

**Design, Development, and Validation of a Remotely Reconfigurable Vehicle Telemetry System for
Consumer and Government Applications**

by

Joshua Eric Siegel

Submitted to the Department of Mechanical Engineering in Partial
Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering
at the

Massachusetts Institute of Technology

June 2011

Copyright © 2011 Joshua E. Siegel. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis and to grant
others the right to do so.

Signature of Author: _____

Department of Mechanical Engineering
May 3, 2011

Certified by: _____

Sanjay E. Sarma
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by:

John H. Lienhard V
Samuel C. Collins Professor of Mechanical Engineering
Undergraduate Officer

[THIS PAGE INTENTIONALLY LEFT BLANK]

Design, Development, and Validation of a Remotely Reconfigurable Vehicle Telemetry System for Consumer and Government Applications

by

Joshua E. Siegel

Submitted to the Department of Mechanical Engineering on May 3, 2011
in Partial Fulfillment of the Requirements for the Degree of Bachelor of
Science in Mechanical Engineering

Abstract

This thesis explores the design and development of a cost-effective, easy-to-use system for remotely monitoring vehicle performance and drivers' habits, with the aim of collecting data for vehicle characterization and traffic shaping.

Vehicular congestion and concerns over fuel reserves, pollution, and carbon emissions have recently emerged as prominent sociopolitical concerns. These problems are formidable, but could be addressed more fruitfully with better information about vehicles and drivers habits, leading to policies such as vehicle-specific congestion charging or an odometer-based road tax.

Despite the proliferation of sensors in cars, data is often hidden due to the antiquated nature of the federally-required On-Board Diagnostics (OBD). Systems to log and process such data exist, but no well known reconfigurable systems augment OBD with additional sensor data and transmit it over a cellular network.

This thesis proposes a system wherein vehicles become distributed sensors, each transmitting a rich supply of information. The standardization of OBD and decreasing cost of bandwidth make now an opportune time to develop a real-time logging system. Inexpensive processors make it possible to provide privacy through onboard calculation, obfuscating much personally-identifiable data.

This document discusses the planning process, experimental configurations of hardware and software, results, and conclusions associated with the development of a cellular diagnostic system capable of supporting an “app” model for information feedback. I present a Bluetooth-OBD logger, a cellular logger, and a web interface capable of representing live and historical data from vehicles, including example applications for calculating congestion pricing.

This project proves the feasibility of capturing data using a remotely reconfigurable controller area network (CAN) to general packet radio service (GPRS) interpreter, visualizing the information in real-time, and writing applications to make use of the incoming data. The hardware and software were proven successful in meeting the goals set for the project.

The hardware proved robust, gathering data without issue for hundreds of miles. The sample data demonstrated low bandwidth use, identified network weaknesses, and pointed out issues with the currently-legislated OBD standard.

This thesis closes by exploring future possibilities suggested by the development of this system, including wireless odometry and next-generation OBD.

Thesis Supervisor: Sanjay E. Sarma
Title: Professor of Mechanical Engineering

Biographical Note

Josh Siegel was born in 1988 in Detroit, Michigan where, from an early age, he was exposed to the automotive industry and was immersed in car culture. As a high school student, he developed autonomous vehicles and restored classic cars, challenges which introduced him to the nuances of automotive hardware and software.

Since coming to MIT, he has held leadership positions in the Electric Vehicle Team and the Entrepreneurs Club, where he focused on advancing electric vehicle efficiency and commercializing low-cost microelectromechanical systems (MEMS) inertial navigation systems. Today, he combines his passion for success with his diverse skill-set in order to develop transformative technologies for vehicles and low-cost rapid prototyping machines.

Acknowledgments

I wish to thank Professor Sanjay Sarma for providing me with guidance and support throughout the evolution of this project, as well as the MIT Mechanical Engineering Department for providing me with the resources and knowledge necessary to complete such a complex task. Ivan Sergeev provided invaluable help developing the real-time software component, as well as error-checking the various hardware revisions.

Finally, I wish to thank my family and friends for their support during the research and writing process – in particular, Maria Frendberg for her gracious help debugging troublesome software, and Yafim Landa for his encouragement and advice while developing the logging server and web interface.

Table of Contents

Abstract.....	3
Biographical Note.....	4
Acknowledgments.....	5
Table of Contents.....	6
List of Figures.....	8
1 Introduction.....	9
1.1 The Need for Better Vehicle Informatics.....	9
1.2 Design Motivation.....	11
1.3 Background of OBD.....	12
1.4 Why Now?.....	13
1.5 Literature Review.....	14
1.5.1 CarTel.....	15
1.5.2 Fleet Tracking at University of Moratuwa.....	16
1.5.3 GPRS and CAN Bus Fault Monitoring.....	16
1.5.4 Real-Time Tracking Management System.....	17
1.5.5 Wireless OBDII Scanner.....	17
1.5.6 Accelerometer-based Diagnostics.....	18
1.5.7 Other Prior Art.....	19
1.6 Proposed Solution.....	19
1.6.1 Primary Goals.....	19
1.6.2 Secondary Goals.....	19
2 Design Methodology.....	21
2.1 Assumptions.....	22
2.2 Milestones.....	22
2.3 Deliverables.....	22
2.4 Outside Sources.....	23
3 Experiment 1 – Bluetooth Dongle & Symbian OS Cell Phone.....	24
3.1 Brief Introduction.....	24
3.2 Experimental Procedure/Hardware Overview.....	24
3.2.1 OBDII Interface - ELM 327.....	25
3.2.2 Bluetooth Transceiver – CSR BC417 and Roving Networks RN41.....	25
3.2.3 Nokia N95.....	25
3.3 Methodology.....	26

3.4	Results & Discussion.....	28
4	Experiment Revision 2 (OBD Reader with Modem, MySQL backend).....	31
4.1	Brief Introduction	31
4.2	Experimental Procedure (Methods and Materials).....	31
4.2.1	Processor - LPC2129	32
4.2.2	Modem & GPS - GM862	32
5	CAN Transceiver – TI SN65HVD232	34
5.1.1	Embedded Software.....	35
5.1.2	Server Software	36
5.1.3	Data Visualizer	37
5.2	Results & Discussion.....	38
6	Experiment Revision 3 (Adding Two-way Communication).....	39
6.1	Brief Introduction	39
6.2	Experimental Procedure	39
6.2.1	Command Architecture.....	39
6.3	Results and Discussion	42
7	Experiment Revision 4 (Writing a Login System and Client Application).....	43
7.1	Brief Introduction	43
7.2	Experimental Procedure	43
7.3	Results & Discussion.....	46
7.4	Summary.....	53
8	Experiment Revision 5 (Embedded Software, Server Revision #2)	55
8.1	Brief Introduction	55
8.2	Experimental Procedure	55
8.3	Results & Discussion.....	56
9	Overall Results and Conclusions.....	58
10	Future Work.....	60
11	Conclusion.....	66
12	Literature Cited.....	67

List of Figures

<i>Figure 1: Commercial OBD scanner</i>	<i>15</i>
<i>Figure 2: Nokia N95 smart phone.....</i>	<i>26</i>
<i>Figure 3: Custom Bluetooth-OBDII transceiver</i>	<i>28</i>
<i>Figure 4: Assembled logging device</i>	<i>35</i>
<i>Figure 5: Screenshot of Zope tracker.....</i>	<i>37</i>
<i>Figure 6: Header/parameter map</i>	<i>41</i>
<i>Figure 7: Main page for login system</i>	<i>46</i>
<i>Figure 8: Administration page.....</i>	<i>47</i>
<i>Figure 9: My Account page.....</i>	<i>48</i>
<i>Figure 10: Edit Account page,</i>	<i>49</i>
<i>Figure 11: Client page</i>	<i>50</i>
<i>Figure 12: Government page</i>	<i>51</i>
<i>Figure 13: Polygon editing page</i>	<i>52</i>
<i>Figure 14 - Table for editing rates.....</i>	<i>53</i>
<i>Figure 15: Bill generation page.....</i>	<i>53</i>
<i>Figure 16: Picture of logger enclosure.....</i>	<i>56</i>
<i>Figure 17 - O2 sensor waveforms</i>	<i>62</i>

1 Introduction

1.1 The Need for Better Vehicle Informatics

There is a dire need for further research in vehicle informatics. Population growth, urbanization, and concerns regarding the costs of fuel and sustainability have led to an ever-heightened demand for automotive efficiency. Recent stories questioning automakers' design processes and quality control have raised the average consumer's awareness of the complexity of automobiles, and of emphatic concerns regarding vehicle failure. An antiquated understanding of the information that vehicles might provide compromises our best response to the demand for greater efficiency and safety. While older vehicles lack electronic sensors entirely, newer vehicles boast hundreds or even thousands of sensors measuring emissions, comfort, and safety, but their data are often obstructed from consumer access. Despite their federal mandating in 1996, the full value potential of on-board diagnostics has not yet been realized. Many vehicles presently on the road are capable of gathering data to address problems, but barriers to diagnostic data access prevent the use of this information.

Increasing traffic congestion is one problem that can be addressed by better use of OBD data. The Texas Transportation Institute's 2010 Urban Mobility Report noted that, in 2009, congestion caused an annual average delay of 34 hours per American and wasted 28 gallons of fuel, for a combined opportunity cost of \$808 per commuter (Texas Transportation Institute 2010). Governments have proposed fixed-rate congestion charges to reduce the amount of traffic entering a given area, but high prices may be harmful to the local economy, while low fixed rates may prove little more than an inconvenience to drivers, bypassing their intended impact. Additionally, resultant fuel economy and CO₂ emissions are a concern for local, state, and federal governments. The negative externalities of excessive single-occupant transit are detrimental to property values and overall citizen happiness, as it portends additional noise, worsened air quality, and higher transit times.

Some cities are exploring variable congestion pricing to load balance the road network. These proposals examine vehicle type, time, location, and trip duration in order to alter the fee associated with

traveling via a certain route, thus incentivizing the use of lesser-travelled roads and shaping the traffic patterns in real-time. These approaches address the problem, but do not make full use of the information that might be made more accessible to planning authorities. A better solution is necessary – one which allows congestion and pollution to be taxed, taking into consideration the particular vehicle being driven, its condition, and road demand. This avoids the unfairness of a flat-rate fee, and instead rewards citizens for driving cleaner vehicles more efficiently, better addressing citizens' interests in fuel economy and high environmental quality, including appropriate air quality. Currently, few wireless odometry systems have the capacity to measure vehicle characteristics, and the systems with this capacity do not optimally use the data as a factor in their calculation. A real-time telemetry system would facilitate such methods of congestion pricing by combining wireless odometry with live diagnostic data.

Diminishing fossil fuel reserves and geopolitical unease have worsened concern over energy dependency. Government bodies, consumers, and industry are advocating for improved fuel economy. This has caused a paradigm shift in thinking about automobiles, moving from a "combustion world" to hopes for a hybrid, electric, and fuel-cell-driven society. For their potential, these technologies also bring new complexities, among them the specter of inadequate diagnostic techniques. Additionally, psychological issues arise, as in the case of "range anxiety" – the fear that one's vehicle might run out of energy before reaching a suitable recharging location. Unlike a gasoline or diesel powered vehicle, when driving an all-electric vehicle, one cannot simply walk to the corner and pick up a can of electricity.

The focus on efficiency has spawned new research opportunities. Consumers now look for greener products; there is a significant interest not only in clean tech, but also in real-time metering of a given technology's cleanliness. Current vehicles do not provide much feedback, with the exception of Ford's SmartGauge technology which does not directly afford the driver an understanding of what control he or she has over the efficiency of the vehicle. Real-time diagnostics would provide consumers, manufacturers, and government the ability to monitor energy consumption and efficiency in ways never before possible. Building a system with this in mind would help address associated concerns, such as how to extend diagnostics to new types of vehicles or how to quell vehicle owners' fears. An intelligent

system could address issues with manufacturing processes and quality assurance, as illustrated by the Toyota accelerator pedal controversy. While it is not clear who was at fault, it is clear that consumers are more aware than ever of the complexity of their vehicles. In many situations, a well-executed monitoring system could help drivers and monitoring systems identify potential vehicular failures before a human ever could.

In the past, the gathering of information required engineers to make assumptions and to create models based on data that predicated upon a small set of observations, as opposed to real-time informatics making use of broader-based monitoring. We are now reaching the limit of prevailing models. Instead of taking surveys and recording only what is available within a single vehicle, I propose that the vehicles themselves be used as data points with a broader systems context – widely distributed sensors, each with a rich supply of information. The existence of a standardized protocol for vehicle communication, combined with the plummeting cost of bandwidth and increasing speed of cellular networks makes this an opportune time to begin developing a wide-area, vehicular data collection network. With the data stored in a central database, we may then analyze and address the problems of our time with a much richer, reality-based perspective.

1.2 Design Motivation

Lack of information is a significant problem, but one that may be readily solved. Under the supervision of Professor Sanjay Sarma, I proposed the design of a system capable of logging data in real-time and its uploading to a central server for analysis, diagnostics, and feedback. The system would be remotely reconfigurable, allowing for accurate, real-time telemetry, gathering only necessary data. The system would facilitate intelligent observation of trends in individual vehicles and the larger vehicle traffic system infrastructure.

This concept relies on the fact that newer vehicles have computers capable of accessing real-time sensor data, and that cellular networks provide an excellent way of transmitting this data to a server for

rapid, off-vehicle processing. Cellular networks are easily interfaced with a vehicle through serial ports, over Bluetooth, or via a variety of hobbyist-oriented modems.

Ultimately, every vehicle equipped with this setup would provide a dynamic data set for analysis. These live data would provide the basis for web “apps” for users demanding easy-to-understand diagnostic data, taking the newly-available information and turning it into a planning aid. The system would provide metrics on energy efficiency, traffic patterns, vehicle failure modes, and vehicular emissions.

Making so much information about drivers' habits available highlights associated privacy concerns. Location tracking is of particular importance, and a concern that plagues extant systems with access to similar information (for example, EZPass). With intelligent data manipulation and onboard data processing, a remotely reconfigurable device may operate as a “thick client,” processing data onboard and only associating critical data – such as distance travelled – with a particular client identifier and checksum for verification purposes. In response to potential privacy concerns, other data may be completely eschewed or anonymized. As an example of how this would be implemented, in applications where gathering certain data is critical, such as for government odometry, well-designed hardware could offer users the choice of a “thick client” (reporting only total miles per month at a high fixed rate) or a “thin client” (using additional data to charge at a rate per gallon of fuel consumed). The thin client user would reap additional benefits by opting to share their data, making it accessible to additional applications for visualization and processing. This research attempts to consider relevant privacy and confidentiality concerns.

1.3 Background of OBD

On Board Diagnostics Version II is a computer system built into all light- and medium-duty vehicles sold in the United States from 1996 onwards. OBD systems are designed to monitor the performance of some of an engine's major components, including those responsible for controlling emissions, though automobile manufacturers often include proprietary information (called EOBD2, for Enhanced On Board Diagnostics 2) over the standardized OBDII connector. The system reports and

identifies malfunctions of critical systems, and also reports live readings from selected sensors. OBDII supports limited storage for “freeze frame” events when a failure is first detected, providing an understanding of why the failure may have occurred.

OBDII was developed by the State of California Air Resources Board (CARB) and the United States Environmental Protection Agency (EPA) and is standardized in the US by the Society for Automotive Engineers (SAE) and worldwide (as E-OBD) by the International Organization for Standardization (ISO). Originally multi-protocol, as of 2008, all vehicles sold in the U.S. were required to utilize a protocol called CAN Bus to reduce the complexity of interfacing with a vehicle.

OBD is not a new field of study, but poor documentation, complicated interface requirements and an industry stranglehold prevent major advances. Recently, hobbyists have begun developing products to log vehicle data to memory cards. These devices are useful in that they make the data more accessible to the average person, but there are few obvious uses for the data. This project aims to take similar data and make it "easy to digest."

1.4 Why Now?

As “hackers” dig deeper into OBD and related communications, and as cellular connectivity becomes more accessible from the design standpoint, this proposed system becomes more feasible, and an ideal way of making use of the information that OBD provides.

As vehicles move away from gasoline and the per-gallon road use tax currently employed, governments are seeking a new way to shape constituent behavior as well as to bill vehicle owners for the social cost of their negative externalities such as traffic and pollution. This system will facilitate new means of billing.

By utilizing a broader base of indicators through intelligent analysis, consumers will appreciate the feedback their car will provide, predicting failures and providing feedback and learning experiences that encourage more fuel efficient driving habits. In times of economic slowdown, even small savings provide significant perceived and actual advantages.

Auto manufacturers seek new diagnostic solutions to accompany the introduction of electric vehicles and hybrids. This research exposes flaws in OBDII, provides stopgap measures, and identifies deficiencies to be addressed in OBDIII.

1.5 Literature Review

Myriad OBD systems exist, but none address the problem of obtaining comprehensive vehicle information. The majority focus on single aspects of data collection, monitoring performance or efficiency but typically not both, and no commercially available system presently supports predictive diagnostics. Some systems simply report location and ignore vehicle status, while others report only vehicle status. Many loggers require a physical connection to a computer for analysis.

These loggers serve many purposes, from fleet management to monitoring teenage drivers, but they focus on a core competency and do not bring new information to the field. No broadly distributed platforms measure engine parameters as the basis for fee calculation, and none provide real-time feedback to improve green driving. None of these systems augments OBDII using historical data, and even among the few diagnostic tools that make use of a cellular connection, not one combines data from various vehicles to determine data trends. Finally, other systems do not allow for the remote reconfiguration of the unit, limiting their applicability to an “app” use model. Before beginning my research, I explored several academic studies on remote vehicle tracking and diagnostic, summarized in the following sections.

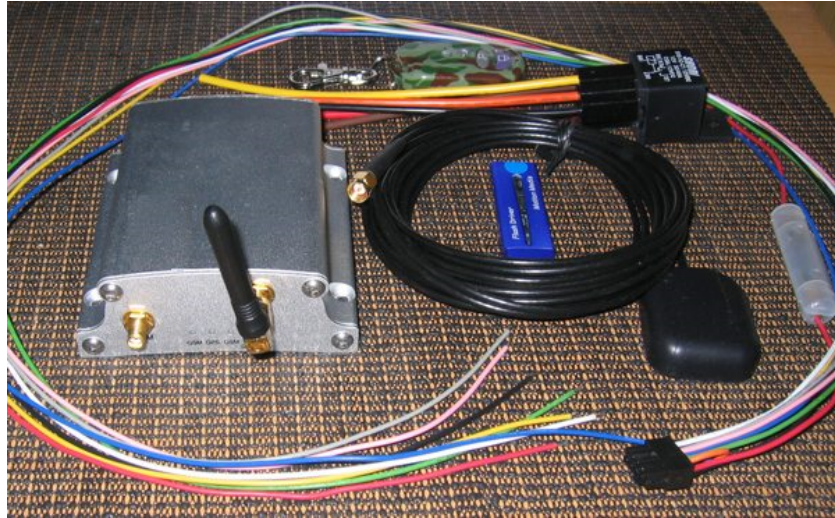


Figure 1: A cellular OBD scanner for speed tracking in fleet vehicles¹

1.5.1 CarTel

CarTel, an MIT Computer Science project, created one such networked vehicular interface. CarTel collects, processes, and visualizes data from mobile sensing units, computing data locally and delivering processed results to a central server using a thick client model. CarTel, however, uses “opportunistic” connections – requiring WiFi networks or cell phones in close proximity to operate. Bret Hull’s 2006 paper in SenSys discusses this in more detail, exploring the networking applications of the system (Hull 2006).

CarTel has uses in traffic mitigation, road surface monitoring, and hazard detection, but lacks a means of providing live feedback with a high guaranteed uptime. While such a project would work for some of the same “apps” as a reconfigurable cellular model, including environmental and infrastructure monitoring, it falls short when it comes to applications requiring high reliability and real-time network access, such as dynamic pricing schemes and mileage based road taxation. The CarTel project has thus far explored monitoring performance and fuel economy claims, and has not attempted to enhance the OBD protocol through novel technologies such as predictive diagnostics, nor has it attempted to identify weaknesses in the existing set of standards. The logger designed for this project is an excellent example of

¹ Retrieved from <http://singapore16.tradenote.net/product/125822-GPRS---GPS.html>

novel networking techniques and data collection, but is limited in its applicability to vehicle informatics by its intermittent connectivity and reliance on archaic protocols.

1.5.2 Fleet Tracking at University of Moratuwa

The University of Moratuwa combined Geographical Information Systems (GIS), Global Positioning Systems (GPS), and General Packet Radio Service (GPRS) to create a fleet tracking system. This design project provided real-time feedback and visualization for a large number of vehicles. Vehicles equipped with tracking devices streamed data over GPRS to a central database where location and use statistics could be calculated. This system focused more on the enterprise aspect, enabling users to track driver time and generate reports on resource utilization (Medagama et al. 2008).

This system demonstrates the utility of tracking systems for fleet management or cost-conscious consumers, but ignores OBD in its entirety, focusing on time management rather than fuel efficiency or vehicle maintenance needs. As with its commercial counterparts, this system is not reconfigurable on the client side, though the server side's use of GIS makes it exceedingly flexible for defining regions of interest. This paper assumed that fleet owners would install these devices on their own vehicles and that compliance would be a non-issue; however, in real-world applications, the hardware would need layers or protection from being disconnected to protect the privacy of the drivers.

1.5.3 GPRS and CAN Bus Fault Monitoring

At the Key Laboratory of Data Storage Systems at the Huazhong University of Science and Technology, researchers developed a fault monitoring and detection system combining the Controller Area Network (CAN) Bus for error detection, and a GPRS modem for remote diagnostic purposes in trucks. This project utilized the Society of Automotive Engineers (SAE) J1939 standard for vehicle communication and attempted to utilize the cellular network as a long-range CAN transceiver (Feng et al.). Most of the research focused on reliability and throughput issues using GPRS (171.2kbps peak) and CAN (250kbps). From a hardware standpoint, this project is unique in that it affords full use of the CAN

Bus over a cellular network. The hardware was, in effect, reconfigurable and live-streaming. Researchers found that for short bursts of data, the device operated well. With higher speed networks, it should be possible to achieve longer data streams without interruption and to incorporate readings from additional sensors into the transmission. This research did not explore any applications beyond utilizing GPRS for remote CAN access.

1.5.4 Real-Time Tracking Management System

The University of Technology North Bangkok explored real-time fleet tracking as a method to reduce fuel consumption. Their research proposed a GPS tracking system designed from commodity hardware, open source software, and a web interface using Google Maps. The research group was successful in creating a reliable GPRS tracker using non-blocking sockets to create a robust multi-client network of remote GPS devices for vehicle tracking (Chadil et al. 2008). However, their hardware did not explore interfacing with OBD, though it did prove that many vehicles could be tracked simultaneously through intelligent software design. Other papers, such as those published in the *World Academy of Science, Engineering and Technology Journal*, describe similar systems employing the use of Short Message Service (SMS) messages as two-way communication is unnecessary for fixed-rate position-only tracking (Muruganandham 2010).

1.5.5 Wireless OBDII Scanner

Mahoney and Keenan of Worcester Polytechnic Institute describe the development of a wireless OBD scan tool capable of connecting to a computer using a serial port interface. This paper explores the commercial applications for a short-range wireless device built into a vehicle. The proposed solution was to provide an in-vehicle “base unit” that would do no calculation or reporting, but rather serve as a logic translator. A professional technician would have an intelligent receiver to scan the vehicle’s device remotely. This would prevent shrinkage of expensive tools, and would allow for the development of a more intelligent back-end system for diagnostics, logging codes to a computer for processing and data

lookup. The group also explored the use of similar devices for improving the efficiency of remission testing. In testing, the group was successful in producing a set of devices utilizing the ZigBee protocol for short range data transmission (Mahoney and Keenan 2008). This is similar to the GPRS and Can Bus Fault Monitoring system in that it does no processing onboard, but the platform offers great potential for data collection and analysis.

1.5.6 Accelerometer-based Diagnostics

Beyond OBD loggers, some universities have done research on techniques for utilizing external sensors on vehicle to provide diagnostic data. The Technical University of Opole is one such university. Their research was based on the fact that not all vehicles have OBD, and in all likelihood, older vehicles without the system are the vehicles most likely to have mechanical problems. This research explored using an accelerometer and a microprocessor mounted to the vehicle to analyze engine power, powertrain health, and braking efficacy. Brohl and Mamala's paper in *Ultragarsas* explores the use of various filtering methods to rid the accelerometer of noise, and the effect of orientation on sensing engine conditions. Their experiment was able to identify engine speed using only the accelerometer, maintaining its accuracy under acceleration or braking (Brol and Mamala 2006). Such a system could prove useful in determining problems with an engine, such as misfires or bad engine mounts, by looking for anomalies after processing the accelerometer with a Fast Fourier Transformation. With a fast enough sampling rate, the accelerometer may even be able to identify specific vehicles.

However, this system is located entirely on-vehicle and is not networked, making the information useful in a laboratory or diagnostic setting only. Utilizing this data in real-time would provide rich datasets for better understanding automotive failure modes, and how specific user behaviors alter vehicle performance. As accelerometers become better (with higher signal to noise ratios, faster sampling rates and better resolution), these data potentially might yield rich information about the drivability of a vehicle without needing to be plugged into the vehicle's network or the need to be programmed with vehicle-specific information.

1.5.7 Other Prior Art

Many other OBD projects exist, from creating OBD-based gauge clusters in vehicles (Grutzmacher et al. 2003) to Global Positioning System (GPS) vehicle tracking and fleet management various Customer Relationship Management (CRM) systems (Mohan and Balan) and remote bus tracking (Bonzon 2007), but none of these systems seek to expand the utility of OBD by employing two-way communication, inputting OBD sensor results to a database, or providing a means of reconfiguring the device remotely. By making a system capable of logging data in such a manner, useful applications for every aspect of vehicle ownership, maintenance, and monitoring might be developed.

1.6 Proposed Solution

The purpose of this research was to create hardware and software to interface a vehicle with a cellular network in order to collect data for later analysis. This required the development of a vehicle to modem interface, as well as an understanding of what data would be useful to collect.

As a proof of concept, I proposed producing data visualizers to demonstrate the functionality of sample applications. A “consumer” application would provide a real-time digital dashboard, while the “government” application would calculate the amount of fuel burned within a geographic domain in order to calculate and apply intelligent congestion taxes.

1.6.1 Primary Goals

The goals of this project ranged from providing real-time telemetry to making existing information more useful to consumers by providing easy to understand results such as green metrics. The data logged to the server would be used for congesting taxing, or some data – such as traffic patterns – could be useful for city planners attempting to re-sculpt currently suboptimum traffic patterns.

1.6.2 Secondary Goals

In addition to providing these new functionalities, this research would support a better understanding of present diagnostic systems, allowing for the discovery of weak points that may then be

addressed in future drafts of vehicle diagnostic standards. This would allow for the determination of how to improve the utility of future diagnostic tools.

2 Design Methodology

I began the engineering process by identifying the core requirements of the product. From these, I could determine the most effective means of achieving those goals.

It was clear that I needed to interface a microcontroller with a vehicle supporting OBDII, design a means of networking the microcontroller, and implement a means of gathering non-OBD data, including GPS and accelerometer sensor values. I would also need a way to gather data from the device and store it in the short term (in memory, on the device) and long term (in an external database). To visualize the data, I would need to develop a way to display the data in an easily digestible format. Then, to piece everything together, I would need to define communication architecture to facilitate data sharing between the vehicle-modem interface and data collection software in the vehicle and remote data storage. Only then could I explore the writing of applications for data analysis and feedback for the various target groups.

These broad milestone goals included:

- Gathering accelerometer and GPS data on a microcontroller
- Connecting the vehicle to the microcontroller
- Writing data from the vehicle and sensors to local memory
- Writing data to a server remotely
- Displaying stored data
- Displaying live data
- Facilitating two-way communication between device and central computer

By breaking these down further, I could define a product contract with markers for progress and build incrementally. I could analyze these smaller processes and apply my knowledge to draft effective solutions with quick turnaround.

To minimize development time, I decided to design the hardware in Eagle PCB and Schematic Editor, write my code in Notepad++, and design enclosures in SolidWorks. I would attempt to locate open source software as a basis for my development where possible, write software using languages I had used previously, and only venture to learn new languages when absolutely necessary. For aspects of the project with which I had no familiarity, I would have to learn new tools and algorithms as necessary.

2.1 Assumptions

I made several assumptions in this research. I assumed that all vehicles wherein this system will be used are OBDII CAN equipped, while in reality, only light passenger vehicles manufactured after 2008 are legally required to be so equipped. For the later sections, where I discuss this project's applications to vehicles and how government bodies may provide processed information back to their citizens, I assume a willingness to use the system and a vast network. In reality, this would require answering questions regarding privacy and making the system friendlier for average consumers.

2.2 Milestones

To determine my progress, I laid out a series of testable, small-scale design components. Though not exhaustive, this list kept me focused on target throughout the development process. In order, I attempted to design and test the:

- Electronic hardware to interface with vehicle
- Electronic hardware to interface with cellular network
- Software server to receive data over cellular network
- Combined electronic hardware, capable of sending GPS and OBD data over a cellular network
- Software server and parser, capable of storing GPS and OBD to a database
- Integration of an accelerometer to the hardware and software
- Creation of software capable of reconfiguring device remotely
- Implementation of a login system
- Implementation of a consumer dashboard capable of displaying data in real-time
- Implementation of a government dashboard, capable of determining amounts owed based on fuel consumption and distance travelled while within a user-defined polygon
- Single vehicle testing
- Multiple vehicle testing

Despite my best efforts, this list was not exhaustive, and had to be revised and updated throughout the project.

2.3 Deliverables

To accompany the list of milestones, I came up with a list of deliverables for the project that would determine whether or not the project was a success. The list included:

- Electronic hardware capable of interfacing a vehicle with a cellular network
- Server capable of processing data received from vehicles as well as clients and directing it to the appropriate database or executing the proper command
- Login system with password encryption and different user permission levels
- Client dashboard capable of showing at least five parameters simultaneously, in real-time
- Government dashboard capable of defining billable regions and rates, as well as generating billing statements from data in database

2.4 Outside Sources

I made extensive use of SAE and California Air Resources Board (CARB) standards and specifications to develop the vehicle interface. Some of the login software was derived from freely-licensed, publicly available software. Ivan Sergeev agreed to help with embedded software development and early stage hardware design validation. Additionally, much of my software support and debugging was obtained through hardware design forums, component distributor technical support, and computer science message boards.

3 Experiment 1 – Bluetooth Dongle & Symbian OS Cell Phone

3.1 Brief Introduction

The first revision of the system was designed to meet as many of the goals as possible in a summer. To maximize the functionality of this setup, I used a phone as a modem to reduce the amount of hardware required, and to keep cost and complexity manageable.

The Bluetooth solution was never intended as a complete one, but rather to provide a platform for learning how to interface with a vehicle and understand data transmission. The data collected by this revision would also be useful in simulating remote clients for future experiments.

The demonstration application of the Bluetooth logger was to monitor the energy use and path-efficiency of a vehicle. By combining this hardware with a smart phone capable of GPS localization, I could log critical parameters as a vehicle moves, including engine speed, vehicle speed, fuel consumption, and emissions data, all of which would be analyzed and displayed to the end user as a form of efficiency feedback.

The hardware for this project consisted of a phone with an integrated Bluetooth transceiver, as well as a Bluetooth-capable OBD device. This section contains an overview of OBDII transceivers, Bluetooth transceivers, the Nokia N95, and the process of measuring engine and other vehicle parameters.

3.2 Experimental Procedure/Hardware Overview

I began by exploring various smart phones capable of accessing the Bluetooth Serial Port Protocol. I turned up with a list of the iPhone 2G, Google Nexus One, or Symbian OS phones and settled on the iPhone due to its wide acceptance and open architecture, under the assumption that the yet-unreleased iOS3 would support Bluetooth SPP. I purchased a developer's license and began to learn Objective-C, but Apple's "Made for iPod" program was too cumbersome to deal with for a one-off prototype, and I was forced to switch platforms.

Android was still in its infancy - it had potential, but the early Bluetooth Application Programming Interfaces (APIs) were barely usable. Symbian OS had excellent documentation and a wide codebase. While Symbian was on its last legs, Professor Sarma had such an equipped phone on hand and

I already had had general Python experience. As this was primarily a hardware development project, Symbian proved to be the most logical choice.

3.2.1 OBDII Interface - ELM 327

To access the information available from OBDII, I needed a means of communicating with the vehicle. As a result of the standardization of OBD, many companies produce integrated circuits for this purpose. Most communicate with legacy (non-CAN) OBDII standards, and some can even read OBDI codes.

The best-known and best-supported transceivers for OBDII are the ELM series, made by ELM Electronics. This research made extensive use of the ELM327, a multi-protocol OBDII to serial IC. Other circuits are faster and less expensive, but the ELM is known for its ease of interface. The company provides good support, an excellent datasheet, and has the widest user base of any OBD transceiver. An additional plus, its hobbyist origin (a PIC18 core) makes it easy to reverse engineer. For my application, the slow sample rate was deemed sufficient.

3.2.2 Bluetooth Transceiver – CSR BC417 and Roving Networks RN41

Getting data out from the ELM327 is just as important as getting data in from the vehicle. This required a Bluetooth transceiver. The unit purchased for testing relied on a CSR BC417 radio, but this was not available in small quantities. Research yielded the Roving Networks RN41, which acts as a socketed serial port, allowing easy serial communication and transparently encapsulating data. Pairing is also easy, as the device requires only a passcode to initiate a connection. The device worked well at 38400 baud – fast enough to avoid being the bottleneck in the system.

3.2.3 Nokia N95

Professor Sarma had a Nokia N95 available for data processing. The phone had excellent documentation, support for Python, and built-in Bluetooth. Under Symbian OS Feature Pack 3, the phone had sufficient processing power to convert ELM data to human readable text before writing it to a file and uploading the data. The support forums on the Nokia website offered invaluable support.

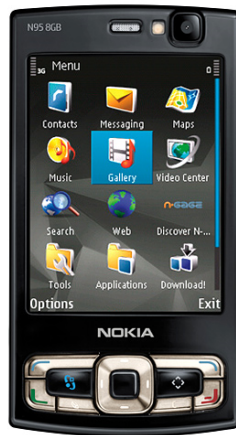


Figure 2 - Nokia N95 smart phone, the device used to test the custom Bluetooth-OBD transceiver ²

3.3 Methodology

I began developing software for the Nokia concurrently with designing transceiver hardware. The decision to use the expensive ELM327 (approx. \$25) meant it was imperative the design be capable of supporting a PIC (approx \$2.50) with minimal changes, should we ever produce the device. To this end, I designed the board to fit an ELM with the same footprint as the PIC – so I could build two, one with known good hardware and one with an improvised ELM clone. I made a list of specifications the ELM met that I wanted to duplicate, and a list of what I could do without.

I ordered an ELM327 to Bluetooth module for testing at the start of summer. This would allow me to test the phone software early, while I waited for the circuit boards' and components' long shipping times. This would allow me to explore the functionality of an existing device and determine the feasibility of reverse engineering the ELM.

Upon receiving the Bluetooth dongle, I examined the hardware and designed a smaller, more cost-effective interface. With the ELM, the cost hovered around \$70 for parts alone. With the PIC, it could be made for \$22 – less than the cost of a single ELM IC. I ordered boards and worked on the software while I waited.

² Retrieved from <http://www.letsgodigital.org/en/16412/nokia-n95-8gb/>

Following SAE specifications, I drafted software to log raw data from the vehicle. I tested my software with the consumer logger, but the phone would disconnect after a few minutes. I located an open source program called SymbtELM, a text-only Symbian OBD reader, and studied its code, before determining where the bug was in my connection routine. After minor tweaks, I was able to interface the phone with the vehicle and began exploring which sensors to sample to best generate meaningful log files.

What information could I glean from certain sensors? How fast would I have to sample them? What parameters would need to be calculated, and what data would those calculations require? I made a list of what I believed to be critical values and rotated through them, while generating log files in the hopes that it would become clear which data was truly important. This data became the basis for simulated clients in later experiments.

Threading on the phone proved difficult, so I wrote a test program with a 40Hz loop polling the serial port and interrupts for GPS and accelerometer sampling. The program logged these data to separate comma-separated files. The software worked at 0.4Hz per OBD sample. With tweaks, 38400bps connections enabled me to sample six parameters at 1Hz (six total samples per second). This worked but was not fast enough for long term data collection. I spent weeks optimizing timing, yielding a final sample rate of 9.6Hz on the same vehicle.

Toward the end of the summer, the circuit boards arrived. After some initial confusion about the jumper configuration for the RN41, the ELM devices connected properly. The devices worked for several weeks of testing. I never tested the PIC version; the project was moving away from Bluetooth.

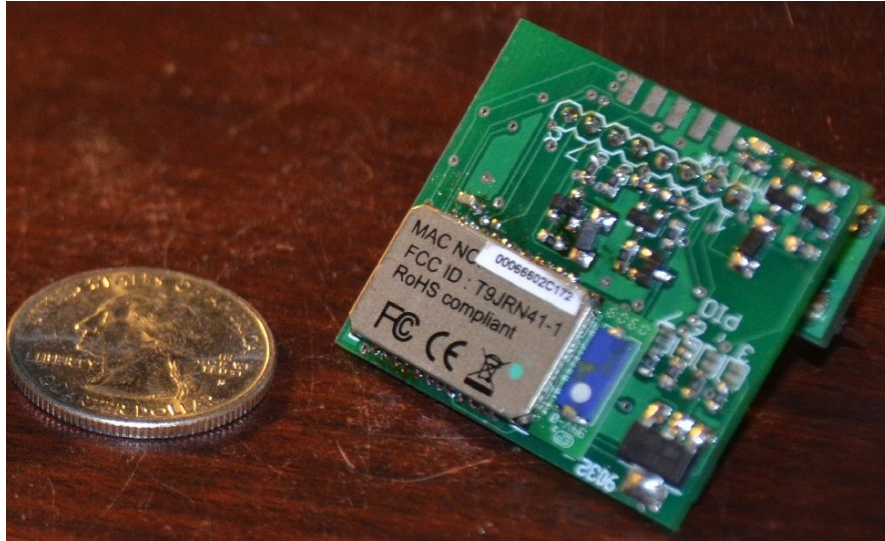


Figure 3: Picture of the custom Bluetooth-OBDII transceiver, without enclosure. This device connected a vehicle to a cell phone for local data analysis.

3.4 Results & Discussion

My research proved that it is possible to transmit OBD data over Bluetooth. At the end of the summer, my system even slightly bested the performance of the commercial ELM327 scanner. With additional engineering development, this setup could have met all of the project deliverables but would be overly complicated and slower than a dedicated device.

While it worked, Bluetooth was a substantial bottleneck in the data logging process. The data logging also revealed deficiencies the SAE specification for OBD, as non-CAN vehicles cannot process multiple parameter requests in a single query.

The phone hardware had non-trivial issues, too. The inaccuracy of the GPS meant velocity calculations were useless, and the phone's inability to obtain satellite lock during cloudy weather presented a significant problem. Rolling down the windows helped obtain a lock, and the signal was located more quickly after repeated use, but for a consumer product these flaws would be unacceptable.

The accelerometer in the phone yielded surprising results. The resolution was too low to provide localization data, but was high enough to pick up engine noise. Between this and the phone's varying orientation, any future revisions would require the implementation of software filtering.

Interestingly, the Bluetooth component of the system worked well enough to cause an unexpected problem. I could consistently connect to my vehicle from my fourth floor apartment when I was parked outside. This could be a significant security issue, and papers such as those presented at the IEEE Symposium on Security and Privacy are now starting to suggest Bluetooth unnecessarily exposes vehicle networks (Koscher et al. 2010).

After extensive testing, the device performed as well as expected, but Bluetooth had taken the project as far as it could. It required the user to carry a phone in the vehicle at all times, which was impractical. It drained the battery, and aside from the bandwidth bottleneck, some phones are unable to pair multiple devices simultaneously, making use of a headset with the device impossible. I also learned that File Transfer Protocol (FTP) uploading was of little value, as the data arrived “stale.” Future versions would require a true live streaming mechanism.

One benefit of this early testing is that I learned multiple ways to evaluate certain parameters. Based on the age of the vehicle, sensor availability, and accuracy requirements, calculating certain values – such as barometric pressure and fuel economy – could be derived a number of different ways.

Learning about Python for Symbian OS made writing future vehicle interface applications simpler, as the framework for OBD communication is device agnostic. The hardware designed during the course of this research also suggested the potential for a small and low-cost alternative to existing vehicle-to-Bluetooth transceivers.

In many respects, this project was too ambitious, and taught me to scale back the future experiments. The next revision required a better GPS and accelerometer setup with higher gain antennae. The ELM was not a bottleneck in the system, so it could be carried over to future revisions, though all model year ‘08 and newer vehicles make use of one of the CAN protocols, meaning I had been using a lot of additional hardware for little additional benefit. What I needed was something that was not a phone, but rather a dedicated device.

Ivan Sergeev had been working on a similar logger at the time (reading to SD card on a dedicated device, rather than making use of a phone). He was kind enough to pass the hardware on to me to develop

the next revision, with an integrated modem. Given that most of Ivan's hardware and software had been tested, I could think towards other aspects of the project, including brainstorming applications for the device. This early demonstration proved there is a lot of potential in this field, particularly in the segment of data processing and analysis.

By logging the most critical vehicle parameters under a variety of conditions, I could access information that, prior to cell-phone vehicle diagnostics, would have been unimaginable. For example, using the data gathered by the phone, it was possible to predict component failures based on symptoms the engine computer reports.

4 Experiment Revision 2 (OBD Reader with Modem, MySQL backend)

4.1 Brief Introduction

Experiment two involved refocusing the project. This new revision was based on Sergeev's logger, with an added GPS receiver and high gain antenna as well as cellular capabilities. The primary goals for this hardware were to have the system entirely powered by the vehicle, to log to an SD card, to support only CAN OBDII (enabling a fast sample rate), and to build all the hardware into a single circuit board. This platform would be able to provide real-time feedback or send encoded data out (e.g. aggregate fuel economy within a certain region). Having the modem onboard meant that the hardware would not need to pair with a phone using Bluetooth, offering the opportunity to improve security and perhaps even bundle data plans with the hardware (similar to the model of the Amazon Kindle). GPS, fuel economy (how much fuel one's car has used while within certain geographic boundaries), and time of day would allow us to construct a more valid, variable pricing scheme for congestion in urban areas. A server setup to capture this data would display the information in real-time and display the data on a remote computer.

This device would provide improvement over the Bluetooth device because the real-time data would allow for live feedback that could be used in traffic shaping applications. Higher resolution data samples from the accelerometer, as well as an overall faster sample rate, would hold the proverbial magnifying glass up to the characteristics we are tracking and enable us to see more subtle patterns emerge, providing information to potentially address a variety of problems.

The basis for the hardware had been tested, as had the low level vehicle communication software. Both of these components worked, but could be improved - increasing speed, increasing the number of parameters read, increasing storage. The hardware redesign and software tweaks would address these issues.

4.2 Experimental Procedure (Methods and Materials)

The experimental procedure began by defining a set of goals. The previous project taught me that "feature creep" would significantly slow down progress, so I started by making the decision to ignore two-way communications and focus solely on retrieving data from the vehicle. Focusing only on CAN

vehicles further simplified the hardware and software design process. I would aim to get data from the modem to the cellular network first, then data from the vehicle to the modem, then from the cellular network to a server.

With that in mind, I set up goals for the hardware and the server that would meet these objectives, but still allow for growth towards later steps of the project. The vehicle interface hardware would simply be a one-way, long range CAN Bus transceiver. The data collection server would serve only one purpose – providing a target for incoming device connections and storing the parameters received to a database. After making these decisions, I evaluated Sergeev’s hardware and identified the core components.

4.2.1 Processor - LPC2129

Sergeev had been using the LPC2119 processor. He preferred the ARM processor core to the PIC, as he believed that ARM was a more commercially viable processor to use while PIC caters to hobbyists and short run component manufacturers. Regardless, the LPC2119 had a lot going for it. It had two serial UARTs, it was fast, had great documentation and examples, could be programmed over ISP using freeware called FlashMagic, and had a real-time OS (FreeRTOS) written for it. The problem was that the LPC2119 was low on flash memory. Fortunately, there was another processor – the LPC2129, with double the flash memory – available in the same package. The choice of processor was one of the easiest in the entire project.

4.2.2 Modem & GPS - GM862

The first step to locating a modem was answering the question of GSM versus CDMA and choosing a carrier. I first did an analysis of various carriers to get an understanding for how much the device would cost to run if it were to be mass produced.

I looked at business analyst reports for a similar device, the Amazon Kindle. For the Kindle 1, using Sprint, the cost was \$0.12/Mb. The second generation Kindle cost was \$0.15/Mb, worked backwards from Kindle’s reported two dollar average revenue per user and the typical number and size of books purchased according to a Nielson study. Using these numbers and applying them to the various

cases of a thick client that reported a single, preprocessed number, versus a thin client that would pass on all reportable parameters, I was able to come up with a reasonable cost estimate.

In the worst case, with a 1Hz update rate and a 512 byte buffer, the device would cost \$8/month to run for one hour per day. This number was encouragingly low, and the trends were in the right direction (increasing bandwidth limits and decreasing cost). Coupled with the fact that GSM modems were less expensive and that there were no prepaid CDMA data plans, GSM was the obvious choice. AT&T offered the most compelling plan, including a free SIM card, and they assured me that GPRS access would be unfettered.

With the network determined, I could begin looking at GSM modems. I was looking for something hand solderable with high signal strength. Integrated GPS would be a plus, as would a common supply voltage with the rest of the existing hardware.

I identified the SM5100B module that met these criteria and was low cost, but the module was hard to locate in quantity and the documentation was lacking. I could not locate any way to reach the manufacturer for support, so I began to look elsewhere. I discovered RoundSolutions, a network consultancy and distributor of Telit GSM modems. The availability of support and FAQs on their forums guided me to the GM862-GPS, a modem with an integrated GPS receiver, a Python interpreter, an analog to digital converter built-in, and a well defined AT command set supporting HTTP, FTP, SMTP and socketed connection requests. The GM862 was hand solderable and competitively priced. It even offered a ball grid array version for future work, the GM864. I purchased a development board for the module from MikroElektronika. The board allowed me to test the device with a PC before making the decision to build it into the final hardware revision.

I used RealTERM to issue AT commands and test the viability of the modem. The GPS, though slow, worked very well – and the increased cost of purchasing a separate GPS module that updated faster than 1Hz, as well as the increased bandwidth costs, made sticking with the GM862 a wise design decision.

5 CAN Transceiver – TI SN65HVD232

The CAN transceiver required less thought than the selection of a modem. Because CAN is a basic specification, nearly any transceiver IC would serve our purposes. I looked for a well-documented, small footprint 3.3V transceiver. I found the SN65HVD232 which supported up to 1Mbps communication, meaning it would be compatible with all OBDII CAN protocols. It also had fail-safe detection built in, making one less component that could be exposed to damage from voltage transients the regulator did not snub at the alternator.

After component selection, it was a matter of redesigning the layout to incorporate the new components and their support hardware, such as a high frequency switching power supply. This was my first foray into cellular hardware design, and it took a lot of reading to learn how to avoid interference and keeps the signal from dropping out, especially considering the poor regulation of an automotive alternator. I used National Instrument's power supply calculation tool and paid careful attention to power routing on the PCB, adding large ground planes and ensuring that I used high-value tantalum bypass capacitors anywhere that could be affected by voltage ripples. Unlike with low power circuit layouts, the modem had thermal issues to contend with. A large ground plane and low density component packing made this a non-issue, though it did get generate heat during operation.

From the previous experiment, I knew that the orientation mattered. Mounting the device under the dashboard would force us to contend with the possibility that the device would not fit cleanly in the port (its cross-section was larger than the federally required cutout) or would create a hazard for the driver's knees. Given that I had designed for external antennas, I decided to put the hardware on a cable of its own to allow for easy positioning. The orientation issue would be corrected for in software with a high pass filter. I ordered the circuit board and components and assembled the device in a homemade reflow oven. I found a problem with interference between the SD card holder and the GM862 casing, but trimming a tertiary ground contact solved the issue and I was able to download a test program successfully. I programmed it with an FT232RL USB to serial cable, using FlashMagic.

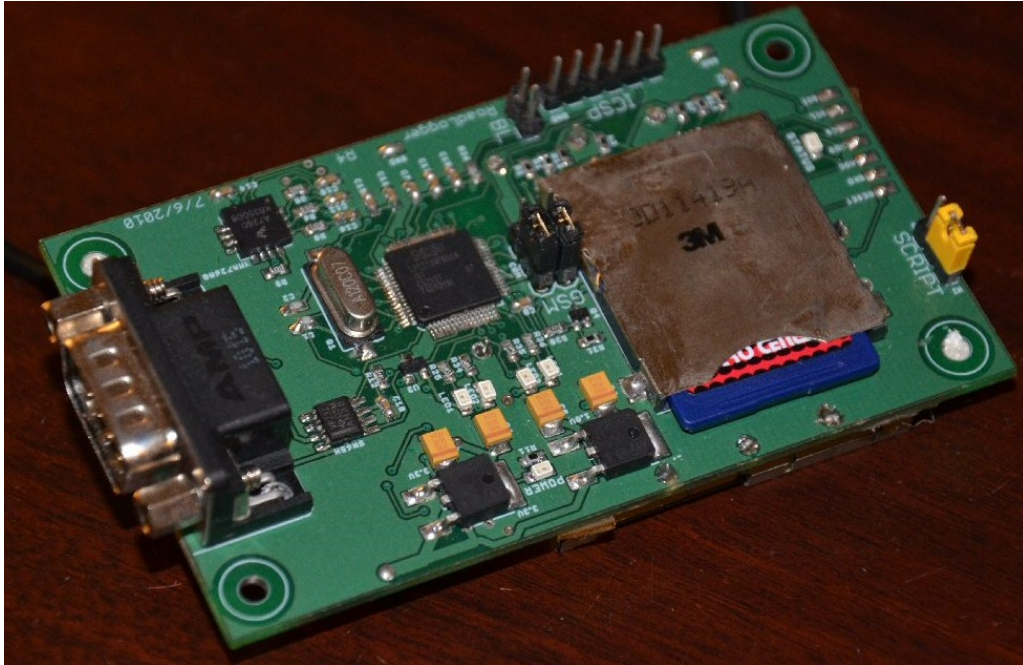


Figure 4: Assembled road logger, no enclosure (antennas, cables not pictured). This device features CAN for vehicle communications and GPRS for cellular connectivity. It also logs to SD card to make development easier and to “fill in” data gaps created by the GPRS network delays.

5.1.1 Embedded Software

After designing the hardware, I wrote software to transmit latitude, longitude, and a device ID from the hardware to a PC. I had designed in jumpers on the circuit board to allow me to connect the device over USB, making debugging significantly easier than attempting to do it over the cellular module from the start. Once I had it sending data to the PC, I was able to begin debugging the connection code and working with Sergeev on developing a framework for the embedded system software. We knew the hardware would have to work with CAN vehicles, and that there are 11 and 29 bit vehicle communication systems. The hardware was the same, so I wanted to make the device work with as many MY08 and newer vehicles as possible by designing for both protocols from the start. I also wanted variable sample rates for the accelerometer, GPS, and OBDII, so we would have to integrate a real-time operating system to schedule the tasks.

5.1.2 Server Software

From the Bluetooth logger, I knew that uploading data after a drive limited the amount of analysis I could do. Even breaking it into large packets would make it impossible to realize many of the benefits afforded by live streaming data. For this reason, I opted to connect the hardware using a socketed connection that behaved as a wireless, long-range serial port. I could have used PHP to parse HTTP requests, but that required additional overhead that could significantly increase bandwidth costs and would not allow for remote change of mode easily. An SMS would have worked, but could take too long to deliver. An encapsulated socket was the best choice.

This required another decision – what type of connection would I use? UDP has low overhead, but there is no guarantee of delivery receipt. There is also no means of ordering incoming packets. TCP solves these problems at the expense of additional overhead. As data transmission is constantly becoming less expensive, I opted to increase the packet size in order to have more reliable transmission. This also ensured that charges generated using this system would be as accurate as possible.

I wrote the server in Python using my knowledge of sockets on Symbian OS. The server connected to the devices and logged the input to text files reliably. The next step was to store the incoming data to a database. I chose MySQL 5 because of my familiarity with it, and used the MySQL-Python module to make it compatible with my logging server. The first version of the device sent an ID and passed along entire \$GPRMC strings from the GPS module, which were split at each comma on the server side. It was inefficient, but it worked and captured the necessary parameters (latitude and longitude) to a database.

Unfortunately, there was a problem with the server: it could only handle a few clients at a time before the blocking calls began to cause errors. I needed to migrate to something multithreaded or asynchronous, but my programming skills weren't up to the challenge of doing it by myself. I located the Twisted framework for Python, which worked remarkably well and was extremely well documented.

With the server working reliably, I was able to add individual OBD parameters to the string, sending along speed and RPM with the location. This was enough data that I could start working on the visualization software.

5.1.3 Data Visualizer

The problem with showing the data is that it comes in quickly. Showing parameters that do not change or that change slowly is easy, but I wanted something that demonstrated the full potential of the device. I decided to show a live updating map with clickable markers to show velocity and RPM, to demonstrate that the system was working and to get myself familiar with designing web pages that update live.

I had heard good things about Zope, and decided to use it to create my tracker. Using Zope meant I would not need to install a complicated server, and passing data from the database to the web page was easy using the included handlers. The online tutorials taught me how to pull SQL, and I had testable software in short order. The biggest stumbling block was connecting Zope with the MySQL database. With that figured out, everything worked as intended.

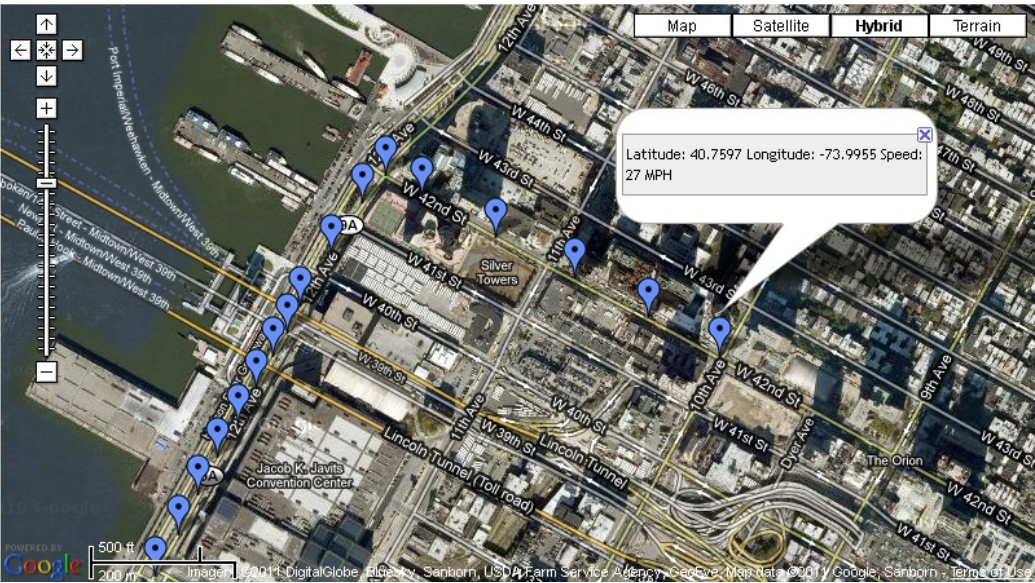


Figure 5: Screenshot of Zope tracker showing map markers with speed data. This demonstrated one-way communication using the GM862 cellular modem.

5.2 Results & Discussion

The results of the one-way communications test were encouraging. The GPS was accurate and updated at 1Hz, even in poor weather. The server was robust and could handle twenty simulated clients hammering the server with requests as frequently as 5 Hz. The real-time visualization was helpful, the OBD values were updating as expected, and the database was immensely useful for post-processing results to generate maps showing engine speed or RPM at various locations.

A test drive from Boston to Detroit showed that aside from network outages, the hardware worked reliably, and signal strength was strong even in rural areas. The testing did reveal a bug that was easily addressed before the next experiment: after losing a connection, the hardware would refuse to reconnect for hours. I thought it was a socket that was not closing properly, but in reality there was a parameter stored in the RAM of the modem that needed to be flushed before the device could reconnect. With this fixed, I could build toward bidirectional communication in the next revision.

6 Experiment Revision 3 (Adding Two-way Communication)

6.1 Brief Introduction

One of the goals for this project was to create a remotely reconfigurable OBD logger. To do this requires being able to accept reconfiguration commands. To this point, the data has only been sent from the device to a server and binned into a database. To move forward, I had to rewrite the server to allow for outbound communications and design a command architecture that would allow for several devices to connect to the server simultaneously. The server would also have to allow for the uploading of varied parameter groupings while still binning them correctly, as well as the individual addressing of connected nodes. This was no easy feat, especially since IP addresses over cellular networks can change quickly. The server would have to be able to locate a vehicle before it could change its mode, and would have to be able to contend with constantly changing data streams that might have network-based concurrency issues.

6.2 Experimental Procedure

I first made a list of revised goals for the server. It would maintain an up-to-date list of which clients are connected and their current IP address, a list of configuration parameters for each device to facilitate mode-setting based on last-used parameters, and would parse incoming data into the appropriate locations in a MySQL database. As a new, secondary purpose, it would also serve as a target for web clients, allowing owners of vehicle informatics hardware to change the parameters requested and vary the update rate of their device remotely. To that end, I developed a new architecture that could be tested using simulated clients.

6.2.1 Command Architecture

With multiple devices connecting to a single server, every vehicle needed a unique identifier. Fortunately, HS-3000 provides us with easy access to such an identifier: the VIN number. The first thing the hardware did when turning on was read the VIN number and status of the check engine light to memory. The device connected to a server which replied and then began communication by sending a string in order to store the current IP of the vehicle in a database. The server replied with either a default

mode command (if the device had never before connected), or the last used command. The mode command started and stopped tasks on the processor.

Once the string was received, the device verified the VIN against the variable stored in RAM and if it matched, it stopped the old tasks and created new tasks sampling sensors, uploading them, and logging them to an SD card. The tasks were identified using headers and each task had a sampling period associated with it. The device uploaded the results from these tasks with headers for parsing and binning to an SQL database on the server side. The tasks continued until they were changed, or until the device was shut off.

To be fully reconfigurable, a client needed to be able to tell the server to change the mode of a given device while in operation. A second change of mode command allowed for this. A client sent the server a mode command, and the server attempted to locate the relevant node. The server did a VIN/IP lookup to determine the most recent IP address for the target vehicle. The server attempted to send the mode command to the device, and set the “last used” mode in the database to the new command. This ensured that if the connection failed due to network errors, the device would switch to the new task set as soon as it reconnected. The use of headers on every uploaded parameter avoided the potential for concurrency issues this structure could cause.


```

`A0100` VARCHAR(32) DEFAULTNULL, /* AA - PIDS SUPPORTED (01-20) - BIT ENCODED */
`A0101` VARCHAR(32) DEFAULTNULL, /* AI - STATUS SINCE DTCS CLEARED */
`A0102` VARCHAR(64) DEFAULTNULL, /* AJ - DTC THAT CAUSED FREEZE FRAME STORAGE */
`A0103` VARCHAR(16) DEFAULTNULL, /* AK - FUEL SYSTEM STATUS */
`A0104` DOUBLE(5,2) DEFAULTNULL, /* AL - CALCULATED ENGINE LOAD VALUE */
`A0105` INT(3) DEFAULTNULL, /* AM - ENGINE COOLANT TEMPERATURE */
`A0106` DOUBLE(5,2) DEFAULTNULL, /* AN - SHORT-TERM FUEL TRIM (BANK 1) */
`A0107` DOUBLE(5,2) DEFAULTNULL, /* AO - LONG-TERM FUEL TRIM (BANK 1) */
`A010A` INT(3) DEFAULTNULL, /* AR - FUEL PRESSURE */
`A010B` INT(3) DEFAULTNULL, /* AS - INTAKE MANIFOLD ABSOLUTE PRESSURE */
`A010C` DOUBLE(7,2) DEFAULTNULL, /* AT - ENGINE RPM */
`A010D` DOUBLE(5,2) DEFAULTNULL, /* AU - VEHICLE SPEED */
`A010E` DOUBLE(3,1) DEFAULTNULL, /* AV - TIMING ADVANCE */
`A010F` INT(3) DEFAULTNULL, /* AW - INTAKE AIR TEMPERATURE */
`A0110` DOUBLE(5,2) DEFAULTNULL, /* AX - MAF AIR FLOW RATE */
`A0111` DOUBLE(5,2) DEFAULTNULL, /* AY - THROTTLE POSITION */
`A0113` VARCHAR(8) DEFAULTNULL, /* BA - LOCATION OF O2 SENSORS */
`A0114A` DOUBLE(4,3) DEFAULTNULL, /* BB - BANK 1, SENSOR 1 O2 VOLTAGE */
`A0114B` DOUBLE(4,1) DEFAULTNULL, /* BB - BANK 1, SENSOR 1 SHORT TERM FUEL TRIM */
`A011C` VARCHAR(8) DEFAULTNULL, /* BJ - OBD STANDARD VEHICLE CONFORMS TO */
`A011D` VARCHAR(8) DEFAULTNULL, /* BK - OXYGEN SENSORS PRESENT */
`A011F` DOUBLE(6,1) DEFAULTNULL, /* BM - RUNTIME SINCE ENGINE START */
`A0121` DOUBLE(6,1) DEFAULTNULL, /* BN - DISTANCE TRAVELLED WITH MIL ON */
`A0124A` DOUBLE(4,3) DEFAULTNULL, /* BQ - BANK 1, SENSOR 1 O2 EQUIVALENCE RATIO */
`A0124B` DOUBLE(4,3) DEFAULTNULL, /* BQ - BANK 1, SENSOR 1 O2 VOLTAGE */
`A012F` DOUBLE(5,2) DEFAULTNULL, /* CB - FUEL LEVEL INPUT */
`A0130` INT(3) DEFAULTNULL, /* CC - WARM-UPS SINCE CODE CLEARED */
`A0131` INT(5) DEFAULTNULL, /* CD - DISTANCE TRAVELED SINCE CODE CLEARED */
`A0133` INT(3) DEFAULTNULL, /* CF - BAROMETRIC PRESSURE */
`A0134A` DOUBLE(4,3) DEFAULTNULL, /* CG - BANK 1, SENSOR 1 O2 EQUIVALENCE RATIO */
`A0134B` DOUBLE(6,3) DEFAULTNULL, /* CG - BANK 1, SENSOR 1 O2 CURRENT */
`A013C` DOUBLE(5,1) DEFAULTNULL, /* CO - BANK 1, SENSOR 1 CATALYST TEMPERATURE */
`A0142` DOUBLE(5,3) DEFAULTNULL, /* CT - CONTROL MODULE VOLTAGE */
`A0143` DOUBLE(7,2) DEFAULTNULL, /* CU - ABSOLUTE LOAD VALUE */
`A0144` DOUBLE(6,5) DEFAULTNULL, /* CV - COMMANDED EQUIVALENCE RATIO */
`A0145` DOUBLE(5,2) DEFAULTNULL, /* CW - RELATIVE THROTTLE POSITION */
`A0146` INT(3) DEFAULTNULL, /* CX - AMBIENT AIR TEMPERATURE */
`A014D` INT(5) DEFAULTNULL, /* DE - RUN TIME WITH MIL ON */
`A014E` INT(5) DEFAULTNULL, /* DF - TIME SINCE CODES CLEARED */
`LAT` DOUBLE(9,7) DEFAULTNULL, /* LA - LATITUDE */
`LON` DOUBLE(10,7) DEFAULTNULL, /* LO - LONGITUDE */
`ALT` DOUBLE(7,2) DEFAULTNULL, /* ZZ - ALTITUDE */
`LIVE_ACCEL_X` DOUBLE(5,3) DEFAULTNULL, /* IX - REAL-TIME ACCELERATION */
`DTCS` VARCHAR(20) DEFAULTNULL, /* TR - DTCS ACTIVE */

```

Figure 6: Example data structure and header/parameter mapping from the MySQL create statement. Notice that most of the OBD parameters are emissions related.

With the architecture figured out, I wrote a simulator based on this configuration. The simulator used the socket library in Python as it did not need to be asynchronous. I wanted to test the server's ability

to manage connections, so rather than keeping a socket open, I opened and closed the socket before and after every transmission. This simulated some of the worst network conditions imaginable. Testing with a few simulators running would stress test the server as though dozens or hundreds of vehicles were attempting to connect simultaneously.

To gather data for the simulations, I utilized data from the Bluetooth logger and reformatted it to fit the new architecture. In some cases, I worked backwards from Google Maps, defining a route and making educated guesses about various parameters at small time steps.

The simulation worked. The server was able to process change of mode requests and send them to the connected client directly if the connection was still live, and worked as expected when the client was not connected at the time of the change command and sent it the command upon the next login. Even simulating concurrency issues, the header parsing worked reliably.

6.3 Results and Discussion

The result of this experiment was an architecture capable of successful two-way communication, proper parsing based on transmission headers, a server capable of handling mode requests, and an easy way to store lots of incoming data to MySQL. The server initially had problems handling parameters that reported two values, but rewriting the dictionary for those parameters alleviated the problem. There was also an issue with processing negative numbers that turned out to be a problem with the regular expression parsing the incoming strings. With that fixed, the server was able to handle twenty-five clients constantly connecting and disconnecting at 1Hz without measurable lag or missed data packets. With the parsing and storage to MySQL working fine, the next experiment could begin – developing a means of displaying the incoming data in a user-friendly format.

7 Experiment Revision 4 (Writing a Login System and Client Application)

7.1 Brief Introduction

To this point, I was able to collect data but I had done no front-end development. Experiment four focused on designing a way to convey all the information the road logger reports, as well as a user interface for the various client types.

This required the development of a multi-level login system, a client application for data visualization and a government application for data processing and analysis. Each application had certain required capabilities. The login system would allow various permissions based on user level, making certain applications accessible to each user category. The client application would show the vehicle owner live-updating parameters from key sensors, as well as read trouble codes and link to an external resource to identify possible causes of problems. The government application would allow for the creation, editing, and deletion of billing zone polygons, as well as rate schedule definition and bill generation. This would all have to be accomplished in a way that made the user feel comfortable with sharing their information, and in a way that was as secure as possible.

7.2 Experimental Procedure

As with the other experiments, experiment four began by defining a set of milestones, in this case the login system, the client application, and the government application. After deciding to use Apache to host the page due to its ease of use and multiplatform support, I had to start developing the site. I did not want to reinvent the wheel, so for each application, I looked to see what freely licensable software I could make use of, or if there were well documented algorithms I could incorporate into my own code. Then, I determined the best method for integrating these code snippets, or I worked backwards based on the functionality required and determined the most appropriate programming language and methods to use.

For the login system, I aimed to anticipate three broad user types – clients, government users, and administrators, each with access to specific web pages. Each user would have a login and a password stored in an encrypted database. The most complicated part about this system would be making it impossible for non-authorized users to access restricted pages. This meant it would have to render

server-side security and could not pass any parameters through the URL. I knew PHP has good MySQL compatibility and rendered server-side, so I began to look for a PHP login system. I located JPMaster77's "PHP Login System with Admin Features" on evolt.org and found that it did nearly everything I needed. After some tweaking, writing CSS to make changing the layout cleaner, and locating Ivan Novak's (now defunct) JPMaster77 Wiki for help, I had a login system with multiple user categories in place.

The client page had a unique set of goals. I wanted to create an entire virtual car interior, from gauges to windshield, as a way of demonstrating the potential for this technology. I wanted the gauges to be reconfigurable – both in type and in location. To make the gauges move, I incorporated a virtual windowing system written in JavaScript and making use of jQuery. This allowed the user to create, move, resize, and close gauges as necessary. For the gauges, I located Bindows JavaScript gauges and wrote code to make use of Asynchronous Javascript and XML (AJAX) and Javascript Object Notation (JSON) to update in near real-time. A drop-down box changed the script, meaning each window could display a number of parameter representations (including vehicle speed, RPM, text representations of latitude and longitude). I incorporated a modified version of the live map tracker from experiment 2, and made a second map using Google Map's "Street View" images. This looked at start and finish points to determine orientation of the vehicle and simulate the view through the windshield. The check-engine light script used AJAX and a JSON derived from a PHP MySQL query to trigger a virtual "check engine" light and point the user to the OBD-Codes.Com web page for that particular error code by passing the code as a parameter in the URL. Finally, the accelerometer graph was sourced from HighCharts and again was fed by JSON data, updating every three seconds.

The government website was the most complex to write. Drawing polygons on Google Maps is easy since the V3 API was released, but storing polygons to a database and reloading them is difficult. I located sample code that allowed for editing of polygons written in JavaScript that I liked because it created midpoints when endpoints were dragged, allowing the government user to increase the resolution of their defined region without redrawing the entire polygon. Modifying the script, I created a save feature using PHP to save the polygon to a database, and a load feature that ran a JavaScript SQL query on the

page load to populate the map with the previously saved polygons. This worked after a significant amount of debugging, and the government user script worked to add, edit, and delete polygons while keeping a database of vertices up to date on the server.

The next requirement was to be able to define the pricing structure on a per unit distance, per gallon of fuel consumed, or per unit time within boundaries with granularity of two hours. I located a PHP script capable of updating an arbitrary MySQL database using text entry, and repurposed it to allow the government user to define a series of rates for each polygon in two hour blocks.

Finally, the government user had to be able to generate a bill. Using MySQL to populate the VIN field and a JavaScript calendar to determine the start and end dates, I was able to do a query and pull out the latitude, longitude, timestamp, and parameters necessary to calculate the fuel economy from the requested vehicles. These parameters were passed on to a PHP script where the calculation would occur.

Generating accurate bills required developing multiple algorithms to examine points to determine if they were in one of the polygons that had previously been defined, looping over these points by time, and determining which points were continuous in time before calculating the amount due.

The determination of whether or not a point was inside the polygon was done by first executing a bounding box check as part of a MySQL query. If the point was inside the box, I ran another algorithm that tests points by counting the number of times a line in any direction from the point intersects the polygon boundary. This of worked intermittently, and I realized that the points in the polygon had to be defined in a clockwise order for the algorithm to work. I determined this was outside of the scope of the project, so I made a note on the polygon drawing page to be mindful of the orientation of the polygon.

For integrating distance, the distance between GPS readings was treated as a straight line and added up. Time was read straight from the MySQL timestamps, and fuel economy was integrated as a left-hand Riemann Sum (underestimating in the user's favor). Testing this software with the simulator from experiment three proved that, after debugging, it worked as expected. The calculated results for bills seem accurate, though the error has not been experimentally validated.

7.3 Results & Discussion

This section documents the results of the software development through descriptions of the various web pages and informative screenshots.

Main Page

The first page prompted users for a login. Upon logging in, users were taken to their user level's main page (the features displayed varied based on user level). From here, a high level user could see every page – from the administrator interface to the client interface and everything in between.

Login

Username:

Password:

Remember my information for the future

[\[Forgot Password?\]](#)
Not registered? [Sign-Up!](#)

Member Total: 4
There are registered members and guests viewing the site.

[jsiegel /](#)

Logged In

Welcome **jsiegel**, you are logged in to Joshua Siegel's Vehicle Tracker (Rev 10292010_1611).
[\[My Account\]](#) [\[Edit Account\]](#) [\[Admin Center\]](#) [\[Government Application\]](#) [\[Client Application\]](#) [\[Logout\]](#)

Member Total: 4
There are registered members and guests viewing the site.

[jsiegel /](#)

Figure 7: Screenshots of main page for logger, showing all possible links (what an administrator sees)

Admin Center

The administrator interface was the management interface for the system– including changing user permissions. Clients could see the virtual dashboards of their own vehicles. The government could

edit billable regions and rates, as well as generate end of month reports for individual or multiple VIN numbers. Administrators were the only users who could edit other users' accounts.

User Table Contents identified all the registered users and their permission level.

Update User Level allowed the administrator to type in a username and promote or demote its permission level.

Delete User allowed the administrator to delete users.

Delete Inactive Users deleted users who had not logged in recently in order to keep the database clean.

Ban User disabled a user's account.

Admin Center

..... Logged in as **jsiegel**

Back to [\[Main Page\]](#)

Users Table Contents:

Username	Level	Email	Last Active
jsiegel	3	jsiegel@st.edu	1303170068
ksullivan	3	ksullivan@shantok.usps.edu	1294965823
mlford	1	mlford@st.edu	1302719589

Update User Level

Username: Level:

Delete User

Username:

Delete Inactive Users

This will delete all users (not administrators), who have not logged in to the site within a certain time period. You specify the days spent inactive.

Days:

Ban User

Username:

Banned Users Table Contents:

Database table empty

Delete Banned User

Username:

Back to [\[Main Page\]](#)

Figure 8: Administration page, showing how administrators manage accounts

My Account

My Account showed the user his or her “vitals” - name, associated VINs, email, and currently selected VIN. This was tailored to client users, but every user had a “My Account” page.

Edit Account was a link to the “Edit Account” page.

User Info

My Account

Name: Josh Eric Siegel

Username: jsiegel

Email: j_siegel@mit.edu

VIN 1: 1GCHK29102E229241

VIN 2: 1G1FP22PXS2100001

VIN 3:

Active VIN: 1GCHK29102E229241

[Edit Account Information](#)

[\[Main\]](#) [\[Admin Center\]](#)

Figure 9: My account page, showing name, e-mail, and other parameters associated with each account.

Edit Account

This page allowed the user to change any of the values associated with their account. "Active VIN" was the VIN the generated dashboard gauges were associated with. The user made changes and clicked “edit account” to apply them.

User Account Edit : jsiegel

Name:	Josh Eric Siegel
Current Password:	
New Password:	
Email:	j_siegel@mit.edu
VIN 1:	1GCHK29102E229241
VIN 2:	1G1FP22PXS2100001
VIN 3:	
Active VIN:	1GCHK29102E229241 (Selected) ▼

[\[Main\]](#) [\[My Account\]](#)

Figure 10: Edit account page, where the user updates his or her information

Client Application

This page allowed an end-user to see his or her active vehicle's status. The main page presented a few options. "Change VIN" allowed the user to update the vehicle the gauges reflect. "Change Parameter Configuration" was a text-entry way of configuring a device remotely (the user would enter a command string indicating which parameters to send to the server and how frequently). It queried the server for the IP of the particular VIN and sent it a new list of requested parameters.

The page served two main purposes. First, there was a once-per-minute query that checked the MIL status and reported if the check engine light was on. If it was, a "DTC" gauge indicated what went wrong and probably causes.

The virtual gauges were the second key feature. Clicking 'Create new gauge/graph' brought up a jQuery window. The user could move it and resize it before selecting what to populate it with. The user could create as many unique gauges as they wanted, but opening too many would lag the server. This is why the default gauge configuration is blank. Alternatively, clicking "fixed layout" would take the user to an optimized page with every possible gauge open and updating at a slower refresh rate.

Gauges

Blank (default) – this was for gauge creation, resizing and repositioning purposes only.

Error codes - showed what triggered the malfunction indicator light, if was active. This linked to an external database, but could have accessed an internal database.

Temperature – showed the current vehicle temperature.

Overview - showed all federally mandated parameters in text format in a single window.

Engine load –displayed current airflow divided by peak airflow.

Speedometer – displayed vehicle speed in MPH, sampled from OBD.

Tachometer – displayed engine speed, in RPM.

Throttle Position – a percent, from 0-100. Non-zero at idle is normal.

Mass airflow rates – reported airflow in grams/second.

Gas gauge – on vehicles supporting extended PIDs, this would display the level of fuel in the gas tank.

Map - showed a real-time updating map indicating the location of the vehicle. A toggle button allowed this to be switched to Streetview to simulate the view out the windshield of the vehicle.

Acceleration - showed three axes of acceleration on a scrolling, expandable graph. Axes were able to be turned on or off or even printed.

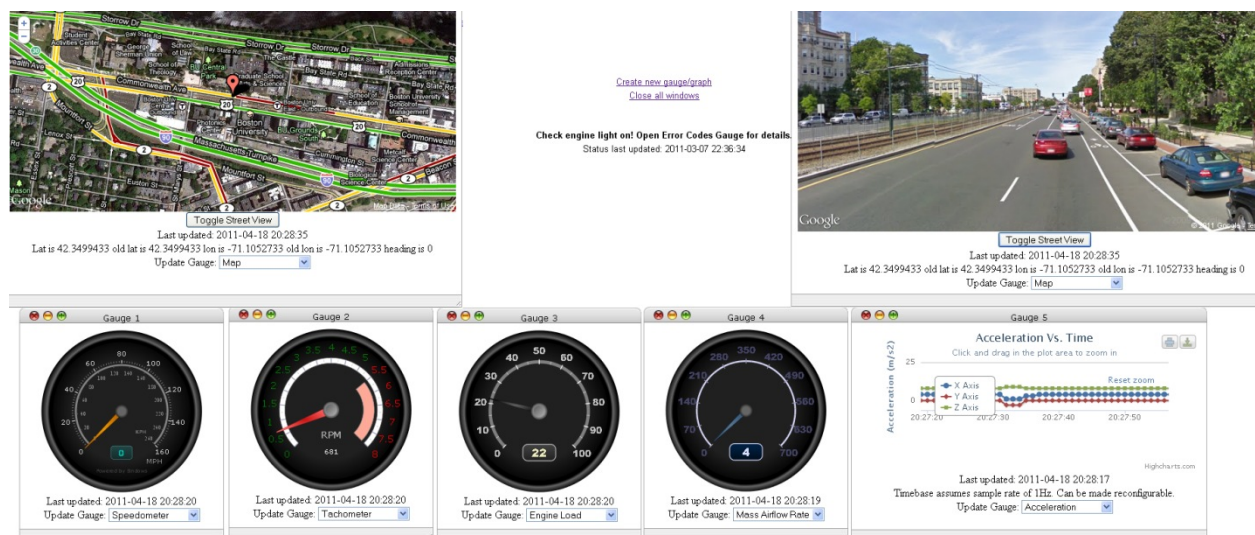


Figure 11: Client page, with a sample display of gauges including speed, engine speed, engine load, mass airflow rate, acceleration, error codes, and a virtual windshield.

Government Application

The government application was where billing polygons, rates, and fees due were edited and calculated. Its primary purpose was to demonstrate an application capable of facilitating real-time traffic shaping.

Welcome **jsiegel**, you are logged in to the government application.

This is where you can create, edit, and delete polygons defining billable space. Rate schedules are also determined here.

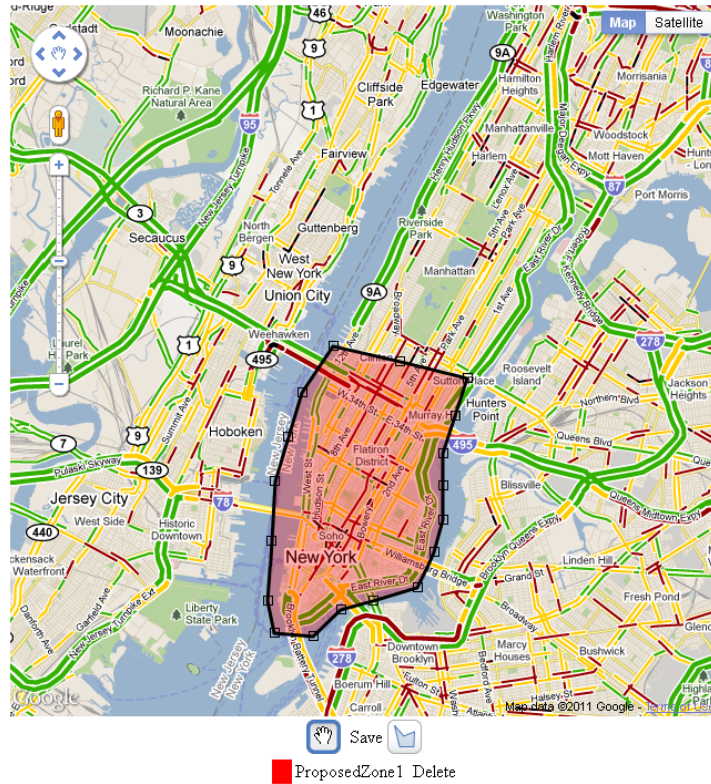
[Edit Billing Polygon](#) [Adjust And Apply Rates](#) [Examine Current Billing Period](#)

[\[My Account\]](#) [\[Edit Account\]](#) [\[Logout\]](#)

Figure 12: Government page, showing options to edit polygons or to create billing statements

Edit Billing Polygon

This is where the billing polygons were defined. There were a few useful tools here, and several important rules. The hand allowed the user to move the map and grab points to edit polygons. Clicking on a blank area allowed the user to drag the base map around and moving the hand over a point on an existing polygon allowed the user to edit an existing polygon. When editing, moving a point caused the two lines touching the vertex being dragged to create midpoints, increasing resolution. The polygon tool enables users to create new polygons. These had to be created clockwise (the algorithm to determine whether or not a point was in a polygon required a specific orientation to function for both concave and convex polygons), and due to MySQL limitations, these had to be named without spaces or special characters. To create a polygon, the user would select the tool, click a point down, then another, and continue until the polygon was complete. The software automatically closed the polygon. Polygons could be deleted by clicking 'delete', or take focus on by clicking their names. Clicking "save" took the user to the "Review and Apply Updates" page, where the data was saved.



Note: Changes to map are retroactive for future rate calculations. Care must be used when updating rates or polygons to first export the amounts due before applying new rules.
 Note 2: For the moment, polygons MUST BE DRAWN CLOCK WISE.

Figure 13: Polygon editing page, demonstrating a sample geofence. Government and administrators may define billable regions using this interface.

Review and Apply Updates

Here, the government user defined the billing type (dollars per gallon of fuel used, per hour driven, or per mile travelled) as well as the hourly rates of each polygon. The user would type the name of the polygon into the text field and click 'edit.' A new page appeared where a rate could be entered for each two hour subdivision of time. Selecting the billing type and clicking 'update the record' allowed the user move on. From here, the user could update additional polygons or examine the current billing period.

Please input name of polygon to edit as it appears below:

Edit

Overview

Name:	12A-2A	2A-4A	4A-6A	6A-8A	8A-10A	10A-12P	12P-2P	2P-4P	4P-6P	6P-8P	8P-10P	10P-12A	Billing Type:
ProposedZone1	4.000	0.000	1.000	5.000	5.000	1.000	1.000	1.000	5.000	5.000	1.000	2.000	PerGal



[Return to Edit](#) [Examine Current Billing Period](#)

Figure 14: Table for editing rates, indicating the ability to charge per mile, per gallon of fuel burned, or per unit time spent within the bounded region.

Examine current billing period

This page generated financial statements for clients as well as for government use, printing them to a local browser. The user selected a start date and an end date by clicking the calendars, and selected the VIN of interest or picked ‘*’ to select all VINs. The subsequent page displayed the results of the billing calculation.

Currently Monday 18th of April 2011 07:50:31 PM

Calculate from: 18-Mar-2011  to: 18-Apr-2011  for VIN

(Note: * means ALL vehicles)

Submit

Querying values from to where VIN is ANY

VIN 1GCHK29102E229241 owes \$12.97 for the selected billing period.

VIN 1G1FP22PXS2100001 owes \$7.04 for the selected billing period.

Return to [main page](#).

Figure 15: Bill generation page and sample output

7.4 Summary

The system worked well and demonstrated the full functionality of the command architecture. It identified several bugs, most of which were fixed, but some (like the orientation of the polygon disturbing the functionality of the point-in-polygon test) were not fixable within a reasonable timeframe and were

left as-is. The PHP.ini file and mysql.conf files had to be altered for the web page to work as expected, primarily changing timeouts to deal with scripts that were programmed inefficiently or simply had too much data to process within the default 30 seconds. This experiment could well have gone on forever, and the biggest lesson I learned from it was to watch out for "feature creep" and focus on a single competency. The experiment left me with a framework that worked reliably and is easily extensible.

8 Experiment Revision 5 (Embedded Software, Server Revision #2)

8.1 Brief Introduction

With the backend software working well, I received news that I would have to go back to the drawing board. Ivan Sergeev called and told me that the LPC2129 could not handle all we were asking of it while running FreeRTOS. This meant that we would have to dramatically restructure the architecture and reduce the functionality of the device.

8.2 Experimental Procedure

The first step was offloading some of the processing from the device to the server. Instead of sending human-readable data over the network, the device would send raw data from the OBDII port, in effect making it a long-range ELM327. This would have the added benefit of reducing the load on the network, and could make the data more reliable thanks to the addition of a checksum.

This required rewriting the data processing server. In many respects it was the same as before, making use of headers to identify which data is incoming and binning it appropriate. The difference is that, rather than simply needing to know the data type, a second lookup table was required with the formulas for every reportable parameter.

The mode command was also restructured to minimize the amount of processing onboard, and to keep the processor running smoothly, OBD sampling had to be converted into a task with a single sample rate for all parameters. With the embedded software working, Sergeev and I worked together to design a new architecture. We settled on sending fixed length strings with headers to identify data type. The OBD strings passed along their requested parameter as well as the raw hex data associated with the request.

I developed a new Python server and tested it using data from the SD card on the device. It worked as expected, and using a dictionary composed of functions worked properly for converting incoming data to human readable text.

The last step was black-boxing the device. From the drawings of the PCB, I designed an enclosure for the board and 3D printed it to give it a more professional look. This took several revisions

due to uncertain nature of plastic shrinkage on filament deposition 3D printers, but eventually turned out well.

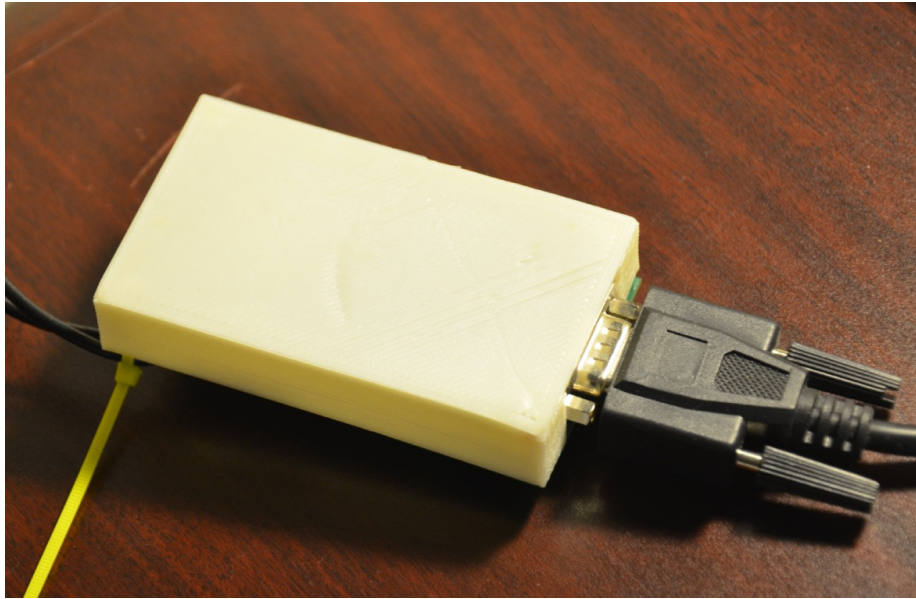


Figure 16: Picture of road logger enclosure. Plastic pins fit through holes in the circuit board and hold the two halves together.

8.3 Results & Discussion

I tested the device in numerous ZipCar vehicles from Toyota, Subaru, and Kia and found it to generally perform well for basic OBD scanning and data logging. Testing the device for more complicated tasks, however, revealed a slew of problems. On many vehicles, the VIN querying did not work, and neither did trouble code reading. At other times, the results from a parameter request would yield invalid results (this turned out to be an issue with defining the most significant byte in the embedded code). While faster than previous revisions, the GPS time-to-lock was still sometimes measured in minutes rather than seconds, and network delays ranged from one to six seconds. TCP connections and timestamps helped avoid significant issues, but the new format increased overhead and meant that the hardware was pushing bandwidth limits on its 2G connection, increasing transmission cost. This required splitting packets while uploading, making the parsing more complicated, and resulted in sockets closing sloppily and preventing the device from reconnecting reliably.

There were positive results from this experiment, too. The network worked superlatively when the parameters were adjusted to avoid overloading the network. The new format, which was the same on the SD card as it was on the cellular network, meant historical data could be turned directly into data for simulation, and comparing it against uploaded data validated that the network was properly transmitting vehicle parameters. This saved immeasurable development time when refining the various applications. The mode change command worked as expected, meaning the device truly was reconfigurable and capable of supporting an app model. From a design standpoint, the enclosure worked well, making the device look more professional, and did not present any heat-related issues even after extensive testing.

9 Overall Results and Conclusions

This project proved the feasibility of capturing data using a remotely-reconfigurable CAN to GPRS interpreter, visualizing the information in real-time, and writing basic applications to make use of the incoming data. The hardware and software met every goal defined in the early stages of the project, making the experiments successful. The hardware proved robust, gathering data without issue for hundreds of miles. The sample data demonstrated low-bandwidth use, identified network weaknesses, and pointed out issues with the currently legislated OBD standard. Among the deficiencies discovered with OBDII, I noted that sampling sensors from the vehicle was excessively slow, that OBD only stores a single freeze when an error code triggers, and the vehicle completely eradicates the freeze frame when it is cleared. The existing standard lacks provisions for hybrids and electric vehicles, and OBD does not require the reporting of many sensors that are now common in vehicles (e.g. accelerometers) or others that are easy to integrate (e.g. GPS) that would yield far more valuable information than the minimum required by law.

Other issues included that certain “simple” parameters, such as odometer readings, are not legally required and we therefore may not assume all vehicles have them. This makes it complicated to verify distance travelled, fuel level, and other metrics that would be computationally trivial to implement. There are additional issues with the standard. For example, even though all MY08 or newer vehicles are CAN, the difference between 11 and 29 bit CAN addressing means there still is not a single unified standard. There are no provisions to easily access low level systems such as ABS sensors, trouble codes cannot always be cleared when the engine is running, and there is no standardization of when the port is on or off – there is no signal for if engine is running or not. Data transmissions from the vehicle to the reader lack checksums, and, shockingly, there is no central repository or even requirement to disclose manufacturer specific Parameter Ids (PIDs). It would be easy for a manufacturer to integrate various tells to determine

if a device is plugged in and operational, but as things stand, spoofing or confusing vehicle logging hardware is a relatively trivial affair.

The experiment uncovered other issues in testing, too. Cellular networks lack inexpensive channels for text-only transmission, and network reliability makes keeping open a socketed connection difficult, leading to problems determining where a given device is connected.

Perhaps the biggest lesson I learned throughout this project was that it is easy to get distracted by "feature creep" at the risk of never finishing the original objectives. One cannot design to address every available opportunity, but by focusing on the key opportunities, it is possible to solve something simply and effectively. However, it is critical to frame a set of soluble and important problems appropriately.

10 Future Work

This project has myriad future applications. The task was to find the best way to deliver solutions, permitting both vehicle-specific diagnostics, but also have information on vehicles operating in a traffic system. The more comprehensive and relevant data we have, the better equipped we are to deliver an optimized solution. The device, its software, and its communicative attributes allow it to gather vast amounts of real-time data. The trends in the market are moving in the right direction for a long-range data logger since bandwidth is getting less expensive and vehicles are incorporating more sensors every day.

The software from this project served its purpose, but is incomplete in its potential functionality. I wish to add a fuel economy gauge, better error handling, query efficiency improvements, and a more complete database of users. From a user interface standpoint, I wish to revise the CSS and add a .csv and .kml export option for the data, as well as the ability to visualize graphs and average/typical values for all parameters.

The backend could use tweaks, such as VIN-determined vehicle manufacturer specific enhanced data collection (e.g. GM LAN), reducing the number of simultaneous database connections, optimization of the point-in-polygon determination algorithm, and determination of whether or not the mode change setting was received by the device, which has proven difficult to do in an execute-once PHP script. Incorporating a content management system to facilitate the addition of new applications and dynamic mode setting (automatically setting parameters based on open applications) would round out the web software tweaks.

The hardware could also stand improvement. If I were to do it over again, I would redesign around 4G cellular hardware, integrate a backup battery, reduce the footprint of the device, and build in chip antennas with high gain amplifiers. With tweaks to the embedded software, it could be configured to turn on and off automatically, as well as to upload data over FTP once the vehicle is shut off to provide a richer data set.

Once we answer the question of the true cost to operate the logger hardware (using real-life testing as data points), we may begin to apply the technology to new fields. With the system working well, it has the potential to provide information to spawn any number of new technologies. State and local governments realize the potential value here, but consumers will never accept such a technology without personally realizing the benefits and having their privacy and confidentiality concerns fully addressed.

One additional important consideration is making the value of this technology for an individual consumer palpable, even when a specific concern or complaint is not salient. To that end, how should the consumer value of these applications be presented? That depends on whom we considering, and how they might be using this technology. For cellular networks, our system provides valuable metrics. Every vehicle is part of a network of distributed nodes streaming to a server, thereby providing constant feedback on network quality and weak points that need to be addressed. For consumers, vehicle monitoring has massive potential, from determining when a car actually needs an oil change to when the tires ought to be rotated. Additionally, failure prediction has huge potential as an application, and is one that is very likely to happen.

With access to additional sensors already in many vehicles, such as yaw rate, steering position sensors or traction control computers, it would be possible to determine even more about the state of the vehicle – are the tires wearing evenly? Is the alignment bad? Using characteristic traces, it will be possible to predict when, where and how a vehicle will break down, a hot topic of late due to the Right to Repair Act. We could sell ads to local repair shops or specialty shops to monetize the system.

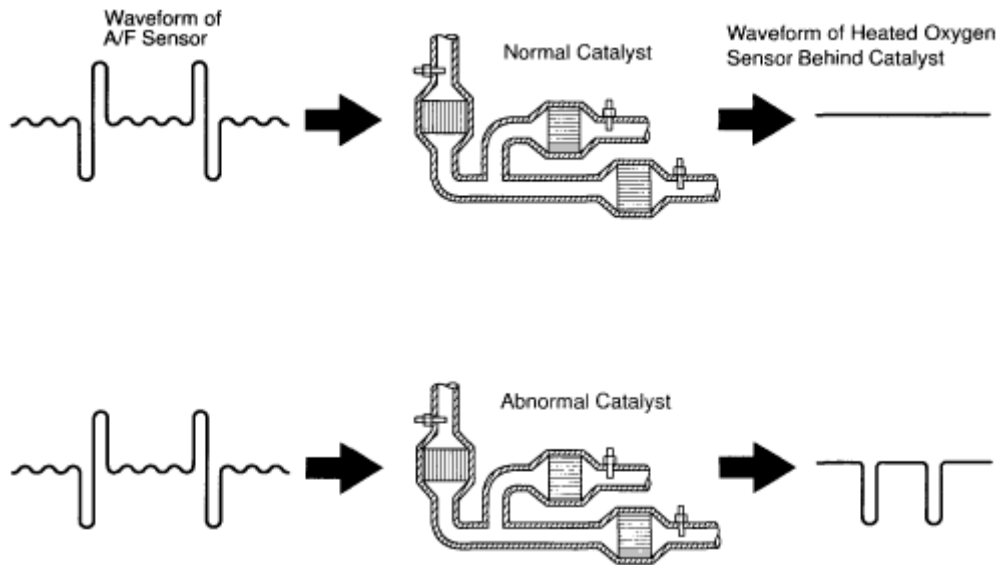


Figure 17: O2 Sensor waveforms demonstrating the ability for OBD to determine sensor failure using live data³

In the event that a failure occurs, we could have a database explaining the trouble code, likely causes and solutions, and relative urgency of repair. Such data could even be reported back to vehicle manufacturers or the NHTSA for failure analysis and early recall determination, or when authorized by the vehicle owner could even enable the request of bids for repair from authorized-participating repair agents. Data from sensors could even enable smarter, dynamic start-stop systems (along with reporting to local governments about the efficacy of the timing of traffic lights).

Other consumer applications include green metrics (what is the ideal gear to drive in, what is the most efficient route, where do I idle, and where does my CO2 go), predictive algorithms (where do I typically go on a Tuesday morning), or algorithms for usage based insurance, something Progressive Insurance has been testing. The system could provide driver feedback, or be optimized to driver characteristic or situational values, such as how much a driver regards fuel economy versus saving time. Similarly, one could use the data for smart route planning based on time of day, day of week, and real-time traffic information, optimizing for the lowest toll or the shortest distance or the smoothest ride. For resale, in part with historically archived data, this sort of system could determine if a car is well-maintained or not, how hard it was driven, how much of the warranty is left, and so on.

³ Retrieved from <http://alflash.com.ua/OBDII/P0420/P0420.htm>

Cellular OBD loggers could be utilized as an EZ-Pass replacement, or as a means of vehicle cost accounting (e.g. drive this way to be more efficient and save your catalytic converter and your oil life, saving you this much per mile, or inflate your tires to desired pressure and improve your gas mileage), theft prevention and recovery, crash detection, and performance monitoring. The accelerometer is of particular use as it can monitor road conditions (identifying ice) and driver conditions (is this driver sober?). The accelerometer could analyze traffic patterns, determining congestion and avoiding it while reporting back in real-time how much time you have actually saved. The system even allows the government to address some of the deficiencies of OBDII, by allowing for remote emissions testing or the storage of multiple freeze frames on a central server.

To run these applications would require an "app store" model, providing the application for free, for a one-time fee, or on a subscription basis. One could charge developers for access to the data on a per query, per cell, or other fee structure to incentivize code optimization, and or simply bill for server time utilized, as Amazon has done.

Alternatively, this same hardware could be taken any number of different directions. It could be ported to trucks supporting the J1939 specification, or reprogram engine control computers remotely and on the fly (both applications make use of the same hardware).

The most likely short-term application of this product is a set of thin and thick client hardware for wireless odometry as an early stage mileage-based road tax. The thin client would pass all of the data the device reads to the central server, whereas the more anonymous thick client would do computation of fares onboard using an updatable lookup table and simply report a VIN number and a dollar amount due. Both systems solve the problem of counting miles while enabling discounts for people who share more data and may therefore be routed to avoid significant congestion charges (they are part of the solution, not part of the problem).

The biggest issue with these devices is spoofing, creating fake devices or software programs designed to emulate a device outputting a certain result, and it is a difficult problem to eliminate. Certain "tells," like a characteristic voltage fluctuation, might help identify whether a device is in a vehicle or not,

as could state awareness and a backup battery (e.g. using the accelerometer to infer what the GPS readings should look like and determining if the GPS is being spoofed). Partnerships with camera systems, like those utilized by EZ-Pass or American Traffic Systems, could help identify cheaters but that would require significant licensing or infrastructure development. There are simpler ways to determine if a device is in a car, for example by sending “heartbeat” parameter requests at random intervals over OBD. The returned value would be checked for type and to ensure that it is not a repeated value, meaning that any spoofing device would have to have CAN communication abilities and implement a complete ECU simulator. A backup battery would allow the device to “scream” if it is unplugged or if the heartbeat responses are incorrect. IMEI registration would make all-in-one spoofing devices difficult to create, and hardware potting would make it infeasible to replace individual components.

Because the odometer is not federally mandated by any OBD specification, reading it for determining distance travelled would be fruitless. It would be possible to filter many inputs from the accelerometer, GPS, and speed sensors (for example through a Kalman filter, creating a form of inertial navigation) to converge on the distance travelled, but there might exist an easier solution. OBD may be tricked into revealing otherwise inaccessible data.

One of federally-mandated parameters is the distance since the trouble codes were cleared. In some vehicles, it is possible to trick the car into sharing this otherwise inaccessible information. One such way is to:

1. Turn on when engine is turned on, determined by reading RPM values
2. Read the trouble codes – are there any? Which codes?
3. Store the codes, if any, to a file (these can be uploaded so as to have a "live" emissions testing procedure, displayed to a text LCD with additional data, or any number of things)
4. Clear the trouble codes, resetting the distance since MIL cleared counter
5. Run a heartbeat process to ensure the device is connected (sampling parameters at random, at random time intervals, and checking that the expected type is returned - spoofing is not impossible, but is certainly difficult - especially if provide the ability for the government to ping the device remotely)
6. Report the distance travelled when the car is shut off (determined by reading RPM values)
7. Repeat, and increment a counter of miles driven in our database

After testing, some cars require the car to be on but the engine to be turned off in order to reset

trouble codes. A backup battery may solve this issue. One vehicle failed to respond to this trick, but the vehicle was not CAN-enabled. It will take significant testing to determine a reliable way of measuring distance driven, but a cellular OBD logger is sure to be a key component.

Wireless odometry and variable pricing to shape traffic patterns are hot topics and are emerging now as a field of study. I plan on taking my knowledge from this project and applying it to the solving of these problems in my graduate studies.

11 Conclusion

This project proved the feasibility of capturing data over a cellular network from a remotely reconfigurable diagnostic device. The CAN network worked well in the vehicle, and the GPRS connection, while slow, was reliable for data transmission. The hardware and software were successful in meeting the goals set for this project, from visualizing data in real-time to demonstrating how the system might be used to generate congestion fees based on vehicle operating parameters.

The hardware and network architecture provide a strong base for a diagnostic platform targeted at addressing environmental concerns and the human factors pertaining to vehicle ownership and operation. It does not replace existing on board diagnostics, but rather identifies weaknesses in the current system and attempts to augment them with external sensors and unique transformations of data. Most importantly, the system gathers information that would have otherwise been lost and attempts to turn it into something useful.

Even with the small sample of vehicles I tested, I was able to witness firsthand the fascinating trends in fuel economy and local congestion through little more than a live mapping application. With the inclusion of more vehicles in the study, the applications of such data would be limitless. Even for those drivers who do not wish to share their full status, the system has proven powerful enough to calculate the results for basic applications onboard and provide the driver with additional privacy, keeping his or her data locally stored and never transmitting it.

This research explored vehicle informatics, but just scratches the surface of the potential in this field. Live-updating diagnostics are becoming a prominent area of research, as are predictive diagnostics. These technologies push OBDII's aged computer structure toward its limit, and drive researchers to develop new technologies to meet the needs of a better connected, better informed public with paradigm shifting automobiles. I believe that for these projects and more, remotely reconfigurable cellular vehicle diagnostics will prove an invaluable tool.

12 Literature Cited

Aisenberg, A., & Andromalos, M. (2009). *OBDII Project*. Retrieved February 20, 2011, from EE476: http://courses.cit.cornell.edu/ee476/FinalProjects/s2009/ama64_maa66/ama64_maa66/index.html

Bonzon, F. (2007, February). Internet -based Tracking of GPS Equipped Buses (Master's Thesis). Ecole Polytechnique Federale De Lausanne.

Brol, S., & Mamala, J. (2006). Assessment of Passenger Car Driveability with use of two axis accelerometer mounted on car body. *Ultragarsas*, 2 (59).

Chadil, N., Russameesawang, A., & Keeratiwintakorn, P. (2008). Real-Time Tracking Management System Using GPS, GPRS and Google Earth. *ECTI-CON*.

ELM327 Bluetooth OBDII Wireless Transceiver Dongle. (n.d.). Retrieved February 28, 2011, from DealExtreme: <http://www.dealextreme.com/p/elm327-bluetooth-OBDII-wireless-transceiver-dongle-16921>

Feng, D., Lu, C., Zhou, K., & Wang, F. (n.d.). Transfer Sensor Data on a motor vehicle with GPRS Modem and CAN bus. Huazhong University of Science and Technology.

Fiducia, K., Lance Mahaz, J., & Smith, G. (2006, December 4). Vehicle Data Logging Device (Design Document). University of Central Florida.

Grutzmacher, P., Harmon, C., Muniak, R., & Siladie, J. (2003). *Real-Time Driving and Diagnostics Display*. University of Akron.

Hasan, K. S., Rahman, M., Haque, A. L., Rahman, M. A., Rahman, T., & Rasheed, M. M. (2009). Cost Effective GPS-GPRS Based Object Tracking System. *International MultiConference of Engineers and Computer Scientists*. Hong Kong.

Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., et al. (2006). *CarTel: A Distributed Mobile Sensor Computing System*. Massachusetts Institute of Technology.

J.P. (2004). *PHP Login System with Admin Features*. Retrieved February 28, 2011, from Evolt: http://evolt.org/PHP-Login-System-with-Admin-Features?from=2550&comments_per_page=50

Keenan, J. I. (2009, April 30). Creating a Wireless OBDII Scanner (Bachelor's Thesis). Worcester Polytechnic Institute.

Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., et al. (2010). Experimental Security Analysis of a Modern Automobile. *2010 IEEE Symposium on Security and Privacy*, (pp. 447-462). Berkeley, California, USA.

- Lawrence, J. (2005). Automotive Telematics: Is it time for a Renaissance or an Obituary? In J. Groebel, E. M. Noam, & V. (Feldmann, *Mobile Media Content and Services for Wireless Communications* (pp. 45-53). Taylor & Francis, Inc.
- Lyons, A. (2010). *The Future of Remote OBD*. Napa: CARB.
- Mahoney, S., & Keenan, J. (2008). *Creating a Wireless OBDII Scanner (Major Qualifying Report)*. Worcester Polytechnic Institute.
- McCarthy, M. (2005, September 13-15). Update on Light Duty OBD II. Pasadena, California, United States.
- Medagama, M., Gamage, D., Wijesinghe, L., Leelaratna, N., Karunaratne, I., & Dias, D. (2008). GIS/GPS/GPRS and Web-based Framework for Fleet Tracking. *National Conference on Geoinformatics Applications Sri Lanka*.
- Meschtscherjakov, A., Wilfinger, D., Scherndl, T., & Tscheligi, M. (2009). Acceptance of future persuasive in-car interfaces towards a more economic driving behaviour. *1st International Conference on Automotive User Interfaces and Interactive Vehicular Applications* . New York: ACM.
- Mohan, J. V., & Balan, M. (n.d.). Fleet Management System (Major Project). College of Engineering, Munnar.
- Muruganandham, P. M. (2010). Real Time Web based Vehicle Tracking using GPS. *World Academy of Science, Engineering and Technology* (61).
- Perez, A. J. (2009). *The Use of Onboard Diagnostics to Reduce Emissions in Automobiles (Bachelor's Thesis)*. Massachusetts Institute of Technology.
- Plandor, D. (2008). *Automotive Diagnostics Using Bluetooth Interface (Bachelor's Thesis)*. Technical University of Ostrava.
- SAE. (2009). *SAE On-Board Diagnostics for Light and Medium Duty Vehicles Standards Manual*. SAE International.
- Texas Transportation Institute. (2010). *Urban Mobility Report 2010*. Texas Transportation Institute.