EZCab: A Cab Booking Application Using Short-Range Wireless Communication*

Peng Zhou[1], Tamer Nadeem[2], Porlin Kang[1], Cristian Borcea[3], and Liviu Iftode[1]
[1] Department of Computer Science, Rutgers University
[2] Department of Computer Science, University of Maryland, College Park
[3] Department of Computer Science, New Jersey Institute of Technology
[1]{pzhou,kangp,iftode}@cs.rutgers.edu, [2]nadeem@cs.umd.edu, [3]borcea@cs.njit.edu

Abstract

EZCab is a proof-of-concept ubiquitous computing application that allows people to book nearby cabs using their cell phones or PDAs equipped with short-range wireless network interfaces. EZCab discovers and books free cabs using mobile ad hoc networks of vehicles. We have implemented an EZCab prototype on top of Smart Messages, a middleware architecture based on execution migration, which we had developed to provide a common execution environment for outdoor ubiquitous computing applications. The experimental and simulation results have demonstrated the feasibility of EZCab.

1 Introduction

For many years, people have discussed about mobile ad hoc networks (MANET) and have proposed numerous routing algorithms [23, 31], but the technology has not been mature enough to support the deployment of such networks in the real world. Recently, short-range wireless networking has started to live up to its expectations, and we see a large deployment of products based on either IEEE 802.11 standards or Bluetooth (a low-cost, low-power alternative to IEEE 802.11 family of protocols). While short-range wireless technology is on its way to become ubiquitous in the near future, little has been done to develop real-life services or applications over mobile ad hoc networks.

This paper presents EZCab, a real-life ubiquitous computing application built over MANET, which allows people to book nearby cabs in densely populated urban areas using their cell phones or PDAs equipped with short-range wireless network interfaces. Current cab booking systems rely on centralized schemes for cab dispatching such as making phone calls to a taxi company or sending short messages (SMSs) to a certain server over cellular links [1, 2]. Although under the traditional centralized solution cab dispatching is guaranteed, this solution is not scalable due to: 1) all requests have to go through one or multiple cab dispatchers, which introduces waiting time for the clients, especially during periods of peak cab requests, and 2) in order to dispatch the nearest cab to the client, all cabs in the city have to be monitored to find the closest one to the client's location.

The EZCab dispatching system, on the other hand, is simpler, faster, and more scalable since it works in a completely decentralized fashion, and there is no need to gather the locations of all the cabs in real-time. EZCab provides all these benefits because it defines a system architecture in which the clients and vehicles communicate using only short-range wireless network interfaces. This design decision, however, makes EZCab a "best effort" service. Clients can switch to the standard centralized methods to book a cab if they fail to get a cab using EZCab within a short period of time. Hence, the EZCab system can be incrementally deployed and coexist with current centralized systems. EZCab is useful in cities with high density of cabs, such as New York or Tokyo, where the contention to get cabs during certain periods (e.g., people getting out from a show) is very annoying.

EZCab is made possible by two recent technology trends. The first is the transformation of PDAs (e.g., HP iPAQ [5], Toshiba PocketPC [10]) and cell phones (e.g., Ericsson P900 [7], Motorola A760 [6]) into relatively powerful mobile computers equipped with short-range wireless capabilities. The second is the increasing presence of powerful embedded systems, GPS receivers, and even wireless network interfaces in modern vehicles. For instance, GPS has been successfully used to track vehicles and provide accurate position information [11, 21]. A taxi service, based on real-time GPS information collected by

*This work is supported in part by the NSF under the ITR grant number ANI-0121416

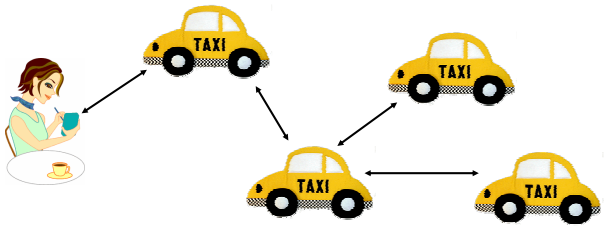

Figure 1. EZCab: Booking a cab over a mobile ad hoc network of cabs

a centralized dispatching center over cellular networks, has also been implemented in Singapore [28].

We have implemented an EZCab prototype on top of Smart Messages (SM) [13, 24], a middleware architecture based on execution migration, which we developed to provide a common execution environment for outdoor distributed applications. An SM carries its own routing code and routes itself at each node in the path toward a node of interest. To perform routing, SMs store routing information in a memory addressable by names at nodes. The SM self-routing mechanism [12] is especially useful for EZCab because it allows on-demand deployment of new routing algorithms and changing the routing algorithm during execution. This feature enables EZCab to adapt to highly dynamic network configurations. The testbed for EZCab consists of ad hoc wireless networks composed of HP iPAQs running Linux and communicating through IEEE 802.11 network interface cards. The experimental results show that EZCab is a viable solution for booking cabs in densely populated urban areas. We have also evaluated and compared through simulations multiple mechanisms for discovering free cabs under different traffic scenarios.

The rest of this paper is organized as follows. We present the design of EZCab in Section 2. Section 3 describes the EZCab prototype, its SM-based routing algorithms, and the experimental results. Section 4 presents the simulation results comparing different routing algorithms. Finally, we discuss related work in Section 5 and conclude the paper in Section 6.

2 Design

EZCab consists of a mobile ad hoc network of computers embedded in taxis and client handheld devices, which communicate using short-range wireless network interfaces such as IEEE 802.11 as illustrated in Figure 1. Instead of booking cabs through a centralized dispatcher, EZCab clients book free nearby cabs by communicating directly with other EZCab nodes (i.e., taxis) over a mobile ad hoc network. This decentralized architecture provides a simple, cheap, and scalable solution to a real-life problem. However, EZCab presents new challenges which do not exist in traditional systems based on centralized dispatching centers. For instance, we need a distributed protocol to ensure that at most one cab arrives at the site of the client who requested the cab. Furthermore, we must ensure that any free cab accepts only one client request at any point in time. To provide an automatic booking mechanism, we also need accurate location information for both clients and cabs (e.g., the client's street address has to be present in the booking request). Finally, EZCab needs to provide a mechanism for the client and driver to authenticate each other when they meet. This section presents the EZCab system architecture and its protocol that satisfies these requirements.

2.1 System Architecture

EZCab consists of two types of entities: client stations and driver stations. A client station is a a PDA (or cell phone), while a driver station is a system embedded in the cab. We assume that all driver stations communicate with each other using IEEE 802.11, the de facto standard for high bandwidth, short-range wireless networking. Additionally, each driver station has a GPS receiver that can report its current location when needed. All driver stations that are within the radio range of each other form an ad hoc network of nodes which communicate and perform the distributed computation necessary to book a free cab. The client stations have to join such a network to inject their requests for free cabs. A client station communicates directly with cabs in its transmission range, and the cabs in the network forward the request until a free cab is discovered.

Unlike driver stations which are homogeneous, client stations can have different capabilities. Besides powerful PDAs equipped with IEEE 802.11 wireless network interfaces and GPS receivers, cell phones equipped with low-power Bluetooth interfaces and without GPS receivers can also be used as client stations. In such a case, however, the client station needs to connect to a gateway station that forwards the request into the network of cabs and sends the answer back to the client. A gateway station is a computer equipped with a GPS receiver and multiple network interfaces (i.e., Bluetooth and IEEE 802.11), which can be co-located with *Hotspots* (i.e., IEEE 802.11 access points) at public places such as restaurants, stores, theaters, bus stops, telephone booths, and building lobbies. The role of a gateway station is to "stamp" location information on requests received from lighter clients that cannot incorporate a GPS receiver (i.e., given the range of Bluetooth, the location of the gateway is a good

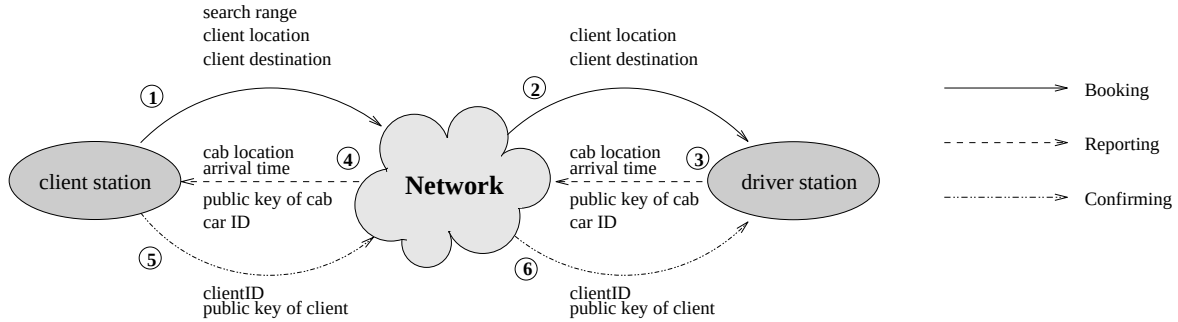

Figure 2. Cab booking protocol

approximation for the location of the client) and perform a change of protocols between client stations equipped with Bluetooth and driver stations equipped with IEEE 802.11.

2.2 EZCab Protocol

The EZCab application starts when a client sends out a request and ends with a validation at the client's location. The EZCab protocol consists of two phases: *Cab Booking* and *Validation*. Figure 2 shows the *Cab Booking* phase. During this, the client and driver stations collaborate with each other to forward the client's request through the network until a free cab is discovered. To accommodate various network configurations determined by density and mobility of the cabs, EZCab assumes an extensible routing layer which supports different routing algorithms as plug-ins (similar to Active Networks [3]). Sections 3 and 4 will present our experiences with developing routing algorithms specific to EZCab.

Cab Booking is a three-way handshake protocol which ensures that a nearby free cab is booked and no more than one client books a cab simultaneously. It starts by sending a request containing the client's information (e.g., current location, destination location) into an ad hoc network of cabs. This request is routed to a free cab by the routing layer. Once the driver of a free cab agrees to the client's requirements, the cab status is changed from free to occupied. The expected arrival time (estimated based on the distance between the client and the cab, the time of the day, and previous experiences of the cab driver) and the license plate of the cab are sent to the client. The handshake protocol completes with an acknowledgment from the client station to the driver station. Once the driver station receives the acknowledgment, the cab driver is notified to pick up the client.

Both the client and driver stations have a timeout mechanism which allows them to cope with highly dynamic network configurations. If the driver station times out before receiving the last part of the handshake, the cab will be made available again. Similarly, if the client station times out before receiving any confirmation from a cab, a new request is injected in the network. While multiple driver stations may be traversed during the execution of the EZCab application, its protocol ensures that a cab which satisfies the client's requirements is finally booked and reaches the client.

When the booked cab arrives at the vicinity of the client, the driver initiates the *Validation* phase to mutually authenticate with the client. The *Validation* protocol uses a challenge-response scheme based on public-key cryptography. The client and the driver stations exchange their public keys during the three-way handshake *Cab Booking* protocol. Once, the cab arrives in the proximity of the client, the driver station sends a random number encrypted using the client station's public key (as a challenge) to the client. Only the client holding the corresponding private key can successfully decrypt the challenge. The client then encrypts the result (as a response) using the driver station's public key and sends it back to the driver station. If the driver station could successfully decrypt the response, both the client and driver stations have mutually authenticated. Thus, no client can claim an already booked cab, and no driver can get somebody else's client.

3 Prototype Implementation

We have implemented EZCab using Smart Messages [13, 24], a middleware architecture based on execution migration, which we developed to provide a common execution environment for outdoor ubiquitous computing applications. This section presents a short overview of Smart Messages, the EZCab prototype, the routing algorithms used by EZCab to find free cabs, and experimental results over ad hoc networks of PDAs.

3.1 Smart Messages

Smart Messages (SM) define a middleware architecture, similar to mobile agents, for programming distributed

applications over mobile ad hoc networks of resource constrained devices. An SM is a distributed application consisting of code, data, and a lightweight execution state. Instead of transferring data among nodes involved in computation, SMs migrate the execution to each of these nodes. An SM carries routing code and routes itself at each node in the path toward a node of interest. Each node cooperates to support the SM execution by providing a virtual machine (VM) for execution over heterogeneous platforms, a shared memory addressable by names (tag space) for inter-SM communication and synchronization, and a code cache for storing frequently executed code.

An SM calls explicitly for migration when it needs to execute on a different node. Upon an SM arrival at a new node, its execution is resumed from the next instruction following a migration invocation. During its execution an SM can spawn or create new SMs. Additionally, an SM can interact with the host or other SMs using tags stored in the tag space. Essentially, the tags are *(name, data)* pairs. Corresponding to their functionality, there are two types of tags: application tags for "persistent" memory across SM executions which can store application-specific data, and I/O tags for interaction with the host's operating system and I/O. Tags can also be used for synchronization between SMs. An SM can block on a tag until another SM performs a write on that tag (i.e., update-based synchronization).

SMs name nodes by tag names (i.e., content-based naming) and migrate to nodes of interest using a high level migration function that implements routing [12]. The routing is executed at each node on the path toward a node of interest; hence, SMs are self-routing applications. The implementation of routing uses information stored by SMs in the tag space and a system-provided primitive for one-hop migration. This primitive captures the current execution control state and migrates it to the next hop along with the code and data.

The SM platform is developed by directly modifying Sun Microsystem's Kilobyte Virtual Machine (KVM); KVM features a small memory footprint (160K) suitable for most embedded devices. The tag space and SM operations are available to applications as a Java API.

3.2 EZCab Prototype

We have implemented the EZCab prototype on top of the SM platform installed on HP iPAQs running Linux. The iPAQs use Orinoco's 802.11 cards for wireless communication, and each of them is connected to a Geko 201 GPS receiver. We have demonstrated in a different project [18] that cars can be mapped with high accuracy on the roads despite the low accuracy provided by raw GPS data (e.g., raw GPS data provides on average an accuracy of 10 meters). Figure 3 illustrates an EZCab driver station.

Figure 3. EZCab driver station consisting of an HP iPAQ equipped with Orinoco 802.11 wireless card and Geko 201 GPS receiver

The EZCab prototype has two types of graphical user interfaces, corresponding to client station and driver station, respectively. Each user interface communicates with the SM platform via special bi-directional I/O tags, called UI tags. These tags persist for the entire duration of the user interface process and behave similar to a producer-consumer circular buffer. We have used the Open Palmtop Integrated Environment (OPIE) to develop the user interfaces of EZCab. OPIE is an open source graphical user environment for PDA's and other devices running Linux.

3.3 EZCab Routing Algorithms

A very important feature of the SM middleware is its self-routing mechanism that enables SMs to use newly developed routing algorithms during their lifetime and change between multiple algorithms dynamically. This feature allows applications to plug-in different routing protocols according to their needs without the need to install such algorithms on the nodes (i.e., SMs carry their routing code). An EZCab SM can switch to different routing algorithms as it traverses the network (based on different network properties such as mobility, network density, and GPS data). Carrying routing code with an SM does not incur additional overhead in the common case because the routing code will be cached at nodes. Thus, the cost of transferring routing code is amortized over time.

The routing algorithms for EZCab have two special properties that make them different from traditional MANET routing: limited geographical scope and multiple possible destinations. First, EZCab routing only looks for a destination node within a limited geographical scope (i.e., a free cab near the client). Second, every free cab node within a limited range of the client serves equally well as

destination of the request.

We have implemented and compared three routing algorithms for finding free cabs: *Flooding*, *Probabilistic On-Demand*, and *Probabilistic Proactive*. All the three algorithms rely on an underlying broadcast mechanism and assume symmetric links. The first algorithm does not cache any routes, while the second and third algorithms cache discovered routes in a routing table on each node for subsequent re-use.

The three-way handshake protocol described in Section 2 is implemented as three different SMs: BookSM, ReportSM, and ConfirmSM. BookSM uses any of the three routing algorithms mentioned above to find a free cab. Upon reaching a free cab, it creates a ReportSM which goes back to the client with the cab's information. Upon reaching back the client, ReportSM creates a ConfirmSM which goes back to the cab driver waiting for client confirmation. Both the ReportSM and the ConfirmSM use geographical routing which allows them to reach the nodes of interest independent of any routing information, except for the geographical positions of the one-hop neighbor nodes. This method is scalable and avoids possible broken routes. In the following, we describe the implementation of the *Flooding*, *Probabilistic On-Demand*, and *Probabilistic Proactive* routing algorithms using SMs.

3.3.1 Flooding

Flooding is the basic mechanism to propagate messages in many MANET protocols (e.g., routing, service discovery). In this scheme, each EZCab client request (BookSM) is broadcasted to all its neighbors recursively, up to a maximum number of hops, *TTL*, or until it arrives at a free cab node, whichever comes first. The client node will only consider the first free cab response (i.e., ReportSM); responses from other cabs will be dropped. The SM waiting at the client for responses will eventually timeout if no free cab is available within *TTL* hops. To prevent routing loops, BookSM also marks visited nodes using a unique tag. In most cases, this simple *Flooding* yields free cabs in the proximity of the client and works well in a less crowded scenario. If the network is very dense, *Flooding* does not work well due to the unavoidable wireless contention. In such a situation, we resort to the next two algorithms.

3.3.2 Probabilistic On-Demand

This routing algorithm builds routing tables on-demand for nodes lying in a *TTL*-hops range from the client. The routing table at a node can be shared among different BookSMs, and it consists of entries with the probability of finding a free cab through the node's neighbors. Since the route to each neighbor could be invalidated due to mobility, each routing entry records its last update time; the system uses this information to age routing entries. The *Probabilistic On-Demand* mechanism is similar to the reactive routing protocols in MANET such as AODV [31].

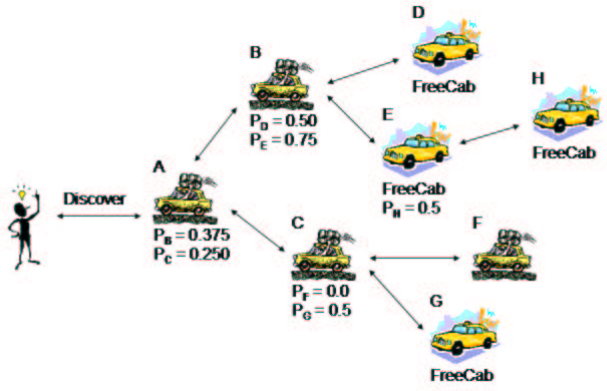

Figure 4. Probabilistic on-demand route discovery

BookSM triggers the *Probabilistic On-Demand* route discovery on any node where the routing table does not exist, the maximum probability is too small, or the communication failed because the node with maximum probability is not a neighbor of the current node any more. To do so, BookSM creates a DiscoverySM, which migrates in the network to look for a free cab and blocks on a routing tag for a timeout value. We have adopted a timeout scheme instead of waiting for a certain number of DiscoverySMs to return (i.e., we could have waited for one DiscoverySM for each direct neighbor) since it is unreasonable to do so in a volatile ad hoc network. If a new BookSM request arrives at the same node, it will block on the same tag instead of creating another DiscoverySM. Thus, only the first request updates the routing table at that node. The second and subsequent requests will use the routing table created by the first request when they continue after timeout.

Figure 4 shows the *Probabilistic On-Demand* route discovery in action. The DiscoverySM spawns itself at each node. The child SM broadcasts itself to its neighbors, and the parent blocks with a timeout waiting for all its children to come back with routing information (i.e., probability to find a free cab through this node). Each child stores the address of its source node (which according to the algorithm is one-hop away). Once a DiscoverySM reaches a node located *TTL*-hops away from client, it will report the probability to its source as either 0.5 or 0.0, depending on whether the cab is free or occupied, respectively. Note that every node traversed by a DiscoverySM, except for the last hop, has a blocked DiscoverySM. When the DiscoverySM unblocks (after timeout), it will gather all the probabilities reported by its child SMs and report back to its source node. This process continues until the top-level DiscoverySM is

reached.

The routes created by this algorithm form a *Directed Acyclic Graph*, rooted at the client station. The DAGs created by multiple client stations could overlap each other. The probability, P_n ($1 \leq n < TTL$), reported by each node located n hopes away from the client to its source is given by:

$$P_n = \begin{cases} \frac{1}{2}\big(1 + max(P_{n+1})\big) & \text{if n is occupied,} \\ \frac{1}{2}max(P_{n+1}) & \text{if n is available.} \end{cases}$$

The timeout for the parent DiscoverySM at each node is defined by:

$$2 \times oneHopTime \times (TTL - currentHops) \times L,$$

where $oneHopTime$ is the time for single hop migration. Thus, $(2 \times oneHopTime \times (TTL - currentHops))$ is the total migration time. To account for different factors affecting the migration time (such as MAC layer contention), we introduced L ($L > 1$) as a fudge factor. L is our attempt to ensure that the child Discovery SMs could get back before the parent SM times out. According to our experimental results, the more contentions, the larger L should be. For instance, a fudge factor $L = 3$ works well for cabs with 6 or less neighbors.

When a client sends out a request, BookSM simply checks the local routing table and migrates to the neighbor with the largest probability. At the same time, it updates the routing table based on the *TTL* used by the DiscoverySM. Once it chooses the next hop, BookSM divides the corresponding probability by $2^{(N-hopCount)}$, where *N* is the estimated number of hops from the client to the free cab (we have chosen $N = \frac{TTL}{2}$) and *hopCount* is the number of nodes already visited on the path from the client to the free cab. For instance, the probability is reduced by $\frac{1}{2^N}$ at the first node and $\frac{1}{2^{(N-1)}}$ at the second node. The probability is reduced by $\frac{1}{2}$ for each node located farther than $\frac{TTL}{2}$ hops away from the client. This process repeats until a free cab is reached or no further route is available. If a free cab is found and the driver has agreed to pick up the client, the BookSM creates a ReportSM. This SM uses geographical routing to send the booking information back to the client. On the other hand, if no further route is available and no free cab is found, BookSM performs a fresh route discovery from the current node.

The overhead of this routing scheme is expected to be lower than that of *Flooding* when nodes move slowly and many requests occur within a limited region and time interval (e.g., when many people book cabs outside a theater when the play just finished).

3.3.3 Probabilistic Proactive

In a highly dynamic network, the *Probabilistic On-Demand* algorithm could perform poorly. For example, the entry with the largest probability can become invalid before it is utilized (i.e., the node moves out of wireless range). This situation triggers DiscoverySM, a relatively expensive process since the BookSM has to wait for $2 \times TTL$ node traversals while routing tables are reconstructed from scratch.

To have "fresher" information in the routing table, we propose a third routing algorithm: *Probabilistic Proactive*. Proactive routing algorithms are known to perform poorly in highly mobile networks [14]. This algorithm, however, has the potential to work well for EZCab because EZCab does not need end-to-end transfers of bulk data, but rather to find any free cab in a given region.

Similar to the *Probabilistic On-Demand* scheme, this algorithm builds routing tables with probabilities to find free cabs in each entry. The routing updates, however, are triggered proactively by the presence of free cabs, not by on-demand requests from client stations. Every free cab using this algorithm broadcasts an UpdateSM periodically. UpdateSMs contain a small TTL value to limit the scope of flooding and a unique identifier *freeCabID* of the free cab which generated it. Upon receiving this update, the direct neighbors of this free cab update their routing table with (*freeCabID*, 0.5) and decrement the TTL of the UpdateSM. The UpdateSM is then further propagated until the TTL reaches 0. Each hop receives a pair (*previousCabID*, $\frac{1}{2^N}$) in the UpdateSM, where N, ($1 \leq N \leq TTL$), is the number of hops from the free cab. If *previousCabID* is not in the routing table, a new entry will be created. Otherwise, the current probability is updated with the new value. Similar to the *Probabilistic On-Demand* scheme, an aging interval is used to purge old information.

This algorithm generates overlapping DAGs rooted at each free cab. When a client requests a free cab, BookSM will migrate hop-by-hop through the entries with the largest probability (up to *TTL* hops) until a free cab is found. If a free cab is found, EZCab will proceed with the next phase of the three-way handshake protocol. A BookSM migrating to a free cab updates the probabilities in the routing tables in the same way it does in the *Probabilistic On-Demand* algorithm. If no free cab is found after *TTL* hops, BookSM will report the failure to the client. On the other hand, if a broken link is discovered (i.e., the target node disappeared), a DiscoverySM using *Flooding* will be injected to find a free cab instead of rebuilding the whole routing table.

3.4 Experimental Results

We measured the completion time for a single EZCab request using three ad hoc wireless network topologies

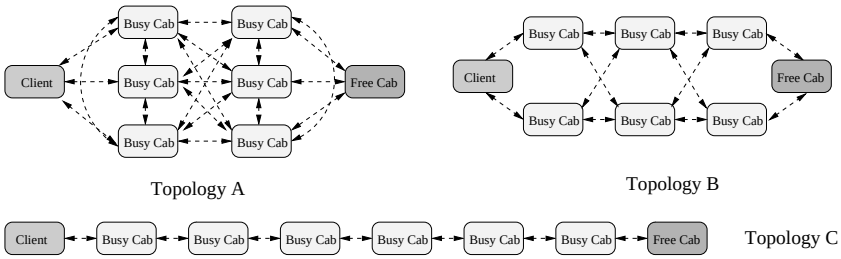

Figure 5. Topologies used in our experiments

Topology	Response Time (ms)
A	444.18
B	401.57
C	356.60

Table 1. Completion time for EZCab using three topologies

consisting of eight HP iPAQs running our prototype. Figure 5 shows the three topologies, and Table 1 presents the average response time for each of them. Response time is defined as the total time needed to complete the three-way handshake protocol which starts with the client sending a cab request and ends with a cab receiving a booking confirmation. Although a request in topology C has to travel more hops than in the other two, EZCab completes faster for this topology because of low contention. Since the scale of the network is small (only 8 nodes), we did not expect to see significant differences in the response time for different route discovery algorithms. Hence, we have used just a simple flooding to discover a free cab. Moreover, we did not use geographical routing for ReportSM and ConfirmSM in these experiments. The experimental results demonstrate the feasibility of EZCab for small scale networks, showing that the application can successfully book a cab in a reasonable amount of time.

4 Performance Evaluation

To compare the performance of the three EZCab routing algorithms under different scenarios, we have simulated EZCab using the *ns-2* simulator [9], enhanced with the CMU-wireless extensions [8]. Different scenarios are used to test sensitivities of the algorithms to various application and network parameters. In this section, we describe our experiments and present the corresponding results.

4.1 Scenario Generator

We have developed our own scenario generator tool based on *setdest*, a generator tool for random-way point mobility model, developed at Carnegie Mellon University to generate traffic scenario for cities with grid roads (e.g., Manhattan). The scenario generator accepts as parameters the simulation time, the width and length of the city grids in meters, number of horizontal roads, number of vertical roads, number of lanes per road, average speed of the cabs in meters/sec, average gap distance between cabs on the same lane, and the number of clients in the city.

In our traffic model, we use an even number of alternated single direction roads along each dimension of the grid. Cabs can change their lanes within the same road independently of each other. The probability of staying on the same lane is 0.6, whereas the probability of changing the lane is 0.4 either to the left or to the right. Similarly, cabs can switch their roads at intersections. With equal probabilities, a cab chooses either to stay on the same road or change to the road it intersects with. When a cab reaches the last intersection on a road, it is forced to change to the road it intersects with. Clients behave similarly to the cabs at the intersections, except that they can move on both directions of a road. Therefore, clients at intersections can choose uniformly between the three new directions.

The cabs select their speed as $[average\_speed \pm (0.25 \times average\_speed \times rand())]$, where $rand()$ returns a uniformly distributed random number from the range $[0, 1]$. The clients select their speed uniformly from range $[0, 1]$ meters/sec. Initially, each cab moves toward a random destination along its road using its currently selected speed. Once a cab reaches its destination, it selects another random destination along its road as well as a new speed. If the selected destination is behind the next intersection on the road, the cab sets its destination to the intersection. Clients select their new destinations in a similar manner.

For all the simulations, we fixed the width of the grids to 2,000 meters, while the length is 3,000 meters. The roads are distributed uniformly as 6 vertical roads and 10 horizontal roads with 2 lanes per road. The average gap between successive cabs on the same lane is 185 meters. The scenario generator places $[number\_of\_lanes \times (\frac{city\_width}{gap} \times number\_vertical\_roads + \frac{city\_length}{gap} \times number\_horizontal\_roads)] = 410$ cabs and 200 clients evenly distributed on the roads. We used IEEE 802.11b as

the wireless media, with a data transmission rate of 11Mb and a transmission range of 250 meters. During our outdoor experiments, we found out that the wireless transmission range is less than 250 meters. However, we have been able to restore this transmission range using external antennas.

4.2 Simulation Results

For all the simulation runs, the first 200 seconds represent a warm up period (i.e., no cab request occurs within this period). Each client sends a cab request (BookSM) once during the simulation period, at a time chosen randomly after the warm up period. BookSM is unicasted for the on-demand and proactive routing mechanisms, while it is broadcasted for the flooding mechanism. Note that in case a client uses the *Probabilistic On-demand* or *Probabilistic Proactive* routing mechanisms and has no routing table information, it will broadcast the BookSM or DiscoverSM (to simplify the exposition, we will refer only to BookSM).

The maximum number of hops a request can propagate (i.e., TTL) searching for a free cab is set to 20 hops. Once a free cab receives a cab request, it sets its status to reserved for a period chosen randomly between 2 and 5 seconds and sends back a ReportSM. Once a confirmation (ConfirmSM) is received from a client, the cab status is set to booked (occupied) for a random period chosen uniformly over the range [300, 1800] seconds. Regardless of the routing mechanism, a client re-broadcasts a cab request if it does not receive a reply from any free cab within a random period chosen uniformly over the range [2, 5] seconds. The average cab speed is set to 15 meters/second with zero pause time. We set the fudge factor L for the on-demand mechanism to 3. For the proactive mechanism, we set the maximum entries in the routing table ($maxE$) to 20, and the periodic update interval (UPD) for UpdateSM to 5 seconds. The maximum number of hops (i.e., TTL) for UpdateSMs generated by free cabs is set to 3. Each entry in the routing table expires after $2.5 \times UPD$ seconds of the last update.

4.2.1 Effects of Number of Free Cabs

We first look at the effect of the initial number of free cabs. An initial free cab can be booked during the simulation and then switches its status back to free after the booking period. On the other hand, all the cabs initialized as booked at the beginning of the simulation remain booked during the simulation period. The initially booked cabs act only as relays. For these runs, we fix the simulation time to 1500 seconds, with the 200 seconds warm up period. The clients request cabs uniformly over the next 1200 seconds, leaving the last 100 seconds for any unfinished requests.

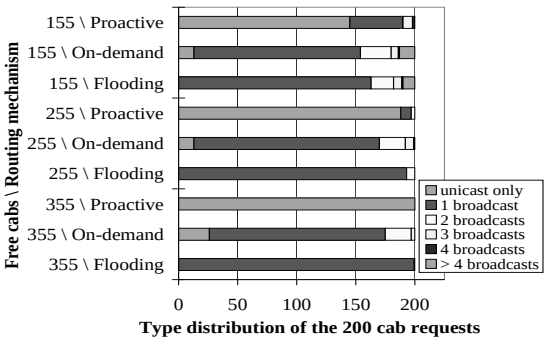

Figure 6. The type distribution of the 200 request messages versus the number of free cabs

Figure 6 plots the type distribution of the requests used by the 200 clients for each routing mechanism. A client starts with a unicast request if possible. If the unicast cannot be used or simply fails, the client switches to one or more broadcast requests until it books a cab. For example, the *'155\On-demand'* bar indicates that only 13 clients succeeded to get a cab using only unicast requests. The rest of the clients had to switch to broadcast requests. Of which, 141 clients succeeded to book cabs using a single broadcast request, while the rest timed out and had to re-broadcast the request (i.e., 26 clients needed two broadcasts to book a cab, 6 clients need three broadcasts, and the remaining 12 clients needed four or more broadcasts).

All cab requests in the flooding mechanism, where no routing tables are used, are broadcast requests. As the number of initial free cabs decreases, a single client request could be repeated several times due to the unavailability of free cabs or dropping of the reply/confirmation messages because of collisions. Most of the cab requests of the proactive mechanism are unicast that exploit the stored routing tables. However, when the number of free cabs decreases, some of those unicasts fail, and the clients use broadcast requests. For the on-demand mechanism, because the clients were distributed over the city, just a few of them could benefit from other requests and succeed to reserve cabs using unicast requests, while the rest of the clients switched to broadcast requests.

Figure 7 plots the cumulative number of clients against the response time of the booking process. A client booking ends when a cab gets a reservation confirmation from its client. A point (t, n) on the plot means that the booking process time for n cabs is less than or equal to t seconds. The proactive mechanism has the best response time, and most of the clients finish their booking in less than 10 milliseconds. Although this time is significantly less than the time measured in the experimental results (our implementation uses execution migration and TCP for one-hop data transfer, while the simulations work directly

on top of IEEE 802.11), we believe that the simulation results provide a realistic comparison among the routing algorithms used by EZCab for large scale networks.

Figure 8 shows the average number of hops between a booked cab and its corresponding client. Interestingly, we found that the cabs booked by the proactive mechanism are the closest ones to their clients, while the other mechanisms tend to book farther away cabs. This could be explained by Figure 9, which shows the average number of reserved cabs per request. Due to the large number of reservations for each request in the flooding and on-demand mechanisms, subsequent cab requests may find that all closer cabs are reserved, and therefore, the requests propagate deep into the network. The on-demand mechanism is the worst because a free cab keeps forwarding the requests aggressively after it replies back to the client in order to enable other clients to update their local routing tables, while in the flooding mechanism, the propagation of the request terminates at a free cab.

Figure 10 shows the overhead of the mechanisms in terms of the average number of messages transmitted per request (including the UpdateSMs for the proactive mechanism). We found that the overhead of the proactive mechanism is higher than the flooding mechanism especially when the number of free cabs is large. This is due to the periodic update messages from the cabs in the system. The on-demand mechanism has the highest overhead because of its aggressive broadcasting mechanism.

4.2.2 Effects of Cab Request Rate

We also tested the performance of the three mechanisms under different rates of cab requests. In doing this, we limit all the cab requests to be within a fixed period after the warm up. The simulation terminates 100 seconds after the end of request period. Changing the period length changes the cab request rate. All the 410 cabs were initialized to free. Figure 11 shows that the proactive routing exploits the routing tables and manages to send many of the requests as unicasts even with high request rates. The other mechanisms suffer from hitting large number of reserved cabs, as shown in Figure 13, that increases the possibility to use broadcast requests several times. Figure 12 shows how the proactive method managed to serve most of the clients in a very short period compared with the other mechanisms. Figure 14 indicates that the proactive mechanism is still able to catch a close cab to the client even with high request rates. Additionally, with high request rates, the overhead of the update messages per request in the proactive mechanism is minimized. Therefore, the proactive mechanism has the lowest overhead with high request rates as shown in Figure 15.

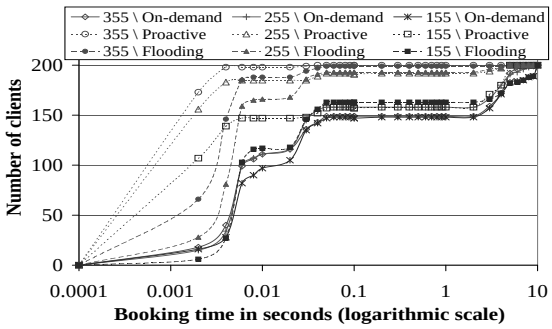

Figure 7. The cumulative number of clients versus the booking time for different numbers of free cabs

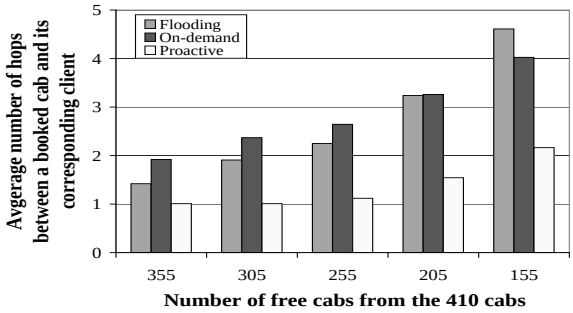

Figure 8. The average number of hops between a booked cab and its corresponding client versus the number of free cabs

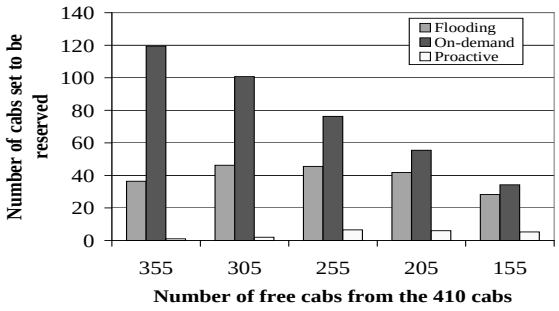

Figure 9. The average number of reserved cabs per cab request versus the number of free cabs

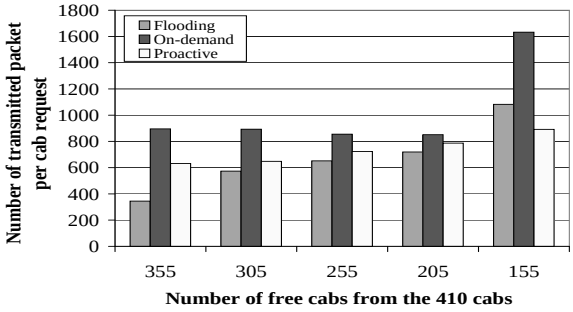

Figure 10. The average number of overhead messages per cab request versus the number of free cabs

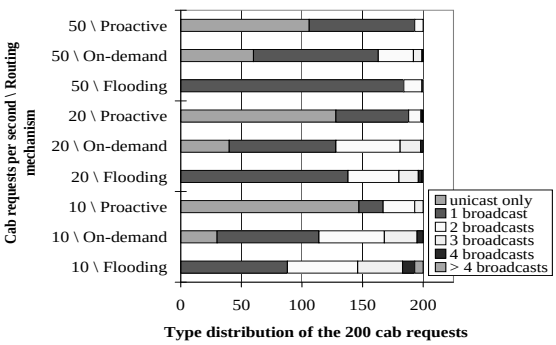

Figure 11. The type distribution of the 200 request messages versus the cab request rate

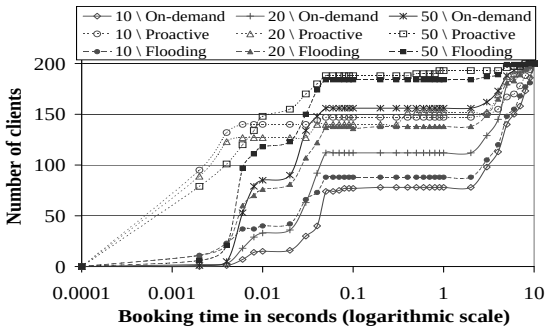

Figure 12. The cumulative number of clients versus the booking time for different cab request rate

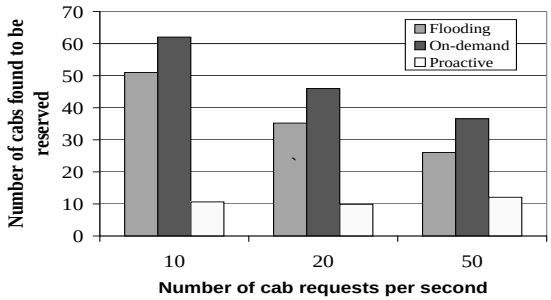

Figure 13. The average number of cabs found reserved per cab request versus the cab request rate

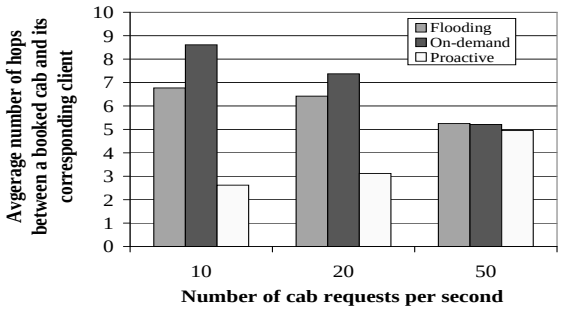

Figure 14. The average number of hops between a booked cab and its corresponding client versus the cab request rate

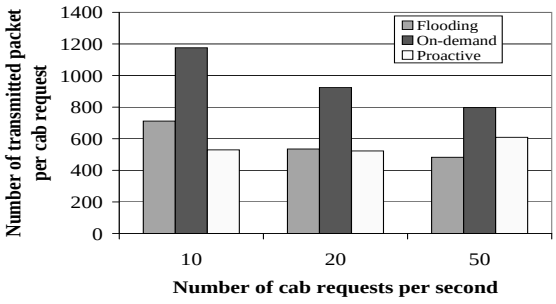

Figure 15. The average number of overhead messages per cab request versus the cab request rate

4.2.3 Effects of Proactive Parameters

To study the performance of the proactive mechanism, we run different scenarios with different values of parameters $maxE$, TTL, and UPD. We found out that increasing the TTL value has negligible effect on the performance, which indicates that most of periodic updates happen only within very few hops of their origin. For these simulations, we fixed UPD to 5 seconds, while $maxE$ varied between 5 and 100 seconds. Similarly, $maxE$ is fixed at 20 for the scenarios with different UPD values ranging from 5 to 45 seconds. Due to space constraints, we just summarize the main results.

Changing the $maxE$ parameter has a small effect on the distribution of the request types, while increasing the interval of the periodic updates (UPD) decreases significantly the number of successful unicast requests because of the outdated information stored in the routing tables. Therefore, the number of clients that finish the booking process in less than 10ms has been significantly decreased from 175 clients to 108 clients when UPD has been increased from 5 to 45 seconds. Increasing UPD reduces the overhead of the periodic update messages, but at the same time, it increases the number of broadcast requests which adds overhead on the system.

From the above results, we conclude that the *Probabilistic Proactive* routing is the most efficient mechanism as it guarantees more cab bookings in shorter time. Also, it guarantees to fetch cabs closer to the clients than the other mechanisms, which impacts favorably on the quality of the service for both cab drivers and clients. We also found that the overhead of the updates sent by the proactive mechanism could be minimized in certain scenarios by tuning its parameters.

5 Related Work

Existing cab booking systems [4, 28, 1, 2] use fixed infrastructures such as Internet or cellular phone networks to relay client requests to a centralized dispatcher and track the location of cabs for optimal dispatch. In contrast, EZCab consists of a mobile ad hoc network of computers embedded in taxis and client handhelds, which communicate through short-range wireless networks such as IEEE 802.11 or Bluetooth. Instead of booking cabs through a centralized dispatcher, EZCab clients book free cabs by communicating directly with other EZCab nodes over the mobile ad hoc network.

Many research groups have looked into applications and system support for inter-vehicular communication [15, 16, 22, 30]. While most of this research has been validated only through simulations, we have designed and implemented a prototype for a real-life ubiquitous computing application which works over mobile ad hoc networks. In doing so, we have leveraged on lessons and insights gained from work on routing in mobile ad hoc networks [23, 31]. To minimize the size of routing table, we use a probabilistic measure to indicate the probability of finding a free cab through each next-hop neighbors. This is possible for EZCab since there is no distinction between different free cabs.

Due to their scalability and robustness against frequent topological changes, geographical routing algorithms [26, 27, 33, 20] are suitable for highly mobile vehicular traffic in EZCab, after a free cab has been discovered. With these algorithms, there is no need to maintain routes from the sender to the destination because the forwarding decisions are based only on local knowledge.

The Smart Messages (SM) platform shares the idea of execution migration with mobile agents [25, 19], and active networks [17, 29, 32]. Mobile agents name nodes by fixed addresses and commonly know the network configuration a priori, while SMs name nodes by content and discover the network configuration dynamically. In contrast to mobile agents, SMs are responsible for their own routing at each node in the path between two nodes of interest. This feature allows SMs to adapt quickly to changes that may occur both in the network topology and the availability of resources at nodes.

Although the SM computing platform shares some of the design goals and leverages work done in active networks (AN), it differs from AN in several key features. AN target improved performance for end-to-end data transfer in relatively stable networks, while the SM platform helps the development of outdoor ubiquitous applications. Unlike AN, the SM platform define a computing model whereby several SMs can cooperate, exchange data, and synchronize with each other through the tag space. In terms of migration, AN do not transfer the execution state from node to node whereas the SM model does. The migration of the execution state for SMs trades off overhead for flexibility to react "on-the-spot" to adverse network conditions.

6 Conclusions and Future Work

In this paper, we have presented EZCab, a real-life ubiquitous computing application for booking cabs in cities. EZCab discovers and books free cabs using only vehicle-to-vehicle short-range wireless communication. EZCab is easy to deploy, cost effective, and scalable since it works in a completely decentralized fashion. EZCab is just one of the outdoor distributed computing applications that we are developing on top of mobile ad hoc networks using Smart Messages as a common middleware architecture. The experimental results over ad hoc networks of nodes running our prototype have demonstrated the feasibility of EZCab. The simulation results using different scenarios show that EZCab with a *Probabilistic Proactive* routing yields the best response time and finds the closest cab to the client.

We are considering several extensions to EZCab as future work. In this paper, we assumed the cabs are cooperative and willing to propagate data between each other. We plan to investigate the performance of EZCab when it uses different classes of cabs in which communication happens only between cabs belonging to the same class. Cab classes model the situation of different cabs companies, where each company is not willing to route messages for a competing company. Another extension will allow occupied cabs to act as free candidate cabs based on their scheduled drop-off location and time. For example, it would be useful to consider a cab that is scheduled to drop-off a client a mere 10 feet from the new client within a minute, which is far better than booking a 15 minutes away free cab. We also plan to study the use of priorities for the free cabs based on, for example, their distances to the client, how long they have been idle, or the number of clients served in the last hour. The goal of such priority system is to guarantee fairness and optimality.

References

[1] http://www.citycab.com.sg/services/nts/sms_booking.html.

[2] http://www.comfort-transportation.com.sg/booking_svcs.html.

[3] Active Networks. http://nms.lcs.mit.edu/activeware/.

[4] Auriga Systems. http://www.auriga.co.uk/.

[5] HP iPAQ. http://h71016.www7.hp.com.

[6] Motorola A760. http://www.motorola.com.

[7] Sony Ericsson P900, http://www.sonyericsson.com/p900/.

[8] The Monarch Group at Rice University. http://www.monarch.cs.rice.edu/.

[9] The Network Simulator ns-2, http://www.isi.edu/nsnam/ns/.

[10] Toshiba PocketPC. http://www.csd.toshiba.com.

[11] D. Ashbrook and T. Starner. Using GPS to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, October 2003.

[12] C. Borcea, C. Intanagonwiwat, A. Saxena, and L. Iftode. Self-Routing in Pervasive Computing Environments using Smart Messages. In *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, pages 87–96, Dallas-Fort Worth, TX, March 2003.

[13] C. Borcea, D. Iyer, P. Kang, A. Saxena, and L. Iftode. Cooperative Computing for Distributed Embedded Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, pages 227–236, Vienna, Austria, July 2002.

[14] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97, ACM Press New York, NY, USA, 1998.

[15] Z. Chen, H. Kung, and D. Vlah. Ad Hoc Relay Wireless Networks over Moving Vehicles on Highways. In *Proceedings of the 2nd ACM International Symposium on Mobile Ad-hoc Networking and Computing*, pages 247–250, Long Beach, CA, Oct 2001.

[16] I. Chisalita and N. Shahmehri. A Peer-to-Peer Approach to Vehicular Communication for the Support of Traffic Safety Applications. In *Proceedings of the 5th IEEE International Conference on Intelligent Transportation Systems*, Singapore, Sept 2002.

[17] D. Wetherall. Active Network Vision Reality: Lessons from a Capsule-based System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP 1999)*, pages 64–79, Charleston, SC, December 1999. ACM Press, New York, NY.

[18] S. Dashtinezhad, T. Nadeem, B. Dorohonceanu, C. Borcea, P. Kang, and L. Iftode. TrafficView: A Driver Assistant Device for Traffic Monitoring based on Car-to-Car Communication. In *Proceedings of the 59th IEEE Semiannual Vehicular Technology Conference*, May 2004.

[19] R. Gray, G. Cybenko, D. Kotz, and D. Rus. Mobile Agents: Motivations and State of the Art. In J. Bradshaw, editor, *Handbook of Agent Technology*. AAAI/MIT Press, 2002.

[20] H. Hartenstein, B. Bochow, A. Ebner, M. Lott, M. Radimirsch, and D. Vollmer. Position-aware ad hoc wireless networks for inter-vehicle communications:the Fleetnet project. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 259–262, Long Beach, CA, 2001.

[21] Jennewein and J. Lexus. Volvo launch GPS-based telematics in the US, 2000. Global Positioning & Navigation News 10, 19.

[22] J. G. Jetcheva, Y.-C. Hu, S. PalChaudhuri, A. K. Saha, and D. B. Johnson. Design and Evaluation of a Metropolitan Area Multitier Wireless Ad Hoc Network Architecture. In *Fifth IEEE Workshop on Mobile Computing Systems & Applications*, Monterey, California, October 2003.

[23] D. Johnson and D. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*. T. Imielinski and H. Korth, (Eds.). Kluwer Academic Publishers, 1996.

[24] P. Kang, C. Borcea, G. Xu, A. Saxena, U. Kremer, and L. Iftode. Smart Messages: A Distributed Computing Platform for Networks of Emb edded Systems. *The Computer Journal, Special Focus-Mobile and Pervasive Computing*, pages 475–494, 2004. The British Computer Society. Oxford University Press.

[25] N. Karnik and A. Tripathi. Agent Server Architecture for the Ajanta Mobile-Agent System. In *Proceedings of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, Las Vegas, NV, July 1998.

[26] B. Karp and H. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, August 2000. ACM Press, New York, NY.

[27] Y.-B. Ko and N. H. Vaidya. Location-Aided Routing(LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 66–75, October 1998.

[28] Z. Liao. Real-time taxi dispatching using Global Positioning Systems. *Communications of the ACM*, 46(5):81–83, 2003.

[29] J. Moore, M. Hicks, and S. Nettles. Practical Programmable Packets. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001)*, pages 41–50, Anchorage, AK, April 2001.

[30] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode. TrafficView: A Scalable Traffic Monitoring System. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM 2004)*, Berkeley, GA, January 2004.

[31] C. Perkins and E. Royer. Ad Hoc On Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1999)*, pages 90–100, New Orleans, LA, February 1999.

[32] B. Schwartz, A. Jackson, W. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart packets: Applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, 2000.

[33] J. Tian, L. Han, K. Rothermel, and C. Cseh. Spatially Aware Packet Routing for Mobile Ad Hoc Inter-Vehicle Radio Networks. In *IEEE 6th International Conference on Intelligent Transportation Systems (ITSC)*, Shanghai, China, October 2003.