# CarTel: A Distributed Mobile Sensor Computing System

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

## ABSTRACT

This paper describes *CarTel*, 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 (GPS, cameras, wireless network monitors, etc.) on cars collect data as users drive around.

CarTel's design meets three sets of challenges: (1) providing a simple programming interface, (2) handling intermittent and variable network connectivity, and (3) handling data heterogeneity (large amounts of data from a variety of sensors). CarTel nodes rely primarily on opportunistic Wi-Fi and 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, sub-sample, filter, and dynamically prioritize data. The portal and the mobile nodes use CarTel's delay-tolerant network stack, *CafNet*, to communicate with each other.

CarTel has been deployed on six cars, running on a small scale in Boston and Seattle for over a year. Three case studies—traffic delay monitoring, metropolitan Wi-Fi network monitoring, and automotive diagnostics—demonstrate the usefulness of the system.

## Categories and Subject Descriptors

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

## General Terms

Design, Measurement, Algorithms, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*SenSys'06,* November 1–3, 2006, Boulder, Colorado, USA.
Copyright 2006 ACM 1-59593-343-3/06/0011 ...$5.00.

## Keywords

Mobile networks, mobility, vehicular mobility, connectivity, delay tolerant databases, query processing, data visualization, network management

## 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 traffic conditions on 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 car's on-board sensors, which can help in preventive and comparative diagnostics.
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.

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 mon-

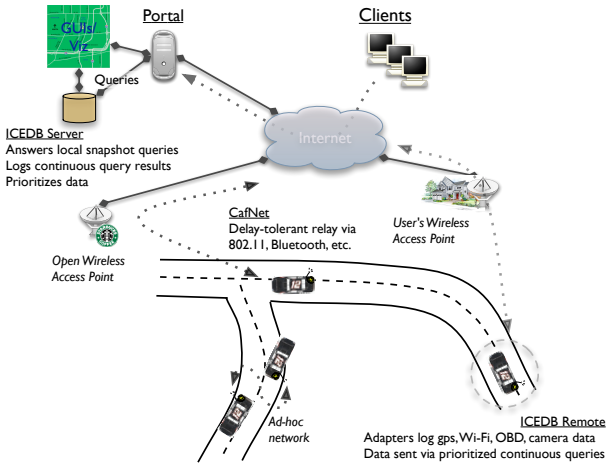


**Figure 1: The CarTel system architecture showing the different components of the platform. Cars collect data as they drive, and log them to their local ICEDB databases. As connectivity becomes available, data on cars is delivered via CafNet to the portal, where users can browse and query it via a visualization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

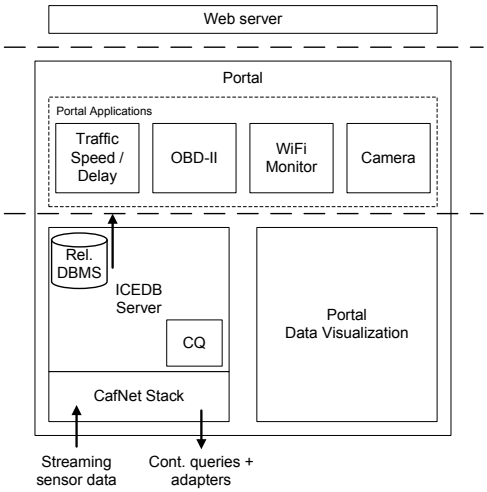


**Figure 2: The CarTel portal software architecture. Applications, such as those that measure traffic delays or request terrain photos from the camera, run snapshot queries against the relational DBMS in ICEDB to access data, displaying the results of those queries on geographic overlays using the data 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.**

itoring [38, 2, 50, 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 Wi-Fi and Bluetooth links. Wi-Fi access points run by users in homes and other locations 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 (several kilobytes per second) 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

[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].

[2]As in emerging metropolitan Wi-Fi deployments, wireless mesh networks, and Fon [20], our belief 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.

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*, a delay-tolerant continuous query processor, and *CafNet*, 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. Specifically, 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 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, 1] 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 sockets 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, particularly 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.

The rest of this paper details these contributions.

# 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 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 must be able to interpret 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.

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.[3] 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 specify the attributes produced by the sensor. In our current implementation, the type field must be a valid PostgreSQL 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. By using adapters to handle these sensors, ICEDB makes managing many data types across a large number of nodes easy.

## 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.[4] To support *continuous* queries in ICEDB, queries include a a *sample rate* specified by a new `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. Our system includes a number of other query language extensions to handle additional challenges in our intermittent, bandwidth-constrained environment.

The 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 via NTP (which uses GPS for synchronization when it is 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.)

One of those challenges is deciding what data to deliver at a particular time. Our observation is that the value of a piece of information is often application dependent (for example, one application may be interested in data that shows times when a car was speeding, whereas another application may be interested in times when a car was subject to unusual slowdowns.) For this reason, our focus is on providing a declarative way for applications to express precisely what data is important, and on building a prioritization framework in ICEDB that translates declarative specifications into a total ordering on local query results that need to be sent over the network.

In order to prioritize data for delivery, our 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.

As results are produced, they are stored into a named buffer. 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. We describe the how local and global prioritization work with this buffer in more detail in the next two sections.

### 3.2.1 Local Prioritization

Local prioritization is achieved by specifying a `DELIVERY ORDER BY` clause at the end of a query. This clause locally determines the order in which buffered result tuples are sent to the portal. Like a traditional SQL `ORDER BY` clause, `DELIVERY ORDER BY` can take attribute names to statically order by those columns. However, when prioritizing delivery for the intermittent network conditions to which CarTel applications are subject, many types of data would benefit from a more dynamic ordering that depends on the entire set of tuples. To enable this, `DELIVERY ORDER BY` can alternatively take the name of a user-defined function that takes as input the entire set of pending results and outputs a new score for each result. Because the `DELIVERY ORDER BY` function has direct

[3] If the executable crashes, it needs to be called again.

[4] 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.

access to the entire result set, the ordering of results can be dependent on the other results in the buffer, something 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 BY` 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

Global prioritization is enabled via the `SUMMARIZE AS` clause, which allows the user to specify a query that will compute a summary of the buffered result tuples. This summary is sent to the portal, where a user-specified function can compute a prioritization over this coarse-grained data and send this prioritization to the node. 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 query partitions the buffered result data into groups representing a low-resolution synopsis of the complete buffered data. On the server, the user specifies a function that orders the summary. 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 to the server.

Server prioritization is most important 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.[5] 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.

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.

[5] As in previous work such as DOA [54], making these identifiers a hash of a public key (and a random salt) would ease message authentication.

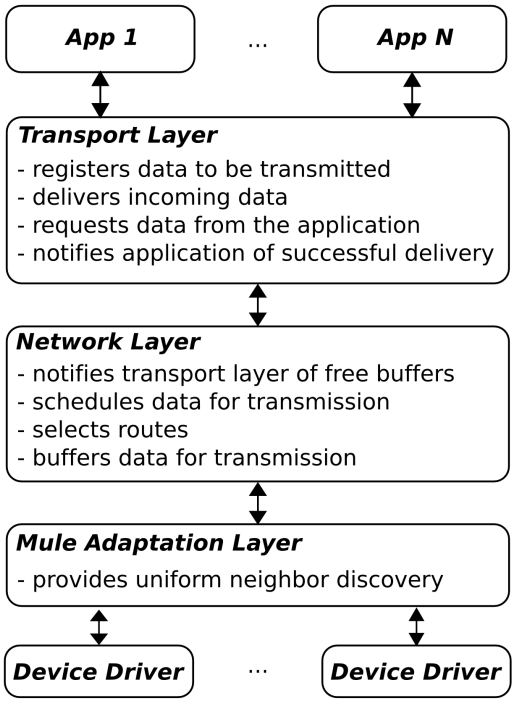


**Figure 3: The CafNet communication stack.**

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. In our implementation, the CTL is a library that applications link against, 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 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 implements peer discovery in the lowest layer (MAL) of its stack because those mechanisms are media-specific. For example, a Bluetooth-enabled cellphone presents itself as a single next-hop contact, and its discovery requires Bluetooth protocols. Discovering Wi-Fi connectivity at vehicular speeds requires fast scans and associations. 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 any host on the Internet is reachable along the link).

## 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. Moreover, in our implementation, the CNL runs in a different process from the application and the CTL, which implies that no process boundary crossings are required to deliver data to the MAL as soon as the MAL detects some network connectivity.

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. The advantage of this modification to the basic stack is that the CNL buffer hides any application delay in retrieving and packaging data prior to transmission, and achieves 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, and 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.).

Another optimization we have recently started investigation is an online algorithm using simple machine learning techniques to predict impending network connectivity. Our preliminary analysis of inter-connection arrival distributions and durations suggests that a practical algorithm is viable, but developing and evaluating the scheme are future work.

# 5. THE PORTAL

Users analyze sensor data in CarTel using web-based applications hosted within the portal environment, which was 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. Applications in the future will 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, ICEDB 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 using a range of styles, including those that repeatedly issue and withdraw continuous queries based on user input, as well as those applications 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 (a Google Map [22] in our current implementation) for a given trace.

Figure 4(a) shows the (anonymized) 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, for example, find all traces that correspond to his commute, doing so would be quite tedious if the data is sorted chronologically. To make mining 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 functionality 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 could easily 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 traffic 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 most of their time. Other examples of applications implemented on the portal include those that visualize engine 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, these types of applications still take advantage of the rest of the portal framework and issue both continuous and snapshot queries.

# 6. CASE STUDIES

This section presents four possible usage scenarios for CarTel system: road traffic monitoring, traffic simulation, wide-area Wi-Fi measurements, and automotive diagnostics, showing that CarTel is a platform that is flexible and versatile.

## 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. Somewhat to our surprise, we have observed users of CarTel quantitatively analyzing various alternative routes by comparing drive times of different routes at different times.

The CarTel system and the portal are well suited for this type of analysis. Using the GPS adapter and a continuous ICEDB query, the portal's traffic delay application can keep an accurate record of the routes a driver takes and the delays along those routes. Moreover, because this application shows quartile progress markers indicating the distance traveled after each successive 25% of elapsed time, users can 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 | Cnt |
|---|---|---|---|---|
| Freeway | 9.94 miles | 19:52 | 02:14 | 7 |
| City Streets | 9.83 miles | 29:34 | 02:19 | 9 |
| Frontage Road | 9.27 miles | 31:51 | 03:54 | 13 |

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 to mix and match routes to find an even faster commute.

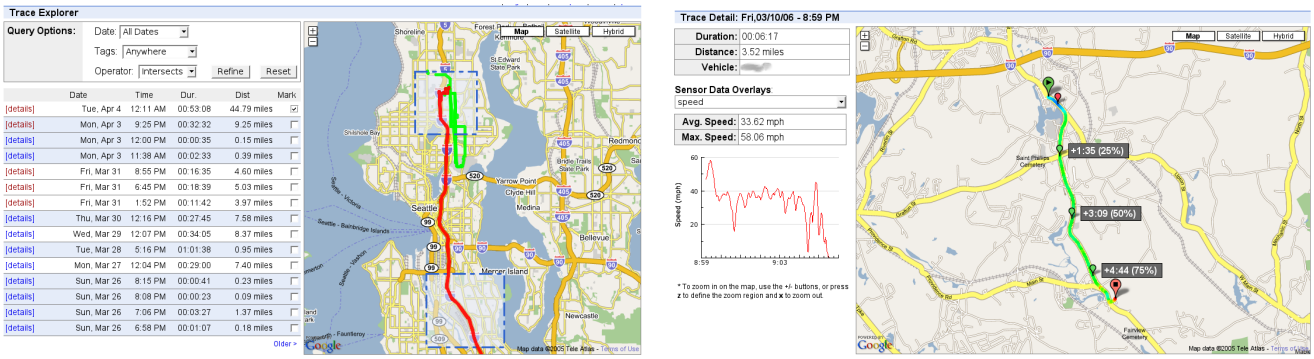


**Figure 4: The CarTel portal, showing a user (a) querying for traces corresponding to his commute and (b) viewing the speed overlay for one trace.**

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?"

### 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" in an area is sufficient to construct a reasonable route that minimizes traffic delays. It is precisely this observation that motivates an application on the portal that calculates and visualizes a user's traffic hot spots.

We calculate traffic hot spots from the GPS records collected once per second via the GPS adapter on each node. 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 every 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 for a given time of day, helping to isolate hot spots during rush hour or lunch time.

Figure 5 shows several 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 several intersections on back-roads display congestion that drivers may benefit from avoiding.

### 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."

CarTel makes it easy to collect the photographs needed to augment driving directions with the requisite landmarks. As mentioned

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

in Section 2, we have integrated a small camera into the sensor package deployed with each node installed in cars. Currently, the adapter for each camera is configured 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 FIFO order, but the large amount of data requires a smarter scheme. One such scheme is the "bisect" DELIVERY ORDER BY function of Section 3. We describe experiments showing this effect in Section 6.2.

Figure 6 shows a street-level view just prior to a turn on a user's route. By including this photograph with driving directions for this route, users will more easily catch this turn if they know to look for an Arco gas station on the left.

## 6.2 Traffic Simulation

To evaluate the performance of our querying and prioritization schemes for managing image and traffic data, we implemented a trace-driven data generator that simulates a number of cars simul-

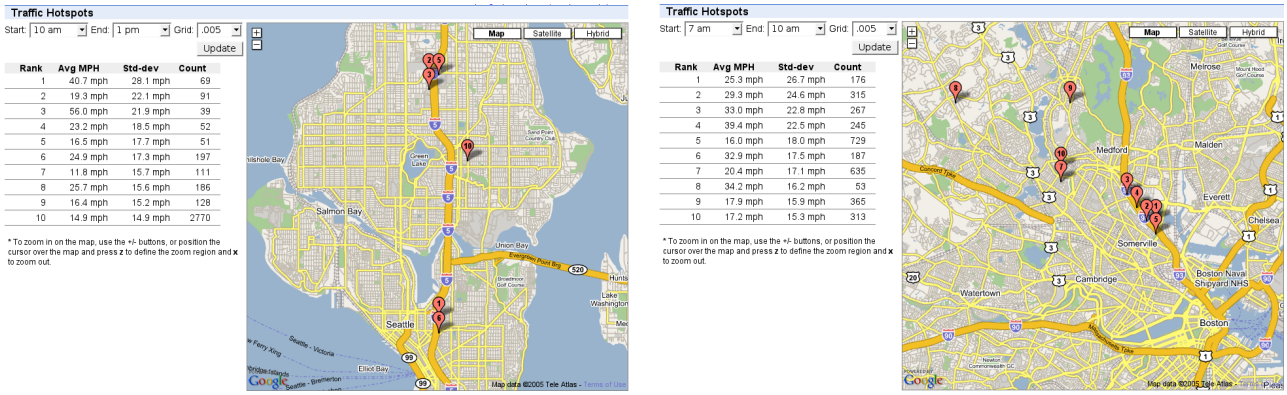


**Figure 5: The CarTel portal, showing users' traffic hot spots for the Seattle area (left) and the Boston area (right).**

taneously driving over the same set of roads and uploading camera images via CafNet. The traces these cars follow are chosen from real traces collected by CarTel users over the past several months, but condensed in time so that drives appear to happen concurrently. Concurrent drivers are important to demonstrate the effectiveness of server-based prioritization schemes.

Our simulation also uses a real distribution of observed wireless access points; we allow simulated cars to upload data based on the distribution of uploads that we were able to achieve driving past our own access points [8]. This distribution has mean of about 100 kilobytes per connection. In a typical half-hour urban drive in this environment, our traces suggest that cars see about 20 access points.

The results of our simulation are given in a related technical report [4]. In this report, we compare the FIFO approach to the local bisect algorithm described in Section 3 and to a summary-based method where the server prioritizes locations that are of most interest to user queries. User queries are generated according to one of several different distributions. Our results show that if cars have even a few tens of kilobytes of data per second to upload (for example, a camera image captured once per second), there is not enough bandwidth to upload all images. Hence, intelligent prioritization is essential if users are interested in rapid response to queries about particular locations or good overall coverage of the areas where drivers are driving.

## 6.3 Wide-area Wi-Fi Measurements

We have used an earlier version of CarTel, one without ICEDB, to collect over 290 "drive hours" of data about Wi-Fi connectivity in urban environments. The details of the study can be found in [8], below we summarize these results as relevant to the capabilities of the data collection system.

In addition to the GPS adapter used for traffic analysis, we have collected data from a Wi-Fi adapter. Our data collection program generated information for the following events:

- *Wi-Fi scans* are reports that list nearby access points(AP).
- *Wi-Fi associations* are attempts to establish a link-level connectivity with APs that can fail for a number of reasons
- *IP Address acquisitions* are attempts to establish IP address, either by means of DHCP or other methods.
- *Ping and upload statistics* are connectivity and throughput statistics inside and outside WLAN.

We have used this data to conduct a number of studies of the density of urban Wi-Fi networks and the feasibility of using open Wi-Fi access points for network access at vehicular speeds. In total, we have discovered 32,000 distinct APs. Association succeeded on 5,000 distinct APs. IP address acquisition succeeded on 2,000 distinct APs.

Using the data collected by means of CarTel we have evaluated association duration and duration of time between successive APs, and TCP upload throughput for "typical" commuting patterns. These numbers are summarized in the following table:

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

From this data we have concluded that it would indeed be feasible to use unplanned urban Wi-Fi networks for Internet connectivity. However, as discussed in Section 4, we believe that network interface would have to be modified to take advantage of this connectivity.

This study of connectivity illustrates the type of infrastructure monitoring that CarTel enables. In retrospect, using a full-fledged database and unified data uploading mechanism on CarTel units allowed us to make iterative corrections with only minor side-effects. In our experience, a lot of later measurements (such as retry counts and timing) were driven by earlier findings, so having a flexible data collection system was of a great benefit.

## 6.4 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.

### *6.4.1 Analyzing Driving Patterns*

The 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 [52, 51], strong correlations exist between both speed and

acceleration, and emission levels. 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 as reported by 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 CarTel users with the accelerations of FTP75 and US06. The graph shows the cumulative distribution functions of accelerations, where the accelerations are grouped into discrete ranges spanning 1 mph per second.[6] 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 experiments can be carried out using the CarTel platform.

### 6.4.2 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 are currently exploring the use of this data in a number of contexts, including studies of the performance of cars over the same roads on over many months to measure performance degradation as well as validation of EPA fuel economy ratings.

## 7. IMPLEMENTATION AND SYSTEM MANAGEMENT

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 omni-directional rubber-duck antenna to each Wi-Fi card. Figure 8 shows this platform. Each node also includes a PostgreSQL database 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. In parallel, we have also experimented with various battery-operated CarTel nodes, both on cars and on bicycle rides.

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

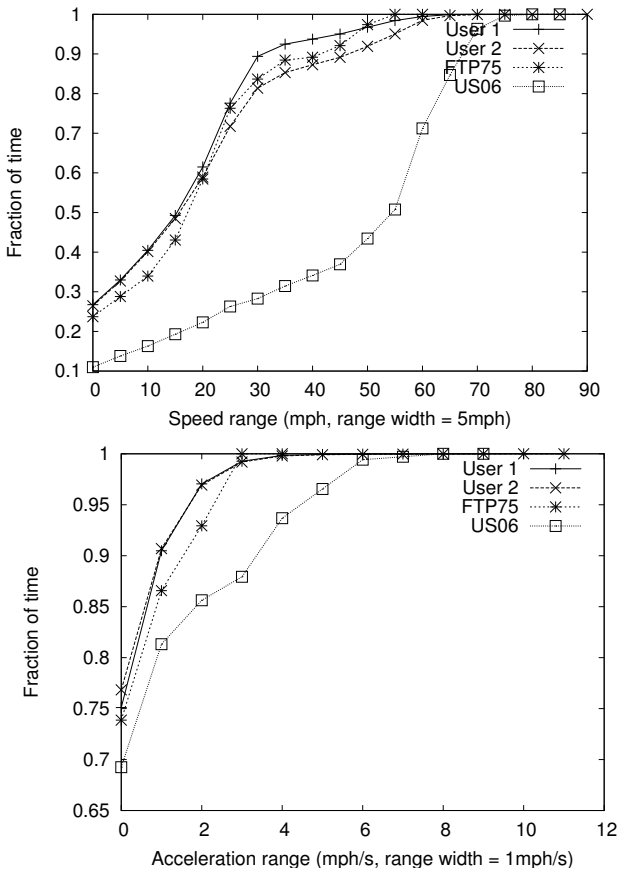


**Figure 7: Comparison of speed and acceleration data from CarTel users and from federal test procedures FTP75 and US06.**

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 [50, 2, 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 [53] has focused on using mobility when it is not feasible to build a dense network of fixed sensors,

**Figure 8: The CarTel node hardware.**

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 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, 23, 29, 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, 46, 30, 36, 43, 56, 25, 35, 45, 24]. 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, but our current focus has been on issues that arise in single-hop networks. In particular, we have concentrated on API design using callbacks to handle dynamic priorities, and on predicting connectivity (even when it is not on known trajectories) to properly size network-layer buffers. Though some related work has looked at statistical models for predicting connectivity [28] and buffer sizes [46], most work has not looked at the relationship between the network protocols and disconnection-aware applications as we have.

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 [23, 47], 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, 21]; we hope to integrate similar solutions into CarTel.

**Query Processing.** Many research projects have noted the need for in-network query processing [37, 55, 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. In CarTel, our query propagation, prioritization, and data collection mechanisms are all specially tailored to the high-rate, mobile environment in which our software operates.

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.

Raman *et al.* 's work on the juggle operator [**?**] as a part of the CONTROL project provides a method for allowing users to prioritze 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 `PRIORTIZE` clause to automatically assign priorities to tuples as they are produced.

Amsaleg *et al.* presented query scrambling [3] 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.

In JamBayes [26] researches developed 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 comple-

mentary 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 [49]. 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 SmarTraveler [48] that aggregate information from various online traffic information sources to present a view of road speeds and hazards in urban areas. These sources typically only provide information about conditions on major highways.

# 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 page, which uses CarTel's geo-spatial visualization interface. Our experience, albeit 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 6.4.2. 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 database techniques to efficiently answer questions about trace similarity and other geographic queries, while being resilient to noisy and missing data. Our current approach has been to process much of this data manually, which clearly does not scale. 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.)

Last but not least, we plan to increase the number of users of our system to about 20 within the next two months, and further by the end of summer—several of our colleagues have been keen to obtain nodes for their cars to improve their commute, provide automotive diagnostics, or monitor their teenage kids' driving habits!

# 10. REFERENCES


[1] 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.

[2] 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.

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

[4] Reference anonymized due to double-blind reviewing. Technical report.

[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.

[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] FON home page. `http://en.fon.com/`.

[21] 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.

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

[23] 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.

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

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

[26] 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.

[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, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *ACM SIGCOMM*, 2004.

[29] 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.

[30] 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.

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

[46] 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.

[47] 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.

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

[49] 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`.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

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

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

[56] 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.