Opportunistic Sensing and Mobile Data Delivery in the CarTel System

by

Bret W. Hull

S.M., Electrical Engineering and Computer Science, M.I.T., 2005
B.S., Electrical Engineering and Computer Science, U.C. Berkeley, 2002

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science
at the
Massachusetts Institute of Technology
February 2010

© 2010 Bret W. Hull. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this document in whole or in part in any medium now known or hereafter created.

Author

Department of Electrical Engineering and Computer Science
January 12, 2010

Certified by

Hari Balakrishnan
Professor
Dissertation Supervisor

Accepted by

Terry P. Orlando
Chair, Department Committee on Graduate Students
Opportunistic Sensing and Mobile Data Delivery in the CarTel System

by

Bret W. Hull

Submitted to the Department of Electrical Engineering and Computer Science on January 12, 2010, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

ABSTRACT

Wide-area sensor systems enable a broad class of applications, including the fine-grained monitoring of traffic congestion, road surface conditions, and pollution. This dissertation shows that it is possible to build a low-cost, wide-area sensor system. Our approach relies on two techniques: using existing motion from such sources of mobility as cars and people to provide coverage (opportunistic mobility), and using the abundance of short duration network connections to provide low-cost data delivery (opportunistic networking).

We use these two techniques to build a mobile sensor computing system called CarTel, to collect, process, deliver, and visualize spatially diverse data. CarTel consists of three key components: hardware placed in users' cars to provide remote sensing, a communication stack called CafNet to take advantage of opportunistic networking, and a web-based portal for data visualization. This dissertation describes the design and implementation of these three components.

In addition, we analyze the properties of opportunistic networking and mobility. To show the viability of opportunistic networking, we studied Internet access from moving vehicles and found that the median duration of link layer connectivity at vehicular speeds was 13 seconds, that the median connection upload bandwidth was 30 KBytes/s, and that the mean duration between successful associations to APs was 75 seconds. To show the viability of opportunistic mobility, we used a simulation and found that after as little as 100 drive hours, a CarTel deployment could achieve over 80 percent coverage of useful roads for a traffic congestion monitoring application.

Dissertation Supervisor: Hari Balakrishnan
Title: Professor
Contents

Contents 7
Figures 9
Previously Published Material 11
Acknowledgments 13

1 Introduction 15
1.1 The Need For Wide-area Sensing 16
1.2 Our Approach 17
1.3 CarTel Applications 19
1.4 CarTel: A Mobile Sensor Computing System 22
1.5 Contributions 24

2 CarTel System Architecture 25
2.1 Data Collection 25
2.2 CafNet 35
2.3 The Portal 40

3 Opportunistic Wi-Fi 46
3.1 Experimental Method and Data Summary 49
3.2 Connectivity Results 55
3.3 Packet Losses and Data Transfers 65
3.4 Discussion 68
3.5 Implementation in Cabernet 73

4 Coverage 76
4.1 CSIM - The CarTel Coverage Simulator 78
4.2 Participatory Monitoring Results 80
4.3 Infrastructure Monitoring Results 81
4.4 Arterial Monitoring Results 83
4.5 Summary 84

5 Related Work 86
5.1 Mobile sensor networks 86
5.2 Delay-tolerant networking ........................................ 87
5.3 Query Processing ..................................................... 87
5.4 Urban sensing ............................................................. 88
5.5 Opportunistic Wi-Fi .................................................... 89

6 Lessons Learned & Conclusion ........................................ 91
   6.1 Lessons Learned ................................................... 91
   6.2 Conclusion ............................................................. 92

References ................................................................. 94
Figures

1.1 Motivating applications .............................................. 16
1.2 Pothole Patrol: Typical pothole accelerometer waveform. .... 19
1.3 Pothole Patrol: Pothole processing and visualization. ....... 20
1.4 Data from mobile sensors can be used for traffic monitoring. 21
1.5 CarTel system architecture ......................................... 23

2.1 CarTel node hardware. ............................................. 26
2.2 CarTel node installed ............................................... 27
2.3 ICEDB remote node data path ..................................... 29
2.4 ICEDB tuple data path ............................................. 32
2.5 Second generation CarTel node hardware ....................... 34
2.6 CafNet and sockets API compared ................................. 37
2.7 CafNet stack .......................................................... 38
2.8 Portal login ........................................................... 42
2.9 Portal trace history .................................................. 43
2.10 Portal trace detail view ............................................. 44
2.11 Portal Wi-Fi coverage viewer ..................................... 45

3.1 Boston Wi-Fi Coverage Map (Seattle inset) .................... 48
3.2 Timeline of network activity as the car nears an AP .......... 52
3.3 CDF of geographic coverage of successful end-to-end pings 54
3.4 CDF showing distribution of various phases of association 56
3.5 Distribution of scan and association times ...................... 57
3.6 CDF of IP address acquisition latency .......................... 58
3.7 CCDF of association duration ..................................... 59
3.8 CDF of average speeds for associations ......................... 60
3.9 Association durations vs. speed ................................ 61
3.10 CDF of the time between connectivity events ................ 62
3.11 CCDF of the number of APs discovered ........................ 63
3.12 CDF showing the fraction of associations to any given AP 64
3.13 CCDF of connection coverage ................................... 65
3.14 CCDF of the fraction of AP pings that succeeded ........... 66
3.15 Mean Wi-Fi packet delivery rate vs. car speed ............... 67
3.16 CDF of duration .................................................... 68
3.17 CCDF of per-connection end-to-end throughput .......... 69
3.18 CCDF of per-connection bytes received at the server . . . . . . 70
4.1 Boston metro simulation area . . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.2 Simulation results: coverage for participatory monitoring. . . . . . . 80
4.3 Deployment results: coverage for participatory monitoring. . . . . . . 81
4.4 Congested road segments in the Boston area . . . . . . . . . . . . . . . . 82
4.5 Simulation results: coverage for infrastructure monitoring. . . . . . . 83
4.6 Arterial road segments in the Boston area . . . . . . . . . . . . . . . . . . . . 84
4.7 Simulation results: coverage for arterial monitoring. . . . . . . . . . . . 85
Previously Published Material


Acknowledgments

Very few projects in life are solo efforts. This dissertation is no exception and would not have been possible without the support and guidance of many people.

First, I would like to express my deep gratitude to my adviser Hari Balakrishnan. I have worked with Hari for over seven years and he has repeatedly inspired me with his passion and drive. From our first projects investigating congestion control algorithms for TinyOS powered Motes, to our most recent work building CarTel, his optimism and relentless curiosity pushed me to be a better researcher. More than that, he has served as a valued and trusted mentor, sharing much wisdom with me that I'm sure will help for many years to come.

I would like to thank committee member and fellow CarTel co-conspirator Sam Madden for, not only the technical guidance he lent to the project, but also for his willingness to listen to any and all ideas. I took advantage of his open door many times.

I would also like to thank committee member Seth Teller for his valuable feedback and help making this dissertation stronger.

The CarTel project would not have been possible without the significant contributions of many people. Vladimir Bychkovsky, a co-author on many papers, was a key member of our team who contributed much to many parts of the project. Yang Zhang’s tremendous talents were crucial for ICEDB. Allen Miu’s experience and hard work was essential for much of our Wi-Fi work. I am also grateful for the hard work of Kevin Chen, Jakob Eriksson, Lewis Girod, Michel Goraczko, Katrina LaCurts, Sejoon Lim, Eugene Shih, and Arvind Thiagarajan. Each of these people have contributed much to the success of CarTel.

My graduate career would not have been as fulfilling or as enjoyable without the many members of CSAIL.

My officemates, past and present, Vladimir Bychkovsky, Nick Feamster, Michel Goraczko, Szymon Jakubczak, Kyle Jamieson, Sam Perli, Stan Rost, Eugene Shih, Mike Walfish, and Yang Zhang have enlightened me with many insightful discussions and helped make coming into the lab enjoyable each day. I also thank Mike Walfish for the beautiful design of his dissertation, which served as the inspiration for mine.

I was lucky enough to work with a gifted set of researchers in the Net-
works and Mobile Systems Group. I thank everyone for their thoughtful
discussions as well as for the many friends I have made.

I thank Sheila Marian for all her fresh-baked treats.

Finally, I would like to express my gratitude to my parents for all of
their love and support over the years.
This dissertation is about building a low-cost, wide-area sensor system called CarTel.

Sensor systems are becoming an increasingly important component in today's data-driven marketplace. Whether it is a scientist's attempts to establish the veracity of a new theory, a business trying to provide users with fine-grained and timely traffic reports, or a city trying to decide how to allocate resources to repair its crumbling roadways, empirical data is crucial for all types of decision making. One broad class of data—environmental data—requires measuring the physical world using sensors. These sensors are used to gather data to quantify such real world phenomenon as air pollution, road vibrations, and traffic congestion. Often the various problems being investigated require data from large geographic areas: across cities or even metropolitan areas. Sensor systems help us manage this data collection challenge.

Building sensor systems is particularly difficult when you add two constrains: low-cost and wide-area. Without either one of these, the solution becomes straightforward. If cost is not an issue, traditional computational resources and networks can be deployed in high densities to gather the needed data. Likewise, if the goal is to monitor a small area, traditional approaches using static micro-sensors are more than sufficient. However, when you combine the constraints of cost and geographic area, a new approach must be taken.

This dissertation describes and analyzes our approach to solving this problem. In particular, over the next 6 chapters we answer the following question: How do we build a low-cost computing system to collect and analyze environmental data from a large urban area?
1.1 THE NEED FOR WIDE-AREA SENSING

The motivation for wide-area sensing comes from both a technology push, which is rapidly making the underlying hardware components available, and an application pull, which generates the demand for such systems. The technology push is driven by the commoditization of cheap, embedded, sensor-equipped computers and mobile phones. When connected to cars and carried by people, these devices can form a distributed mobile sensor computing system.

The application pull is driven by our increasing desire for high fidelity data from larger and larger areas to enable:

— Traffic delay monitoring, by using mobility traces to infer congestion. Some estimates [80] put the annual cost of traffic congestion in the US citizens at over $87 billion\(^1\). Moreover, during peak travel periods, over two-thirds of major roadways in urban areas experience congestion. Crucial to improving the situation is understanding exactly which roads are congested so that drivers can choose alternate routes or travel times.

— Environmental monitoring, by using mobile chemical and pollution sensors. As communities pass stricter air quality and environmental standards, they need a low-cost way to verify compliance with these new laws.

\(^1\)This figure is for 2007. The cost of congestion is calculated from drivers’ lost productivity due to delays (estimated to be on average 36 hours per person) plus the increased fuel costs due to these longer trips (estimated to be on average 24 gallons per person).
Civil infrastructure monitoring, by attaching vibration and other sensors to cars to monitor the state of roads (e.g., potholes, black ice).

Automotive diagnostics, by obtaining information from a vehicle’s onboard sensors, which can help in preventive and comparative diagnostics. This information can also be used to monitor bad driving tendencies.

Geo-imaging, by attaching cameras on cars and using mobile phone cameras to capture location-tagged images and video for various applications, including landmark-based route finding.

Data muling, by using cars (and people) as “delivery networks” for remote sensor networks, sending data from these networks to Internet servers.

These examples underlie the need for large-scale, distributed sensing. Of course, static sensor networks have been successfully used for some of these applications, particularly in environmental and civil monitoring [1, 10, 16, 55, 81]. Mobile sensor networks, however, offer the potential to instrument a much larger physical area with a smaller number of sensors, relying on node movement to cover different areas in time. This approach may be particularly useful for certain chemical and biological sensing tasks where the sensors are costly, or where the number of sensors is so large that a static deployment is either too expensive or too cumbersome to establish and maintain.

1.2 Our Approach

The traditional approach to acquiring environmental data involves placing static sensors in close proximity to the phenomena being sensed, and having that data delivered with low-latency back to a central location for analysis. For example, if a city wanted to measure vehicle congestion, road crews would need to place inductive loop sensors at every intersection to measure traffic flow and engineers would need to build a data network to link the sensors back to a central monitoring office. The primary limitation of wide-area static sensor deployments is that they are costly. A city-wide deployment might take hundreds or thousands of individual sensors. Each of these sensors needs to be securely installed, powered, and maintained.

For example, mass spectrometers, the best known way to detect various organic pollutants, cost tens of thousands of dollars [62].
What underlies our approach is a simple observation: many sensing applications have relatively low sampling and delivery requirements. Often sampling a given region a few times per hour instead of hundreds of times per second, is more than adequate. For example, measurements of traffic congestion on any given street need only occur every few minutes, and samples of air quality could be taken even less frequently. In addition, this data need not be delivered immediately to a central server for analysis. Holding onto it for minutes or hours is often tolerable and allows for more flexibility. Given these relaxed sampling and delivery requirements, rather than use a large number of static sensors attached to a low-latency network, CarTel uses a smaller number of occasionally connected mobile sensors that take advantage of opportunistic mobility and opportunistic networking. These two novel aspects form the core of our approach. We discuss them in more detail below:

**Opportunistic Mobility.** Mobile sensors are not necessarily a cost-effective solution. If we were to build robotic vehicles to spatially multiplex our sensors over a large area, one might argue that we are simply pushing the cost and complexity to the mobile platform itself. Instead, we make one additional observation: cities offer an abundance of robust mobile platforms, namely, cars and mobile phones. These platforms go just about everywhere you would want to sense and have other uses that necessitate their existence (than being a sensor platform). By piggy-backing on these platforms—taking advantage of opportunistic mobility—we can do cost-effective city-scale sensing.

We use the term “opportunistic mobility” to refer to motion whose purpose and direction is independent of our monitoring goals. On a daily basis, city streets are filled with taxi cabs, delivery trucks, and municipal vehicles that, during the course of fulfilling their normal duties, cover a substantial amount of the vehicular network. Rather than deploy cars and drivers to methodically collect sensor data, we advocate taking advantage of the seemingly random motion of everyday life to help us take a snapshot of various aspects of our urban environment. Automobiles, in particular, are an attractive platform for mobile sensing. In addition to being pervasive, cars provide an environment with abundant energy and space. Utilizing opportunistic mobility is a way of further amortizing the sunk cost of vehicle-centered communities to enable low-cost, pervasive sensing.

**Opportunistic Vehicular Networking.** Wireless data connections are quite expensive, often costing more than $500 per-year, per-device. There is a good reason for this: maintaining thousand of towers to provide pervasive high-bandwidth, low-latency connections to the Internet is quite expensive. Mobile voice applications often need the predictability that the
these planned wireless networks offer. However, there is also a broad class of sensing and messaging applications that don’t have such strict network requirements. Moreover, the world is blanketed with unstructured wireless networks, such as home access points, municipal Wi-Fi, Bluetooth, or even the physical motion of USB keys. We advocate taking advantage of these low-cost, unstructured networks—opportunistic networking—to deliver data for delay tolerant applications. Using these unstructured networks is not easy. We need a network stack that can advantage of connections that only last for a few seconds, but in aggregate offer substantial data transfer opportunities as well as an API that is a natural fit for this environment. In the end though, we envision the use of opportunistic vehicular networking as important tool for substantially reducing the cost of delivering data from the field.

1.3 CarTel Applications

The next several chapters discuss the details of CarTel, its system architecture, and underlying assumptions. To provide more context for that discussion, let us first explore two applications that we envision being built using CarTel.

The Pothole Patrol [26]. Cities spend millions of dollars every year maintaining and repairing their roadways. Roadway maintenance is a challenging and ongoing problem because harsh weather conditions, unexpected traffic load, and normal wear and tear all degrade road surfaces at unexpected rates. Given limited budgets, the key challenge for cities is deciding which roads to fix. Moreover, it would be useful to notify drivers of potentially hazardous conditions before road crews are able to make repairs.

Road surface monitoring is ideally suited for CarTel. First, it funda-
mentally requires mobility. Road surface conditions are naturally sensed from a moving entity that can measure impulses and vibrations. Second, the time scales over which roads degrade is weeks, if not months. Clearly, we don’t need static, high-fidelity, low-latency sensing.

Using the CarTel approach, this problem could be solved by attaching multi-axis accelerometers to city owned or affiliated vehicles (garbage, road, mail delivery), using the opportunistic mobility provided by these vehicles to cover the needed area without any additional cost. These accelerometers would continuously record vibrations as employees go about their normal work routine. Whenever a driver hits a pothole or other road anomaly, as shown in Figure 1.2, its distinct waveform could be picked out and its location noted. As the vehicles encounter connectivity throughout the day, these sightings would be uploaded to a central server, where additional analysis could be performed on the aggregate data to discard false-positives. This connectivity could be provided by a municipal Wi-Fi network, if present, or strategically placed access points on city owned buildings, all requiring little additional cost. Finally, city managers could be presented with a map showing the most severe road surface anomalies, as shown in Figure 1.3. Road crews could then be sent to the most pressing
problems in a timely manner.

**Real-time Traffic Monitoring [54].** Traffic is the bane of most peoples' commutes. Cities and companies have taken countless approaches to monitor and mitigate traffic congestion, from helicopters providing live coverage of known hotspots, to crowd-sourced traffic reports from drivers, to traffic flow reports from sensors embedded in freeways. However, all of these approaches fail in that they do not provide pervasive coverage. Traditional approaches may be quick to pick up the peaks—ten car pile-ups that stop traffic for miles—but often fail to pick up the long tail of congestion that adds a few minutes to each of the segments of a commute.

Again, CarTel is particularly well-suited to solve this problem by enabling pervasive, real-time traffic monitoring as shown in Figure 1.4. Traffic is a perfect problem to crowd-source: opportunistic mobility is not only a cost saving strategy, but a natural way of recording congestion. The increasing prevalence of location aware smartphones makes gathering location from a large number of users straight-forward. One way to incentivize users to install your application would be to provide users with a log of their driving, accessed via a commute portal. This would allow users to monitor route performance along various commute paths as well as keep track of mileage related vehicle maintenance.

Once data is flowing, providing delay samples for a large cross sec-
tion of streets and times of day, it is possible to build statistical models to predict future congestion and make more informed route predictions.

1.4 CARTEL: A MOBILE SENSOR COMPUTING SYSTEM

With these observations and applications in mind, we designed a system to make it easy to collect, deliver, and visualize data from a set of remote, mobile, and intermittently connected nodes. This system, CarTel, provides a simple, centralized programming interface, handles large volumes of heterogeneous sensor data, and copes with variable, intermittent network connectivity, hiding these details from the application developer. Because an interesting class of mobile sensing applications are enabled by automotive sensor networks, we use CarTel nodes embedded in cars as the test case and develop schemes to cope with vehicular movement.

This dissertation focuses on the design, implementation, and evaluation of CarTel, a mobile sensor computing system.

CarTel architecture. First, in Chapter 2, we lay the conceptual groundwork for the design of CarTel. The system relies on the abundance of both opportunistic mobility and opportunistic Wi-Fi. By placing small computing nodes equipped with sensors in peoples’ cars and delivering sensor data to our servers using the existing, ad-hoc arrangement of access points in peoples’ homes and businesses, we build an inexpensive data collection system. Figure 1.5 schematically shows this architecture. There are three high-level components to CarTel: remote nodes operating in the vehicles and their associated data collection software, a network stack designed for variable connectivity, and a data processing portal for visualizing results. Taken together, these components provide a system that enables a wide range of applications. We have implemented several sample applications that allow users to view their most recent drives, making it easy for them to optimize commute routes. We also show an application that aggregates all of the user’s drive data to determine potential points of congestion. We discuss in detail the design and implementation of each of these components.

Opportunistic, Vehicular Wi-Fi. The network architecture of CarTel relies on using open access points for data delivery. In Chapter 3, we present a detailed study of Internet performance using access points when connecting from moving vehicles. We show that, not only is it possible to build a data delivery network in which connections last for just seconds, but that the performance and coverage is much better than expected. In fact, the median connection time exceeded 13 seconds, with the
mean inter-arrival duration between open, associable APs being 75 seconds, even from moving cars. This chapter also provides an analysis of the CarTel architecture, as we use the system itself to measure the performance of these networks.

**CarTel deployment considerations.** Finally, in Chapter 4, we answer several questions relating to the deployment densities we might need for various types of applications. Real world analysis has been limited to at most tens of cars. If a city were to actually deploy this system on their fleets of municipal vehicles or taxis, we show through simulation that hundreds, but no more than a few thousand vehicles, are needed for many applications.

It should also be noted that this dissertation does not fully address potential privacy concerns users might have. We readily concede that most users will not want open access to detailed traces of their vehicular movements. With that in mind, all data is individually controlled and password protected. However, should users' location data be used in aggregate or otherwise shared, suitable anonymization would need to be performed to ensure that individually identifiable movement is not discerned from the
data.

1.5 Contributions

The over-arching contribution of this dissertation is showing that it is possible to build a low-cost, wide-area sensor computing system using opportunistic mobility and opportunistic, vehicular Wi-Fi. Specifically, by designing and building CarTel, we were able to show the following:

An architecture for large-scale, mobile data collection. First, we show that it's possible to effectively monitor a large geographic area piggybacking on peoples' everyday travels. Moreover, we present the design, implementation, and evaluation of this system.

Opportunistic, vehicular Wi-Fi. Second, we show the viability of using opportunistic Wi-Fi as a data delivery network. With the abundance of Wi-Fi in urban areas and a network stack tuned for short duration connections, we can support latency tolerant applications at a relatively low cost. Our results show that the mean inter-arrival duration between open, associate APs is about 75 s, and that the median end-to-end TCP throughput that these APs can support is about 30 KBytes/s.

Coverage properties of opportunistic mobility. Finally, we show through simulation how a system that relies on opportunistic mobility, such as CarTel, might scale. We show that, with just 100 drive hours, over 80% of the traversed road mileage is sampled at least 10 times – a finding that supports using this type of motion for a real-time traffic congestion monitoring service.
2

CarTel System Architecture

This chapter describes the system architecture of CarTel. The system has three components: (1) a set of data collection scripts that run on a small, embedded computer (or smartphone) placed in each car, (2) a communication stack designed to deliver data using short, variable duration network connections, and (3) a server-based data visualization analysis portal running on an Internet server. Below, we describe each in more detail.

2.1 Data Collection

As noted in Chapter 1, CarTel relies on opportunistic mobility — in our implementation we focus on taxis and private cars as our mobility platform. One of our primary concerns for our in-car hardware was ensuring that the packaging could unobtrusively fit in the cabin. In addition, we wanted a system that could run a standard version of Linux and was sufficiently powerful to run programs written in interpreted languages, such as Python and Perl.

Below we describe the hardware installed in each vehicle to run our data collection software. This part of the system underwent a substantial evolution as the project progressed. Building embedded, networked software systems is an art as much as it is a science: the key difficulty is matching software demands to hardware capabilities. Over the many design iterations of CarTel, our requirements changed as the applications and vision for this system shifted. We describe the several versions of the data collection software to show several points in the design space.

2.1.1 Remote Sensing Hardware

First, before we get into the details of data collection, we need to understand the hardware environment on which the software runs. The core of the node hardware is a Soekris net4801 that has a 586-class processor run-
CPU: 266MHz, 586-class
Memory: 128 MB
Storage: 1 GB flash
Wi-Fi: 802.11b, Prism 2.5
Antenna: 5.5 dBi rubber-duck
GPS: Rayming TN200

FIGURE 2.1—CarTel node hardware.

ning at 266 MHz with 128 MB of RAM and 1 GByte of flash memory. Each embedded computer has a high-powered 802.11b miniPCI Wi-Fi card attached to a omni-directional rubber-duck antenna and a GPS receiver. Occasionally, we attached a USB digital camera for imaging applications, an accelerometer for road quality studies, and an ODB-II interface for monitoring vehicle performance. Figure 2.1 shows this low-profile platform. The remote node software runs a standard Linux kernel, complete with a PostgreSQL database and scripts for data acquisition from various sensors.

To power each node, we plug it directly into the vehicle's cigarette lighter socket. Figure 2.2 shows one unit installed in a vehicle. In most vehicles, these ports are active only when the ignition is on. Hence, our remote nodes are powered only when the engine is on. This property turns out to be attractive because we do not have to worry about inadvertently draining a car's battery and because it naturally demarcates the data into useful chunks. However, we must also write fault tolerant software as the power could go out an any moment. We have also experimented with various battery-operated nodes, both on cars and on bicycle rides.

2.1.2 Data Collection V1: CarTelCollect

Our first version of the data collection software, called CarTelCollect, is a set of scripts and utilities installed on the remote nodes that acquire sensor readings and deliver them to our central sever. Each sensor, whether
a GPS receiver, Wi-Fi card, accelerometer, or camera, has a small acquisition script that reads samples from the device and stores them in a local database. Some scripts simply poll the device periodically, while others run continuously and do substantially more processing to generate a sample. For example, the GPS receiver sends location updates once per second, whereas AP monitoring runs continuously scanning each frequency, but only updates the database when a new access point is discovered.

We chose to use a full-fledged database on the remote nodes because, in addition to providing a convenient interface and well-known semantics, such a system provides a clear upgrade path for future in-network query processing.

In order for this data to be transmitted to our central server, the CarTelCollect software extracts the samples from the local database whenever connectivity is present.

In the initial version of our software, this extraction phase would take place immediately after a usable connection was found. This phase would involve selecting all new samples from the database and exporting them into fixed-length, compressed text files. Next, each file would be sent to the central server using an HTTP POST. After receiving each file, the central server would unzip them and insert the samples into its master database. If the data was successfully inserted, the central server would return an OK to the remote node, allowing it to delete the sensor data.

After deploying a few remote nodes, we noticed that extracting samples from the database took tens of seconds. Unfortunately, connections often lasted less than 10 seconds, due to vehicle mobility. This meant that many usable connections went unutilized due to our slow extraction rate.

As a result, we modified CarTelCollect to decouple sample extraction from transmission on both the remote node and the server. Switching to an asynchronous mode of operation required us to develop a new mechanism for deciding when data had been successfully inserted into the central database. Instead of relying on the synchronous OK from the server, we
added a column to the local database on the remote node that describes the number of times each record had been extracted into a local file. When deciding which samples to extract for transmission, we select those with the fewest number of extractions. Whenever a connection is found, the samples are already compressed and on disk, ready to be sent out immediately. Likewise, on the central server, when a file is received from a remote node, it is simply stored on disk, allowing the next data file to be sent immediately. A separate process uncompresses and inserts sensor samples into the central server’s database. Because we don’t use end-to-end acknowledgments, data is deleted once the remote node’s disk reaches capacity, with records that have the highest probability (extraction count) of being on the central server being deleted first from the database.

After experience using CarTelCollect, we discovered several key system limitations:

1. **Data prioritization.** Many different data streams flow through each remote node. Given the variable nature of the bandwidth, much of the collected data may not make it off the remote node until much later. Consequently, it would be quite useful to be able to set stream priorities. For example, it is desirable to send out streams producing engine warning events before raw acceleration readings.

2. **Sensor schema management.** Adding new sensors is a painful process. The code must be manually updated to support any new sensors or data types. The system could benefit from a general way of describing sensors that could be pushed out to cars as hardware or application requirements change.

3. **Snapshot queries.** There is significant bandwidth variability between the individual nodes. Moreover, many sensors generate data at a much higher rate than could be realistically delivered. We would like to be able to send a low rate stream of events back to our central server, and then asynchronously request high fidelity snapshots of the data for events that interest us. For example, an engine warning event might cause us to request detailed RPM readings around the time of the event, or after a traffic accident we might request high resolution imagery from cars in that area at the time of the incident.

### 2.1.3 Data Collection V2: ICEDB

The limitations described in the previous section caused us to re-evaluate our data collection approach. Might a database-centric approach more naturally lend itself to the problem and allow for far more flexibility?

The concept of viewing sensor networks as a streaming database is not new [53]. However, there are two crucial differences between the op-
FIGURE 2.3—the ICEDB remote node data path. Adapters acquire samples from attached sensors. These samples are forwarded both to disk, where they can be later queried if needed, and to the continuous query engine, where a low resolution version of the stream can be sent back to the central server over CafNet. The ICEDB server sends out updated continuous and snapshot queries as needed.

Operating environment of CarTel and that of most systems. First, rather than having a constant network connection, the inherent mobility of our system means that nodes experience extended periods without connectivity. Second, in most systems there is a careful balance between producers and consumers of data. However, in CarTel, media-rich sensors are able to collect far more data than can be sent back even over the most robust of links. These two properties lead us to design ICEDB [88], an Intermittently Connected Embedded Database. ICEDB uses two core ideas to address these challenges:

1. **Delay-tolerant continuous query processing.** Rather than treating disconnection as a fault, in ICEDB the local query processor continues to gather, store, and process sensor data. We use a buffering mechanism and a protocol for managing and staging the delivery of query results from the mobile nodes to the portal.

2. **Inter- and intra-stream prioritization of query results.** ICEDB introduces a set of SQL extensions to declaratively express inter- and intra-stream prioritization, in an effort to make the best use of available bandwidth. These extensions allow for both local (on the mobile node) as well as global (portal-driven) prioritization of results.

* * *

ICEDB distributes query execution and result delivery between the ICEDB server running on the Portal and the remote nodes. The ICEDB server
maintains a list of continuous queries submitted by applications that are pushed to the remote nodes using CafNet. The nodes in the field run ICEDB remote to process the sensor data and return the query results using CafNet, prioritizing the result streams in order of importance. Finally, as the ICEDB server receives results from remote nodes, it places them into a per-query result table in the relational database at the Portal. Applications can access the partial results of any continuous query by submitting ad hoc queries to the ICEDB server.

**Data Model.** ICEDB supports heterogeneous data types and makes the addition and removal of sensors relatively easy. Figure 2.3 shows how data flows inside a ICEDB on a remote node. ICEDB’s mechanism for handling new sensor types and managing schemas is a meta-data package called an adapter, which consists of the attributes of a sensor as well as an executable program (usually a script) that interfaces with the sensor and triggers data collection. These attributes provide ICEDB with enough information to: (1) automatically create local tables to store sensor readings (i.e., without any manual configuration on the remote node), (2) acquire tuples from the sensor, and (3) parse sensor readings to store them in the database and process them as specified by subsequent continuous queries. This scheme is similar to the wrappers found in Mediation systems [86].

Applications can define adapters programmatically. Adapters can also be specified by the CarTel administrator using a Web form interface in the Portal. Once defined, adapters reside inside the ICEDB server on the portal and are pushed out to remote nodes using CafNet.

CarTel has adapters for node diagnostics, the GPS receiver, the OBD-II interface, the Wi-Fi interface, and the digital camera. There may not be a one-to-one correspondence between adapters and physical sensors; a single physical sensor may be abstracted using multiple adapters. For example, the Wi-Fi interface uses three adapters, which handle the data resulting from access points scans, access point connections, and network configurations.

**Continuous Query Model.** Queries in ICEDB are written in SQL with several extensions for continuous queries and prioritization. These queries are run over data as it is produced by the adapters. To support continuous queries in ICEDB, queries include a sample rate specified by a RATE clause. For example, consider the query:

```sql
SELECT carid,traceid,time,location FROM gps WHERE gps.time BETWEEN now()-1 mins AND now() RATE 5 mins
```

Here, each car will report its last one minute of GPS data once every five minutes. These batches of results will be delivered whenever the car is
next able to send data to the portal.

To ensure that readings captured across cars are comparable (e.g., in join or aggregate queries run over the data stored at the portal), cars synchronize their clocks using GPS (when available). Readings are acquired when the clock is divisible by the \texttt{RATE} (so if the current time is 10:02 AM, the above query would acquire readings at 10:05, 10:10, etc.)

\begin{center}
\texttt{****}
\end{center}

In an intermittently-connected, bandwidth-constrained environment, delivering all data in FIFO order is sub-optimal. The "value" of any data is often application-dependent (for example, one application may be interested in data that shows times when a car is speeding, whereas another application may be interested in times when a car is subject to unusual slowdowns). For this reason, ICEDB provides a declarative way for applications to express what data is important. ICEDB uses these specifications to develop a total ordering on the local query results that need to be sent over the network.

To prioritize data for delivery, the ICEDB query language assigns each result tuple a "score" corresponding to its delivery priority. \textit{Local prioritization} produces scorings of data tuples that can dynamically change over time based on other data present at the local node. However, local prioritization is limited because it cannot receive feedback from the portal, which has a global view of the data and can hence make more informed choices regarding what data to send first. \textit{Global prioritization} is a scoring of tuples influenced by feedback from the portal. In order to achieve global prioritization, each time a node establishes a connection, it sends to the portal a synopsis of its query results, and the portal responds with a global prioritization of this coarse representation of the data.

On each node, query results are stored into a named buffer as they are produced. The different prioritization schemes result in different orderings of this buffer; as connections occur, this buffer is drained in order. We have chosen to specify these different prioritization schemes via additional statements attached to the continuous queries in the system. There is nothing fundamental about coupling the query language and prioritization language in this way; prioritization statements could also be sent separately from queries, but it is convenient to use the query language to express dynamic priorities. Figure 2.4 schematically shows how these features work together inside of an output buffer to provide the needed prioritization. Below we go into more detail about schemes for local and global prioritization.

\textbf{Local Prioritization.} Local prioritization uses two language extensions for specifying the local transmission order of query results: \texttt{PRIORITY} and
DELIVERY ORDER.

The PRIORITY clause is specified at the end of a query and assigns a numeric priority to the query’s result buffer. ICEDB transmits query result buffers strictly in order of priority, ensuring that high priority queries (e.g., small, event detection queries) are transmitted before low priority queries (e.g., raw GPS data).

The DELIVERY ORDER clause allows the remote node to locally determine the transmission order of results within a given query buffer. Like a traditional SQL ORDER BY clause, DELIVERY ORDER can take attribute names to statically order by those columns. However, when prioritizing delivery for intermittent network connectivity many types of data would benefit from a more dynamic ordering that depends on the entire set of tuples. To enable this dynamic ordering, DELIVERY ORDER can take the name of a user-defined function that takes as input the entire set of pending results and produces a new score for each result. Because the DELIVERY ORDER function has direct access to the entire result set, the ordering of results can depend on the other results in the buffer, which cannot be done with a traditional SQL ORDER BY clause.

As an example, when collecting a car’s GPS position reports, the user may wish to order the points such that an application on the portal can construct a piecewise linear curve approximating a particular trace. One simple implementation would be to recursively bisect (in the time do-
main) the trace: first, our DELIVERY ORDER function would transmit the
endpoints of the trace; then, it would send the point exactly between those
endpoints to bisect the trace, and then continue recursively bisecting the
sub-traces in exactly the same manner. Simple ORDER BY cannot do this,
however, because the score it assigns to each tuple cannot depend on the
other tuples in the buffer—meaning, for example, the score of a midpoint
of a segment cannot depend on previously chosen endpoints. Using the
bisect approach, the resolution of the route is progressively enhanced as
more data is received. This bisection algorithm and other commonly used
prioritization functions are available in a standard library, and users can
implement their own local prioritization functions.

**Global Prioritization.** ICEDB applications express global priorities us-
ing the SUMMARIZE AS clause, which specifies a query that will compute
a summary, which consists of a set of tuples that summarize the entire
buffer of result tuples. When connectivity occurs, before any query results
are transferred, this summary is sent to the portal. The portal applies a
user-specified function to order the tuples in the summary, and send this
prioritization back to the node, which it then uses to order the entire set
of result tuples. The basic syntax of this clause is shown in this query:

```sql
SELECT ...
EVERY ...
BUFFER IN bufname
SUMMARIZE AS
    SELECT f1,...,fn,agg(fn+1),...,agg(fn+m)
    FROM bufname WHERE pred1...predn
    GROUP BY f1,...,fn
```

The SUMMARIZE AS clause uses grouping and aggregation to partition the
buffered result data into groups and compute summary statistics over
each group. For example, if cars are collecting tuples of the form <lat,
lon, roadname, speed>, the summary query might partition the data by
roadname and compute the average speed over each road. On the server,
the user specifies a function that orders the summary – in our example, it
might order roads according to those which it has heard the least informa-
tion about in the past day. Once this ordering is returned from the server,
the remote ICEDB instance automatically orders the result tuples in the
same order as specified in the server’s ordering of the summary (using a
join query between the server’s summary table and the raw data.) Once
this total ordering has been computed, the complete set of in-order results
are delivered in order to the server.

Server prioritization is useful in situations in which there are several
nodes collecting similar data about the same location, or when a portal
application has changing information needs. The server requests that data
be returned in an order that will provide the most information about areas
other cars have not observed or that are of particular interest to current
portal applications.

In the end ICEDB proved to be quite flexible. However, our implementa-
tion using Python and PostgreSQL pushed the limits of our hardware.
Subsequent versions of the remote node hardware were moving towards
less computing power in an effort to reduce size and cost. We also chose to
focus more heavily on measuring traffic congestion, and hence the expres-
siveness offered by ICEDB and its prioritization primitives proved to be
more than we needed. Consequently, ICEDB was never widely deployed,
although we feel many of the ideas demonstrated by ICEDB are useful for
systems designers.

2.1.4 Data Collection V3: dPipe + QuickWifi

The adoption of a new hardware platform that was smaller and computa-
tionally limited spurred the development of the final version of the Car-
Tel data collection software. Our second generation hardware platform,
shown in Figure 2.5, uses a commodity wireless access point plus a cus-
tom daughter-board for interfacing with external sensors. This reduced
environment meant that we could no longer run ICEDB. This turned out
to be a blessing in disguise, at it forced us to focus our efforts.
First, we reduced our schema to only those fields needed for location, wireless scan records, and acceleration readings. Second, we decided to store sensor data in flat text files to be eventually uploaded to a central server for analysis. Finally, we built a tool, similar to UNIX pipes, called dPipe, which allows us to easily transfer piped data across an intermittently connected network. These delay-tolerant pipes, written by the authors of and briefly mentioned in [26], would buffer outgoing data on disk and then send it to the central server whenever a connection was found using an integrated network stack called QuickWifi. We describe QuickWifi in more detail in Chapter 3.5.

In the end, this final design iteration proved to be the most successful, with various versions being deployed across dozens of vehicles.

2.2 CafNet

Data transport in CarTel is handled by CafNet, a general-purpose network stack for delay-tolerant communication. Recall from Chapter 1, the networking environment for CarTel is quite distinct from your typical corporate LAN. With vehicles traveling in the tens of meters per second, Internet connectivity from inside these vehicles will be fleeting at best. Moreover, there are many instances when connectivity is non-existent, such as between major population centers or inside tunnels. Consequently, this challenging environment calls for a network stack that opportunistically takes advantage of all forms of connectivity. This includes transparently utilizing disparate network technologies, such as Wi-Fi or Bluetooth, as well data-muling protocols that allow data to be forwarded using one or more intermediate, physically moving mules (i.e. car-to-car transport or USB keys).

Applications can use CafNet to send messages across an intermittently connected network. Its mechanisms allow messages to be delivered across two kinds of intermittency: first, when end-to-end connectivity is available between the sending and receiving application, but is intermittent; and second, when the only form of connectivity is via one or more intermediate mules. In CarTel, the portal and the mobile nodes communicate with each other using CafNet across both forms of intermittent connectivity.

It may not be immediately apparent why we need CafNet, but it fills a particular niche. In a continuously connected, high bandwidth environment, TCP and other Internet protocols provide a reasonable compromise between efficiency and responding aggressively to potential congestion. However, the network landscape of CarTel is different. The mobility of our collection nodes make connection intermittency almost inevitable, bandwidth variable, and message latency potentially unbounded (Chapter 3
evaluates this type of connectivity in much greater detail). TCP provides a poor transport layer abstraction for applications dealing with this environment. The rest of this section describes how and why we built CafNet to address the unique nature of this environment.

* * *

All CafNet nodes are named using globally unique flat identifiers that don’t embed any topological or organizational semantics. CafNet offers a message-oriented data transmission and reception API to applications, not a stream-oriented connection abstraction like TCP. As previous work has shown [15, 27], a message abstraction is better suited to a network whose delays could be minutes or hours.

The unit of data transport in CafNet is an Application Data Unit (ADU) [20]. Each ADU has an identifier; the combination of source, destination, and ADU ID is unique. (The terms “message” and “ADU” refer to the same thing.)

Unlike the traditional sockets interface, a CafNet application does not call send(ADU) when it has data to send. The reason is that if the host is currently not connected to the destination, this message would simply be buffered in the protocol stack (e.g., at the transport layer). Such buffers could grow quite large, but more importantly, all data in those buffers would end up being sent in FIFO order. FIFO packet delivery is a mismatch for many delay-tolerant network applications, including ICEDB, which require and benefit from dynamic priorities. In general, only the application knows which messages are currently most important.

What is needed is a scheme where the network stack buffers no data, but just informs the application when connectivity is available or when network conditions change. If all data buffers were maintained only by the application (which already has the data in RAM or on disk), and if it were able to respond quickly to callbacks from the network stack, then dynamic priorities and fine-grained departures from FIFO delivery order would be easier to achieve. CafNet adopts this basic approach: CafNet informs the application when connectivity is available or changes, and in response, the application decides what data to send “at the last moment”, rather than committing that data to the network in advance. Figure 2.6 shows a comparison between CafNet and socket APIs.

CafNet defines a three-layer protocol stack. In this stack, the CafNet Transport Layer (CTL) provides this notification to the application. In the basic version of the stack, the API consists of just one callback function: cb_get_adu(), which causes the application to synchronously return an

---

1 As in previous work such as DOA [84], making these identifiers a hash of a public key (and a random salt) would ease message authentication.
FIGURE 2.6—The buffering and data-flow of the CafNet (left) stack differs from that of many socket APIs (right). With CafNet, once (1) a connection is received the (2) the application receives a call-back requesting an ADU. The application (3) returns the next ADU to be transmitted, which is then transmitted over the recently established connection. By contrast, with sockets, the application initially (1) enqueues its data in the network stack. This data (potentially stale) is drained (2) once a connection is received and the data (3) can be transmitted.

ADU for (presumably) immediate transmission. The CTL also provides a (standard) input() function to receive messages from the lower layers of the stack.

CafNet hides the details of the communication medium (Wi-Fi, Bluetooth, flash memory, etc.) from the CTL and the application. All media-dependent tasks are performed by the lowest layer of the CafNet stack, the Mule Adaptation Layer (MAL), which presents a media-independent interface to the higher layers. The MAL implements media-specific discovery protocols, and sends and receives messages across several possible communication channels (TCP connections to Internet hosts, TCP or media-specific protocols to mules across a “one-hop” channel, writes and reads of data on portable disks, etc.). When the MAL detects any connectivity, it issues a callback to the higher layers informing them of that event. This callback propagates until the application’s cb_get_adu() returns an ADU for transmission to some destination.

Bridging the CTL and the MAL is the CafNet Network Layer (CNL), which handles routing. In our current implementation, the CNL implements only static routing (it can also flood messages to all mules it encounters). On any intermediate node muling data, the CNL also buffers messages. In the basic version of the stack, the CTL, CNL, and MAL on the sending application’s node do not buffer more than one message at a
Figure 2.7—The CafNet communication stack.

Section 2.2.1 describes some additional details of these three layers. In Section 2.2.2, we describe an important set of optimizations to improve the performance of this basic stack, which requires some buffering in the network stack as well as an API extension.

2.2.1 The Basic CafNet Stack

Figure 2.7 depicts the CafNet communication stack. The functions shown in the picture for each layer are for the version that includes the performance optimizations; for now, assume that all the message buffering is in the application alone. The CTL can be implemented as a library that applications link against or as a separate process that communicates with the application using remote procedure calls, while the CNL and MAL are separate daemons that the CTL library communicates with over a socket interface. This architecture means that it is possible to implement the stack without making kernel changes.

The CTL provides optional delivery confirmation service. The application can specify what type of delivery confirmation it wants by setting a flag (NONE or END2END) on the ADU header when it returns the ADU in the cb_get_adu() call. END2END requires the CTL to periodically retransmit a given ADU until either: (1) an acknowledgment is eventually received from the destination node, or (2) the ADU is “canceled” by the sending
application, or (3) a certain maximum number of retransmissions have been attempted.

The CNL's API is simple: when the CTL gets an ADU from the application, it can call the CNL's send(dest, ADU) function, which forwards the ADU towards the destination. The CNL uses its routing tables to decide how to forward the message. The CNL's send() provides only best effort semantics.

In addition to send(nexthop, ADU), which sends a given ADU to the node with ID nexthop, the MAL invokes a callback function implemented by the CNL to update the list of currently reachable CafNet nodes. This cb_neighbor_list(neighbor_list) call always provides a complete list of reachable neighbors to save the higher layers the trouble of detecting if any given CafNet "link" is working or not.

CafNet provides peer discovery in the lowest layer (MAL) of its stack because those mechanisms are media-specific. For example, our current implementation includes a MAL layer for Wi-Fi; in order to provide Wi-Fi connectivity at vehicular speeds, it provides fast scans and associations. We are implementing other MALs, which will require other media-specific support. For example, a Bluetooth-enabled cellphone might present itself as a single next-hop contact whose discovery requires Bluetooth protocols. A passive device such as a USB Key would present itself as a set of peers that it had visited in the past. Any connection to the Internet would present itself as a list of CafNet-enabled peers (or a more concise "Internet" peer, saying that the link has Internet connectivity).

### 2.2.2 Optimizations and Enhancements

The above design is "pure" (no network buffering), but performs poorly when the average duration of connectivity is not significantly larger than the time required for the application to package and return data in response to a cb_get_adu() call. This problem is not academic—for some ICEDB queries, it takes several seconds to package data, reading tuples from a relational database on the mobile nodes. At vehicular speeds, Wi-Fi connectivity often lasts only a few seconds.

To solve this problem (which we experienced in our initial implementation), CafNet introduces a small amount of buffering in the stack. The CNL (rather than the CTL) is the natural place for this buffering, because intermediate mules already require such buffers.

Applications no longer receive callbacks upon discovering connectivity, but do so as soon as any space is available in the CNL buffer. This notification from the CNL, clear_to_send(nbytes), allows the CTL to send() up to nbytes worth of messages to the CNL. This modification to the basic stack allows CafNet to achieve high network utilization when connectivity is fleeting.
Setting the CNL buffer to be too large, however, hinders the application’s ability to prioritize data. For example, because ICEDB dynamically re-evaluates the importance of each chunk of data based on the latest queries and sensor inputs, a problem arises when priorities of data already buffered for transmission need to change. A plausible solution might be to expand the CafNet interface to make the CNL buffer visible to the application, allowing it to change priorities of buffered messages. Unfortunately, this approach is both complicated and violates layering.

To mitigate the problem, CafNet simply allows the application to set a desired size for its CNL buffer. Applications that require dynamic priorities set a buffer size just large enough to mask the delay in re-prioritizing and packaging data when network connectivity is made available.

The above API focuses on the novel aspects of our design and is not complete; for instance, it does not include the data reception path, which is similar to traditional protocol stacks. It also does not include some other details such as the application being informed of what destinations are now reachable in the callback invocation, functions to manage the CNL buffer, functions to cancel previous transmissions, etc.

### 2.3 The Portal

Users navigate sensor data in CarTel using web-based applications hosted within the Portal environment. An example of such an application is shown in Figure 2.10 in which a user views the velocity and location of his car overlaid on a map. In general, CarTel applications use the three main components of the Portal environment: (1) the Portal framework, (2) the ICEDB server to retrieve sensor data, and (3) a data visualization library to display geo-coded attributes.

#### 2.3.1 Architecture

The Portal framework provides the scaffolding for building applications that share a common user authentication mechanism and a common look-and-feel. Currently, to alleviate privacy concerns, users are only allowed to view sensor data collected from remote nodes that they host. Some applications may also report aggregate or anonymized statistics from many users.

The Portal maintains a number of database relations, as described in Table 2.1, to process and display sensor data. Every 15 minutes, a custom script that resides in the CarTelCollect database executes, looking for new sensor data. If new sensor data is found, it is cross-referenced with the GPS data and geocoded, and the requisite Portal tables are updated to support the various data layers.
TABLE 2.1—Portal database tables.

<table>
<thead>
<tr>
<th>users</th>
<th>user management and passwords</th>
</tr>
</thead>
<tbody>
<tr>
<td>preferences</td>
<td>user preferences</td>
</tr>
<tr>
<td>unit_history</td>
<td>maps remote units to usernames</td>
</tr>
<tr>
<td>traces</td>
<td>summary of users' trips</td>
</tr>
<tr>
<td>gpslog</td>
<td>tracks which GPS samples have been processed</td>
</tr>
<tr>
<td>wlanconnlog</td>
<td>tracks which Wi-Fi samples have been processed</td>
</tr>
<tr>
<td>sensor_wifi</td>
<td>localization for Wi-Fi samples</td>
</tr>
<tr>
<td>obdlog</td>
<td>tracks which ODB samples have been processed</td>
</tr>
<tr>
<td>sensor_obd</td>
<td>localization for OBD samples</td>
</tr>
<tr>
<td>cameralog</td>
<td>tracks which camera samples have been processed</td>
</tr>
<tr>
<td>sensor_camera</td>
<td>localization for camera samples</td>
</tr>
<tr>
<td>overlay_types</td>
<td>available overlays for trace detail page</td>
</tr>
<tr>
<td>overlays_altitude</td>
<td>summary of altitude overlay</td>
</tr>
<tr>
<td>overlays_speed</td>
<td>summary of speed overlay</td>
</tr>
<tr>
<td>overlays_wifi</td>
<td>summary of Wi-Fi overlay</td>
</tr>
<tr>
<td>overlays_rpm</td>
<td>summary of OBD overlay</td>
</tr>
<tr>
<td>overlays_camera</td>
<td>summary of camera overlay</td>
</tr>
</tbody>
</table>

Applications communicate with CarTelCollect to issue continuous queries and to view the results of these queries using snapshot queries on the relational database. Once submitted, the ICEDB server pushes these continuous queries out to the remote nodes. Because the results of each continuous query are stored in a table on the ICEDB server, applications can display intermediate results at any time using values from a query's result table. We envision applications interacting with the ICEDB server in different ways, including those that repeatedly issue and withdraw continuous queries based on user input, as well as those that derive all necessary sensor data from a few long-running continuous queries.

Because a large class of collected data is geo-spatial, a natural way for users to interact with the data is using a visual interface. To this end, the Portal provides a library that applications can use to display geographic overlays. The fundamental data segmentation abstraction in this visualization library is called a trace. Traces are designed to encompass all sensor data collected during a single trip (i.e., between “ignition on” and “ignition off”). This library provides two classes of functions: (1) an interface for searching for traces using spatial queries and (2) an interface for overlaying geographic attributes on a map (Google maps [32] in our current implementation) for a given trace. Not all applications will find the abstractions made by the visualization library appropriate for displaying their results. For example, one such application displays the top ten traffic congestion hot spots seen by a user. For this type of application the trace
abstraction does not make sense because its goal is to present an aggregate view of the user's driving experience. However, such applications still take advantage of the rest of the Portal framework and issue both continuous and snapshot queries.

2.3.2 User Interface

The CarTel Portal provides user-level authentication and viewing of sensor data. As shown in Figure 2.8, each user must login with his or her username and password and is only allowed to view data collected from vehicles that the user registers with the Portal.

Once logged in, the Portal presents the user with an overview of recently collected traces, as shown in Figure 2.9. On the left-hand side of the page, a list summarizes the user's most recent drives, providing id (which can be useful finding or recording a specific trace), date, duration, and distance. If there are more than 30 traces in the users history, the list is paginated. The right-hand side of the page, the trajectory of each one of these traces is shown on a map, zoomed to the geographic extent of the data. When the user clicks the check box next to any trace, the associated
trace on the map is highlighted in red.

From this view, the user can also search his trace history using the query box across the top of the page. The query language consists of an arbitrary length list of operator:argument pairs. The valid operators include:

- `<after:date>` Traces starting after `date`.
- `<before:date>` Traces starting before `date`.
- `<on:date>` Traces starting on `date`.
- `<longer:dist>` Traces longer than `dist` miles.
- `<shorter:dist>` Traces shorter than `dist` miles.
- `<id:id>` Trace with matching `id`.
- `<region:operator>` The operator can be one of (`intersects`, `disjoint`, `within`)
FIGURE 2.10—Portal trace detail view. Screenshot showing a user viewing the speed overlay for one trace.

The search interface becomes useful once the number of traces becomes large. For example, if a user wants to find all traces that correspond to his commute, doing so would be quite tedious if the data is sorted chronologically. To make it easier to mine the traces to answer these sorts of questions easier, we allow users to "visually query" their data using graphically defined "interest regions" and operators. This feature is shown in Figure 2.9 where the user has selected two regions—the rectangles—that correspond to the beginning and end of his commute. The operator region intersects is automatically added to the query search field. Should the user want to search using a different type of region operation, he can change the argument to be disjoint or within. Additionally, if the user is only interested in those traces from the last month, filtering by date can be specified in the query options. When the user pushes the refine button, only those traces that intersect both interest regions and are from the last month are returned.

When a user finds a trace of interest, he can click on the details link to view the sensor data associated with it. Each application can export a geographic overlay that a user selects from within this detailed view of
the trace data. Figure 2.10 shows the travel delay application being used to show the speed overlay for a trace in which a color-coded sequence of line segments corresponds to the car’s route and velocity. This application also places a marker at the position of the vehicle for each quartile of elapsed time, giving users an idea as to which segments of their routes account for their time. Other examples of applications implemented on the Portal include those that visualize OBD-II data, Wi-Fi connectivity, street-level imagery, and altitude.

In addition to trace-based data viewing, the Portal provides an environment that allows applications to work on the data in aggregate. For example, in Figure 2.11 we see an application that shows all discovered Wi-Fi access points, geo-coded and displayed on a map. Each pinpoint is color coded according to the type of access points and tool-tips are used to show the AP’s ESSID.
Opportunistic Wi-Fi

This chapter evaluates one of underlying assumptions of the CarTel system: that the density of Wi-Fi access points is high enough and that our network stack can be made reactive enough to take advantage of opportunistic Wi-Fi from a vehicular context. Just as important, though, is that we use the CarTel system itself to measure the performance of in-situ Wi-Fi networks in a realistic deployment.

First, let us consider the case for opportunistic Wi-Fi. The penetration of Wi-Fi in homes and offices around the world has been phenomenal. In 2008 alone Morgan Stanley estimates that over 319 million Wi-Fi chipsets were sold in the United States. In addition, Jupiter Research estimates the number of home-deployed Wi-Fi access points (APs) in the United States to be 26.2 million (37% of online households) and growing. Many home users deploy Wi-Fi in their homes and connect to the Internet over broadband cable modem, DSL, or even fiber. Because these Wi-Fi networks and upstream broadband access links are often idle, they could potentially be used to offer Internet access to other users (for the moment, imagine homes being able to function as “micro-ISPs”). We are interested in understanding what sort of performance one might expect from these unplanned community networks. In particular, could the blanket of radio connectivity provided by home Wi-Fi networks actually provide reasonable network coverage and performance, even for highly mobile users (e.g., those traveling in cars)?

It is important to note that there are several issues concerning policy, business decisions, and law [33] that must be resolved before this vision of an “open Wi-Fi” Internet access network can become real. We believe, however, that with the increasing deployment of open urban Wi-Fi networks in many cities around the world [79], community mesh networks (e.g., Roofnet [70], Champaign-Urbana Community Wireless [21] and commercial activity (e.g., Fon [28], AT&T Wi-Fi, T-Mobile) in this space, such networks may become real in the next few years.
The goal of this evaluation is to answer the following over-arching question: *What is the expected performance of open Wi-Fi networks for mobile users, particularly for users in automobiles, as they move in urban and suburban areas where APs are currently widely deployed?* Answers to these performance questions are largely unknown, and it is in a vehicular context that we believe connectivity and performance problems are most likely to arise.

Some of other questions we address include:

- What is the distribution of the duration of connectivity per AP? What is the distribution of the duration of disconnectivity (i.e., the time between associations to different APs)? How long does it take for a client to scan, associate, and obtain an IP address?
- What is the distribution of the coverage region of an AP?
- What is the distribution of packet loss and data transfer rates?
- What is the effect of a car’s speed on these metrics?

We answer these questions by running a measurement study over a set of *in situ* open APs deployed currently in and around the Boston metropolitan area (some of our data also comes from a small area in and around Seattle). Nine distinct cars, each outfitted with the CarTel hardware and software, collect data about Wi-Fi networks during the course of their owners’ normal driving. These computers attempt to associate with open APs deployed nearby. If the association succeeds, then the mobile node attempts to obtain an IP address, and then initiates an end-to-end ping (with several retransmissions until the first success or a timeout) to a well-known IP address. If the ping succeeds, then the node starts a set of periodic local AP pings to the first-hop IP router. In addition, for a subset of associations, the mobile node initiates a TCP transfer (upload) to the Internet site. This experimental apparatus, described in more detail in Section 3.1, allows us to gauge both radio channel conditions and end-to-end wide-area connectivity from moving cars.

In this evaluation, we focus on data uploads from cars rather than downloads to cars. There are two reasons for this. First, several emerging applications treat cars as data sources in mobile sensor networks, including the CarTel system itself, where a variety of sensors (GPS, cameras, on-board diagnostics, etc.) acquire and deliver data about cars and the surrounding environment. Second, it is very likely that the download performance will be at least as good as uploads, because most broadband links have more bandwidth in the download direction [25]. In any case, many of our results concern the bi-directional properties of the radio channel itself, and those findings should apply equally to both data transfer directions.
We analyzed over 290 "drive hours" of data collected over 232 different days over nearly a year. Figure 3.1 shows the geographic area covered by our study.

We divide our results into two broad categories: connectivity (Section 3.2) and data transfer performance (Section 3.3). First, we analyze the link-layer and end-to-end connectivity properties of our measurements, finding that the distribution of the duration of link-layer per AP (as measured by the time between the first and last successful local AP pings) is 13 seconds. We find that acquiring IP addresses using DHCP has a median delay of 1.83 seconds and that a simple caching scheme that uses the AP's MAC address and honors DHCP lease times can reduce this latency to less than 350 ms (but the cache hit rate is only around 20%, leaving room for considerable improvement). We find that our cars were able to successfully associate with APs and also transfer data with a uniform probability at all speeds between 0 and 60 km/hour, showing that urban vehicular speeds need not preclude Wi-Fi use. In addition, we find that the mean inter-arrival duration between open, associable APs is about 75 s. These APs appear in clusters and are not uniformly distributed.

We then turn to the factors affecting data transfer performance, investigating both link-layer packet loss rates and end-to-end TCP through-
put. Our end-to-end TCP upload experiments had a median throughput of about 30 KBytes/s, which is consistent with typical uplink speeds of home broadband links in the US. The median TCP connection is capable of uploading about 216 KBytes of data.

After describing our experimental method and discussing our findings in detail, we describe the implications of our findings on the design of data dissemination protocols for such intermittently connected networks (Section 3.4). We also discuss various issues concerning the viability of an open Wi-Fi Internet service.

3.1 EXPERIMENTAL METHOD AND DATA SUMMARY

Our measurement study uses 9 cars belonging to people who work at MIT's CSAIL. We instrumented these cars with an embedded computer running the CarTel system software, which includes a set of programs, Scanping, to measure Wi-Fi performance. The 9 cars maintained their normal driving patterns and schedule during the course of our study, and did not do anything out of the ordinary. All the data collected and analyzed are therefore subject to the constraints of real traffic and network conditions, a marked departure from the previous studies discussed in Chapter 5.

This study analyzes 290 total hours of driving over 232 distinct days between July 29, 2005 and July 20, 2006. Section 3.1.2 describes the high-level features of our deployment and the data.

3.1.1 The Experiments

For our experiments we used the hardware described in Chapter 2. During the experiments, the Wi-Fi card was used solely to perform measurements and not for any other communication. Scanping probes the GPS device once per second to obtain the current latitude, longitude, altitude, and speed; the GPS device also serves as the time reference when the computer boots up.

The embedded computer draws power from the car, boots up when the ignition turns on, and launches Scanping. It shuts down when the ignition is turned off. Scanping continuously loops through the following sequence of operations:

1. **Scan.** The Wi-Fi interface performs an active scan, which consists of sending probe requests and waiting for responses over all 11 802.11b channels. For each AP that is discovered, Scanping logs its ESSID (a human-readable string that identifies a network), BSSID (a 48-bit
bit-string that uniquely identifies an AP), radio frequency, and the received signal strength of the AP's response to the probe. In addition, Scanping logs the AP's advertised "Privacy Bit" value, which hints if an AP has enabled 802.11 WEP encryption or other security settings [40]. The scan operation repeats in a tight loop until the interface detects at least one AP. When Scanping finds an AP, it proceeds to the next step.

2. Association. Scanping issues a command to the Wi-Fi interface to associate with an AP that responded to its probe. If multiple APs respond, Scanping associates with the AP whose response had the highest received signal strength. We patched the Linux HostAP Wi-Fi driver (v.0.2.4) [56] to override the default roaming procedure to give Scanping full control over initiating and terminating associations with an AP, and to propagate feedback to Scanping about the success or failure of the association attempt. Scanping logs the feedback status along with the start time and the measured duration of this operation.

Scanping then jumps back to Step 1 (Scan) if the association fails. Otherwise, it launches tcpdump on the Wi-Fi interface to monitor and log all subsequent networking activity involving Scanping, and proceeds to the next step. Running tcpdump has proved invaluable in debugging and understanding observed performance.

3. Address configuration. Scanping uses dhcpcd to obtain an IP address. As explained in Section 3.2, caching IP addresses speeds up connection establishment substantially. Scanping therefore uses the AP's BSSID value to query a local cache for the AP's IP configuration information obtained from a previous drive. If an entry exists and has not expired according to the previous DHCP lease, then Scanping uses the cached parameters, and proceeds to the next stage in the program loop. Otherwise, it invokes dhcpcd to obtain a DHCP lease from the DHCP server running on the AP's network. If dhcpcd fails to acquire an address, the client times out after \( T = 5 \) seconds and returns to Step 1 (Scan).

4. Single end-to-end ping. At this point, Scanping has established connectivity with the wireless network. Certain types of APs grant association rights and DHCP addresses to any client, but refuse to forward traffic for them without proper authentication. Examples of such APs include commercial hot-spot networks and wireless LANs configured with MAC address filters or firewalls. Our device can connect to such an AP and potentially waste valuable scan opportunities; moreover, by including such APs in our analysis, we might
over-estimate the connectivity available in today’s unplanned Wi-Fi deployments.

Our connectivity test simply pings our central server’s IP address (to avoid a time-consuming DNS lookup). To combat potential packet losses, which can be especially high while entering an AP’s coverage area [63], Scanping attempts this end-to-end ping every 200 ms, until the first successful one, or until 2 seconds elapse. Scanping only considers APs for which this end-to-end test succeeds in estimating end-to-end connectivity durations.

5. Connectivity and uploads.

If the previous step succeeded, Scanping starts two separate processes that run in parallel. The first one pings the first-hop router, while the second one attempts TCP uploads.

(a) AP pings. Our primary interest is in understanding the nature of vehicular connectivity and estimating what the capacity of unplanned Wi-Fi networks is likely to be.

To measure the properties of the wireless channel, Scanping sends ping packets to the first-hop router every 100 ms. We picked this periodicity because it is similar to that of 802.11 AP beacons, but it is fine-grained enough to understand the characteristics of the channel as observed by the car.

Scanping logs the time at which each AP ping was done and whether the ping succeeded or failed.

(b) TCP uploads. In a subset of cases, Scanping opens a TCP connection to the central server and delivers data. During this upload, the car can move out of range an AP and cause the wireless interface to disassociate. At this point, Scanping needs to terminate the TCP connection as soon as possible, so that it can proceed to discover new APs in the next iteration of the loop. The default TCP connection termination time is many minutes, which is too long. Unfortunately, the low-level mechanism for detecting AP disassociation events from our Wi-Fi card and driver is unreliable. Hence, Scanping uses the ping connectivity probes described above, and terminates the entire association (and hence the upload) if no responses are heard for 3 seconds.

Scanping logs the time at which each TCP upload was done, the total number of bytes uploaded, and the duration of the connection. Scanping currently attempts no downloads.

All the results reported in this paper use an 802.11 transmission bit-rate of 1 Mbit/s, the lowest bit rate. We picked this rate because it tends to have a
bigger region of coverage than higher rates, and because we expect most home broadband access links to have upload bandwidths that are smaller than this number. However, it's interesting to note that in [25] they found that 11 Mbit/s to be optimal when not limited by the upload caps that many ISPs impose. We configured the Wi-Fi interface to use the maximum transmit power (200 mW) and used a link-layer retransmission limit of 8.

3.1.2 Data Summary

Figure 3.2 shows the typical timeline of various events of interest as a vehicle moves into the region of coverage of an access point. The salient features of our data set are shown in Table 3.1. This data spanned a large geographic area, as shown in Figure 3.1 (each box there is a cell 1.4 km × 1.6 km in size; boxes here are relatively large so they can be seen). Figure 3.3 shows the same information as a CDF of successful pings grouped by smaller cells of size 260 m × 270 m (this size is roughly the "typical" AP range). The CDF shows that our data set samples about 700 total cells (corresponding to an area of 49 square kilometers or a "distinct linear distance"—assuming we traveled along the diagonal of each cell once—of more than 260 km). 50% of the successful pings come from 42 cells (about 15 km of distinct linear distance).

Scanping stores all collected data in a PostgreSQL database on each car; another process periodically sends the data to a centralized back-end database and analysis system when no experiments are being done.
No. of cars & 9 \\
Drive hours analyzed & 290 \\
Start date of analyzed data & July 29, 2005 \\
End date of analyzed data & July 20, 2006 \\
No. distinct days analyzed & 232 \\
No. of traces (drives) & 1605 \\
Traces with non-empty scans & 1176 \\
No. of non-empty scans & 154981 \\
No. of APs discovered & 32111 \\
No. of join attempts & 75334 \\
Successful join attempts & 19576 \\
Joins that acquired IP address & 6410 \\
Joins responding to e2e ping & 4997 \\
Distinct APs joined & 5112 \\
Distinct APs that gave IP & 2259 \\
Distinct APs that responded to AP ping & 793 \\
Distinct APs that responded to e2e ping & 1023 \\

**Table 3.1—Data summary.**

(specifically, when a car entered an MIT parking garage, the CarTel software in the car used the Wi-Fi network there to deliver the data collected to the central site; that Wi-Fi network was not included in our data set). The back-end also uses PostgreSQL and includes a map-based visualization framework that produces graphs and location-based pictures over Google maps (see Chapter 2).

The database stores information about all non-empty scans and association attempts in corresponding tables. Information about observed access points is extracted from the scan records and stored in a separate table. Location information produced by the GPS is also stored in a separate table. Wi-Fi and GPS data are cross-referenced through time stamps.

### 3.1.3 Lessons Learned

We learned several lessons from our measurement and analysis work. First, we found that our experimental setup stretches the limits of existing software and hardware, because they are not designed to operate in a challenging environment where power disruption is frequent and where the low-level Wi-Fi operations of scanning and association are controlled by a user-level process and occur at such high rates. To cope with failures while our experiments were running “in the field”, we instrumented Scanping to detect and recover from database corruptions (in particular, we found that frequent power disruptions and flash failures can corrupt
FIGURE 3.3—CDF of the geographic coverage of associations with successful end-to-end pings in our data set. We divide the world into cells of size 260 m × 270 m, sort the cells in descending order of the number of distinct successful associations, and plot the resulting CDF. The top 42 most popular geographic cells accounted for 50% of all successful associations; the remaining 50% came from over 650 other cells.

the filesystem). Scanping also proactively reloads the wireless interface driver whenever there is over 60 seconds of inactivity, because the Wi-Fi subsystem does get "wedged" occasionally at high rates of scanning and association.

Second, the CarTel embedded computer (a Soekris net4801) has a high process overhead; it often takes multiple seconds to start new processes (e.g., new Perl programs) when the rest of the system (including PostgreSQL) is running. As a result, the two programs started in Step 5 above do not start as soon as they are initiated. We did not realize this shortcoming until after all the data was collected.

On the positive side, our use of CarTel and the reliance on a database (as opposed to flat files with scripts for analysis) was a good decision. Nearly all of our graphs are generated through short, declarative SQL statements, and PostgreSQL (and the associated PostGIS packages) make it easy to perform relatively sophisticated analyses over geo-coded data.

Data analysis in this environment proved to be extremely time-consuming, both in terms of running sanity checks to ensure consistent and explainable results, and to remove a variety of outliers. For example, when the GPS on our devices acquires a satellite fix, it automatically sets the time on the in-car device. If we are in the middle of a connection when this happens, it can cause connections to appear to be extremely long (or
have a negative duration.) In the middle of a drive, this behavior can also cause the disconnection duration sample to be wrong. Many outliers are also generated by AP associations and end-to-end connections that partially complete or timeout, leaving some state in the database that must be filtered out. In addition, data analysis was also complicated by our experiments evolving with time and being refined iteratively.

3.2 Connectivity Results

This section investigates the time it takes to perform the different steps of the experiment specified in Section 3.1.1. For the first three steps—Scan, Association, and IP address acquisition—we show the distribution of latencies. We then show the distribution of the time between a successful association and a successful end-to-end ping reception, and the distribution of latency between the first and last successful AP ping. This latency captures the duration over which the car maintains Wi-Fi connectivity (the time between the end of Step 2 through the end of Step 5).

Figure 3.4 shows four CDFs: the latency after a successful association to (1) acquire an IP address, (2) receive a response to the end-to-end ping, (3) receive the first successful AP ping response, and (4) receive the last successful AP ping response. The data used to plot each line in this graph consists only of those associations that successfully reached the corresponding phase of the association.

Figure 3.4 shows that the median time between AP association and IP address acquisition is about three seconds (this includes associations with and without the DHCP caching optimization discussed in Section 3.2.1 below.) The median time to first AP ping from the time of association is about 8 seconds and the minimum is about 5 seconds. This several-second delay is surprisingly high. Upon investigation, we found that the primary reason is the high overhead on our embedded platform for starting a separate Perl process in Step 5 to launch the AP pings. The platform has only a small amount of memory (128 MB), and several other processes (including PostgreSQL) use up fair amounts of it. A secondary factor are the end-to-end pings, but that CDF is almost the same as the IP address acquisition CDF (end-to-end pings are launched from the same process), suggesting that they do not contribute significantly to this delay.

Figure 3.4 also shows CDF of the time at which Scanping receives the last successful AP ping. The median is about 24 seconds. Thus, although an artifact of our setup is the inability to usefully probe or transfer data during the first few seconds following an association, we are able to carefully characterize and analyze performance over the remainder of the association. The distribution of overall association times has a heavy tail and ranges from just a few seconds to several minutes (presumably represent-
FIGURE 3.4—CDF showing the distribution of times for various phases of an association following a successful association with an AP. The small offset between IP acquisition and a successful end-to-end ping shows that, if the end-to-end ping succeeds, then the response usually comes quickly. The delay between the end-to-end ping response and first AP ping response is surprisingly high, and is an artifact of the high process initiation overhead of our experimental setup.

In the rest of this section, we analyze the Wi-Fi association and IP address acquisition times, overall connectivity duration (including the effects of car movement), periods without connectivity, and the distribution of an AP’s coverage region in more detail.

3.2.1 Wi-Fi Association and IP Address Acquisition

Scan and association latency. Figure 3.5 shows the CDF of scan and association times. The minimum, mean, and maximum scan times are 120 ms, 750 ms, and 7030 ms, respectively. The minimum, mean, and max-
Figure 3.5—Distribution of scan and association times observed by our moving cars.

Minimum association times are 50 ms, 560 ms, and 8970 ms, respectively. Both distributions don’t have heavy tails; in both cases, the 95th percentile is not significantly higher than the median. The outliers in the scan times are probably anomalies, while the outliers in the association data are probably due to the retries attempted after failures to associate.

The scan duration depends on the card but our measured numbers are consistent with previously published microbenchmarks. Our association times seem to be slightly higher than those reported in some previous studies [58], possibly because our network conditions are less controlled and harsher.

DHCP latency. We found that the time to acquire an IP address is often significantly higher than the scanning and association latencies. The “DHCP acquisition” curve in Figure 3.6 shows the CDF of the time to acquire an IP address after a successful association when using DHCP (without the caching optimization discussed below).

The main reason for the relatively high median and tail DHCP latencies is that when the car successfully associates with an AP, it may still be at the “edge” of the area covered by the AP. In addition, DHCP requests are sent to the broadcast address and can’t benefit from link-layer acknowledgments and retries. As a result, some link-layer frames corresponding to the DHCP request or its response may be lost. (The spikes in the CDF at 0.5, 1, and 2 seconds are artifacts of experimental runs when we set the DHCP timeout to those values.)
Accelerating initialization. These results show that when a car encounters an AP for the first time, it takes a non-negligible amount of time before it can actually send or receive data using that AP. We now investigate whether caching can reduce this latency.

To reduce scan and association latency, we considered a scheme that would cache mappings between GPS coordinates and a set of APs (with their channel parameters) based on past drives, and use that to avoid re-scanning when the car is near the same location again. Our experimental setup, however, obtains GPS coordinates only once per second, and so the location will be slightly out of date. It is possible to develop techniques that predict the likely current location using the past few location samples, but we didn’t experiment with that idea because the scan and association times are currently a relatively small fraction of the overall idle time. However, such an optimization may be even more beneficial for 802.11a than for 802.11b, because of the larger number of channels in 802.11a.

Caching the mapping between an AP’s MAC address (“BSSID”) and an IP address granted by the corresponding DHCP server from the previous drive should be a significant win, because a cache hit can eliminate the DHCP delay altogether. Caching the IP address that worked the last time, while honoring the DHCP lease rules, avoids the need to negotiate a DHCP address. The “IP cached” line of Figure 3.6 shows that a successful cache hit greatly reduces IP address acquisition latency (the median reduces from 1.83 seconds to 346 milliseconds). We also found that for our
drives the cache hit rate was between 17% and 22%.

This caching scheme can be improved in several ways. For instance, we might tag the cache with flags that show to what extent the address worked (successful end-to-end ping, successful AP ping, etc.). Such tagging would be useful because cached addresses in the current scheme turn out to have a higher probability of not leading to successful AP or end-to-end pings than when DHCP provides an address. In addition, we might also try to use a cached address even after the expiration of the lease, after running arping—a network probing tool using ARPs—to check if the IP is currently being used.

3.2.2 Connectivity Duration

We estimate the duration of connectivity by calculating the time duration between the first and last successful AP pings for each association (Figure 3.7). As explained earlier, owing to the process overheads of our platform, the time to successfully send the first AP ping is several seconds long, and that is reflected in the durations being a few seconds smaller than suggested by Figure 3.4. We have verified that the duration between a successful end-to-end ping and the last successful AP ping is typically about five seconds longer than between the first and last successful AP pings.

One might wonder whether the connectivity results obtained so far are biased by the amount of time our cars spend traveling slowly. For
example, does the bulk of our successful associations occur when the cars travel slowly or when they are stationary (e.g., at a busy intersection or a traffic light)?

To answer this question, we analyzed the probability of a successful association as a function of car speed. Figure 3.8 shows the results. Somewhat surprisingly, the number of associations is fairly uniform across all speeds between 0 and 60 km/hour. Note that only a few of the associations were made at speeds above 70 km/hour, and in particular the number of successful associations at highway speeds is small.

There are three reasons for this lack of success at higher speeds. First, most of the driving was done on surface roads in these measurements. Second, open wireless APs are common to residential areas that are often distanced from major highways. Third, we have not optimized our measurement system to achieve sub-second associations, which would improve connectivity at highway speeds (that said, we still noticed a few successful associations at speeds above 80 km/h). Hence, we believe that the possibility of using unplanned open Wi-Fi APs from highways might still hold promise, especially if the penetration of Wi-Fi APs near highways rises. For more challenging environments we recommend using the QuickWiFi network stack, described in Section 3.5 and in [25].

Even though association establishment success is not directly affected by speed, associations made at lower speeds tend to have longer durations (see Figure 3.9). This result is as expected, and implies that more data
FIGURE 3.9—A plot of association durations vs. speed. Error bars indicate 95% conf. intervals. The maximum association duration is shorter at higher speeds, as expected.

would be transferred per AP at lower speeds.

3.2.3 Periods without Connectivity

This section investigates the inter-arrival time observed between connectivity events, for different types of connectivity. We start with end-to-end connectivity as observed in situ in our data set. The mean time between successful end-to-end pings in our traces is only 260 seconds. The corresponding CDF is shown on the “end-to-end” line of Figure 3.10. The median is considerably lower than the mean, and the CDF shows that end-to-end connectivity events occur in clusters. In fact, the standard deviation of this duration is large, about 642 seconds. Thus, some intervals without connectivity are extremely long (witness the heavy tail in the CDF), but when successful connectivity occurs, it does so in quick succession. A cursory analysis suggests that the long durations correspond to times when drivers are in areas of low human population or traveling at high speeds.

We are also interested in knowing what would happen to the time duration between connectivity events as more APs participate. To answer this question, we successively refine the set of APs to include ones for which our mobile nodes successfully obtained a local IP address, successfully associated, and simply encountered (whether open or not). The CDFs for these inter-arrival durations, as well as the mean times between events and their standard deviations are shown in the remaining lines of Figure 3.10. If all APs were to be used, the mean time between connectiv-
FIGURE 3.10—CDF of the time between connectivity events for four types of events: successful end-to-end pings, successful local IP connectivity, successful associations, and association attempts. Whereas today only the first type can actually be used, the other three events show improvements in connectivity that can arise if more access networks participate. Ultimately, observe that the mean time between association attempts is comparable to the mean time for which connectivity lasts (24 seconds from Figure 3.4), although connectivity is not uniformly distributed in space.

Of course, because these events occur in bursts, one cannot conclude that the system would provide continuous connectivity. One can conclude, however, that even when restricted to the number of APs that currently permit associations or even those that provide local IP connectivity, cars traveling in areas in and around cities are likely to encounter several usable APs on any drive that lasts more than a few minutes. This finding implies that it may be possible to get timely updates from (and to) vehicles in urban areas using an unplanned open Wi-Fi infrastructure.

Section 3.4 discusses some incentives that might enable more participation in a potential open Wi-Fi network. Although it is highly unlikely that all, or even a majority, of these APs would participate, even a 20%
participation rate (e.g., those APs that allow successful associations today) would reduce the mean time between connectivity events by a factor of over 3x.

3.2.4 AP Coverage

Figure 3.11 shows the number of APs discovered per successful AP scan (i.e., any scan that found one or more APs). More than two APs are discovered over 65% of the time, attesting to the high density of APs in many areas. APs are generally found in clusters. Also note that our measurements provide a lower bound on the number of APs, because Scanping does not scan when it is either trying to associate or has associated successfully with an AP.

Figure 3.12 shows a CDF of the fraction of all associations that were made to each of the 5112 APs that we associated with, where the APs are sorted in descending order of the number of associations. The top 100 APs account for about 20% of all associations (these are the darker geographic cells in Figure 3.1).

We are also interested in the coverage of an AP. To compute this quantity, we take the set of GPS coordinates at which the car received an AP ping response. We then compute the smallest bounding box around those coordinates and log the length of the diagonal of this bounding box. This definition does not account for possible coverage holes within the region formed by these coordinates (for that reason, we don’t report an area but...
rather the diameter).

Figure 3.13 shows the complementary CDF of the AP coverages observed in our data. The median AP has a coverage of 96 meters and the top 10% of associations have a coverage of more than 300 meters (recall that our mobile nodes transmit at a relatively high power level of 200 mW, though we have no control over the AP settings). We note that the coverage shown in this graph will often be a lower bound on the true coverage of the APs, because our experiments do not attempt to find the maximal coverage region. Moreover, it is very likely that the coverage is not circular or symmetric about the AP, but is likely to be quite erratic and non-convex. Hence, these numbers should be treated with these caveats in mind.

Our results are generally consistent (except for the shape of the tail) with Akella et al.'s recent Pittsburgh study [4], but are considerably lower than the maximum 802.11 AP range of 800 meters reported in previous work [63] for a single access point placed directly along a road. This difference is not surprising given that our access points are likely in buildings and could be far from the road; moreover, we do not include the distance from an AP to a car in these estimates because we don't attempt to precisely localize APs. These coverages would also likely reduce at higher radio bit-rates.

These relatively large coverages suggest that a relatively small number of access points can cover a large area, if we are willing to tolerate some holes. For example, the city of Boston is about 130 square km, so in theory,
FIGURE 3.13—CCDF of connection coverage. The coverage of an AP is the length of the diagonal of the smallest bounding box that encloses all GPS points from which some communication (lower bound) was possible with the AP.

according to our measurements, the entire city could be covered with just 2,000 (properly placed) APs. Of course, the real number is likely to be bigger because of the vagaries of real-world radio propagation.

3.3 PACKET LOSSES AND DATA TRANSFERS

This section describes observed wireless packet loss rates and the end-to-end performance of TCP uploads.

3.3.1 Wi-Fi Packet Loss Rates

We now examine the bi-directional loss rates of AP ping packets to assess the channel quality as cars move into and out of range of an AP. Figure 3.14 shows a complementary CDF of the probability of successful AP ping over all connections, for connections in which at least one end-to-end ping succeeded. The median connection has a delivery rate of 78%; assuming symmetric losses, this translates into an approximate uni-directional delivery rate of about 90%.

To better understand the distribution of losses throughout the connection, we studied the loss rate of our AP pings in different phases of long connections. We found that end-to-end loss rates are only about 10% at the time AP pings begin (recall that this is several seconds into the con-
connection, as shown in Figure 3.4), and that loss rates remain at this level until the last few seconds of the connection, when they become quite high, often exceeding 80% as the car moves out of range of the AP. We speculate that if we did not have our initial address acquisition delay or end-to-end ping, we would have seen high loss rates at the beginning of connections as well, as has been observed in prior work [63], as well as confirmed subsequently in [25].

Figure 3.15 plots the mean AP ping success rate observed at different speeds. Our data set has little data at higher speeds, which is why some speeds do not have points. There appears to be no correlation between speed and packet delivery rates. This result is not too surprising; because the car-to-AP latency is at most a few milliseconds, the distance traveled by a car in this time is small.

3.3.2 TCP Throughput

This section analyzes the performance of TCP uploads from moving cars. These uploads were done only on a small subset of all AP associations. We begin by looking at the time from initial association until we receive the first TCP acknowledgment (ACK). This time distribution, shown in Figure 3.16 (broken down by IP address acquisition method), captures the time taken for short single-packet reliable transfers. Observe that the minimum time until any data is successfully transmitted is about 5 seconds, which as mentioned before is mainly an artifact of our resource-constrained implementation platform. This time also includes other over-

![Figure 3.14—CCDF of the fraction of AP pings that succeeded per connection. The median delivery rate is 78%.

0.8
0.6
0.4
0.2
0

Packet delivery rate

Fraction of associations

0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

FIGURE 3.14—CCDF of the fraction of AP pings that succeeded per connection. The median delivery rate is 78%.
heads, including first-hop router address resolution using ARP (about 1.5 seconds in our experiments) and DHCP (if used).

Once a connection has been established, Scanping begins transmitting data from the car to the server. Figure 3.17 shows the complementary CDF of the per-connection end-to-end throughput. The median connection throughput is about 30 KBytes/s. 80% of all the connections (between the 10th and 90th percentiles) observe TCP performance between 10 KBytes/s and 50 KBytes/s, a fairly narrow range. These results are not surprising in light of the fact that in urban Wi-Fi networks, most APs reside on home networks that receive their Internet connectivity through cable modems or DSL links. Typically, these types of connections offer upstream bandwidths that are roughly similar to the values we observe in this data. Because we use a 1 Mbit/s radio rate, we don’t see any throughput results that are higher than 100 KBytes/s.

The relatively high packet loss rates shown in the previous section may also impair TCP throughput, although the connection attempts that actually succeeded in transferring some data over TCP probably had a lower average loss rate than those that only succeeded in transmitting AP pings. Moreover, as we discussed, the average reported in the previous section is not uniform over the duration of the connection — the middle of a connection, when most data is sent, has a lower packet loss rate than the end.
Figure 3.16—CDF of duration between association and first TCP data ACK reception at the client, broken out by IP acquisition method. Our results show that for APs whose IP addresses we were able to cache, the median time to the first payload data ACK is reduced by about 4 seconds (to 9.13 seconds from 12.9 seconds). Only 53 connections used caching here, so we consider these numbers to be preliminary.

Figure 3.18 shows the CDF the total number of bytes transferred via TCP per connection. The median connection transfers about 216 KBytes of data, which, at 30 KBytes/sec, suggests a connection duration of about 8 seconds. This number is consistent with our analysis of server logs, which show that the typical connection has packets arriving at the server for about this much time. Some connections (about 10%) transfer only one packet, and in some of those cases the TCP ACK was not received by the mobile unit. Notice from the CDF that the number of bytes delivered has a heavy tail, presumably corresponding to times when a car is at a stop light or is in heavy traffic.

3.4 DISCUSSION

The measurements in previous sections show that there is a surprising amount of connectivity in metropolitan areas even with the small fraction of access points that are currently open. Our data suggests that the idea of individual users banding this connectivity together to form a homegrown, community-based, nearly-ubiquitous network holds much promise. While the idea is appealing, there are a number of technological, social, and legal questions that arise in this context, namely:
FIGURE 3.17—Per-connection end-to-end throughput CCDF. The median throughput is about 30 KBytes/sec, which is consistent with the upstream bandwidth of most cable modem backed APs and a lossy wireless link.

— How can we incentivize users to open their networks and opt into such a service? How do we convince ISPs to allow this?

— What will connectivity in such networks look like as more APs join?

— How will the network capacity scale as more mobile clients (e.g., cars) participate?

— What transport protocol features will such a network require?

— How do we prevent roaming mobile users from monopolizing the resources of users who donate their Internet connections?

We address some of these questions below.

3.4.1 Toward Open Wi-Fi Networks

For these kinds of open Wi-Fi networks to become a reality, it is important to provide users and service providers with incentives for opening their APs. There are two potential incentives: promising participants free service, and giving participants money. For example, the recently proposed Fon network provides users who open their APs free access to all other users' APs, but charges users who have not opened their APs to use the system. Fon then keeps some of the money and proposes to redistribute the rest to the ISPs involved, and perhaps also to the owners of the APs (especially those that do not wish to become clients of other owners' APs).
Another example is Meraki Network’s “Free the Net” [57] campaign to blanket San Francisco with Wi-Fi coverage by placing their access points in peoples’ homes.

This approach is not the only possible economic model, and we are certain that other models will emerge in the future (only time will tell what might succeed in the marketplace). We also note that Fon does not target highly mobile users moving in cars, while that is our main focus in this paper.

A tiered security model for Wi-Fi APs would also be a good idea; home users should be able to encrypt their traffic, while leaving APs usable by subscribers. A simple approach would be to use three tiers, one for the owners, one for subscribers, and possibly a third one for everyone else.

To reduce the impact on an AP owner’s data transfers in an open network of this kind, owners should be able to set different rate limits for these tiers. Furthermore, as we discuss below, there are a number of optimizations that can be made to further reduce the potential impact of greedy mobile users.

ISPs will presumably be willing to participate in such an arrangement provided they are given sufficient financial incentives. An ISP’s primary concern is likely that users will use the open network rather than paying for an Internet connection of their own; a proper fee schedule or connection time limits on the open network can obviate these concerns.

Fortunately, the legal issues in this case favor users who open the networks, at least in the US. Here, so-called “safe harbor” laws protect par-
participants from being liable for the actions of users of their access points, just as they protect ISPs (as long as the participants agree to comply with law-enforcement officials in stopping or tracking down malicious users on their APs). In fact, networks that allow home users and ISPs to share access with roaming users has garnered interest in the research community, with several projects ([38, 71]) already exploring this model.

As an alternative to community-driven open networks, it is possible that municipalities will simply install a large number of open access points, as they are starting to do in some cities [79]. Though in some cases these networks will be free, it seems very likely that many cities will charge some fee to partially subsidize the cost of deployment and maintenance. Furthermore, such networks are likely to span only the dense cores of cities, rather than smaller communities or suburban areas. Hence, we expect that the future metropolis will consist of a mix of commercial and municipal Wi-Fi networks, as well as community-driven “open” models. Many of the results reported in this paper apply to municipal and community Wi-Fi networks.

3.4.2 Connectivity and Network Transport in Open Wi-Fi Networks

In either the open community or municipality-driven model, it is interesting to explore what connectivity will look like from the perspective of a mobile user. One possibility is that connectivity will be continuous. Based on our analysis in Section 3.2.3, however, we believe that it is unlikely that at vehicular speeds it will be possible to continuously maintain connectivity with at least one AP.

If connectivity for mobile users does turn out to be continuous, the transport layer solutions will most likely consist of Mobile IP with some set of techniques for fast connection handoff [13, 72]. If connectivity is discontinuous, however, there are a number of open issues. First, the API that applications use to connect to each other will change, since disconnectivity suggests that connection-oriented protocols are no longer appropriate. Instead, a natural solution is to expose the non-continuous nature of the underlying connectivity by means of application level framing [20], where applications express their transmissions in application defined data units (ADUs.) Previously in Chapter 2 we discussed CafNet, one such solution to this problem.

3.4.3 Fast, Friendly Connection Establishment

Regardless of whether such networks are continuous or intermittent, an essential feature of future network stacks tuned for these mobile environments is that they provide fast connection establishment that provides
fair sharing of available bandwidth. We describe three possible optimizations in this area, related to: (1) timing of TCP connection initiation, (2) time-based fairness, and (3) aggregating bandwidth across multiple access points.

**Connection initiation timing.** There has been substantial work on using TCP over wireless [6]. Most of this work focused on wireless clients with little or no mobility. As previous work suggests [63], wireless LAN connections at vehicular speeds go through a set of phases as the client moves relative to the AP. High losses at the beginning of a TCP connection could dramatically reduce the overall throughput of the connection. In our experiments, TCP connections start only a few seconds after a successful association. Optimizations to the transport protocol (e.g., [64]) might be able to better address this issue.

**Fairness.** One way to improve fairness and avoid over-utilizing the connections of users who donate their access is to use rate limiting at APs or on clients, or to use cooperative transport protocols like TCP Nice [83] (appropriately tuned for wireless access).

Another fairness related issue is that most Wi-Fi interfaces are designed to adapt their bit-rates according to the link condition, selecting lower bit-rates to overcome frame errors and signal-to-noise ratio in a lossy link. Our experiments show that a significant fraction of connections suffer from high loss rates. Therefore, the wireless interfaces on the urban network clients are likely to operate in low bit-rates most of the time.

Unfortunately, although lower bit-rates help reduce link losses, they have the side effect of reducing the capacity of Wi-Fi networks [35, 78]. In certain situations, a home client’s throughput might reduce to an eighth of its original value as a result of an urban client’s connection. Such anomalies can occur even when there are no packet losses in any of the links. Thus, the anomaly is strictly a link-layer problem that cannot be addressed by higher layers. To solve this problem, Tan and Guttag [78] suggest that time-based fairness be used for scheduling access to the channel. We believe that using a similar mechanism for Wi-Fi clients will limit the impact of bit-rate differences on throughput in open Wi-Fi networks.

**Aggregating Bandwidth over Multiple Access Points.** The authors in [67, 69] describe different systems that aggregate bandwidth across multiple cellular towers of several different cellular networks. Their measurements show that bandwidth aggregation can provide a several-fold improvement in TCP throughput over connections that use only a single cellular tower.

Our measurement results suggest that the same techniques can be used to improve throughput for unplanned Wi-Fi networks. In 65% of the AP
scans, we find at least two APs. When an urban network client finds multiple APs after a scan, it can use a system like MultiNet [17] or multiple wireless interfaces to simultaneously associate with different APs and use them.

One premise of bandwidth aggregation is that the concurrent connections should operate in orthogonal channels so that simultaneous transmissions do not interfere with each other. Indeed, a Wi-Fi measurement study discovered that over 60% of open AP deployments are configured to operate in 3 channels so that it is very likely that any two nearby APs may operate in an overlapping channel [3]. In this case, the concurrent uplink transmissions would be serialized at the link-layer, as the multiple interfaces contend to use the same or overlapping radio channel.

Despite serialized transmissions, connecting to multiple APs can still improve throughput as long as the connections sharing the overlapping channels do not saturate the wireless link. In Section 3.3.2, we show that the median throughput of 30 Kbytes/sec remains well below 802.11b’s saturation point for a TCP connection at the lowest bit-rate of 1 Mbit/s [78]. Thus, the wireless link may have spare capacity to support concurrent connections using overlapping channels.

3.5 IMPLEMENTATION IN CABERNET

Cabernet [25] incorporates many of the ideas presented in this chapter and extends them to build a data delivery system for moving vehicles. Although this work was done by a different set of researchers, we present Cabernet as an illustration of what a system incorporating the lessons learned from the preceding sections might look like. Cabernet makes contributions in three main areas: (1) fast connection establishment via QuickWiFi, (2) improved end-to-end connectivity over lossy links using the Cabernet Transport Protocol (CTP), and (3) a static bit-rate selection based on experimental results. What follows is a brief discussion of each of these contributions and how they compare to the results from this chapter.

* * *

QuickWiFi. As discussed in Section 3.2, establishing a connection between a mobile host and an AP can take 10s of seconds. The majority of the delay is not intrinsic to the connection process, but due to suboptimal scanning, inappropriate timeouts, and a lack of parallelism. To address these limitations, the Cabernet authors built QuickWiFi, a lightweight tool that integrates scanning, authentication, association, DHCP, ARP, and link monitoring into a single process to speed up the connection process. In addition, the timeouts for retrying each stage of the connection process
are reduced from seconds to hundreds of milliseconds, and the authentication/association requests are sent in immediate succession. Finally, QuickWiFi employs an optimized scanning strategy that relies on the observation that most APs are configured to use channels 1, 6, and 11, and hence, the remaining channels should be scanned much less frequently. These improvements result in the median connection taking 287 ms to be established—a significant improvement over the 8 seconds reported in Figure 3.4. One consequence of this result is that a system using QuickWiFi can take advantage of many more short duration connections than could our original system. In Cabernet, the median encounter duration is 4 seconds, which is significantly less than the 24 seconds shown in Figure 3.4.

**Cabernet Transport Protocol.** Most TCP implementations work best when loss rates are less than 5%. When connecting to APs from moving vehicles, loss rates typically exceed 20%. Moreover, as vehicles move between areas of coverage, IP addresses and network paths change. The Cabernet Transport Protocol (CTP) is designed to differentiate between congestion losses and wireless losses, as well as provide session migration between networks. CTP detects congestion—as opposed to wireless losses, which are recovered from, but do not affect transmission rates—not by looking at end-to-end ACKs, but instead uses probe packets sent directly to the AP. These probe packets are large enough (1500 bytes) to be affected by congestion and illicit responses (either TCP RST or ICMP responses, depending on the method) from unmodified APs. A lack of response is interpreted as a sign of congestion and causes CTP to reduce its rate in a multiplicative fashion, though different from TCP. Likewise, when CTP receives an ACK, it increases its rate using an additive strategy. CTP also provides network migration using unique, session-level identifiers that remain the same even as IP addresses change. This strategy requires that either CTP runs on the both end points, or there be an Internet connected proxy that maintains the necessary TCP/IP state to communicate with the Internet in a transparent fashion. For the purposes of evaluation, both CTP and TCP were deployed on vehicles in Boston. The mean throughput of CTP was 760 Kbit/s, whereas TCP only achieved 408 Kbit/s. Note, these values are much higher than achieved in CarTel—we used a standard TCP stack and are primarily concerned with uploads, which are typically capped at 240 Kbit/s by ISPs.

**Rate Selection.** IEEE 802.11b supports 4 bit rates: 1, 2, 5.5, and 11 Mbit/s. For the Cabernet system, the authors measured the success rate of sending packet bursts at each of the bit rates over a large number of encounters and APs. They found that, although the probability of success drops as bit rates increase, the fall-off is not significant enough to make up for
the increased transmission time of the lower rates. For example, although transmitting at 1 Mbit/s will increase success by about 10 percent, the packet transmission time increases by a factor of 10, negating any possible performance gains. This trend was also true when considering success probabilities conditioned on previous failures. Consequently, they found that fixing the bit rate at 11 Mbit/s proved to be the best strategy for maximizing throughput. In our study, as mentioned at the end Section 3.1.1, we fixed the bit rate at 1 Mbit/s under unsupported assumption that coverage area would be dramatically improved. However, it is still true that most broadband upload rates are capped at less than 1 Mbit/s, which makes bit rate selection not as important for upload heavy deployments.

* * *

Taken together, these strategies allow Cabernet to deliver a mean throughput of 38 Mbyte/hour (86 Kbit/s) during a deployment in the Boston area. These techniques could easily be integrated into CarTel.
In this chapter we analyze the second novel aspect of CarTel: its use of opportunistic vehicular mobility. The principle advantage of piggy-backing on the existing mobility patterns of cars and trucks is cost savings—we don’t have to hire drivers or build expensive autonomous vehicles. However, this approach is not without trade-offs. In particular, it is unclear what sort of spatial coverage we will get from seemingly random, vehicular motion. The goal of this chapter is to answer this fundamental question: *How many participating vehicles would we need to meet the sampling requirements of a mobile sensing application?*

We answer this question against the backdrop of a traffic monitoring service. Throughout our work, the feedback we’ve received from users and drivers has shown us the importance of traffic monitoring and how much it remains an unsolved problem. One of the crucial challenges in building such a service is getting an adequate number of delay samples from each of the road segments being monitored. Based on our experience and discussions with people in the field, we’ve set the needed number of samples per hour between 10 and 20. Throughout our analysis we’ll present graphs showing these two curves, representing upper and lower bounds for a reasonable range of sampling requirements. As you might expect, this value is highly dependent on the specifics of application. We are primarily concerned with performing a first-order study of opportunistic mobility.

In addition to looking at a range of sample rates, we analyze three monitoring approaches designed to mirror how our traffic monitoring service—or, any other sensing application using CarTel—might be deployed:

**Participatory monitoring.** In the first approach, we consider a *participatory* focused deployment in which our goal is to monitor those areas visited by vehicles carrying our device. This mirrors a deployment in which the monitored area grows with the number of units deployed. For exam-
ple, if you and your neighbor both commute into Boston for work, we’d only be interested in monitoring the roads the two of you touch. Note, there’s a good chance there is a substantial overlap between your drives. If your neighbor left 15 minutes before you, data from his drive might give you good idea what traffic is like for the bulk of your drive, even if you have different destinations. With this type of deployment, our primary concern is delivering useful data to participating members.

Infrastructure monitoring. In the second approach, we broaden our focus and consider an infrastructure focused deployment in which our goal is to monitor a specific region. This is how a municipality might deploy CarTel to monitor congestion along major arteries or to monitor road surface conditions along residential streets. This type of monitoring could require more cars than the participatory case because we are targeting a potentially much larger set of roads to monitor, thereby helping all drivers and not just those using our system.

Arterial monitoring. In our final approach, we consider a modified version of the previous formulation: we simply select those roads whose speed limit exceeds 25 mph, our proxy for classifying a road as a major artery. In our experience, these high speed roads can easily be congested and result in high travel time variance for routes that incorporate them.

We explore all three of these approaches using a purpose-built simulator called CSIM. In addition, we analyze data collected from a Boston taxi fleet to further validate our assumptions.

Let us begin by defining a few terms that will be used in this chapter.

First, we use the term coverage to describe the fraction of road miles that meet a minimum sampling requirement. We say a road segment (an uninterrupted road between two intersections) is covered if, over a given time period \( t \), that road segment is traversed at least \( k \) distinct times. If we assume our vehicle trips are independent, then \( k \) needs to be at least 10 to estimate the mean for our traffic service. Note, this number is somewhat arbitrary, and various applications use a number between 10 and 20. The time period \( t \) will depend on the specifics of the application and how long a significant number of samples are valid. For our traffic monitoring service, \( t \) is an hour. However, for other applications, such as road quality monitoring, \( t \) might be on the order of days if not a month. For our purposes, the time of day is not explicitly modeled. For a traffic service you would want to primarily monitor the hours around rush hour.

Second, we will report our results in terms of drive hours, rather than number of vehicles. One drive hour equals one vehicle driving for one
hour. Likewise, two vehicles driving for one hour equals two drive hours. Any given amount of coverage will require a certain number of drive hours. However, any given number of drive hours can be achieved with a wide range in the number of physical vehicles, depending on the driving schedule. Using drive hours reduces confusion and allows us to easily extrapolate to different sized fleets with varying duty schedules. For example, when we report coverage for the 10 sample coverage, 100 drive hour mark, this value could be achieved by having 100 vehicles driving for each of the hours of the day you wish to monitor. If these vehicles are not driving for the full time, then an appropriate number of additional cars would be needed to achieve this level of coverage.

What follows in the next section is a discussion of the simulator architecture, followed by an analysis of the results from several different workloads.

4.1 CSIM - THE CarTel COVERAGE SIMULATOR

CSIM is designed to simulate the route selection patterns of vehicles in a road network. Our goal is to determine a reasonable projection for how many participating vehicles we might need to meet different application requirements. The simulator is not designed to model low-level vehicle dynamics, such as traffic light timing or affects of cars on each other's driving behavior. Although this type of simulation would be important for understanding why specific streets experience congestion, our goal is to present a first order analysis of opportunistic mobility using cars.

We use map data from NAVTEQ to construct our simulation road network. To focus our analysis, we chose a subset of the Boston Metro area, representing 2,234 km of roadway. This region is shown in Figure 4.1.

The motion model for CSIM is designed to mirror a taxi service. Each trip starts at the location where the previous ended. Based on our own experience, destinations such as airports or central business districts tend to attract a disproportionate share of traffic. Our motion model takes this skew into account by selecting routes that traverse between these high traffic clusters, with the occasional trip to an outlying destination. For the purposes of our analysis, we selected 10 such destination clusters, as shown in Figure 4.1. One of the inputs to CSIM is the probability to select one of these 10 clusters. Otherwise, a random node on the map is selected. Most of our subsequent analysis will be done with a cluster selection probability of 90%.

Conceptually, CSIM is straightforward. It takes as input a map topology, a list of destination clusters, a probability for selecting among destination clusters as opposed to random nodes, and the duration of the simulation. It outputs the map topology annotated with the visit times.
CSIM models the roadway network as a directed graph in which the nodes represent intersections and the edges represent the streets between them, with separate edges for each direction of traffic. At the start of the simulation period a vehicle begins at a random location. To select the next destination, we probabilistically select a random node from either the map at large or from a constrained set consisting of nodes within a quarter-kilometer of one of the destination clusters. The vehicle then travels along the shortest path between these nodes at the posted speed limit. We record the visit times for each segment. This entire process is repeated until the end of the simulation. Note, we do not include a rest period between successive trips as our goal is to record the drive hours needed to reach a certain coverage level. Rest periods can be factored in later when applying the results to a known driving pattern.
4.2 PARTICIPATORY MONITORING RESULTS

In this section we explore the coverage properties of participatory deployments. As mentioned earlier, the backdrop for this analysis is a hypothetical traffic service that requires between 10 and 20 samples per hour per road segment for accurate congestion reports.

In a participatory monitoring deployment we want to monitor the roads driven by our users. For any given number of drive hours, we calculate the coverage for roads traveled on at least once. If a road hasn’t been traversed, there isn’t much reason to monitor it.

Figure 4.2 shows the fraction of the distance driven that is covered with at least 10 or 20 samples. This is from a simulation run where we cluster with 90% probability to 10 of our cluster locations. What we see is a steep growth in coverage for the first 200 hours, achieving 81% coverage at 100 hours, and then a gradual flattening to more than 90% coverage at 500 hours.

One question we might have is: How sensitive is our motion model to cluster size and selection probability? Below is a table showing how coverage at the 100 and 200 hour mark changes depending on the driving model.

<table>
<thead>
<tr>
<th>Cluster Prob.</th>
<th>Cluster Count</th>
<th>100 Hour Cov.</th>
<th>200 Hour Cov.</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>10</td>
<td>81%</td>
<td>89%</td>
</tr>
<tr>
<td>90%</td>
<td>2</td>
<td>87%</td>
<td>90%</td>
</tr>
<tr>
<td>50%</td>
<td>10</td>
<td>63%</td>
<td>79%</td>
</tr>
<tr>
<td>50%</td>
<td>2</td>
<td>66%</td>
<td>79%</td>
</tr>
<tr>
<td>10%</td>
<td>10</td>
<td>55%</td>
<td>76%</td>
</tr>
<tr>
<td>10%</td>
<td>2</td>
<td>55%</td>
<td>76%</td>
</tr>
</tbody>
</table>
This table shows that coverage tends to be greatest when cluster probability is high. This result follows from our intuition, as the best route between any two clusters tends to be fixed, resulting in a larger overlap between independent trips. We also see that with a given cluster probability, the number of clusters does not result in significant coverage differences.

In addition to simulated results, we wanted to evaluate coverage using driving from a real fleet of vehicles. To do so, we collected location traces encompassing 500 drive hours from a taxi fleet in the Boston area and performed a similar analysis as done in Section 4.2.

Figure 4.3 shows the fraction of the distance driven that is covered by at least 10 samples. What we see is a steep growth in coverage for the first 100 hours, achieving just under 80% coverage, and then a gradual leveling off to about 90% coverage at 500 hours. These results are on-par with our simulated results.

### 4.3 Infrastructure Monitoring Results

In this section we explore the coverage properties of infrastructure focused deployments. In these types of deployments we monitor a fixed set of roads. The roads we select depend on the specifics of the monitoring application. For our traffic congestion service we want to monitor the top 5% of congested roads. To determine which roads experience significant congestion we analyzed previously collected location traces from a fleet of taxis in Boston. We used the following process to find congested
FIGURE 4.4—Congested road segments in the Boston area.

segments:

1. For each segment in our street map graph, we calculated the mean travel time based on our taxi traces.

2. For each taxi trip, we compared the actual delay for each segment in the trip to the mean delay. If it exceeded the mean by both a constant factor (30 seconds) and relative factor (50%), we deem that segment congested, and make note of the time of day.

3. Next, we ranked the segments according to the number of hours in a given day that the segment experienced congestion.

4. Finally, we chose the top 5% (125 km) of segments, representing the segments that experienced congestion for the greatest number of hours per day.

Note, this is one of many potential schemes for picking congested segments. The rest of our analysis centers on determining how many hours of driving are needed to adequately cover these most congested segments.

Figure 4.4 shows the congested segments, drawn in bold, which represent about 5% (125 km) of the roadway distance. For those familiar with the roads in Boston, we see that many of the major arteries, such as Massachusetts Avenue, Cambridge Street, and the McGrath Highway, are all selected, which follows our intuition and personal driving experiences.
Figure 4.5—Simulation results: coverage for infrastructure monitoring.

Figure 4.5 shows how well our simulated vehicles covered these congested segments. What we find is that after 100 drive hours of operation, 30% of the hotspots meet our requirement of at least 10 samples. After 300 hours, we see coverage level off at just under 50%. These coverage values are less than those for participatory monitoring, which is to be expected. Congested segments tend to get a substantial amount of traffic—however this is not always the case with many outlying streets, making it much more difficult to completely cover the selected segments.

* * *

We also validated these simulation results using our fleet of taxis, as described in the previous section. Although the taxi results were slightly lower than that of the simulation, the difference was slight. The primary difference is coverage does not level out at 300 hours. This is to be expected, as the congested segments were derived the taxi traces. The graph is omitted for brevity.

4.4 Arterial Monitoring Results

The final scenario we analyze is monitoring major arteries. In our specific case, we want to monitor all roads whose posted speed limit is above 25 mph.

Figure 4.6 shows these major arteries, representing about 193 km, or 8 percent of the network. Note, this includes most of the main traffic thoroughfares in Boston, including Storrow Drive, Memorial Drive, I-90, and I-93.
Figure 4.7 shows how well our simulated vehicles covered the arter-ies. What we find is that after 100 drive hours of operation, 30% of the hotspots meet our requirement of at least 10 samples. After 300 hours, we see coverage level at just over 40%.

We also validated these simulated results against those derived from a set of taxi traces. The results were similar, with the taxi traces showing just under 25% coverage at 100 drive hours, and near 35% coverage at 300 drive hours.

These slightly lower values than those provided by infrastructure mon-itoring are to be expected. Arteries are often used for traveling into or out of the monitoring region. Our simulation and taxi traces have most of their trips completely within the monitoring region.

4.5 SUMMARY

In this chapter we looked at coverage for three different types of deploy-ments: those for participatory-, infrastructure-, and arterial-focused mon-itoring. We found participatory monitoring to be the easiest to satisfy, as we are only trying to monitor those roads a given subset of drivers use. With 100 drive hours, our cars achieved over 80 percent coverage. With infrastructure monitoring, our goal was to sample a subset of the road network deemed important—in our case those segments that frequently experience traffic congestion. Adequate coverage proved to be much more
difficult to achieve, as the 100 hour mark only resulted in 40 percent coverage. However, we still think this is adequate for many applications. Finally, we looked at monitoring major arteries. At 100 drive hours, only 30 percent of our desired roads were covered. This could be due to the fact that major arteries are mainly used for moving into or out of the monitoring region—our simulation focused on traffic that started and ended mainly within the region. We also validated all of these scenarios against real taxi traces, showing similar results.

Taken together, these results lend credence to the notion that opportunistic mobility is a reasonable strategy for achieving wide-area coverage in a low-cost sensor system. The details of each specific application will necessitate a different number of cars to meet the needed sampling requirements. However, the high level point of this chapter is that hundreds, or even the low thousands, of cars is all that is needed to make such a system work—a number that is quite achievable with today’s technology and device cost structure.
5

Related Work

Over the past few years, advances in wireless networking and embedded computing have led to the “first generation” of wireless sensor networks, including some impressive field deployments [1, 16, 55, 81]. In general, these are for monitoring or tracking applications characterized by low data rates and static deployments. The CarTel system, among others, represents a new generation of sensor networking, that focuses on solving the challenges related to mobility, delay-tolerant networking, and urban sensing. The following sections explore recent work in these areas.

5.1 Mobile Sensor Networks

Recent work in the NIMS project [46, 48] and underwater sensing [82] has focused on using mobility when it is not feasible to build a dense network of fixed sensors, due to sensor or instrumentation costs or a large geographic area that needs to be covered. In these cases, mobile nodes are typically robots that follow a controlled movement pattern to collect data about regions of interest.

ZebraNet [45] exploits inherent mobility in the sensing deployment. By placing sensors on animals roaming in Africa, researchers have been able to observe their movement and socialization patterns. However, researchers had the added challenge of fitting their hardware on collars around the animal's neck and had to deal with the limited energy budget such an environment would offer.

BikeNet [23] is a mobile sensor system designed to do cyclist experience mapping. Bikes have been outfitted with an array of sensors whose readings are delivered to a central visualization portal. Applications include pollution monitoring, cyclist fitness assessment, and route tracking. Bikes deliver their sensor data to the portal using deliberately placed sensor access points as well as by data muling using nearby bikes.

CarTel shares many of the same motivations of these system, but in-
stead focuses on applications for wide area sensing that occur in a vehicular context.

5.2 Delay-Tolerant Networking

Many researchers have studied the potential throughput and energy benefits of muling [7, 31, 43, 47, 48]; though energy constraints are not an issue in the current implementation of CarTel, we exploit the throughput advantages that muling offers in CafNet.

There are several mule-based, delay-tolerant network architecture proposals in the community [27, 34, 36, 44, 50, 51, 64, 73, 74, 89]. These systems typically provide some mechanism for buffering data that applications want to send while disconnected, possibly with some custody transfer [27] whereby intermediate nodes accept responsibility for reliably delivering data connected by remote endpoints. Much of this related work focuses on issues related to routing over multiple hops in such networks; we plan to utilize this work as we move forward with our CafNet implementation. Thus far, we have concentrated on API design using callbacks to handle dynamic priorities.

Several research groups have been exploring the use of Wi-Fi or other short-range networks to provide connectivity. For example, in work on Infostations [31, 75], researchers have studied networks in which there are pockets of good connectivity. Their focus is on analyzing the throughput and latency of such networks rather than on designing data management and application infrastructures for them.

Finally, there has been some research into using mobile nodes for emissions and pollution monitoring [30, 60]; we hope to integrate similar solutions into CarTel.

5.3 Query Processing

Many research projects have noted the need for in-network query processing [42, 52, 87] in sensor networks. Like CarTel, these systems are typically motivated by a need to reduce the bandwidth consumption that collecting all data from a network would require. Unlike CarTel, however, these systems have typically focused on low-data rate, well-connected sensornets.

ICEDB also bears some similarity to previous work on stream processing for continuous queries [14, 18, 59]; however, intermittent connectivity is not a failure case in CarTel as it is in these systems. Furthermore, dynamic prioritization of results and the simple SQL extensions to express priorities are important features of ICEDB that are largely missing from other systems. In a few cases, prioritization schemes are used to decide
what data to cache on clients when connectivity is available [8, 19, 49] rather than on what data to transmit, as in ICEDB.

The juggle operator [68] developed as part of the CONTROL project provides a method for allowing users to prioritize the delivery of results from particular groups in long running queries over disk-based data. Their approach is only suitable to aggregate queries, and requires users to prioritize results as query results arrive (typically via a GUI). In ICEDB, we are concerned with all types of queries, and need a prioritization approach that does not require users to specify priorities for tuples as they stream into the portal. Hence, we chose a declarative approach that allows the system to use the PRIORITIZE clause to automatically assign priorities to tuples as they are produced.

Mediation systems [86] serve as an intermediate layer between data sources and the user applications that query for data, accessing and merging data from multiple potentially heterogeneous data sources. ICEDB's mechanism of adapters are similar to the wrappers found in mediators, which transform the data at each distinct data source into a common, uniform representation and semantics so that the mediator can integrate the homogenized data.

Amsaleg et al. presented query scrambling [5] as an approach to query processing where data arrival may be delayed. By reordering and restructuring the query plan, the query processor can perform other useful work while waiting for data from a data source. Query scrambling addresses initial delays that arise from difficulty connecting to the data source, or when the data source experiences heavy load, and assumes stable connectivity thereafter. ICEDB handles delays that may be considerably longer.

5.4 URBAN SENSING

Using sensor networks for road traffic monitoring has recently become a hot topic. For example, in the TrafficLab project at Rutgers [22, 61], researchers use an ad hoc networks of cars to collect and disseminate traffic information to cars on the same road. Their system is largely focused on networking issues, however, rather than on the sensing and data collection issues that are at the core of CarTel. In particular, CarTel does not currently use car-to-car communication.

JamBayes [37] is a probabilistic traffic forecasting service. They used historical and real time traffic data to build models that predict the onset of congestion up to an hour in advance for freeway bottlenecks throughout the Seattle area. The service sends alerts to users' smartphones and can forecast unexpected delays along user-configurable routes. CarTel is a complementary system that could be used to collect and analyze traffic data for roads outside of the highway network that are not instrumented.
The PATH project [65] at UC Berkeley has investigated a number of issues related to smart transportation systems, including the use of sensor networks for on-road monitoring [24]. On-road networks present an alternative to the monitoring approach taken in CarTel: they provide relatively fine resolution about a small area of the roadway, whereas CarTel provides spottier information about a much larger geographic area.

There has also been recent interest in using cellular phones as traffic monitoring devices: by using the location features in most cellular devices, it is possible to determine how fast different roadways are moving [77]. Although this approach is likely to be good for road speed monitoring (modulo privacy concerns), it does not offer the ability to collect other types of information that CarTel also monitors. We are targeting cellular phones and other handheld devices as a future platform for CarTel software; we envision mobile users collecting information about the environment just as cars do in our system today.

Finally, there are specialized traffic services like Inrix [41] and SmarTraveler [76] that aggregate information from various online traffic information sources to present a view of road speeds and hazards in urban areas. In addition, Google Maps for mobile [9] recently started using GPS enabled smartphones that run its software to anonymously collect velocity and location samples from drivers on roads in the US. These samples allow them to annotate their maps with live traffic estimates of US highways and major arteries.

5.5 **OPPORTUNISTIC WI-FI**

The performance of TCP and UDP in wireless network scenarios from stationary clients has been fairly well-studied [2]. For the most part, however, few previous measurement studies have attempted to characterize wireless “LAN” network performance from moving vehicles. Ott and Kutscher [63] study the behavior of network connections that are initiated over an IEEE 802.11b channel from a moving car. This study involved a small number of bi-directional measurements over both UDP (RTP) and TCP. The goal was to understand the impact of the car’s speed, transmission rate, 802.11 bit-rate, and packet size on throughput and delay. The experiments used a fixed, carefully planned test-bed, one in their lab and one in a limited two-AP deployment.

The authors break a TCP connection into three phases: the “entry” phase, the “production” phase, and the “exit” phase. During the entry and exit phases, the car is too far from the AP and throughput is low. Once the car is within a certain range (~200 m in their experiments), throughput rises as the car enters the production phase. The rise is rather dramatic, but the throughput is highly variable. Although a significant volume of data
can be transferred, the authors believe that proxies or changes to protocols may improve performance further. For example, they show in their more recent work [64] that it is possible to avoid startup overheads in transport protocols like TCP by using a proxy to hide short periods of disconnection from the transport layer.

Gass et al. [29] demonstrate the feasibility of using off-the-shelf IEEE 802.11b wireless networks for TCP and UDP transfers to and from a moving car.

Their experiments are also conducted in a planned environment—they measure performance from an “in-motion” client to a single access point in the California desert, where interference from other radios and vehicles is non-existent and there are no obstacles. This environment allows them to measure performance in a controlled mobile setting.

In the first part of the paper, the authors measure the performance between the client and the AP only. As in [63], they conclude that packet losses are low within 150 m of the AP and that for a wide speed range (5–75 mph), there is a region of good connectivity and throughput. Then, they measure the end-to-end performance of various protocols by connecting a server to the AP and changing the delay and bandwidth of that link, finding that Web performance degrades when a delay of 100 ms is added to the end-to-end link. They suggest that aggregating multiple HTTP requests will mitigate the impact of this delay; we believe that this delay could be avoided by simply issuing pipelined requests, an option already present in HTTP/1.1. (The authors use wget, an HTTP/1.0 client that does not appear to support pipelined requests.)

Similar performance studies have also been performed for cellular networks [67, 69]. Cellular networks, however, have characteristics that are very different from the urban and suburban in situ IEEE 802.11b deployments we are examining in our work.

Several “war driving” studies have mapped APs in various areas around the world (e.g., http://placelab.org, wifimaps.com, http://wardriving.com or http://wigle.net), but these studies have generally not gone beyond characterizing the location of APs. Akella et al. [3] also measure the supported bit-rates, whether encryption is turned on, and (more recently) [4] the coverage region of each AP for a few thousand APs in Pittsburgh.
6

Lessons Learned & Conclusion

With hundreds of millions of automobiles to which embedded computers can be attached, and billions of mobile phone-equipped people in the world, cars and humans may turn out to be the carriers of the world’s largest and most dynamic sensor networks in the coming years. Such mobile sensor networks have the potential to sense large expanses of the world at much finer fidelity and scale than possible by static deployments. This dissertation presents CarTel, which is a step towards a general-purpose mobile sensor computing system to realize this vision.

The bulk of the work for this dissertation was performed three years ago. In these intervening years, we’ve had the chance to iterate several versions of the system, incorporating many lessons learned from hands-on experience. Below is a brief discussion of what we learned.

6.1 Lessons Learned

A well-provisioned back-channel is essential for debugging. As the size of our deployments grew, so did the effort required to manually install software updates. An intermittent connection, especially one using a network stack that you are actively developing, made delivering large firmware updates difficult. Consequently, we outfitted each remote node with a 3G cellular modem, to provide a continuous connection for delivering large updates as well to provide a console for remote debugging.

Opportunistic, vehicular Wi-Fi works. Before starting this project there were real questions about the viability of opportunistic networking. Were there enough open access points? Could we connect fast enough and stay connected long enough to transfer a non-trivial amount of data? Chapter 3, as well as several other follow-on projects, have shown that it is possible to deliver significant amounts of data over these unplanned networks. For many messaging applications, the bandwidth and latency is
more than sufficient. Opportunistic networking provides an alternative to expensive cellular data connections, particularly for passively connected devices, or in less-developed parts of the world.

**Keep data delivery abstractions as simple as possible.** The data collection system for CarTel took many forms. Initially, we designed a purpose-built set of Python scripts to deliver a few data types. As our experience with the system and our ambitions grew, we saw the need for data prioritization and support of snapshot queries. We built a distributed database for intermittently connected environments called ICEDB. Unfortunately, we over-estimated our hardware capabilities as well as the actual need for such a complex system. In the end, as the hardware platform was further slimmed down, we went back to flat files and shell tools. This served as a reminder to us as to how important it is to keep the scope of your software closely aligned to the applications you are solving.

**Traffic congestion detection remains the killer app.** Although a CarTel-like system could be used to collect many different types of sensor data—pollution, road quality, imagery—collecting location traces for traffic aware routing resonated almost immediately with our user base. Much on-going work is dedicated to more efficiently collecting these location traces using a number of devices and phones. In addition, the algorithms for traffic aware routing are much more complex and nuanced than initially envisioned.

### 6.2 Conclusion

Wide-area sensor systems enable a broad class of applications, including fine-grained monitoring of traffic congestion, road surface conditions, and pollution. This dissertation showed that it is possible to build a low-cost, wide-area sensor system. Our approach relied on two techniques: using existing motion from such sources of mobility as cars and people to provide coverage (opportunistic mobility), and using the abundance of short duration network connections to provide low-cost data delivery (opportunistic networking).

We use these two techniques to build a mobile sensor computing system called CarTel, to collect, process, deliver, and visualize spatially diverse data. CarTel consists of three key components: hardware placed in users' cars to provide remote sensing, a communication stack called CafNet to take advantage of opportunistic networking, and a web-based portal for data visualization. This dissertation described the design and implementation of these three components.

In addition, we analyzed the properties of opportunistic networking
and mobility. To show the viability of opportunistic networking, we studied Internet access from moving vehicles and found that the median duration of link layer connectivity at vehicular speeds was 13 seconds, the median connection upload bandwidth was 30 KBytes/s, and that the mean duration between successful associations to APs was 75 seconds. To show the viability of opportunistic mobility, we used a simulation and found that after as little as 100 drive hours, a CarTel deployment could achieve over 80 percent coverage of useful roads for a traffic congestion monitoring application.

CarTel has been deployed on over a dozen cars, running on a small scale in several metropolitan areas in the US for over a year. Over this time, we have collected hundreds of hours and thousands of miles worth of data from drives, including data about road traffic speed and delays, the quality and prevalence of Wi-Fi access points on drive routes, images from an attached camera, and on-board automotive diagnostic data using the OBD-II interface. All this data is accessible to users via a Web site, which uses CarTel's geo-spatial visualization interface. Our experience, though limited, suggests that CarTel's three components—the portal, ICEDB, and CafNet—are an effective way to collect, process, deliver, and visualize data from mobile sensor networks.
References


[65] PATH Project. http://www.path.berkeley.edu/. (Referenced on page 89.)


[75] SMALL, T., AND HAAS, Z. J. The shared wireless infostation model: A new ad hoc networking paradigm (or where there is a whale, there is a way). In MOBIHOC (2003), pp. 233–244. (Referenced on page 87.)


