CarTel: A Distributed Mobile Sensor Computing System

Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko,
Allen Miu, Eugene Shih, Hari Balakrishnan and Samuel Madden
MIT Computer Science and Artificial Intelligence Laboratory
cartel@nms.csail.mit.edu

Abstract

CarTel is a mobile sensor computing system designed to collect, process, deliver, and visualize data from sensors located on mobile units such as automobiles. A CarTel node is a mobile embedded computer coupled to a set of sensors. Each node gathers and processes sensor readings locally before delivering them to a central *portal*, where the data is stored in a database for further analysis and visualization. In the automotive context, a variety of on-board and external sensors collect data as users drive.

CarTel provides a simple query-oriented programming interface, handles large amounts of heterogeneous data from sensors, and handles intermittent and variable network connectivity. CarTel nodes rely primarily on opportunistic wireless (*e.g.*, Wi-Fi, Bluetooth) connectivity—to the Internet, or to "data mules" such as other CarTel nodes, mobile phone flash memories, or USB keys—to communicate with the portal. CarTel applications run on the portal, using a delay-tolerant continuous query processor, *ICEDB*, to specify how the mobile nodes should summarize, filter, and dynamically prioritize data. The portal and the mobile nodes use a delay-tolerant network stack, *CafNet*, to communicate.

CarTel has been deployed on six cars, running on a small scale in Boston and Seattle for over a year. It has been used to analyze commute times, analyze metropolitan Wi-Fi deployments, and for automotive diagnostics.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed applications

General Terms

Design, Implementation, Experimentation

Keywords

Sensor networks, mobility, intermittent connectivity, data management, query processing, data visualization

1 Motivation

CarTel is a distributed sensor computing system motivated by the hypothesis that an important and emerging category of sensor networks is *mobile* and involves *heterogeneous sensor data*. The motivation for mobile, heterogeneous sensor networks 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. These systems can sense the environment at much finer fidelity and higher scale than static sensor networks, particularly over large areas. For example, to monitor commute delays on our roadways, one approach would be to deploy static sensors on roads. While this approach may be tenable for the major roadways in any area, given the large expanse of backroads that many commuters use, it may not be a practical way to cover the entire area around a city. A complementary approach, which we adopt, is to instrument each car with a GPS sensor to opportunistically obtain information about traffic delays observed as cars move and to use that information in traffic monitoring and route planning applications.

In addition to traffic monitoring, mobile sensors, particularly on vehicles, can be used for:

1. *Environmental monitoring*, by using mobile chemical and pollution sensors.
2. *Civil infrastructure monitoring*, by attaching vibration and other sensors to cars to monitor the state of roads (*e.g.*, potholes, black ice).
3. *Automotive diagnostics*, by obtaining information from a vehicle's on-board sensors, which can help in preventive and comparative diagnostics. This information can also be used to monitor bad driving tendencies.
4. *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.
5. *Data muling*, by using cars (and people) as "delivery networks" for remote sensornets, sending data from these networks to Internet servers.

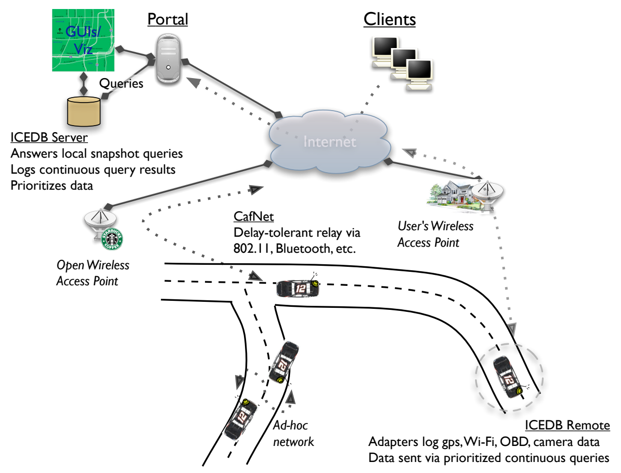

Portal
Clients
GUIs/Viz
Queries
Internet
ICEDB Server
Answers local snapshot queries
Logs continuous query results
Prioritizes data
CafNet
Delay-tolerant relay via 802.11, Bluetooth, etc.
User's Wireless Access Point
Open Wireless Access Point
Ad-hoc network
ICEDB Remote
Adapters log gps, Wi-Fi, OBD, camera data
Data sent via prioritized continuous queries

Figure 1. The CarTel system architecture showing the different components of the platform. Cars collect data as they drive, and log them to their local ICEDB databases. As connectivity becomes available, data on cars is delivered via CafNet to the portal, where users can browse and query it via a visualization interface and local snapshot queries.

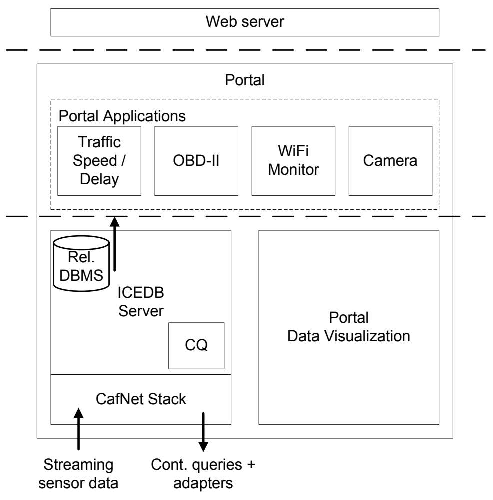


Figure 2. The CarTel portal software architecture. Applications, such as those that measure traffic delays or request terrain photos from the camera, run snapshot queries against the relational DBMS in ICEDB to access data, displaying the results of those queries on geographic overlays using the data visualization API. They can also issue continuous queries and create adapters that are pushed to mobile nodes via CafNet. Mobile nodes stream sensor data back as connectivity permits; this data is stored in the relational database.

These example applications motivate our design of CarTel. Of course, static sensor networks have been successfully used for some of these applications, particularly in environmental and civil monitoring [38, 3, 52, 7, 11]. Mobile sensor networks, however, offer the potential to instrument a much larger geographical 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,[1] or in situations where the number of sensors is so large that a static deployment is either too expensive or too cumbersome to establish and maintain.

CarTel makes it easy to collect, process, deliver, and visualize data from a collection of remote, mobile, and intermittently connected nodes. It 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.

The rest of this paper describes the design, implementation, and evaluation of CarTel, and discusses some case studies. We start by presenting an overview of the system and the contributions of this paper.

2 Overview and Contributions

CarTel provides a reusable software platform that can be used to build many mobile sensing applications. Each node in CarTel is a mobile, embedded computer coupled to a set of sensors. This computer runs software that functions as a gateway between the sensors attached to it (either physically or via short-range radio) and the rest of the system.

The technical design of CarTel meets the following goals:

1. **Provide a simple programming interface.** Our goal is to centralize and simplify the development of mobile sensor network applications. CarTel applications should be as easy to write as standard Web applications, and application developers should not have to deal with distribution or mobility.
2. **Handle large amounts of heterogeneous sensor data.** CarTel should not constrain sensor data types, and should make it easy to integrate new kinds of sensors, new mobile nodes, and new data types into the system. Moreover, since the rate at which data is collected by media-rich mobile sensors such as cameras will often exceed the available network bandwidth to Internet hosts, CarTel requires local buffering and processing on the mobile nodes.
3. **Handle intermittent connectivity.** The primary mode of network access for mobile CarTel nodes is via opportunistic wireless (e.g., Wi-Fi, Bluetooth). Wi-Fi access points run by users in homes and other locations

[1]For, example, mass spectrometers are the best known way to detect various organic pollutants, and even "low-cost" spectrometers cost tens of thousands of dollars [42].

can opt in to the CarTel system, allowing mobile CarTel nodes to communicate with Internet hosts.[2] In most urban areas, such connectivity will be intermittent, alternating between periods of relatively high bandwidth (tens of kilobytes per second today) and no connectivity. In addition, a CarTel node can use mobile storage devices such as USB keys and flash memory (*e.g.*, on Bluetooth-equipped mobile phones) as "data mules", relying on those mules to deliver data in best-effort fashion. Finally, as the density of CarTel-equipped nodes increases, they may be able to exchange data with each other as they move via Wi-Fi or Bluetooth.

2.1 CarTel Components

The system has three main components. The *portal* is the central location that hosts CarTel applications and functions as the point of control and configuration for the distributed system. The portal is also the "sink" for all data sent from the mobile nodes. CarTel applications use two other CarTel components to specify how the mobile nodes should collect, process, and deliver sensor data: *ICEDB* (intermittently connected database), a delay-tolerant continuous query processor, and *CafNet* (carry-and-forward network), a delay-tolerant network stack. Figure 1 shows a high-level view of the CarTel distributed system.

The CarTel programming model is centralized and simple, and is shown schematically in Figure 2. Applications running on the portal issue continuous queries using an API exported by ICEDB. These queries cause mobile nodes to stream responses using CafNet's data delivery mechanism. Unlike in traditional stream processing applications, however, results are not directly sent to the querying application. Instead, CarTel uses the following alternate approach:

1. Queries specify what sensor data must be acquired and at what rate, how the data should be sub-sampled, filtered, and summarized on the mobile node, and in what (dynamic) priority order results should be sent back to the portal. Because sensors often produce more data than the network can promptly deliver to the portal (*e.g.*, due to insufficient bandwidth or lack of connectivity), applications on the portal need a way to specify how to prioritize data (*e.g.*, preferring summaries to be delivered before detailed values). The ICEDB continuous query interface allows an application to express intra- and inter-query priorities.
2. Query results stream in across an intermittently connected network and populate a relational database at the portal.
3. Applications issue SQL queries on the portal's relational database to retrieve data they need for further analysis, visualization, etc. These are snapshot queries that run on whatever data is currently available. Applications do not wait synchronously for the results of continuous queries.

This design is well-suited to intermittently connected operation, as well as to achieving low-latency responses to application queries (step 3). The continuous queries are logically centralized, but run in distributed fashion on each node. ICEDB uses CafNet to deliver these continuous queries to the remote nodes in situ whenever the node connects to the portal and new queries need to be "pushed" to the node. The node begins executing queries as soon as they are received.

CarTel handles heterogeneous sensor data, allowing the set of sensors to be expanded without requiring major software changes on the remote nodes. Each sensor has an *adapter* running on the node that handles the details of configuring and extracting information from that sensor and converting it into a normalized form. To ease management and deployment, when a new sensor is added, or when the functions of an adapter need to be modified, only the adapter module needs to change. CarTel also provides the facility for more massive software upgrades (*e.g.*, to fix bugs in CarTel or node's the operating system), but these are expected to be much less frequent than updating adapters.

The portal includes a geo-spatial data visualization system that stores sensor data from cars. It organizes data in terms of *traces*, which are sets of sensor readings collected during a particular drive. Users are given a simple graphical query interface for selecting the traces they are interested in and visualizing various summaries, statistics, and maps within or across the traces they select.

2.2 Contributions

CarTel builds on the results of much previous work on mobile systems, sensor data management, and delay-tolerant networks (Section 8 discusses related work). Our primary contribution is the synthesis of ideas—some of which are novel—in each of these areas, into a single system that satisfies the goals mentioned at the beginning of this section. The result is a working system that has been running on six cars in a small-scale deployment for over a year. This system currently collects road traffic data, monitors the quality of Wi-Fi access points on routes, captures images along drives, and gathers a variety of data from the On-Board Diagnostic (OBD-II) interface on cars. Although our experience with these applications is somewhat limited, we have found that CarTel greatly simplifies the task of collecting, processing, delivering, and visualizing data in this environment.

This paper makes the following contributions:

- Extending the notion of continuous queries [39, 13, 2] to handle intermittent connectivity. In particular, this extension requires data streaming from the mobile nodes to be buffered locally, and the ability to dynamically prioritize data both on a single stream and between different streams. Section 3 describes this scheme in the context of ICEDB.
- Enabling modular upgrades to integrate new sensors and data types using *adapters* (Section 3.1).
- The CafNet "carry-and-forward" delay-tolerant network stack that delivers data in intermittently connected environments (Section 4). Unlike the traditional sock-

[2]As in emerging metropolitan Wi-Fi deployments and wireless mesh networks, our belief (perhaps misguided) is that users will opt in because of financial incentives or because of the mutual benefit it provides. In our current deployment, communication occurs when users drive by access points controlled by the set of CarTel users, which turns out to be enough to upload data collected on drives within a few minutes to a few hours of the drive.

ets interface, the CafNet interface uses callbacks across all its layers. By issuing callbacks whenever network conditions change, CafNet makes it possible for the sender to dynamically prioritize data. At the same time, CafNet's network layer provides some buffering to achieve high utilization when network connectivity is fleeting (*e.g.*, a few seconds), a common situation at vehicular speeds.

- The design of the portal and its visualization interface (Section 5).
- A discussion of the results from four case studies—road traffic monitoring, traffic simulation, Wi-Fi network monitoring, and automotive diagnostics (Section 6)—that illustrate the ease with which a variety of sensor data processing applications can be built using CarTel.

3 ICEDB

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. The rest of this section describes the ICEDB data model and its delay-tolerant continuous queries.

3.1 Data Model

ICEDB supports heterogeneous data types and makes the addition and removal of sensors relatively easy. Because all results are eventually stored in a relational database, this requirement implies that the system be able to parse and store tuples that sensors produce and must be able to evolve schemas as users add new sensors and application developers introduce new data types.

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 [57].

The CarTel administrator and application developers can specify attributes on a per-adapter basis. A typical adapter includes the following attributes:

1. *ID and name*: Each adapter must be uniquely identified.
2. *Type*: An adapter can either *push* data to ICEDB over a local TCP socket, or ICEDB can *pull* data by invoking the executable at a specified rate. ICEDB invokes the executable for each push adapter once, so that it runs as a daemon in the background. Pull adapters, on the other hand, produce readings each time ICEDB invokes the executable program.
3. *Rate*: For pull type adapters, the rate at which ICEDB should invoke the executable.
4. *Forwarding flag*: Specifies whether or not raw data should be forwarded to the portal. Setting this flag is a shortcut for specifying a continuous query that selects all attributes from the sensor.
5. *Schema*: A list of (name, type) pairs that specifies the attributes produced by the sensor. In our current implementation, the type field must be a valid PostgreSQL [45] data type.
6. *Priority*: The priority assigned to forwarded tuples when the forwarding flag is set (priorities are explained in the next subsection).

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 currently 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.

3.2 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.[3] To support *continuous* queries in ICEDB, queries include a *sample rate* specified by a `RATE` clause. For example, consider the query:

```
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 (for example, 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 `RATE` (so if the current time is 10:02 AM, the above query would acquire readings at 10:05, 10:10, etc.)

In an intermittently-connected, bandwidth-constrained environment, delivering all data in FIFO order is suboptimal. 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

[3] In our current implementation, data produced by the adapters is stored in a local database table that continuous queries run over. This allows us to implement a simple continuous query processor that repeatedly invokes queries over a local relational database.

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. *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. *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.

3.2.1 Local Prioritization

Local prioritization uses two language extensions for specifying the local transmission order of query results: `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 domain) 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.

3.2.2 Global Prioritization

ICEDB applications express global priorities using 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:

```
SELECT ...
  EVERY ...
  BUFFER IN bufname
  SUMMARIZE AS
       SELECT f_1,...,f_n,agg(f_{n+1}),...,agg(f_{n+m})
       FROM bufname WHERE pred_1 ... pred_n
       GROUP BY f_1,...,f_n
```

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 information 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.

4 CafNet

CafNet is a general-purpose network stack for delay-tolerant communication. Applications can use it 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.

4.1 Overview

All CafNet nodes are named using globally unique flat identifiers that don't embed any topological or organizational semantics.[4] 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 [10, 19], 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) [15]. 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.

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

[4]As in previous work such as DOA [56], making these identifiers a hash of a public key (and a random salt) would ease message authentication.

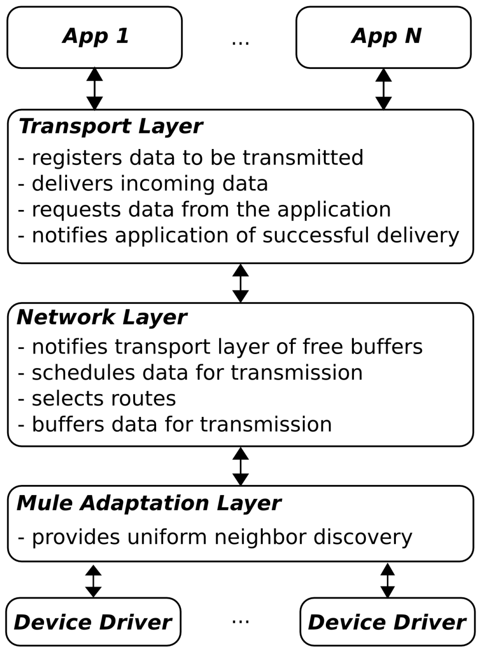

App 1
...
App N
Transport Layer
- registers data to be transmitted
- delivers incoming data
- requests data from the application
- notifies application of successful delivery
Network Layer
- notifies transport layer of free buffers
- schedules data for transmission
- selects routes
- buffers data for transmission
Mule Adaptation Layer
- provides uniform neighbor discovery
Device Driver
...
Device Driver

Figure 3. The CafNet communication 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 time.

Section 4.2 describes some additional details of these three layers. In Section 4.3, 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.

4.2 The Basic CafNet Stack

Figure 3 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. No kernel changes are required.

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).

4.3 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.

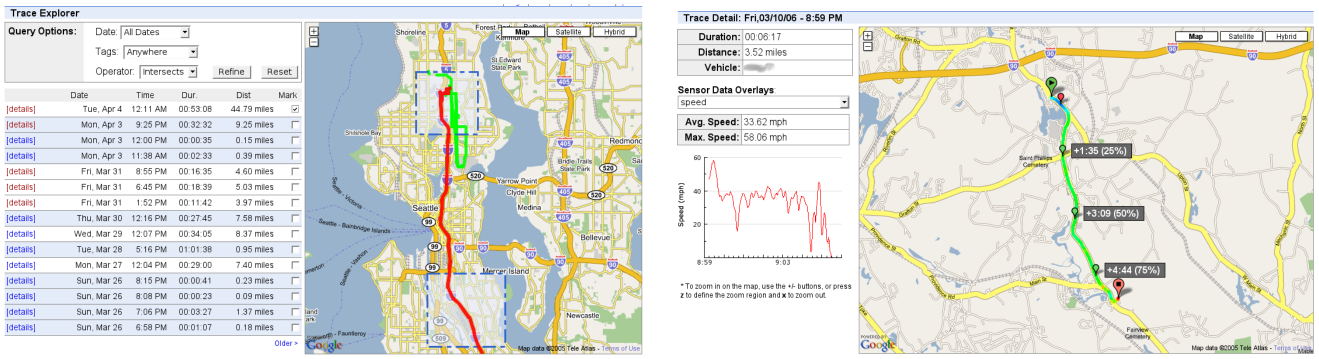
The above API focuses on the novel aspects of our design and is not complete; for instance, it does not include the data reception path, which is similar to traditional protocol stacks. It also does not include some other details such as the application being informed of what destinations are now reachable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

5 The Portal

Users navigate sensor data in CarTel using web-based applications hosted within the portal environment, shown schematically in Figure 2. An example of such an application is shown in Figure 4(b) 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.

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.

Applications communicate with ICEDB 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.


Trace Explorer

Query Options: Date: All Dates Tags: Anywhere Operator: Intersects Refine Reset

	Date	Time	Dur.	Dist	Mark
[details]	Tue, Apr 4	12:11 AM	00:53:08	44.79 miles	✓
[details]	Mon, Apr 3	9:25 PM	00:32:32	9.25 miles	☐
[details]	Mon, Apr 3	12:00 PM	00:00:35	0.15 miles	☐
[details]	Mon, Apr 3	11:38 AM	00:02:33	0.39 miles	☐
[details]	Fri, Mar 31	8:55 PM	00:16:35	4.60 miles	☐
[details]	Fri, Mar 31	6:45 PM	00:18:39	5.03 miles	☐
[details]	Fri, Mar 31	1:52 PM	00:11:42	3.97 miles	☐
[details]	Thu, Mar 30	12:16 PM	00:27:45	7.58 miles	☐
[details]	Wed, Mar 29	12:07 PM	00:34:05	8.37 miles	☐
[details]	Tue, Mar 28	5:16 PM	01:01:38	0.95 miles	☐
[details]	Mon, Mar 27	12:04 PM	00:29:00	7.40 miles	☐
[details]	Sun, Mar 26	8:15 PM	00:00:41	0.23 miles	☐
[details]	Sun, Mar 26	8:08 PM	00:00:23	0.09 miles	☐
[details]	Sun, Mar 26	7:06 PM	00:03:27	1.37 miles	☐
[details]	Sun, Mar 26	6:58 PM	00:01:07	0.18 miles	☐

Older >

Trace Detail: Fri,03/10/06 - 8:59 PM

Duration:	00:06:17
Distance:	3.52 miles
Vehicle:	[redacted]

Sensor Data Overlays: speed

Avg. Speed:	33.62 mph
Max. Speed:	58.06 mph

* To zoom in on the map, use the +/- buttons, or press **z** to define the zoom region and **x** to zoom out.

Figure 4. The CarTel portal, showing a user (a) querying for traces corresponding to his commute and (b) viewing the speed overlay for one trace.

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 [21] in our current implementation) for a given trace.

Figure 4(a) shows the user interface to navigate traces. By default, summary statistics for the most recently collected traces are presented alongside a map that shows the geographic extent of this data, as derived from GPS readings. The number of traces becomes large quickly after any significant amount of usage. 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 in Figure 4(a) where the user has selected two regions—the dashed rectangles—that correspond to the beginning and end of his commute. The operator that he selected is “intersects”. Additionally, if the user was 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.

Once a user finds a trace of interest, he can 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 4(b) 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.

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.

6 Case Studies

This section presents three case studies conducted using CarTel: road traffic monitoring, wide-area Wi-Fi measurements, and automotive diagnostics.

6.1 Road Traffic Analysis

CarTel is well-suited for studying issues related to traffic, congestion, and navigation. Each GPS-equipped node on a car serves as a fine-grained probe of the roadways, allowing us to measure delays along road segments and to infer congestion hot spots. Moreover, by equipping cars with cameras, we can build applications that help users better navigate unfamiliar terrain.

6.1.1 Commute Time Analysis

Most people have a handful of heuristics that help them decide which of many routes to take on their daily commute. We have observed users of CarTel quantitatively analyzing various alternative routes by comparing drive times of different routes at different times.

Using the GPS adapter and a continuous ICEDB query, the commute time application keeps an accurate record of the routes a driver takes and the delays along those routes. Users can display a detailed view of any trip that shows a trace color coded by vehicle speed and displays quartile progress markers indicating the distance traveled after each successive 25% of elapsed time. These visualizations help users identify heavily congested segments within a given route that should be avoided.

During our initial deployment, one of our users took particular interest in optimizing his commute. The following table shows sample travel times for his three routes between work and home:

Route	Avg. Dist.	Avg. Time	Std-dev
Freeway	9.94 miles	19:52	02:14
City Streets	9.83 miles	29:34	02:19
Frontage Road	9.27 miles	31:51	03:54

Prior to using CarTel, this user estimated that using Frontage Road would provide the shortest travel times. Also, the user felt that using Frontage Road results in a shorter commute compared to city streets, but this is not true. The user perceived the freeway as being the longest route due to frustrating delays at on-ramps and off-ramps. As this data indicates, the freeway route provides the shortest travel times with the least amount of variance while the frontage road provides the longest travel times. However, this data may be slightly skewed by the user's avoidance of the freeway during peak commute hours, which almost certainly would have increased the average route duration. In addition, this user has reported that he was able to use the quartile markers and color codes to mix and match routes to find faster paths.

The relatively small standard deviation in travel times indicates that routes are reasonably predictable. This predictability suggests that it is possible to build accurate models of traffic delays by aggregating GPS traces across all users. We can use these models to predict travel times for routes and times of day that an individual user may never have driven. This would help users answer such questions as "What time should I leave for the airport to get there by 9?" or "When is the latest I can leave for a meeting downtown and still be within 25% of the minimum travel time?" Answering such questions is an area for future work.

6.1.2 Traffic Hot Spot Heuristics

Although a detailed, street-level model of traffic delays is desirable, in many areas, simply knowing the main "traffic hot spots" is sufficient to construct a reasonable route that reduces traffic delays. This observation motivates an application on the portal that computes traffic hot spots.

We calculate traffic hot spots from the GPS records collected once per second. First, we define a grid on the map (.001 decimal degrees of latitude and longitude, which is approximately 100 meters by 80 meters) and assign each GPS point to the grid cell in which it is located. Next, we examine the velocity of the car at each point, as reported by the GPS unit, and compute the standard deviation of the velocities over all GPS records in a given cell. After filtering out cells with an insufficient number of samples, we place markers on a map at the center of the top ten cells with the greatest variation in velocity. In addition, users can restrict the query to run over a given interval during the day.

Figure 5 shows the top ten hot spots this application calculated for the areas around Boston and Seattle. Not surprisingly, in Seattle, many sections of I-5 show high variation in speed during commute times. Likewise, in Boston, I-93 is a key area of congestion. However, in both areas some intersections on back-roads display congestion too.

6.1.3 Image Acquisition

Driving to an unfamiliar location can be difficult, even with detailed driving directions. Streets are often unmarked and inclement weather can reduce visibility. One way of making turn-by-turn driving directions more useful would be to include photographs of landmarks preceding each turn. People often verbally give these types of directions: "turn left at the red barn" or "you've gone too far if you see the McDonald's."

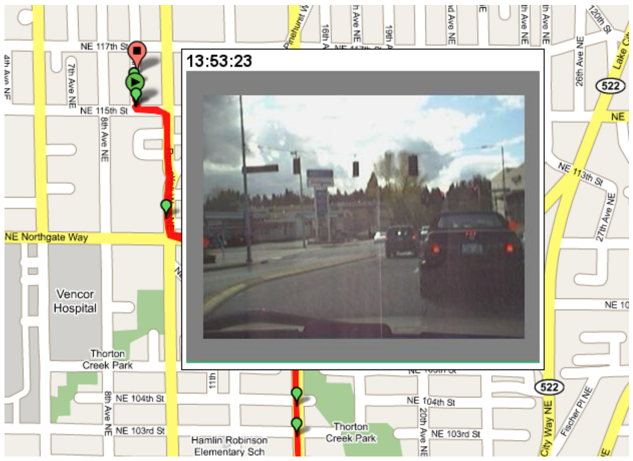


Figure 6. The CarTel portal, showing a street-level view just prior to a turn.

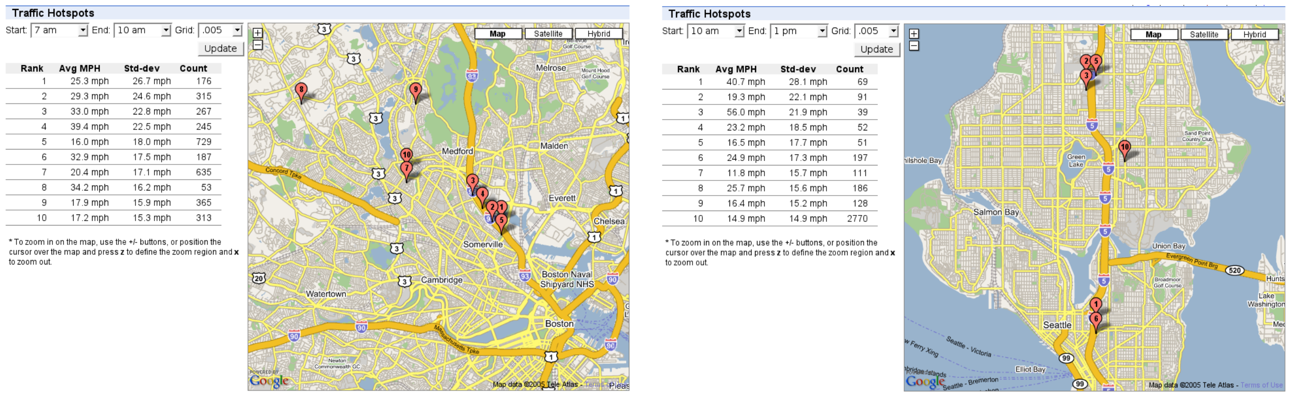
CarTel makes it easy to collect the photographs needed to augment driving directions with the requisite landmarks (we have not build route-finding application yet). As mentioned in Section 2, we have integrated a small camera into the sensor package deployed with one of the nodes installed in a user's car. Currently, a script is used to take a picture every few seconds. The portal uses these images to automatically generate a large repository of geo-coded, street-level images. This archive can be integrated with a driving direction application to provide images just prior to every turn, giving users a set of visual way-points along their journey. Although CarTel does not have any data types or functions specifically designed to manipulate images, PostgreSQL supports binary arrays that are adequate for storing and indexing our image data. Application-specific image processing could easily be added via an adapter that processes the images prior to insertion.

Currently, the images are delivered in no particular order. However, higher resolution images or more frequent acquisition would require a smarter scheme. One such scheme is the "bisect" DELIVERY ORDER BY function of Section 3.

Figure 6 shows a street-level view just prior to a turn on a user's route.

6.2 Wide-area Wi-Fi Measurements

According to Jupiter Research, 65% of on-line households have installed Wi-Fi access points (APs) at home. One could imagine that in areas with reasonably high population density, such APs (and the broadband links connecting them to the Internet) could provide Internet access to other users. In one model, a single large "virtual ISP" empowers the owners of these APs to function as "micro ISPs", providing Internet service. Of course, there are many important legal, business, privacy and other issues that must be resolved before this idea becomes practically viable, but the interesting


Traffic Hotspots

Start: 7 am End: 10 am Grid: .005

Rank	Avg MPH	Std-dev	Count
1	25.3 mph	26.7 mph	176
2	29.3 mph	24.6 mph	315
3	33.0 mph	22.8 mph	267
4	39.4 mph	22.5 mph	245
5	16.0 mph	18.0 mph	729
6	32.9 mph	17.5 mph	187
7	20.4 mph	17.1 mph	635
8	34.2 mph	16.2 mph	53
9	17.9 mph	15.9 mph	365
10	17.2 mph	15.3 mph	313

* To zoom in on the map, use the +/- buttons, or position the cursor over the map and press **z** to define the zoom region and **x** to zoom out.

Traffic Hotspots

Start: 10 am End: 1 pm Grid: .005

Rank	Avg MPH	Std-dev	Count
1	40.7 mph	28.1 mph	69
2	19.3 mph	22.1 mph	91
3	56.0 mph	21.9 mph	39
4	23.2 mph	18.5 mph	52
5	16.5 mph	17.7 mph	51
6	24.9 mph	17.3 mph	197
7	11.8 mph	15.7 mph	111
8	25.7 mph	15.6 mph	186
9	16.4 mph	15.2 mph	128
10	14.9 mph	14.9 mph	2770

* To zoom in on the map, use the +/- buttons, or position the cursor over the map and press **z** to define the zoom region and **x** to zoom out.

Figure 5. The CarTel portal, showing users' traffic hot spots for the Boston area (left) and the Seattle area (right).

question for us is what the performance of such a network is likely to be, particularly for mobile users moving at vehicular speeds.

To address this question, we used an earlier version of CarTel (without ICEDB) to collect over 290 "drive hours" of data about Wi-Fi connectivity in urban environments over several months. Below, we summarize a few of the main results. A more detailed discussion of the study and the results can be found in [8].

In addition to the GPS adapter used for traffic analysis, we collected connectivity data using a Wi-Fi adapter. Our data collection program continually scanned for APs, attempted associations, and collected statistics about the following events as our users traveled on their daily paths:

- *Wi-Fi scans*, which are reports that list nearby APs.
- *Wi-Fi associations*, which are attempts to establish link-layer connectivity with APs.
- *IP address acquisitions*, which are attempts to acquire an IP address using DHCP (optimized with a cache from previous trips).
- *Ping and upload statistics*, which are connectivity and throughput statistics of TCP uploads through open APs.

We used this data to calculate the density of urban Wi-Fi networks to estimate the feasibility of using such networks for vehicular Internet access. In total, we discovered about 32,000 distinct APs, of which we were able to associate with about 5,000 and acquire an IP address from about 2,000. Moreover, the likelihood of a successful association was the same across a large range of urban vehicular speeds up to 60 km/hour.

The following table summarizes some of our local and end-to-end connectivity data:

Mean association duration	25 seconds
Mean time between connections to Internet	260 seconds
Median upload throughput	30 KBytes/s

These findings [8] suggest that such unplanned *in situ* Wi-Fi networks can in fact be used with a delay-tolerant protocol stack such as CafNet to provide connectivity for mobile users traveling at vehicular speeds in urban and suburban areas. This Wi-Fi study of connectivity is a good example of large-scale infrastructure monitoring enabled by CarTel. Using a database split between the central server and the mobile nodes (which eventually became the ICEDB system described in this paper) and a single data-independent upload mechanism on CarTel units allowed us to make iterative changes with only minor side-effects. Because our experiments evolved with time and numerous enhancements were driven by previous findings (as is typical in such studies), having a flexible data acquisition and analysis system proved invaluable.

6.3 Automotive Diagnostics

This section illustrates a driving pattern analysis that we were able to perform using CarTel and briefly discusses some of the additional car sensor data we currently collect.

6.3.1 Analyzing Driving Patterns

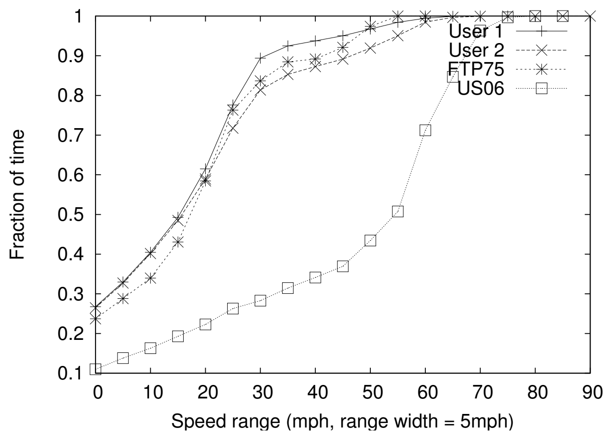
The U.S. Environmental Protection Agency (EPA) maintains a Federal Test Procedure (FTP75) by which cars are rated for fuel economy and emission levels. The procedure performs measurements as the car is driven along a particular schedule of speeds. This *driving schedule* is designed to be representative of typical urban driving patterns including highway driving. The test has been criticized for assuming gentle braking and acceleration that is not representative of actual real-world driving patterns [16]. In 1996, the EPA introduced a new driving cycle (US06) that includes harder acceleration and higher speeds, but this test is not used for fuel economy purposes. According to research in the fields of air and waste management [54, 53], strong correlations exist between emission levels and both speed and acceleration. There appears to be some controversy about which of acceleration and velocity dominates emissions. In our experiments, we compare measures of both from our drives with those of the driving schedules of FTP75 and US06.

We collected speed data from the GPS sensor. This GPS data is sampled once per second and is forwarded to the portal unfiltered. The speed data for FTP75 and US06 is also available in the Code of Federal Regulations [12]. Acceleration is derived from these two different speed data sets from the difference of each pair of consecutive speed readings. We compared the performance of two CarTel drivers to these Federal Standards (the first driver has logged about

2250 miles in CarTel; the second has logged about 1400 miles.)

Figure 7 compares the speeds and accelerations of the two CarTel users with the profiles of FTP75 and US06. The graphs show the cumulative distribution functions of speed and acceleration, where the accelerations are grouped into discrete ranges spanning 1 mph per second.[5] Interestingly, the acceleration distributions of both CarTel drivers are in fact more similar to each other and the FTP75 schedule than the more aggressive US06 schedule. User 1, in particular, has driving habits that are very well matched by FTP75.

In conducting this experiment, no new data had to be collected since historic GPS data was readily available in the database. This analysis was extremely easy and quick to conduct, illustrating the ease with which interesting real-world automotive analysis can be carried out using the CarTel platform.



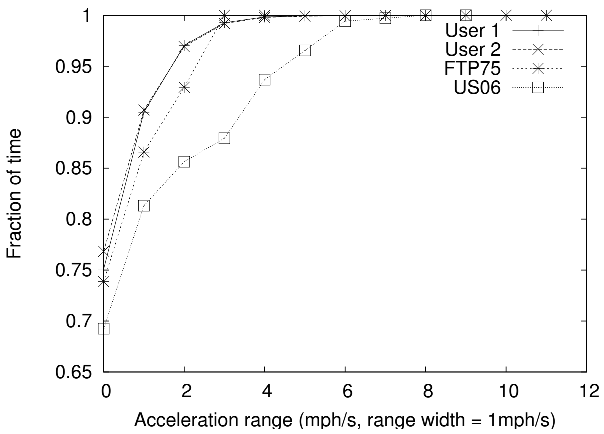


Figure 7. Comparison of speed and acceleration data from CarTel users and from federal test procedures FTP75 and US06.

6.3.2 *On-Board Diagnostic Data*

Our system collects a range of data regarding the emissions, engine status, and fuel consumption of cars over time via the OBD-II interface. We have logged about 60,000 records over the past few months, including data about troubleshooting codes, engine load, fuel consumption and pressure, engine RPMs, engine timing, air intake temperature, engine throttle position, and oxygen sensor status. We plan to use this data to study the performance of cars over the same roads on over many months to measure performance degradation and to validate the EPA fuel economy ratings.

[5]Accelerations exceeding 10 mph per second are filtered out as noise.

7 Implementation

The CarTel node software runs on the Linux 2.4.31 kernel. Currently, the node hardware is a Soekris net4801 that has a 586-class processor running 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, the Senao NL-2511MP with the Prism 2.5 chipset flashed with firmware v1.7.4. We have attached a 5.5 dBi gain omnidirectional rubber-duck antenna to each Wi-Fi card. Figure 8 shows this platform. Each node also includes a PostgreSQL database as part of ICEDB remote and a number of adapters for various sensors, including the Rayming TN200 GPS unit.

To power a CarTel node, we plug it directly into the cigarette lighter socket present in the vehicle. In most vehicles, these ports are active only when the ignition is on. Hence, our 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 the notion of a "trace" is easy to achieve. We have also experimented with various battery-operated CarTel nodes, both on cars and on bicycle rides.

To date, we have installed around ten CarTel nodes into vehicles owned by members of our research group; six nodes are in regular use today. Our plan over the next few months is to scale the system up to more than 20 nodes.

To manage software on remote nodes without requiring them to be removed from cars, we use two mechanisms. The primary mechanism is the adapter framework described in Section 3.1. For software upgrades of lower-level software, we use the Debian `dpkg` package management system and CafNet. To reduce the size of updates, CarTel is partitioned into a small number of packages for each of its major components, including CafNet, ICEDB, and the adapter subsystem. Remote nodes periodically check the portal for new versions of these packages and copy them via CafNet. Packages are installed as they arrive.

8 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 [52, 3, 38, 11]. In general, these are for monitoring or tracking applications characterized by low data rates and static deployments, in contrast to our focus on mobility, intermittent connectivity, and heterogeneous sensor data.

Mobile sensor networks. Recent work in the NIMS project [31, 33] and underwater sensing [55] 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 con-

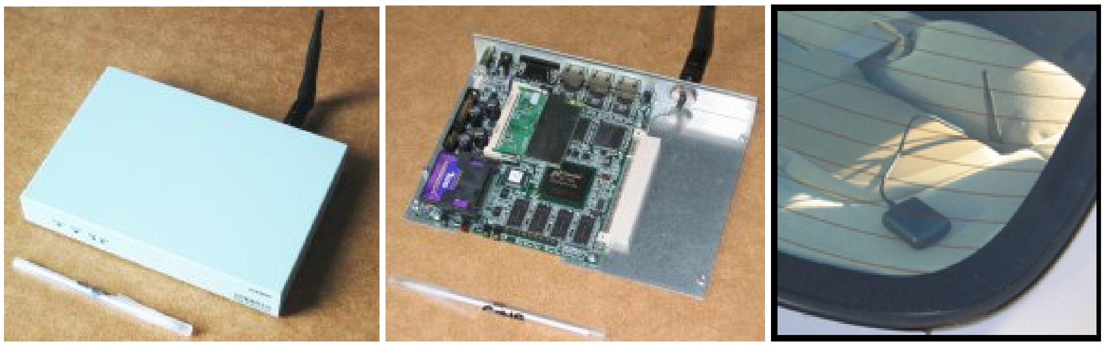


Figure 8. The CarTel node hardware.

trolled movement pattern to collect data about regions of interest. ZebraNet [30] exploits inherent mobility in the sensing deployment, as we do; by placing sensors on animals roaming in Africa, researchers have been able to observe their movement and socialization patterns. CarTel differs in that it provides a more general purpose platform, involves significantly higher local processing using a delay-tolerant query processor, integrates a visualization framework, and handles mobility at vehicular speeds.

Delay-tolerant networking. Many researchers have studied the potential throughput and energy benefits of muling [33, 22, 28, 5, 32]; 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 [19, 48, 29, 36, 43, 59, 24, 35, 47, 23]. These systems typically provide some mechanism for buffering data that applications want to send while disconnected, possibly with some *custody transfer* [19] 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 [22, 49], researchers have studied networks in which there 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 [40, 20]; we hope to integrate similar solutions into CarTel.

Query Processing. Many research projects have noted the need for in-network query processing [37, 58, 27] 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 [39, 9, 13]; 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 [6, 34, 14] rather that on what data to transmit, as in ICEDB.

The juggle operator [46] 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 [57] 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 [4] 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.

Road traffic monitoring. Using sensor networks for road traffic monitoring has recently become a hot topic. For example, in the TrafficLab project at Rutgers [17, 41], 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 [25] 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 [44] at UC Berkeley has investigated a number of issues related to smart transportation systems, including the use of sensor networks for on-road monitoring [18]. 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 [51]. 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 [26] and SmarTraveler [50] that aggregate information from various online traffic information sources to present a view of road speeds and hazards in urban areas. In addition, Dash Navigation [1] is developing a system that uses cars as floating probes to provide real-time traffic reports disseminated via a combination of peer-to-peer networks, Wi-Fi access points, and pager networks.

9 Conclusion

With hundreds of millions of automobiles (to which embedded computers can be attached) and over a billion 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. CarTel is a step towards a general-purpose mobile sensor computing system to realize this vision.

CarTel provides software to collect, process, deliver, and visualize data from sensors located on mobile devices to a portal. Applications specify the properties of the data they want using continuous queries, which are executed using a delay-tolerant continuous query processor, ICEDB, on the remote nodes. CarTel's networking stack, CafNet, delivers data between the portal and the remote nodes in the face of intermittent connectivity. Result streams from the continuous queries populate a relational database at the portal, which portal applications query to obtain results to analyze and process. The portal provides a geo-spatial visualization package for presenting information to users, as well as a management subsystem.

CarTel has been deployed on six cars, running on a small scale in several metropolitan areas in the US for over a year. Over this time, we have collected over 240 hours and 6200 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.

We plan to pursue several avenues of work in the near future, some of which we mention here. First, CarTel currently does not offer a way to aggregate information gathered across different users while also preserving privacy. Users do have password access, so unauthorized users cannot gain access to others' data, but it would be possible to make inferences about another user's location at a given time given the results of certain aggregate queries. Second, our data shows that predicting delays along routes *before* embarking on a trip is likely to have fairly high accuracy, and we are interested in using our data to develop a map-based route finding application that integrates delay information, and answers questions like "How late can I leave home tomorrow morning, and what route should I take, to make sure that I will be at the airport by 8am?" Third, we plan to process and analyze more data obtained from the OBD sensors, as mentioned in Section 6.3.2. Fourth, we plan to incorporate simple routing algorithms into CafNet using information about past movements of mules, and also incorporate the connectivity prediction model as an online algorithm. Fifth, on the portal side, we plan to develop techniques to efficiently answer questions about trace similarity and other geographic queries, while being resilient to noisy and missing data. Sixth, we plan to incorporate a larger number of continuous queries for various applications than we currently have, and also expand the number of applications to include other sensors (acoustics, pollution detectors, video streams, etc.). And last but not least, we plan to increase the number of users of our system in the coming months.

Acknowledgments

We thank Daniel Abadi, Philippe Bonnet (our shepherd), Lewis Girod, Kyle Jamieson, and the other SenSys reviewers for their many useful comments. This work was supported by the National Science Foundation under grants CNS-0205445, CNS-0520032, and CNS-0509261, and by the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan.

10 References

[1] Dash Navigation Inc. home page. `http://www.dash.net/`.

[2] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, N. Tatbul, Y. Xing, and S. Zdonik. Design issues for second generation stream processing engines. In *Proc. of the Conference for Innovative Database Research (CIDR)*, Asilomar, CA, Jan. 2005.

[3] R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, L. Krishnamurthy, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from the North Sea and a Semiconductor Plant. In *ACM SenSys*, 2005.

[4] L. Amsaleg, M. J. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *PDIS*, pages 208–219, 1996.

[5] N. Bansal and Z. Liu. Capacity, delay and mobility in wireless ad-hoc networks. In *INFOCOM*, 2003.

[6] D. Barbara and T. Imielinski. Sleepers and workaholics: caching strategies in mobile environments. In *SIGMOD*, pages 1–12, 1994.

[7] T. Brooke and J. Burrell. From ethnography to design in a vineyard. In *Proceeedings of the Design User Experiences (DUX) Conference*, June 2003.

[8] V. Bychkovsky, B. Hull, A. K. Miu, H. Balakrishnan, and S. Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *12th ACM MOBICOM Conf.*, Los Angeles, CA, September 2006.

[9] D. Carney, U. Centiemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams—A New Class of Data Management Applications. In *VLDB*, 2002.

[10] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, E. Travis, and H. Weiss. Interplanetary Internet (IPN): Architectural Definition. `http://www.ipnsig.org/reports/memo-ipnrg-arch-00.pdf`.

[11] A. Cerpa, J. Elson, D.Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. In *ACM SIGCOMM Workshop on Data Comm. in Latin America and the Caribbean*, 2001.

[12] Code of Federal Regulations. Title 40 Section 86 Subsection AA Appendix I.

[13] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[14] M. Cherniack, M. Franklin, and S. Zdonik. Expressing User Profiles for Data Recharging. *IEEE Personal Communications*, pages 32–38, Aug. 2001.

[15] D. Clark and D. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *ACM SIGCOMM*, pages 200–208, 1990.

[16] Emission Test Cycles: SFTP-US06. http://www.ietf.org/internet-drafts/draft-coene-sctp-multihome-04.txt, Apr. 2004.

[17] M. D. Dikaiakos, S. Iqbal, T. Nadeem, and L. Iftode. VITP: an information transfer protocol for vehicular computing. In *Workshop on Vehicular Ad Hoc Networks*, pages 30–39, 2005.

[18] S. C. Ergen, S. Y. Cheung, P. Varaiya, R. Kavaler, and A. Haoui. Wireless sensor networks for traffic monitoring (demo). In *IPSN*, 2005.

[19] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. ACM SIGCOMM*, pages 27–34, 2003.

[20] M. Ghanem, Y. Guo, J. Hassard, M. Osmond, and M. Richards. Sensor Grids for Air Pollution Monitoring. In *Proc. 3rd UK e-Science All Hands Meeting*, Nottingham, UK, Sept. 2004.

[21] Google Maps API. `http://www.google.com/apis/maps/`.

[22] D. Goodman, J. Borras, N. Mandayam, and R. Yates. Infostations: A new system model for data and messaging services. *In Proc. IEEE Vehicular Technology Conference*, pages 969–973, May 1997.

[23] K. Harras and K. Almeroth. Transport layer issues in delay tolerant mobile networks. In *IFIP Networking*, May 2006.

[24] M. Ho and K. Fall. Poster: Delay tolerant networking for sensor networks. In *SECON*, October 2004.

[25] E. Horvitz, J. Apacible, R. Sarin, and L. Liao. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Twenty-First Conference on Uncertainty in Artificial Intelligence*, July 2005.

[26] Inrix home page. `http://www.inrix.com/`.

[27] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *MOBICOM*, 2000.

[28] S. Jain, R. C. Shah, G. Borriello, W. Brunette, and S. Roy. Exploiting mobility for energy efficient data collection in sensor networks. In *WiOpt*, March 2004.

[29] D. Jea, A. A. Somasundara, and M. B. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *DCOSS*, pages 244–257, 2005.

[30] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proc. Architectural Support for Programming Languages and Operating Systems*, 2002.

[31] W. Kaiser, G. Pottie, M. Srivastava, G. Sukhatme, J. Villasenor, and D. Estrin. Networked Infomechanical Systems (NIMS) for Ambient Intelligence. *Ambient Intelligence*, 2004.

[32] A. Kansal, M. Rahimi, W. Kaiser, M. Srivastava, G. Pottie, and D. Estrin. Controlled Mobility for Sustainable Wireless Networks. In *IEEE SECON*, 2004.

[33] A. Kansal, A. A. Somasundara, D. Jea, M. B. Srivastava, and D. Estrin. Intelligent fluid infrastructure for embedded networking. In *USENIX MobiSys*, 2003.

[34] U. Kubach and K. Rothermel. Exploiting location information for infostation-based hoarding. In *MOBICOM*, pages 15–27, 2001.

[35] J. Lebrun, C.-N. Chuah, D. Ghosal, and M. Zhang. Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks. In *IEEE Vehicular Tech. Conf.*, pages 2289–2293, 2005.

[36] Q. Li and D. Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *ACM MOBICOM*, pages 44–55, 2000.

[37] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. In *proc. of OSDI*, 2002.

[38] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless Sensor Networks for Habitat Monitoring. In *WSNA*, 2002.

[39] R. Motwani, J. Widom, A. Arasu, B. Babcock, S.Babu, M. Data, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Approximation and Resource Management in a Data Stream Management System. In *CIDR*, 2003.

[40] Mobile Pollution Monitoring. `http://www.toolkit.equator.ecs.soton.ac.uk/infrastructure/repository/mobilepollutionmonitor/web/index.html`.

[41] T. Nadeem, S. Dashtinezhad, C. Liao, and L. Iftode. TrafficView: Traffic data dissemination using car-to-car communication. *MC2R*, 8(3):6–19, 2004.

[42] Executive summary of the conference on the prospect for miniaturization of mass spectrometry. Technical report, NSF, 2003. `http://www.nsf-mass-spec-mini-forum.umd.edu/final_report.html`.

[43] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *INFOCOM*, 2005.

[44] PATH Project. `http://www.path.berkeley.edu/`.

[45] PostgreSQL home page. `http://www.postgresql.org/`.

[46] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering for interactive data processing. In *The VLDB Journal*, pages 709–720, 1999.

[47] A. Seth, P. Darragh, S. Liang, Y. Lin, and S. Keshav. An Architecture for Tetherless Communication. In *DTN Workshop*, 2005.

[48] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data Mules: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *Proc. 1st IEEE SNPA Workshop*, 2003.

[49] T. Small and Z. J. Haas. The shared wireless infostation model: A new ad hoc networking paradigm (or where there is a whale, there is a way). In *MOBIHOC*, pages 233–244, 2003.

[50] SmartTraveler. `http://www.smartraveler.com`.

[51] B. Smith, H. Zhang, M. Fontaine, and M. Green. Cellphone probes as an ATMS tool. Technical Report STL-2003-01, Center for Transportation Studies, Univ. of Virginia, 2003. `http://ntl.bts.gov/card_view.cfm?docid=23431`.

[52] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *ACM SenSys*, pages 51–63, 2005.

[53] H.-Y. Tong, W.-T. Hung, and C. Chun-shun. On-road motor vehicle emissions and fuel consumption in urban driving conditions. *Journal of the Air and Waste Management Association*, 50:543–554, Apr. 2000.

[54] A. Unal, H. C. Frey, and N. M. Rouphail. Quantification of highway vehicle emissions hot spots based upon on-board measurements. *Jour. of the Air & Waste Management Assn.*, 54:130–140, Feb. 2004.

[55] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *ACM SenSys*, pages 154–165, 2005.

[56] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *USENIX OSDI 2004*, 2004.

[57] G. Wiederhold. Mediators in the architecture of future information systems. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 185–196. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[58] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[59] W. Zhao, M. H. Ammar, and E. W. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc*, pages 187–198, 2004.