches for *generic* sensor data sharing have a simple multi-tier structure such as ArchRock [36], SensorBase [29], and SensorMap [23], or semi-hierarchical structure such as IrisNet [8], and Global Sensor Networks (GSN) [1]. In ArchRock [36] and SensorBase [29], sensor data from a sensor network is aggregated at the local gateway and is published to the front-end server through which users can share the data. In SensorBase, back-end servers (called republishers) further process sensor data to enable sensor data searching. SensorMap [23] is a web portal service that provides mechanisms to archive and index data, process queries, and aggregate and present results on geocentric Web interfaces such as Microsoft Virtual Earth. In IrisNet [8], each organization maintains database servers for its own sensors, and a global naming service is provided for information access; a similar approach is used in GSN [1] to allow users to query local and remote sensor data sources. GeoServ differs from these approaches in that it focuses on *large-scale participatory sensing* and facilitates information sharing via a scalable structured P2P overlay that efficiently supports location-sensitive data publish/retrieval.

6.2 DHT-based Distributed Storage Systems

Structured overlay networks (or DHTs) such as Chord, CAN, and Pastry (and file systems based on DHTs such as CFS [6] and PAST [30]) provide efficient, scalable, robust methods of locating and storing resources over the overlay network. However, since these systems use *consistent hashing* to map node ID and keyword to key space (i.e., DHT only provides exact match queries), it is non-trivial to support complex queries such as range queries. Chawathe et al. [5] proposed a Prefix Hash Table (PHT) that is a trie-like data structure to provide a range query on top of the DHT layer. PIER [14], a distributed query engine based on DHTs provides rich declarative SQL queries such as equi-join. The major disadvantage of this layered approach is that an extra data structure must be maintained over the DHT.

Instead, researchers tried to solve the fundamental problem of supporting a range query by directly using *keyword* space as DHT key space such as domain names and location (e.g., SkiptNet [13], D2 [24], Mercury [3]). Nodes can be assigned with random node IDs drawn from the key space [3], or with some restrictions such as domain names + random ID [13] and user ID + file IDs [24]. Then, the next challenge is to support multi-dimensional range queries (in our case, geographic range queries). In Mercury [3], each dimension has its own table, and data are published to all tables such that a query can be resolved only using a single table. Maintaining separate tables makes load balancing and storage/consistency management very difficult, because each table may need to store a sheer amount of data (e.g., for a given X coordinate, entire Y coordinate space must be handled). Recently, Rybicki et al. [31, 32] proposed PeerTIS that directly uses geographic coordinates as key space of CAN [26] to support geographic range queries for sensor storage on the mobile-to-mobile Internet overlay. CAN's routing cost is given as $O(dn^{1/d})$ with d-dimensional Cartesian coordinate space, and logarithmic hop can only be achieved when $d = \log_2 n/2$. In 2D space

($d = 2$), however, the routing cost becomes $O(\sqrt{n})$, failing to guarantee logarithmic routing properties.

In GeoServ, we linearize 2D geographic space into fixed size grids with the Hilbert space filing curve (HSFC) [15]. To support location-aware sensor data publish/retrieval, we significantly extend Symphony DHT [19] that only support (key, value)-based unicast routing with consistent hashing. While HSFC was used in other proposals [28, 33], none of these schemes are designed for location-aware sensor data publish/retrieval. The main departure from existing works include that (1) GeoServ is a two-tier architecture where Internet-based fixed servers form a distributed P2P overlay network, through which mobile users publish/access location-aware sensor data; and to minimize data publish/retrieval overhead, mobile clients are associated with the overlay nodes that are responsible for the area where they are residing (like WiFi hotspot association); (2) we formally prove that owing to the recursive construction property of HSFC, GeoServ can always preserve geographic locality; and (3) GeoServ efficiently supports location-aware publish/subscribe services such as sharing traffic information with a group of mobile users.

6.3 Distributed Load Balancing in DHTs

When the keyword space is directly used, it is very important to consider load balancing because keyword space may be skewed. Existing approaches perform load balancing using *virtual servers* as proposed in [9]; i.e., servers located in the less loaded regions are moved to the overloaded regions. SkipNet uses *string name ID space* (augmented with DNS names, e.g., *microsoft.com!skipnet.html*) to preserve content locality and to provide data controllability within domains, yet it can only provide per-domain load balancing based on virtual servers and assumes that popularity distribution among domains is uniform. Bharambe et al. [3] showed that such virtual

server-based load balancing approaches cannot efficiently handle skewed key distribution because after load balancing, the nodes may no longer be uniformly distributed over the key space, which is a necessary condition for the logarithmic DHT operations. Mercury DHT [3] periodically re-arranges routing tables to preserve the logarithmic DHT operations by using the approximate overlay node distribution histogram over the key space. GeoServ adopts this technique to balance load in HSFC space.

6.4 DHT-based Publish/Subscribe Services

There are several publish/subscribe protocols (also known as application-level multicast routing) based on DHT: Bayeux on Tapestry [38], Scribe on Pastry [4], and CAN multicast [27]. These protocols have different methods of building an application-level multicast tree per group. CAN multicast builds a new CAN per group and use flooding over the group-based CAN overlay. Bayeux and Scribe use a rendezvous-based method where there is a rendezvous node for each group. While Bayeux pushes all the subscription information at the rendezvous node, Scribe tries to distribute it along the intermediate nodes when a subscription message is routed toward the rendezvous node. All of these approaches, however, use DHTs that are based on consistent hashing, destroying geographic locality in content-based routing. In contrast, GeoServ uses locality-preserving content-based routing (geocasting). The key innovation is that our protocol partitions the network area in a hierarchical fashion (like hierarchical geographic location services [17, 37]) and thus, the locality of multicast receivers is preserved.

CHAPTER 7

Future Directions

For the future work, we plan to apply GeoServ to traffic information systems (TIS) such as Peers on Wheels [31] and PeerTIS [32] that support the driver of a car in selecting a route, based on traffic information collected by other cars. We will study how Internet-based communication can be leveraged to build a distributed, cooperative TIS. Such a system, implementing well-designed distributed data structures and algorithms, allows to build the decentralized TIS applications.

CHAPTER 8

Conclusion

The main focus of this paper has been to design a scalable sensor networking system that enables location-relevant sensor data sharing among mobile users with smartphones. Given that mobile-to-mobile P2P networking over cellular networks is challenging due to the limitation of cellular networks and constrained resources of mobile devices, we proposed GeoServ, a distributed two-tier sensor networking system that exploits the Internet infrastructure, where mobile users publish/access sensor information through Internet-based distributed sensor storage. The key services of GeoServ include GeoTable, a location-aware sensor data retrieval service that efficiently supports geographic range queries, and GeoPS, a location-aware publishsubscribe service that enables efficient multicast routing over a group of subscribed users. We proved that GeoServ protocols preserve geographic locality and validated their performance via extensive simulations.

REFERENCES

[1] K. Aberer, M. Hauswirth, and A. Salehi. A Middleware for Fast and Flexible Sensor Network Deployment. In *VLDB*, 2006.

[2] R. Baeza-Yates and R. Ramakrishnan. Data Challenges at Yahoo! In *EDBT*, 2008.

[3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *SIGCOMM*, 2004.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A Large-scale and Decentralised Application-level Multicast Infrastructure. *JSAC*, 2002.

[5] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A Case Study in Building Layered DHT Applications (PHT). In *SIGCOMM*, 2005.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area Cooperative Storage with CFS. In *SOSP*, 2001.

[7] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation. In *MobiSys*, 2008.

[8] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a Worldwide Sensor Web. *IEEE Pervasive Computing*, 2(4):22–33, Oct.-Dec. 2003.

[9] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Dynamic Structured P2P Systems. In *INFOCOM*, 2003.

[10] Google Maps Gets Smarter: Crowdsources Live Traffic. `http://www.readwriteweb.com/archives/google_maps_gets_smarter_crowdsources_traffic_data.php/`.

[11] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *IPTPS*, 2006.

[12] J. Härri, M. Fiore, F. Fethi, and C. Bonnet. Vanetmobisim: Generating rrealistic mobility patterns for vanets. In *VANET*, 2006.

[13] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: a Scalable Overlay Network with Practical Locality Properties. In *USITS*, 2003.

[14] R. Huebsch, J. M. Hellerstein, N. Lanham, B. Thau, L. S. Shenker, and I. Stoica. Querying the Internet with PIER. In *VLDB*, 2003.

[15] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *SIGMOD*, 1990.

[16] D. Kessens and T. Savolainen. 3G and IPv6 impact on battery life. In *French IPv6 Worldwide Summit*, 2006.

[17] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *MobiCom*, 2000.

[18] S. Liu, W. Jiang, and J. Li. Architecture and Performance Evaluation for P2P Application in 3G Mobile Cellular Systems. In *WiCom'07*, 2007.

[19] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *USITS*, 2003.

[20] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones. In *SenSys*, 2008.

[21] M. Mun, S. Reddy, K. Shilton, N. Yau, P. Boda, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West. PEIR, the Personal Environmental Impact Report, as a Platform for Participatory Sensing Systems Research. In *MobiSys*, 2009.

[22] The NanoDataCenters Projects. `http://www.nanodatacenters.eu/`.

[23] S. Nath, J. Liu, and F. Zhao. SensorMap for Wide-Area Sensor Webs. *IEEE Computer Magazine*, 40(7), July 2007.

[24] J. Pang, P. B. Gibbons, M. Kaminsky, and S. Seshan. Defragmenting DHT-based Distributed File Systems. In *ICDCS*, 2007.

[25] A. Qureshi, J. Carlisle, and J. Guttag. Tavarua: Video Streaming with WWAN Striping. In *Multimedia*, 2006.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[27] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-Level Multicast using Content-Addressable Networks. In *NGC*, 2001.

[28] S. Ratti, B. Hariri, and S. Shirmohammadi. NL-DHT: A Non-uniform Locality Sensitive DHT Architecture for Massively Multi-user Virtual Environment Applications. In *ICPADS*, 2008.

[29] S. Reddy, G. Chen, B. Fulkerson, S. J. Kim, U. Park, N. Yau, J. Cho, M. Hansen, and J. Heidemann. Sensor-Internet Share and Search – Enabling Collaboration of Citizen Scientists. In *DSI*, 2007.

[30] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-scale Persistent Peer-to-peer Storage Utility. In *SOSP*, 2001.

[31] J. Rybicki, B. Scheuermann, W. Kiess, C. Lochert, P. Fallahi, and M. Mauve. Challenge: Peers on Wheels - A Road to New Traffic Information Systems. In *MobiCom'07*, 2007.

[32] J. Rybicki, B. Scheuermann, M. Koegel, and M. Mauve. PeerTIS: a Peer-to-Peer Traffic Information System. In *VANET*, 2009.

[33] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *HPDC*, 2003.

[34] USA to Add 80 Million New Smartphone Users by 2011. `http://twittown.com/mobile/mobile-blog/usa-add-80-million-new-smartphone-users-2011`.

[35] U.S. Census Bureau. TIGER, TIGER/Line and TIGER-Related Products. Available at. `http://www.census.gov/geo/www/tiger/`.

[36] A. Woo. A New Embedded Web Services Approach to Wireless Sensor Networks. In *SenSys*, 2007.

[37] Y. Yu, G.-H. Lu, and Z.-L. Zhang. Enhancing Location Service Scalability with HIGH-GRADE. In *MASS*, 2004.

[38] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *NOSSDAV*, 2001.

[39] J. Eriksson, L. Girod, B. Hull, R. Newton, H. Balakrishnan and S. Madden. The Pothole Patrol: Using a Mobile Sensor Network for Road Surface Monitoring. In *MobiSys*, 2008.

[40] Pavement Interactive Core: Roughness. `http://pavementinteractive.org/index.php?title=Roughness`.

[41] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazons Highly Available Key-value Store. In *SOSP*, 2001.

[42] B. F. Cooper R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H-A. Jacobsen, N. Puz, D. Weaver, and Ramana Yerneni. PNUTS: Yahoo!s Hosted Data Serving Platform. In *VLDB*, 2008.

[43] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *IPTPS*, 2003.