CloudThink and the Avacar: Embedded Design to Create Virtual Vehicles for Cloud-Based Informatics, Telematics, and Infotainment

by
Joshua E. Siegel

S.B. Mechanical Engineering (2011)
Massachusetts Institute of Technology

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

Master of Science in Mechanical Engineering
at the
Massachusetts Institute of Technology

June 2013

Signature of Author: Signature redacted

Department of Mechanical Engineering
May 10, 2013

Certified by: Signature redacted

Sanjay E. Sarma
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: Signature redacted

David E. Hardt
Chairman, Department Committee on Graduate Theses

[This page almost left blank. Almost.]

CloudThink and the Avacar: Embedded Design to Create Virtual Vehicles for Cloud-Based Informatics, Telematics, and Infotainment

by
Joshua E. Siegel

Submitted to the Department of Mechanical Engineering on May 10, 2013 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

Abstract

This paper describes the development and testing results for a system of Cloud-based mirrors for physical vehicles called "Avacars." Avacars duplicate parameters from On-Board Diagnostics, accelerometers, and GPS sensors installed in a vehicle as part of the MIT CloudThink set of standards. These Avacars may then be used as input and output for a diverse set of applications. Avacars are created by a custom-designed portable cellphone used to instrument a vehicle without user intervention and stored to a secure and private server.

The first section of the document details the background for the Avacar project. It describes available technology and current unmet needs, and presents the solution of an open-standard based application platform for improving access to vehicle diagnostic data and creating new opportunities to build applications.

The second section explains the need for an open platform in the context of end-user and developer feedback along with canonical application examples including vehicle-miles-traveled (VMT) monitoring and generation of fuel metrics to validate programs similar to the United States Corporate Average Fuel Economy (CAFE) standards. This section also explores the value of open and interoperable data as well as transparency in hardware design.

Section three describes the implemented hardware and novel features facilitated by the hardware, including power saving and location-aware application development. This section includes an analysis of the problems faced in the design and deployment process, as well as steps the author might have taken to address these issues prior to their manifestation.

Section four discusses the results of the hardware and platform in testing, and includes visualizations of data collected with the CloudThink platform. The author found that the hardware and platform were capable of addressing the needs of both VMT and fuel economy monitoring applications, though further testing is necessary to validate the results. The author also successfully utilized the platform to extend applications to incorporate non-OBD vehicle sensors and actuators. This allows for the creation of large datasets while providing value to users who chose to test the system, in the form of car applications that repackage information into digestible formats or adding features otherwise not typically available, *e.g.* unlocking from a cell phone.

The paper closes by exploring future use opportunities for the CloudThink platform in monitoring non-automotive sensor enabled devices.

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

Acknowledgements

Thank you to Professor Sanjay Sarma for providing me with guidance and support throughout my Master's research. Thank you also to my family and friends for their support throughout my graduate school experience, especially my parents David and Sharon Siegel, brother Ben Siegel, and girlfriend Katherine Palusci. I additionally wish to thank Dylan Erb for his time and dedication helping design the hardware enclosures and thinking through problems with the power saving routine.

Table of Contents

Table of Figures

1. Introduction: On-Board Diagnostics, the Internet of Things, and the Need for Data

This thesis is about the design, development, implementation, and testing of a platform of "Avacars," or digital mirrors of physical objects (automobiles) for use as input and output for use with an intelligent API supporting the development of web-based applications. Avacars are the first step to proving the utility of digital duplication as a means of building Cloud-centric application platforms, and a demonstration of how elastic computing can power interoperable data caches. They are the first deployed example of a new program called "CloudThink" in development by the MIT Field Intelligence Laboratory, a collection of open standards describing everything from communication between sensors and servers to security to application programming interfaces (APIs). CloudThink aims to unify access and power the development of next-generation Internet of Things (IoT) devices along with cultivating the generation and analytics of Big Data.

There has been growing need for a well-documented, open-standard based approach to digital object mirroring. Further, there is an unaddressed and growing demand for vehicle diagnostics and informatics.[1] This thesis continues the work of the author's undergraduate research in an attempt to address both demands and solve several problems faced by the Internet of Things and vehicle diagnostics industries today [1]. To understand the need for innovative new technologies in these fields, it is necessary to understand the wealth of information present in vehicles, the complexity required to access even basic diagnostic parameters, the state of enabling technologies, and the expected consumer interest.

This report is organized as a background discussion of in-vehicle networks, attempts at standardization, barriers to deployment for the Internet of Things, followed by discussion of the design methodology, testing, and lessons learned while building and testing CloudThink's canonical vehicle interfacing hardware (the CANPuter), embedded software, and server platform. The document closes with a look toward future

[1] Rev: http:/devtoaster.com/products/rev/, Automatic: http://automatic.com/, Torque: http://torque-bhp.com/, Right to Repair: http://righttorepair.org

opportunities to leverage the CloudThink framework to drive Big Data generation and analytics.

1.1 Increasing Efficiency Demands Complicated Vehicles: A Solution to Diagnose and Troubleshoot Problems

Fuel economy became a national issue in the United States after the energy crisis of the 1970's. Fuel shortages demanded more efficient engine operation, while a previous push toward vehicle size and weight increases drove power requirements higher and posed a threat to auto manufacturers' abilities to meet new efficiency targets [2]. The historic solution to improving power had been to increase fuel delivery and engine displacement, but this increased fuel consumption would contrast with the federal government's 1975 Corporate Average Fuel Economy (CAFE) standards [3]. Since carbureted engines were incapable of meeting these targets, manufacturers instead implemented fuel injection and deployed computers to allow the use of multi-variable fuel maps to improve efficiency without sacrificing power. A number of suppliers manufactured these engine control computers, and per-manufacturer On Board Diagnostics (OBD) followed shortly after to diagnose computer and emission related issues (see General Motor's Assembly Line Diagnostic Link [ALDL] as an early example). These OBD systems allowed neighborhood mechanics to diagnose these more complicated systems without special training. While these systems began to address the needs of mechanics and consumers for diagnostic data, no single system was well defined, and many cars required special software, adapter cables, and hardware interfaces to complete even the simplest diagnosis.

The Society of Automotive Engineers (SAE) standardized a generic form of OBD in 1988, and the California Air Resources Bureau (CARB), recognizing the utility of a universal process for testing vehicular emissions, mandated a specific implementation of the diagnostic interface on all vehicles sold in the United States beginning in 1991 [4]. Automobile manufacturers took this redesign mandate as an opportunity to deploy additional sensors and actuators while the vehicle architecture design was undergoing rework in an attempt to catch up and future-proof. As was the case with early diagnostic systems, manufacturers created their own proprietary network protocols [5]. This made the limited set of legislated data difficult to capture, requiring an array of tools, adapters, and software packages. Although OBD has evolved since 1988, key information remains obfuscated, though market forces are driving towards a convergent diagnostic standard.

1.1 OBDI: A First-attempt at Computerized Vehicle Diagnostics

OBDI's migration from analog to digital signaling simplified the automotive wiring harness by eliminating the need to run miles of wire. No longer would every new sensor require two or three wires feeding it; instead, data were multiplexedand early addressing and decoding schemes developed. The use of networked digital meant that data never before sensed as part of a vehicle's network could be made available to the dealership and mechanic as live or stored sensor readings. Though limited storage was available and many sensors could not be read in real-time, this move also allowed for further integration of vehicle computer systems and marked the beginning of the now-ubiquitous vehicle intranet.

OBDI was a minor success for emissions-monitoring purposes despite the fact that every manufacturer offered a unique implementation. New data were available, but a neighborhood mechanic could not keep up with the volume of software and hardware required to analyze and service every make, and more often than not, would become frustrated with new vehicles and troubleshoot without making use of OBD. There was a need for a new generation of standardization, termed On-Board Diagnostics II (OBDII). By design, OBDII would provide a more unified method for accessing emissions data and would have a more structured deployment, limiting the freedom manufacturers could take in defining their networks' hardware and software. OBDII would also bring with it more advanced component monitoring, eschewing threshold checks for active sensor analysis [6].

1.2 OBDII: The Logical Progression Toward a (More) Unified Diagnostic System

OBDII was a joint effort of the California Air Resources Board, the Environmental Protection Agency, and the Society of Automotive Engineers. It built upon OBDI's near-binary parameter reporting and defines several modes of data access ranging from live and freeze-frame data to descriptive trouble codes. OBDII documentation defines standard methods for accessing emissions related parameters along with a wide list of parameter definitions that are standardized but for which implementation is not required [7]. Manufacturers were also encouraged to include additional Parameter IDentifiers (PIDs) not defined in the standards description, but these manufacturers would not have to disclose their proprietary identifiers and scaling factor, making their implementation useless without costly database licenses [8]. This lack of freely available enhanced data made it difficult to perform complex diagnostics and limited the utility of the legislated data to basic visualization or simple diagnostic routines not significantly more advanced than those available with OBDI. While interesting parameters – such as fuel level, steering wheel angle, or accelerometer data – are likely present on a network, the lack of legislation surrounding these metrics limits their utility in enhancing informatics.

OBDII is a call and response protocol typically built upon a form of serial data transfer defined in SAE standard J1979. In many implementations, a user enters a *mode* and *parameter* into a scan tool and this tool sends these data to the network, awaiting a response. Devices on the network recognize responsibility and transmit a response for display on the device. Modes $01 (live data) and $02 (freeze frame data) rely upon a documented list of parameter identifiers and associated scaling factors to report data. Replies may include proper data, a null data identifier, or an invariant response, depending on the computer architecture and error handling implemented in the system [7].

Despite the wealth of documentation surrounding OBDII, the data remain difficult to access. It appears that mechanical engineers played a large role in the development process, and OBDII was built around legacy systems from early automotive networks where data rates were low and Micro Electromechanical Systems (MEMS) devices and flash memory were not available. There was a lack of foresight in design: for example, call and response parameters must be read sequentially – there are no multi-sample requests, leading to a constantly active data bus when attempting to monitor several parameters. Data transfers are serial rather than parallel, on-board storage is minimal, and there are no provisions for increased storage or easy diagnostic system upgrades as new sensors are invented and deployed. There are no true learning algorithms for predictive failure reporting, and things that developers would expect to be available universally – such as the reading on a fuel gauge – seem to have been afterthoughts. Further, many devices on the network that are not emission-related are simply inaccessible using OBDII, despite these sensors sharing a common communication interface and the presence of sufficient bandwidth to provide these data [7] [9].

Part of the reason for the difficulty in accessing data follows manufacturers' different OBDI implementation. Early OBDII hardware had to support five distinct hardware and software implementations of in-vehicle networks, ranging from Pulse Width Modulation (PWM) signaling to serial data streams to Controller Area Networks (CAN). A vehicle could have any of the five networks, making diagnostic hardware costly, cumbersome, and often unreliable due to complexity.

In the United States, as of model year 2008, the complexity of accessing diagnostic data was reduced. The standards still did not require the availability of non-emissions parameters, though there has been progress in that only one network may be used for diagnostics in newly manufactured vehicles: CANBus [10]. The use of a single network simplified data access, provided higher speed communications, and ensured the availability of less expensive hardware through economies of scale. CANBus was well documented and provided a reliable supply chain for hardware developers, though there were limitations in that CANBus followed the outdated OBDII standard (however, with CANBus more than any other installed network, there is hope that other sensor data might be present and easily readable due to the higher bandwidth and robust addressing scheme).

To expand on that claim, many automobile manufacturers deployed additional sensors and actuators when CANBus was required with the intent of leaving the majority of the intranet infrastructure unchanged for years to come. This design strategy was largely effective, and today, modern vehicle computer modules run millions of lines of code and complete millions of calculations every second performing everything from stability control to preventing skidding and maximizing fuel economy and power simultaneously.

1.3 Non-OBD Networks: What, Where, and How

The network found in vehicles today contains significant non-emissions data [9] [11]. The engine computer is one of dozens of computers in a car, reading sensors, logging data, and handling more and more of the task of keeping a vehicle in good operating shape. The typical new car has computers, hundreds of sensors, and actuators for everything from checking to see which seatbelts are buckled to measuring the barometric pressure and adjusting fuel mix accordingly. These devices communicate across one of several networks, depending on their bandwidth requirements and input/output needs.

A typical vehicle might have a high-speed, safety critical network and four to five other networks handling less critical tasks. These networks never sit idle, with engine computers talking to gauges, security modules interfacing with locks, and parking sensors communicating with navigation units, among other connections. Sometimes, communication occurs across networks by means of a gateway device, which has been programmed with special rules and conversion hardware to allow data to seamlessly flow from one location to another. This device keeps networks largely standalone and isolated from one another for security purposes and to allow networks to operate without crosstalk preventing critical communication. A gateway device is the vehicle's closest analog to a firewall. Every car is different, except for the high-speed bus's central functionality, which is locked down as the arbiter for OBDII [12].

A typical vehicle features dozens of computers residing on one of several subnets, which these subnets consisting of a high-speed, "critical" network, a medium-speed infotainment network, and a low-speed "comfort and convenience" network [13] [14].

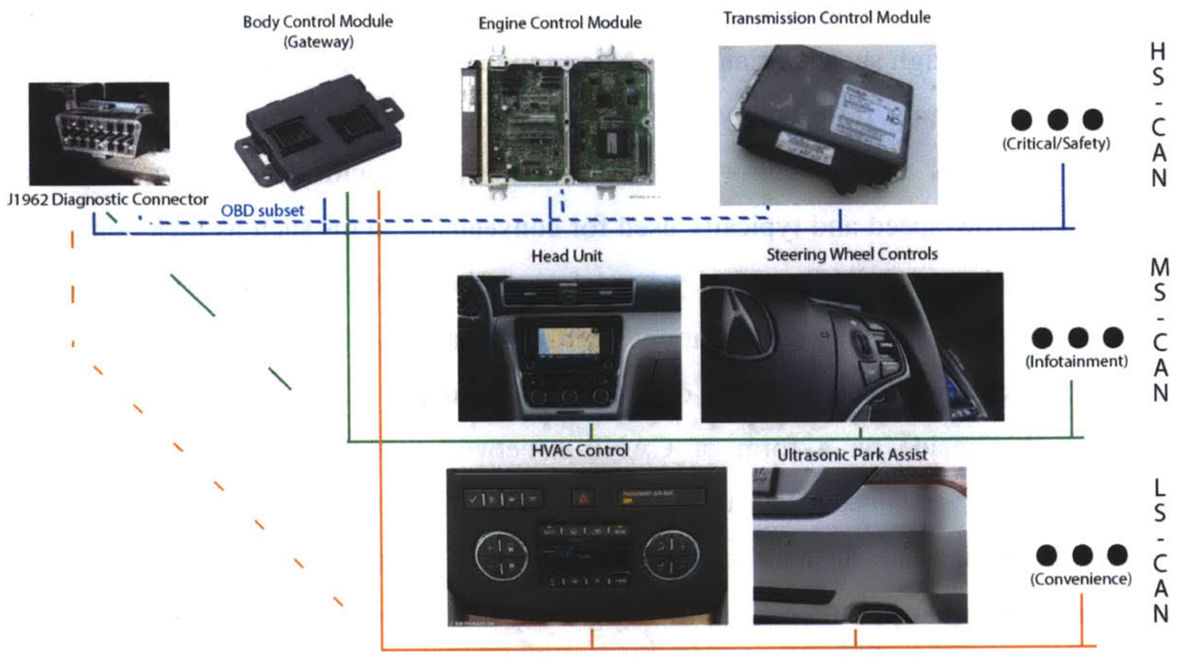


Figure 1: *A representative image depicting the three major networks found in modern vehicles, a High-Speed (HS) network for "critical" data such as OBDII, a Medium-Speed (MS) network, typically for infotainment, and a Low-Speed (LS) network, typically used for comfort and convenience features (C&C). Dotted lines indicate discretionary connections.*

Some common automotive networks are similar to TCP/IP connections. Generally, these intranet protocols define data validation routines in conjunction with an arbitration scheme to specify message importance to ensure high importance messages are received by their intended recipients and that messages are not flooded to the point that they cannot transmit on a particular bus. The devices on this network operate like various web interactions, either broadcasting at constant intervals (like a keep-alive packet), transmitting upon event (or state changes, like button presses), or are set up as a call/response to serve up the most recent data. This last configuration is similar to (Asynchronous Javascript and XML) AJAX in that it requests asynchronous broadcasts of data across sources for use and display at another node's location.

The most commonly found networks in vehicles are Controller Area Network (CAN), Local Interconnect Network (LIN), and Media-Oriented Serial Transfer (MOST). CANBus has multiple single- and dual-wire variants and is known for its high maximum bandwidth, well-defined arbitration schema, and intelligent addressing scheme (OBDII uses a dual-wire, high speed variant). CAN is used in every modern vehicle and has been applied to everything from safety-critical systems like brake controllers to passenger infotainment. LIN is a single-wire, UART-like master/multi-slave network. These networks are low-speed and typically used for convenience items such as lock control or window switches. MOST is typically used for media devices and in very high-speed networks but is not used in other locations due to its relatively high cost.

Due to the unification and de-duplication of design by automotive suppliers, many sensors today reside on a form of CANBus network and harbor the possibility of providing richer diagnostic data to compatible tools. Regardless of network, however, non-emissions data are difficult to access reliably and repeatedly without manufacturer help – and auto manufacturers are unwilling to share, having come to see software as a key brand differentiator and fearing safety and liability concerns – even though these networks lack encryption and are viewable with inexpensive tools. Access to these data will ultimately require documenting manufacturer-specific software and the development of custom hardware to access this wealth of data.

1.4 CANBus Explained: Unified Access for Next-Generation Sensors

CANBus, or the Controller Area Network (bus), is the most broadly deployed in-vehicle network and the basis of OBDII for all vehicles manufactured after 2008. CANBus is notable for its low cost, dual wire signaling, arbitration schema, and the fact that it does not require a host microcontroller to negotiate network traffic thereby reducing cost and software complexity. CAN is a multi-master broadcast serial bus.

A CAN packet consists of an arbitration ID, defining the priority, and perhaps the source and destination address (*e.g.* a sensor and an actuator or control module), along with up to eight data bytes that are sent serially. CAN has four typical data frames: data, remote, error, and overload frames.

A **data frame** is either base or extended format, with 11- and 29-bits, respectively. These frames contain a start-of-frame, identifier, transmission request, idle, reserved bit, and length bit, followed by the data field from 0-8 bytes. A CRC15, ACK, and EOF series of bits ensures the data were transmitted properly. If messages are longer than 8 bytes, ISO-15765-2 multi-frame messages are sent. The first frame includes a header identifying the start and total message length, and the transmitting node waits to receive an acknowledgement of a multi-frame reply prior to transmitting beyond the first frame [15].

A **remote** frame is similar to a data frame, but is typically sent by a sensor at a regular interval or on an autonomous scheduler. A data frame is dominant to a remote frame.

An **error** frame defines an error message that may trigger other events, such as a delay or a retransmission. The error frame also increments a counter to help identify systemic failures in hardware and software, when counts climb quickly.

An **overload** frame request notifies a node that the message was not processed, as the receiver was not ready to handle the incoming data. The difference between an overload frame and an error frame is that the overload frame does not request retransmission of data and does not increase the error count. The event is "lost."

In a CAN network, any node may send or receive data, though not at the same time as any other node. This is because all nodes share a serial interface, and therefore all data is broadcast to each connected node simultaneously. A node may begin to send when the network is silent, while a priority header determines which message proceeds in the event two messages are transmit at the same time. This is accomplished through the use of two types of bits, a dominant (0) and recessive (1). These bits are operated upon with a digital "AND" operator, so that any node conflicting with a high priority message is able to immediately determine a collision during the transmission of the arbitration ID and let the dominant message finish, ensuring no delay in transmission and no requirement to retransmit the dominant/critical message. A dominant, high-priority message therefore has a low-value arbitration ID [16].

The CANBus network is unencrypted, so it is accessible to monitor or transmit onto freely. This enables easy study of existing in-vehicle networks and provides a wealth of data insight into vehicle operating state.

1.5 Consumers are Unsatisfied with OBDII Hardware, Diagnostics

There is a market for diagnostic data, but consumer needs are not being met by present hardware and software solutions. OBDII hardware is constrained by the standards and must operate within the confines of federally mandated data, while more advanced fleet management systems are limited primarily to GPS-based tracking with the possible inclusion of software geofence triggers and require complex installation. Mechanic-style diagnostic tools are improving and able to show more compelling live and recorded visualizations of data though are still lacking in actual, value-driven content generation and "smart" diagnostics, and cost thousands of dollars – on an annual basis. Bluetooth OBDII interfaces for mobile devices have great supporting apps like Rev and Torque, but there is a fundamental problem with their communication technology: Bluetooth requires users to bring their mobile device at all times and prevents the driver from pairing the phone with a second Bluetooth device on older devices. The potential for "man in the middle" attacks is also higher than with other network options, and data throughput is limited [17]. A few platforms are beginning to scratch the surface of using manufacturer-specific diagnostic data and actuation control, namely OnStar's new Open API and Ford's OpenXC[2]. However, these platforms are fundamentally flawed as they remain closed silos (data remains within the vehicle), APIs are on a per-vehicle/per manufacturer basis, and mobile device interaction is limited. These silos inhibit interoperation, real-timing and crowdsourcing which prevents the capture and analysis of truly Big Data.

All of these tools are either for individual vehicle diagnostics or informatics, or the few web-based systems keep the data reined in – nothing is sharable, everything is locked into a single platform, and the user is at the whim of the service provider which ultimately leads to a poor user experience. These devices only make use of the most basic data from vehicles, ignoring the recent easy access of other sensors and connected devices (like MEMS accelerometers or GPS receivers), and therefore generate poor data sets even before being crippled by the lack of data portability.

[2] http://openxcplatform.com/ and https://developer.gm.com/

The author's undergraduate design thesis ("Design and Validation of a Remote Telemetry Unit") attempted to address these problems by creating hardware with a direct, always-on GPRS connection to allow self-reporting and remove user reliance on a secondary device. However, this system had significant shortcomings due to the complexity of the software and the lack of user-friendliness which caused problems with software task scheduling, raised concerns about data security, suffered from poor power management and had insufficient processing for thick clients where data abstraction calculations are done onboard. Later versions, immediately following the publication of the thesis, explored non-OBD parameters on the primary CAN network (not yet tapping into secondary or tertiary networks) and failed to find value in increased data availability or actuation possibilities due to limited data and the lack of an easy-to-use API. The previous development was a step toward self-reporting sensors driving open data, but between the hardware errors and the lack of a development platform for logged data, suffered limited utility. The system did not make logging vital vehicle statistics easy and transparent – it was clear there was a device in the car, which challenged gathering true use data, and the bugs in the connection routine led to a requirement for constant supervision. Unlike Facebook, the value of a sharing system for diagnostic data comes from the "ugly pictures" – as these provide the most compelling insights. None of those events get captured if the user unplugs the device or if data capture fails due to software errors.

1.6 End-users Demand Extended Diagnostics

Despite commercial attempts to address data needs, there remains an unmet demand for an open diagnostic platform. These needs for data emerge from scientists and fleet managers, a consumer need for improved telematics, and an increasing consumer desire to utilize data (demonstrated by the significant push to pass Right to Repair, which seeks the publication human readable diagnostic data and manufacturer-specific "decoding" parameters) [18].

Fleet management is a growing market. Beyond the obvious truck, taxi, and police monitoring systems, there is a consumer demand for fleet management. Salespeople, realtors, and consultants, for example, benefit from fleet tracking with improved analytics and audit trails to ensure maximum tax deduction for work-related travel. Existing services require complex hardware installation and are therefore not portable and expensive to operate. These devices are also "single function" and the data are stuck within the application.

Telematics is big business, and aftersales become an increasing source of revenue for auto manufacturers, illustrated by Verizon's acquisition of Hughes Telematics [19]. The market is growing by leaps and bounds, and nearly 16% of vehicles sold in 2013 have factory-installed telematics systems [20]. Here again, these data are siloed and stuck within a single location. Consumers lease the use of their data from service providers.

Vehicular congestion and concerns over fuel reserves, pollution, and carbon emissions have emerged as prominent sociopolitical issues. Fuel economy concerns are more pressing now than ever with increasing fuel prices. Small vehicle sales are increasing and more economical engine choices, like Ford's EcoBoost, have proven incredibly successful. The introduction of a family of Prius vehicles, the Nissan Leaf, the Tesla Model S and the Chevy Volt also indicate a growing demand for highly efficient vehicles [21]. 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.

Consumer-facing application sales are also growing rapidly, with the proliferation of OBDII software sales for mobile devices, such as Torque for Android, and Rev for iPhone. The data for such applications remains tied to the phone with which the interface device is paired, preventing crowdsourcing and making real-time data difficult. With the recent passage of Right to Repair in many states, consumers will continue seeking to liberate vehicle diagnostic data for consumer use on any and all devices for which they have access – and removing digital boarders and claiming ownership to data generated will be the first steps.

1.7 The Cloud is an Enabler: Next-Gen Diagnostics Meet Elastic Computation

Many of the applications sold today do not meet expectations due to the failure to leverage data across platforms or clunky user experiences. However, the Cloud is an enabler to connecting computers, sharing across devices, and providing affordable, scalable computing for complex computation.

Imagine a user sitting at a desk supporting a monitor, keyboard, and mouse. There is no desktop computer, just a small box with a WiFi antenna stuck to the back of the monitor – and yet, the screen looks like a desktop is connected, and the software on screen reacts as responsively as if a workstation were connected. This computer has no real processor, but is able to access storage and vast processing power from the Internet. Now, the user needs to relocate, but she cannot take her monitor with her. She walks to her meeting and pulls a portable, tablet-sized display out of her bag. She turns on her tablet, logs in, and the exact same workspace from her monitor appears on the tablet display. She continues working with the documents she had open on her desktop monitor, as if she had not left at all.

This service demonstrates the key concepts motivating the Cloud. The Cloud connects computers together using central storage, and facilitates the use of programs no one computer could ever hope to run through the use of elastic computing platforms that scale to meet demand.

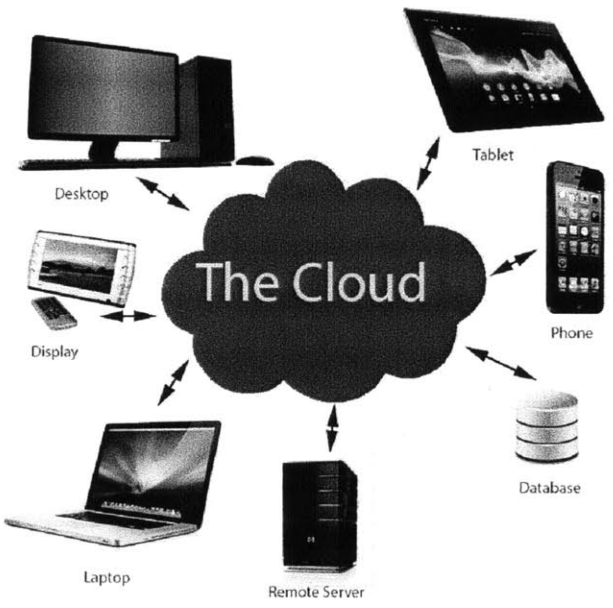


Figure 2: *Cloud client devices connect to servers located around the world to perform complex tasks inexpensively*

The Cloud is here today, and companies are already enjoying its advantages: infinitely scalable storage, the ability to ramp up computation speed, and paying only for the processing power necessary to complete a job. Average consumers also enjoy the benefits of the Cloud, sharing their files across devices or playing a game or streaming a movie. The Cloud, in essence, is a way of having an infinitely large, always-ready IT department. It is similar to leasing computers, but clients get to enjoy the latest technology with economies of scale possible only when a single provider purchases the aggregated computing power necessary for thousands of users at once.

The Cloud has its own language. The client is the end user's device. That device relies on a service as a means of machine-to-machine (M2M) interaction. The service facilitates infrastructure, a platform, or software. The infrastructure simply provides machines – network switches and servers, for client applications. Platforms deliver operating systems or preconfigured hosting environments. Software delivers a particular application to the user. Storage handles data delivery between clients. All of these models provide significant value, and ability to horizontally scale that no other computing setup can come close to [22] [23].

The Cloud is an enabler for all things connected and all things Big Data, and thrives in the presence of open standards. It is an exciting technology that helps consumers realize their dreams of having data and computation power accessible anywhere and for any purpose at a moment's notice. When using a Cloud server, data storage and processing capabilities are limitless – and the data are without boundaries, accessible from any device.

1.8 Internet of Things: Connecting Devices, Once Standards Emerge

The Internet of Things (IoT) is a technology that is somewhat parallel to the Cloud. The IoT describes a network of connected "smart" devices, leveraging pervasive computing to improve quality of life and deliver value to consumers. A refrigerator that talks to a microwave while paired with a smart phone to make a grocery list and plan dinner is a great example of an IoT network. Cloud computing has made the IoT feasible, and instrumenting devices by turning them into sensors enables the capture of rich data and new supporting applications [24].

With the Cloud, MEMS sensing, flash memory, and other recent technological developments, the timing is right to start deploying new IoT software, hardware, and services. However, most of the recent advancements in IoT development have been incremental, failing to succeed due to a lack of standardization in development processes.

The most intriguing uses for networked devices cannot be realized until developers have an understanding of user needs and until standards are in place. The IoT cannot be ready for primetime until the goals of IoT development are well understood and a road to implementation has been standardized.

1.9 A Combined Approach: Cloud + Car = Digitally Mirrored "Avacars"

An ideal solution to the problems described is an open platform for data generation and analysis. Developing a platform of hardware and software addresses this latent demand for vehicle diagnostic data, and democratizes the utilization of these data in the process. Leveraging the common implementation of CANBus in vehicles today, it becomes possible to build digital duplicates of vehicles in the cloud, mirroring all key parameters of these physical vehicles and storing them in near real-time on an elastic computing platform. Using "plug and forget" hardware, every car may be duplicated as a "virtual vehicle" in the cloud, a digital mirror of all the key parameters of a physical vehicle stored in an easily accessible manner that may also be shown as human- and machine-readable. A vehicle-to-cloud standard would define a means of storing and accessing vehicle data on an elastic computing platform, standard security practices, a communications protocol, and canonical hardware for bridging On-Board Diagnostic data to CloudThink databases, answering the call of collecting and processing data and also creating a platform for others to build unique and compelling applications.

The automotive avatar created by CloudThink could become an input/output for various applications, importing real-time and historic data for visualization, analytics, and even infotainment purposes. Without a platform like CloudThink, basic parameters such as those required for fuel economy monitoring could not be logged. These cloud mirrors are all stored on separate virtual drives and are the evolution of the programmatic object, serving as input and output for applications running on web-enabled devices. This concept is fundamentally different than the approach taken by others, as it relies upon an open standard and provides a secure deployment of data that will be accessible from any mobile device. A depiction of the broad architecture appears in **Figure 3**.

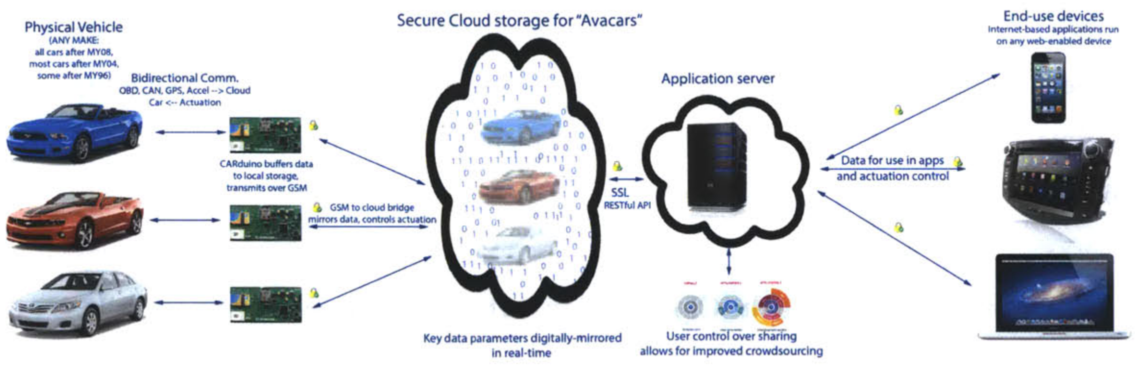


Figure 3: *Graphic representing the cross-platform, open standard for digital object duplication in the cloud and resultant applications.*

The design report that follows builds upon the author's MIT undergraduate thesis and addresses the major pitfalls with a better understanding of consumer needs, standardization requirements, and how to develop an open platform that is scalable, interoperable, and above all else, useful.

2. Design Process, Needs Analysis, and Canonical Use Cases

This section explores the design process, user and developer needs analysis and canonical data use cases that set the stage for the development of the CloudThink platform and Avacar mirrors. These needs emerged from analyzing the unmet demand for diagnostic data in Section 1, intraoffice discussion, collaboration with project partners, and interviews with third parties about what they would like to see in a platform for digital object duplication. The author also spoke with potential lead users who had used existing solutions in a similar field to find out their reactions and solicit feedback about the platform.

2.1 Identification of Platform Needs: Consumer Insight Meets Engineer Intuition

The introduction set the stage for the CloudThink platform and the Avacar virtual vehicle. OBDII provides some diagnostic data that can be expanded upon with intelligent hardware and software solutions, and there is consumer demand to drive the development of a smarter platform for capturing, storing and serving these data. In analyzing the Field Intelligence Lab's IoT development needs of data generation, capture, and analytics, existent technology could meet these needs separately. In-vehicle OBD and MEMs sensors generate data, GPRS and Bluetooth devices capture data, and existing visualization applications demonstrate analytics. However, these technologies are separate and distinct – OBDII cannot read all sensor values, analytics have limited data sets and are not cross-vehicle. Therefore, seeking to understand why this disconnect exists would be paramount to extending existing technology into a data mirroring platform.

Writing user need statements led to several key ideas. Any deployment must be **user-friendly**, for the general public to use. Would a grandmother be able to identify if the device or system had failed, and how might she be able to report or address that failure with minimal knowledge of the subject? The system would have to be **reliable** in recording and reporting, ensuring the accuracy and timeliness of data for applications making real-time use of data. Concerns about privacy led to a system requirement of being **protective** – who can see where a driver has been, and for what purposes? Finally, any system would have to have provisions for **security**, to prevent fraud in cases where data might be tied to financial resources or other services requiring auditability.

From the perspective of hardware and software developers, a different set of needs emerged. Some ideas were similar – like resistance to hacking and ease of installation. Others built upon the user needs, for example, low cost to drive ease of use and replaceability, or the need for data addressability, leading the requirement for vast onboard storage. Still other concepts were aimed at creating value through connected devices, like demanding real-time data access (requiring always-on wireless connectivity) or the need for a robust, open API to abstract complicated data acquisition. These needs would drive the creation of a platform for data collection and application development, facilitating the development of eco-aware, social, infotainment and maintenance applications, among others.

Thinking about the diagnostic and data capture systems available today, a major problem is universal access. Without universal accessibility and data standardization, data must remain where it is generated. An open standard would drive toward universal access and cross-platform capabilities, inviting hardware designers, software developers, OEMs, and networks to participate as partners at various stages in development, deployment, and support.

To ensure data would be used as broadly as possible, a common database was necessary. A Cloud-centric platform would facilitate easy application development owing to central database storage and the ability to perform complex analytics server-side. Leveraging the Cloud, any web-enabled device could be a sensor, and any other web-enabled device can be used to visualize data and informatics – not just the ones developed in this document. This free-flow of data across platforms and devices would drive improved acquisition, analysis, and visualization by removing digital borders and providing transparent access to key parameters.

These open standards could then lead to real-time applications capable of leveraging the power of elastic computing. Combining real-time data with crowd sourced analytics would drive toward the creation of tools for visualization that convert the data generated from distributed vehicle sensors into insights that would have been previously impossible to discern. Crowdsourcing leads the way for more intelligent applications, and with liberated data, more bright minds solving today's most challenging problems.

The combination of manipulable data and standardization would drive a positive user experience and deliver value in the form of rich datasets for analysis. However, end users have typically been wary of data scraping systems, and for good reason. In any implementation, addressing security and privacy concerns are critical. To provide the best user experience, users must be allowed to own their own data, opting-in only when they choose to share, and sharing only the data they wish to disclose with clearly understood partners.

To keep users happy, any system would have to bypass typical machine-to-machine (M2M) and machine-machine-cloud (MMC) rollouts and instead favor machine to cloud to machine (MCM) reporting. This would bypass user reliance on middle devices and yield richer datasets while improving robustness against man in the middle attacks. The MCM configuration does fail when it comes to latency for single-user applications, however, the portability of data and common database benefits outweigh these concerns. The implementation of a secondary network interface (*e.g.*, Bluetooth in conjunction with GSM) further ameliorates this problem.

These need statements were disparate, but addressable by a simple solution summarized in **Figure 4**. All could be met by the creation of a data-centric application platform based on digital duplication of physical objects created by always-on black-boxed hardware. With this system in place, applications and informatics could become "smart" in that they could begin to infer, rather than assume.

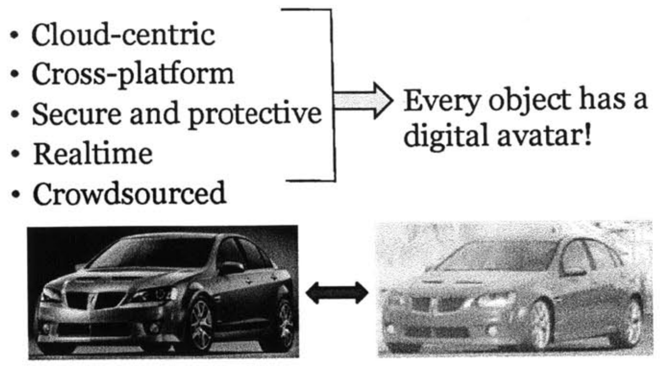


Figure 4: *A distillation of key need statements for digital vehicle duplication*

2.2 Vehicle Miles Traveled Monitoring: A Canonical Problem to Validate CloudThink's Avacar Mirroring

Any platform development project requires a test case to prove whether it is an improvement over existing technologies. In the case of CloudThink and Avacar mirroring, one such example looks at the use of a platform for measuring Vehicle Miles Traveled (VMT) and extension toward VMT logging for auditability and tax purposes. This problem is real and an excellent test case to evaluate the efficacy of the platform, as well as a prime example capable of shaping the design guidelines as it addresses the primary concerns of reliability, security, ease of use, and ability to drive value. Exploration of this topic best begins by understanding the need to monitor VMT.

Since their introduction as an import tax during the Hoover administration, gas taxes have remained a hotly contested topic. In 1932, the United States federal government was facing the prospect of a severely unbalanced budget. President Hoover was exploring options for the Revenue Act of 1932 in order to locate income sources to avoid bankruptcy and foster broader economic growth through infrastructure projects. At the time, a fuel tax seemed an effective and minimally invasive solution – fuel use was increasing not only among industry, but also in the consumer sector due to the advent of modern vehicles. Seizing this opportune moment, many states began to levy their own fuel taxes (Oregon was first to levy this tax in 1919, while other states and the federal government took decades to implement due to bureaucratic red tape).

Despite the gas tax's efficacy in addressing budget shortfalls, some consumers resisted the idea for decades after its introduction, claiming that these tax revenues did not directly benefit the purchasers of fuel. These citizens believed instead that fuel tax revenues went to broader federal programs rather than into maintenance coffers. Irrespective of this attitude, sociopolitical circumstances ensured the bill was renewed time and again, despite the negative public sentiment. Finally, during the Eisenhower administration, the interstate highway system solidified the gas tax's permanent role in American society by concretely demonstrating a compelling use case: a comprehensive, federally run road system [25]. State-elected officials saw the benefits of having an interstate road system, and moreover, saw the benefit of having the federal government pay for maintenance of this infrastructure project from a single central fund. The sheer scale of the interstate infrastructure project made federal fuel taxes significantly more visible targets than state level taxes, but also addressed the big question of tax equality across state lines – now, paying into the fund would ensure functional roadways for all drivers. This shift of focus from state to federal tax, along with the rationally presented use case was effective in making dissent against fuel taxes unattractive. The interstate system was a champion of garnering support for a federal fuel tax, which ultimately would contribute to interstate economic growth. Today, state and federal fuel taxes are one of the primary sources of road infrastructure funding in the United States. Despite this, many citizens today are wholly unaware of these substantial taxes and the present funding scheme.

The Congressional Budget Office (CBO) recently projected that the United States roadworks budget will soon become imbalanced due to overdependence on this tax, leading to serious budget shortfalls. Since the fuel surcharge is levied on every gallon of fuel purchased, government programs to subsidize electric and hybrid vehicles, as well as increasing fuel economy due to more stringent CAFE requirements, has resulted in diminishing revenue while total passenger miles travelled increase year over year (Wolfram Alpha Knowledgebase). This problem is not insignificant, as CBO estimates peg road maintenance revenues at two cents per mile while costs are 10 cents per mile [26]. For a single driver in the United States traveling an average of 12,000 miles per year on publicly maintained roads, this shortfall adds up to nearly $1,000 – and there are hundreds of millions of drivers in the United States! An alternative, "true use" metric is needed in the near future – one that ensures equitable financing of road maintenance based on the actual impact of every vehicle traveling on public roadways. These models will make use of new technology to factor in load-based wear and other negative externalities associated with driving.

Facing such a large potential budget deficit, government organizations and academic institutions have focused efforts on creating a more accurate way of accounting for true impact. There are many proposed solutions to this problem, some more likely to be adopted than others. The Business-As-Usual (BAU) solution would accept increasing travel, decreasing revenue ($10bn/year), an increasing reliance on driving, and a sense of entitlement as given [27]. This tact would require a shift from public road works to privatized toll roads, as our current approach is unsustainable financially. It remains to be seen whether this shift would have significant impact on drivers, but it likely would unless the institutional overhead for a private road maintenance group is appreciably lower than government overheads. Another alternative to capture maintenance revenue is a flat increase in fuel tax. An increased gasoline tax might increase revenues, but would be regressive (impacting poorer families more, as they spend proportionately a larger percentage of total income on fuel, and additionally, these families often cannot afford high-efficiency internal combustion or electric vehicles). A targeted, electric-vehicle specific tax (like an annual registration tariff) is another, less-regressive option, but faces the issue of inadvertently stagnating sales of highly efficient vehicles, which already have difficulty gaining market traction without subsidy. Depending on how policy is written, this may shift the cost of externalities away from road-maintenance and towards carbon capture and sequestration, leaving the original problem unaddressed.

A solution monitoring impact at all times is optimal when it comes to addressing revenue deficiencies in a "pay to use" manner. One "true use" tax is a "vehicle miles traveled" tax (VMT), which many governments and educated drivers find to be a reasonable compromise for ensuring revenue needs are met without severely impacting drivers' quality of life and requiring habit changes. Comprehensive VMT taxation is also considered to be an excellent proxy for cost of maintenance, as damage tracks directly with miles travelled for light vehicles, and factors such as vehicle loading can be factored in whereas gas taxes do not vary directly with weight [28]. A VMT tax is the next logical progression for a gas tax setup when most vehicles had nearly identical fuel economy. Put simply, "the gas tax inadvertently taxes people on their fossil fuel consumption, incentivizing people to drive eco-friendly vehicles — while a VMT [vehicle miles traveled] tax, would work to charge a car on its usage" (Ami Cholia, AltTransport).

There are many forms of VMT taxation implementation. Generally, VMT monitoring systems record miles travelled in a vehicle either using spot-checking (during fuel fill-ups, oil changes, or annual inspections) or real-time data transfer using a wireless network. These wireless models afford more options to bring value to consumers using the same hardware to monitor VMT and to run pay as you go (PAYG) apps (like pay-per-mile insurance or automated toll collection and parking payment), but open up questions about privacy, like how to secure data collected through location tracking and how to bill a user without requiring an interceptable, invariant identifier for each driver. These PAYG apps would maximize the social value from the investment in VMT measuring hardware, but without government coercion, would likely not directly result in adoption of VMT tax programs as a viable tax solution. These devices – thin client, which pass information directly to a server, or thick client, which does calculation onboard and only transmits aggregate metrics – can be more involved than a simple distance traveled tax, accounting for true impact based on vehicle loading, congestion factor, time of day, and other externalities using complex models and variable rate pricing. To be more equitable and auditable than current solutions, these same models factor in non-taxable travel, such as trips out of state or on private land, to allay consumer fears. Technologically, the time is right to deploy these fair "pay to play" taxes, but people are not yet ready to accept the devil they don't know due to a lack of technological understanding [29].

The State of Oregon has been a pioneer in seeking to understand VMT technology and related tax strategy. The state has conducted several, small-scale pilot studies with the intent of collecting information to better understand data collection methods and methods for calculating fair use fees. The state studies are evaluated using the following eight principles: users pay for actual use, the government controls local revenue sources, the system must be sustainable economically, the system must be transparent in its use, there should be minimal burden to non-government parties, the system must be enforceable, it must support the entire road system in the state, and the system must be accepted by the general public [30].

The Oregon trials met with success, but were not without obvious problems: consumers did not like the notion of GPS use, even if point data was never transmitted. Instead, drivers voiced a strong preference for low-tech versions that use a less-equitable, fixed per-mile pricing scheme and visual inspections or thick-client, aggregate data reporting [30].

The Oregon pilot programs answered all the questions within the context provided, proving that VMT measurement and fee calculation is feasible without causing pain points for drivers [31] [32]. The trials were a great success, but the consumer feedback received makes it clear that the road to full, statewide or nationwide deployment for VMT programs will be long and face challenges as the technology and policy grows beyond applicability to self-selected drivers. Oregon projects a minimum three-year trial period, while many other states say it will be five to ten years before even pilot technology is sufficiently well developed [33]. Only when the hardware is developed can deployment begin, though education programs could certainly begin sooner.

The study provides several valuable takeaways. First, understanding that third-party partners and PAYG applications are core to VMT policy adoption means that these applications can be used to accelerate this otherwise slow process. Building an open hardware platform, and choosing what to share, is an ideal solution for protecting consumer privacy while rolling out a system that consumers will actually want to install [31]. The study also proves that educated consumers will grow quickly into accepting VMT tax as an alternative to the fuel tax.

The studies did omit information that would have been valuable, such as consumer reaction and feedback (interview based or otherwise) to various rollout methodologies, as well as non-volunteer/compulsory survey results, results when consumers were not given an example VMT tax budget, and the role of information in driving consumer choices and resultant feedback (*e.g.* does the driver see VMT fees levied in real-time, or only at the end of the month?). The study also raises further questions, such as how to handle interstate VMT monitoring and revenue sharing, and the issue of privacy versus auditability (recently, Oregon proposed an interesting solution using a thick client sending aggregate usage statistics with a usage identifier, and a second, encrypted message containing the regional splits driven by the vehicle operator to remove identifiable driver data while still allowing for proper apportioning) [29]. Not discussed at length, but equally important, is the issue of anti-spoofing to ensure drivers cannot easily evade the tax so that drivers can rest assured their neighbors are also paying taxes. Here, an open platform can ameliorate these big pitfalls and allow data interoperability across physical and digital boundaries (state lines and various third party monitoring programs).

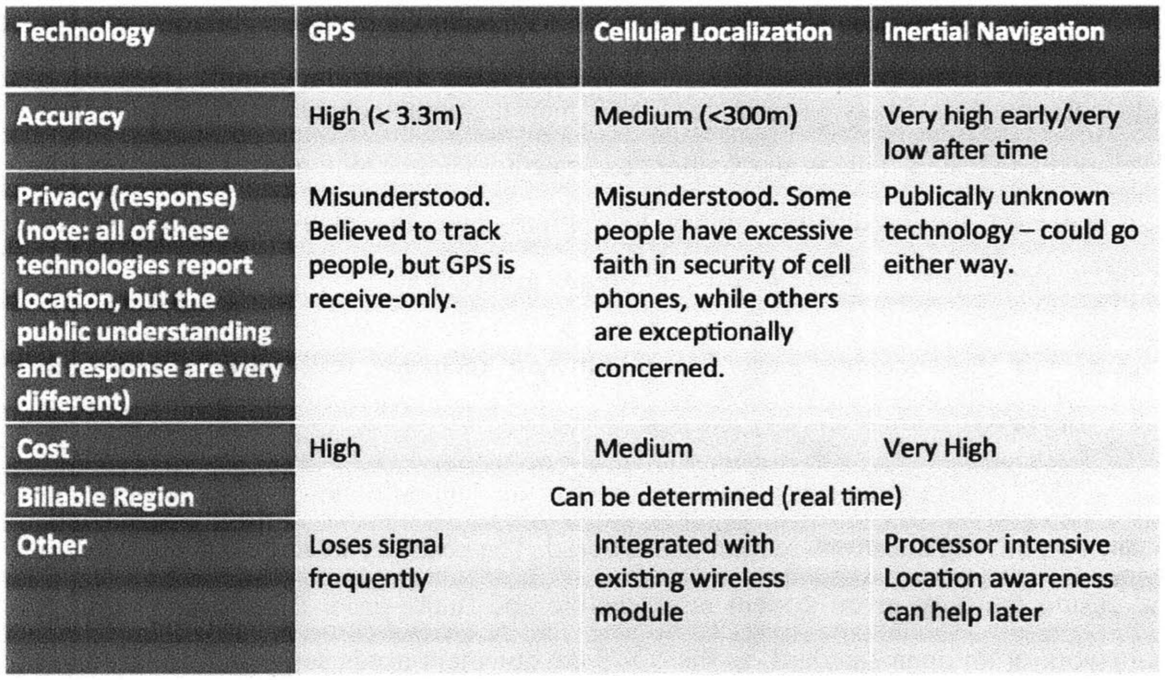
The issue of a central database populated with driver information was a topic of contention, especially in the context of sharing between government organizations or even private entities such as bill collectors. A related Texas VMT proposal suggests that stating a clearly defined goal, data retention guidelines, and allowing data transparency for users alone, may be enough to quell fears during limited deployment. Along with comparison to current technologies (OnStar, *etc.*), privacy and data collection may be a non-issue – or could backfire terribly.

While VMT policy may face challenges, these challenges are not insurmountable and deployment is likely. Therefore, VMT measurement technology will have to be developed and tested. This is an excellent use case for CloudThink and Avacar mirroring, as it will test the hardware's ability to capture data, the platform's usability, and challenge developers to create secure and private applications.

2.3 VMT Measurement Tests CloudThink on Accuracy, Reliability, Privacy, and Security

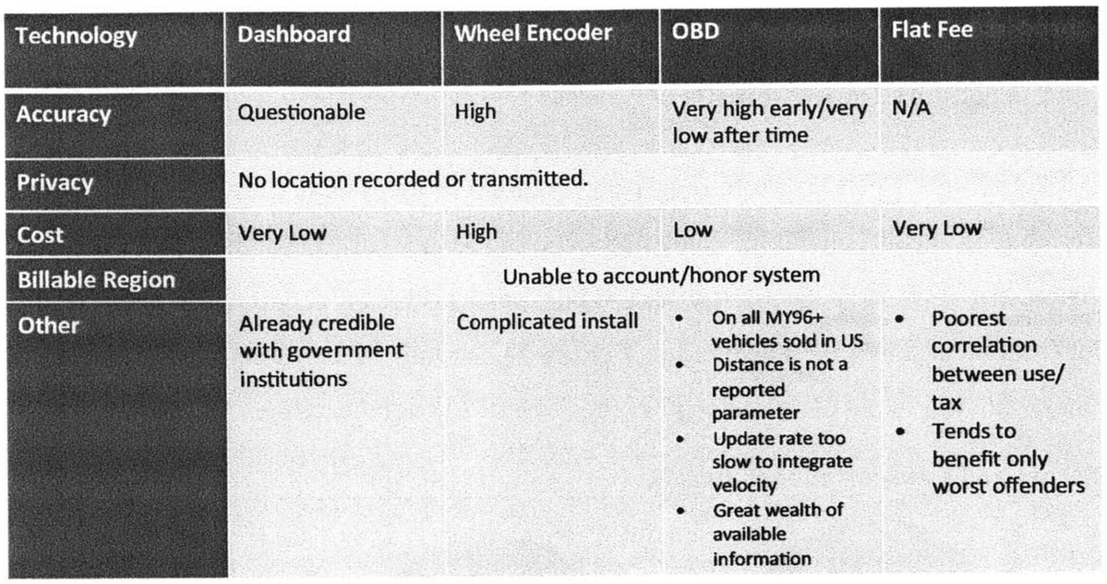
VMT measurement has been deployed in the United States with varying degrees of success, but big questions remain unanswered. The best way to address these concerns and ensure that any given system is extensible and future-proof is to build within the framework of an open standard, as the VMT development needs suggest. Further, digital mirroring directly supports auditability and extension to PAYG applications, meaning this test case is a good launch pad for future application development. The CloudThink platform, if implemented properly, would provide a multi-state, auditable, and private solution. The VMT monitoring problem is a challenge, but an excellent test case for a mirroring platform due to the volume of data, complex computation, and security and privacy concerns that must be addressed. It also provides a "real" use case to convince consumers of the platform's value, and addresses the challenging problem of measuring distance using On-Board Diagnostic data as a primary source.

The use of VMT applications as a test configuration for cloud mirroring provides design challenges that will help to test and refine the development of the embedded hardware and software. VMT technology can be described as location aware, location agnostic, or a hybrid approach. Location aware VMT technology includes GPS, cellular localization, internal navigation, and other similar technologies, whereas location agnostic reporting relies upon dashboard visual inspection, wheel odometers, fixed fees, and On-Board diagnostics. Hybrid approaches combine these tactics, but can add a layer of complexity in that location data may be utilized in processing but never transmitted, or may be transmitted but never used as the basis for calculation. **Figures 5** and **6**, below, compare these technologies.



Technology	GPS	Cellular Localization	Inertial Navigation
Accuracy	High (< 3.3m)	Medium (<300m)	Very high early/very low after time
Privacy (response) (note: all of these technologies report location, but the public understanding and response are very different)	Misunderstood. Believed to track people, but GPS is receive-only.	Misunderstood. Some people have excessive faith in security of cell phones, while others are exceptionally concerned.	Publically unknown technology – could go either way.
Cost	High	Medium	Very High
Billable Region	Can be determined (real time)		
Other	Loses signal frequently	Integrated with existing wireless module	Processor intensive Location awareness can help later

Figure 5: *Comparison of location-cognizant measurement techniques for vehicle miles traveled*



Technology	Dashboard	Wheel Encoder	OBD	Flat Fee
Accuracy	Questionable	High	Very high early/very low after time	N/A
Privacy	No location recorded or transmitted.			
Cost	Very Low	High	Low	Very Low
Billable Region	Unable to account/honor system			
Other	Already credible with government institutions	Complicated install	• On all MY96+ vehicles sold in US • Distance is not a reported parameter • Update rate too slow to integrate velocity • Great wealth of available information	• Poorest correlation between use/ tax • Tends to benefit only worst offenders

Figure 6: *Comparison of location-agnostic measurement technologies for vehicle miles traveled*

A hybrid approach in many cases melds the benefits of location based with non-location based monitoring, in that it can correct for out of billable zone travel (private properly) and keep data private by processing locally. The downsides to this model are complexity, though as embedded processor speeds increase and flash memory becomes less expensive these barriers to deployment will diminish.

One example of a hybrid measurement approach is the merging of diagnostic (OBD) and location (GPS) data. OBD would provide distance metrics, perhaps combined with GPS to ensure validity, and the precise location could be used in an onboard process to identify whether the region was billable. In a thin client model, these data would be passed to a server as raw data, while thick client models would pass data only when GPS indicates that the region is taxable, and might provide aggregated distance travelled only when crossing between billing zone boundaries.

As an extension to this VMT monitoring, and another use case, CloudThink and the Avacar were designed with the consideration of monitoring fuel use in mind. This extended the platform design "stress tests" to include habit monitoring and methods for providing feedback to users.

2.3.1 Fuel is a Big Issue: Extending VMT Monitoring to Test and Push Fuel Economy Improvements

Extending VMT to capture fuel use provides a more complex, more impactful test scenario that is easily relatable. Fuel use is a big issue– in 2009, transportation was responsible for 29% of U.S. energy use, while passenger cars account for 66% of this number (this breakdown is visualized in **Figure 7**), leaving 19% of energy use tied up in consumer vehicles. Most of this energy comes from CO_2-rich gasoline.

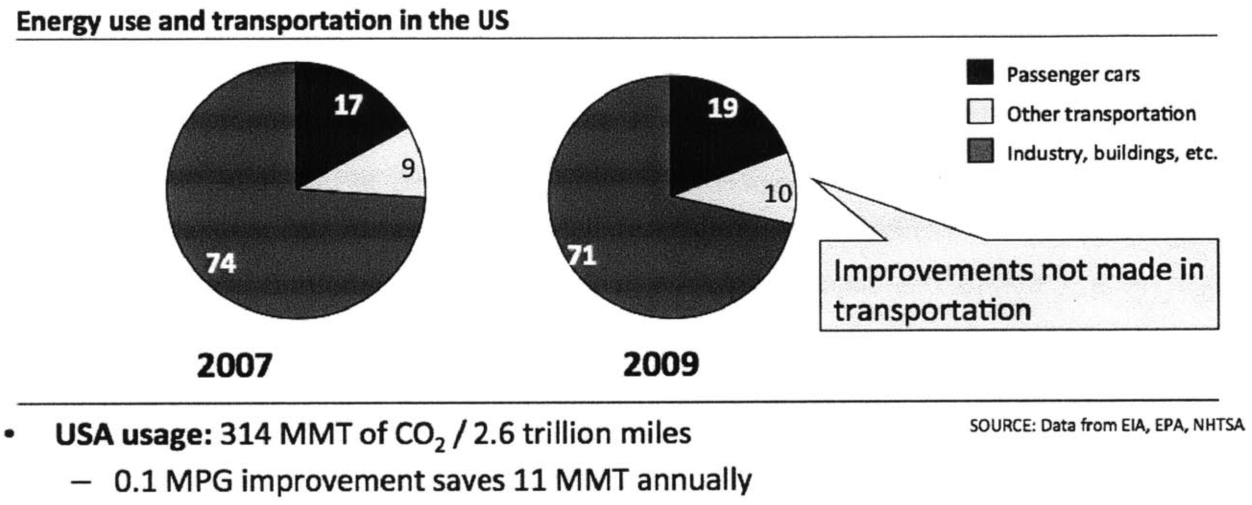


Figure 7: *Energy in the United States by sector in 2007 and 2009, based off of EIA and EPA data. For transportation in particular, much of the energy use is tied to gasoline use.*

Looking at the same data, consumer vehicles released 314 million metric tons of CO_2 over 2.6 trillion miles in 2004, with an average fuel consumption of 19.6 MPG. Since gasoline produces about 19 pounds of CO_2 per gallon combusted, a mere 0.1 MPG increase would save almost 100 pounds of CO_2 per vehicle per year, or over 11 million metric tons of CO_2 annually if the changes were applied across the entire US consumer fleet [34].

Though energy use increased between 2007 and 2009, the relative percentage for transport-related energy use is higher, indicating a proportionate lag of improvements in that sector (data from 2003 and farther back corroborate this notion). Fuel use is an important factor in reducing greenhouse gas emissions and otherwise preserving the environment. This section explores CAFE (Corporate Average Fuel Economy) legislation suggested targets for fuel use within the United States between present day and 2035, as a means of identifying the monumental problem with fuel consumption in the United States and illustrating the need for consumption monitoring and behavior-changing applications that would further test a digital mirroring platform like CloudThink.

2.3.1.1 CAFE Goals: Can Better Data Make a Bigger Impact?

CAFE stands for Corporate Average Fuel Economy, the sales-weighted average fuel economy over a manufacturer's entire portfolio. Sales weighting means that a manufacturer's CAFE number, used as a comparison against CAFE target fuel economy, is actually fairly flexible from manufacturer to manufacturer.

CAFE legislation is not a new idea, and has been implemented with great success in the past. It was drafted in 1975 as a response to the Arab Oil Embargo as a set of incentives and potential fines that, with low latency, took effect and had great impact. With a few tweaks over the years, it effectively brought the fleet average for new cars sold within the United States from 18 MPG in 1978 to 27.5 MPG by 1985. Since CAFE tends to be driven by fuel prices, progress leveled off between 1985 and present day, and modern fuel economy had not improved appreciably between 1985 and 2004 [35] [36]. However, the changes from 1974-2004 demonstrated a remarkable impact, as the whole-fleet EPA-rated average economy improved 78% and this resulted in a 33% reduction in CO_2 emissions per mile [37]. The reason for stagnation between 1985 and 2004 was the consumer push for safety, luxury, and power; all efficiency gains were negated by increases in weight and horsepower [38].

As fuel prices began to climb again recently, CAFE has once more become an attractive piece of legislation and was revamped and redeployed. The US target is 54.5 MPG by 2025, starting from 2011 and moving up in 5% or 3.5% increments annually for cars and trucks respectively. This figure is, as with most CAFE calculations, not entirely representative of on-the-road improvement. The 54.5MPG figure comes from the US EPA's 2025 CO_2 target of 163 grams/mile by 2025 and the requisite equivalent fuel economy, while the "true" target is 49.6 MPG because it is a hybrid of EPA and NHTSA's desires for specific fuel and CO_2 targets. This value takes into consideration the EPA CO_2 standard as well as EPA-approved credits provided to manufacturers for exceeding targets or producing "game changing" technology, like plug-in electric vehicles [39] [40]. Therefore, this number is not an accurate representation of what fuel economy the vehicle might get in use, as a car meeting CAFE's 2025 target might only have a 36MPG combined (city/highway) EPA window sticker [39]. This is because CAFE testing is done on a dynamometer, not the road, and uses the original EPA drive cycle for fuel use evaluation.

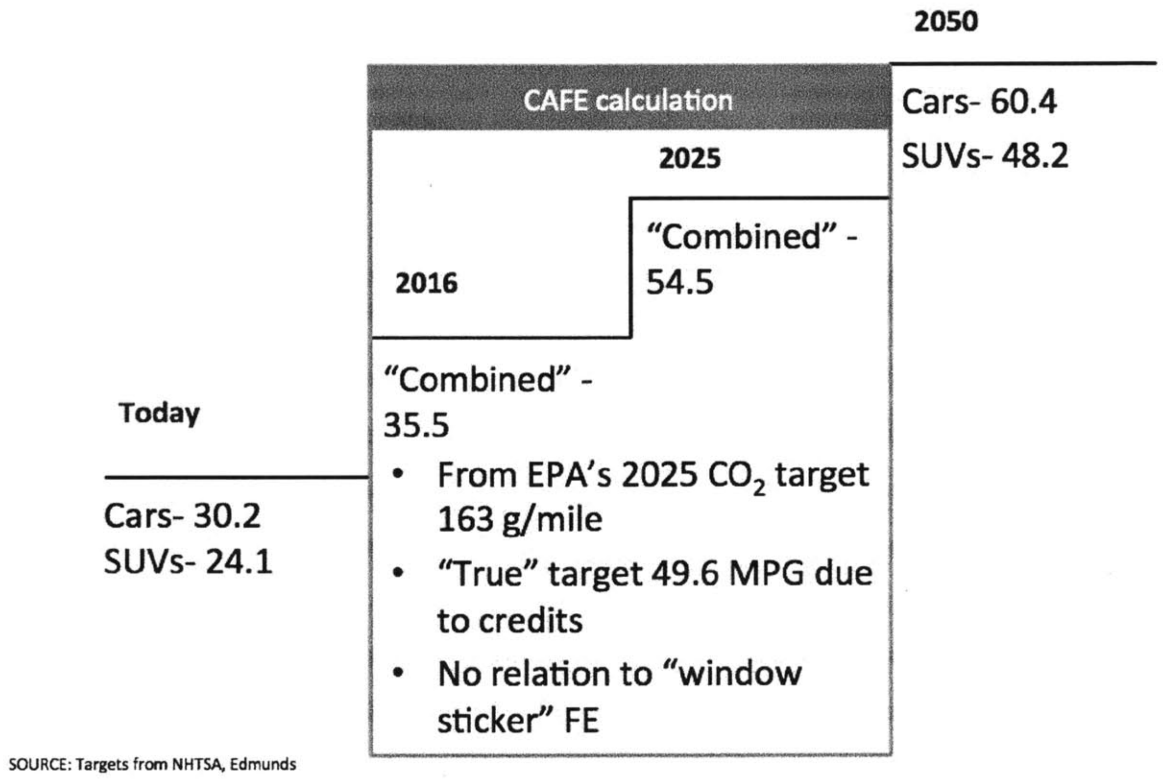

2050
CAFE calculation
Cars- 60.4
SUVs- 48.2
2025
"Combined" - 54.5
2016
"Combined" - 35.5
• From EPA's 2025 CO2 target 163 g/mile
• "True" target 49.6 MPG due to credits
• No relation to "window sticker" FE
Today
Cars- 30.2
SUVs- 24.1
SOURCE: Targets from NHTSA, Edmunds

Figure 8: *CAFE targets from present day until 2050. Note that present day economy values use window sticker economy, while CAFE uses an older EPA test.*

The actual calculation of CAFE averages is accomplished using a harmonic average, the reciprocal of the average of the reciprocal values for fuel economy. This captures the fuel economy realized by driving each vehicle in the fleet for the same number of miles annually, whereas the arithmetic mean captures the fuel economy driving each car using the same amount of gas [41].

CAFE is fundamentally flawed because it relies on imaginary vehicle in imaginary situations. An open platform providing real-time, crowd-sourced diagnostic data could provide far better metrics for fuel use and drive significant improvement relative to today's policy.

2.3.2 Impact of Real-World Data on Fleet Average Fuel Economy

CAFE is a good idea, but needs improvement before it can be effective. CAFE makes use of the now-deprecated EPA drive cycle while almost all other organizations concerned with fuel economy have revised their standard drive cycles to reflect increases in speed and the proliferation of accessory usage. The argument in favor of retaining the old EPA drive cycle is the resultant comparability of fuel economy figures over time, and the argument that a percentage improvement with one particular cycle is still that same percentage improvement, but it is deceptive to report such artificially constructed fuel economies. This is especially true when reporting values to consumers who are unable to fully comprehend the impact of CAFE and what it means for ROI when purchasing a new, more fuel-efficient vehicle. Beyond updating the drive cycle, it is more important than ever to have an accurate measure of fuel economy in alternative fuel vehicles (for example, when the government calculates MPGge, it is unclear what the power source is or if it is corrected for line losses or plant inefficiency). These new fuels and update vehicle use cases must be corrected for, and an open data platform allows for this with present technology.

The outlook for meeting CAFE targets at present is relatively bleak. While it is not easy to address the political shortcomings and consumer desire behind these policy developments, a cloud-based monitoring system would avoid loopholes in calculation by providing better, real-world data composed of real information from the deployed US fleet. For example, it would be possible to identify the type of fuel a vehicle is using – and eliminate unjust alternative fuel credits – or examine a typical use case rather than a federal drive cycle. With cloud mirroring, legislators and manufacturers could examine the true efficacy of policy and restructure legislation to ensure measurable and meaningful impact. Loopholes could be tightened – for example, by eliminating the SUV exemption based on typical routes driven – and fines cold be levied for manufacturers more equitably. As an added value, manufacturers could receive better data as to the use case of their vehicles and optimize their development to better meet these needs in the context of reducing emissions and fuel use, while government agencies could gather anonymized data representing the true use case of the United States vehicle fleet, weighted by miles driven and cumulative total of negative externalities. These data would drive the creation of more representative drive-cycles, and perhaps most importantly, these data could have an impact on consumers.

Consumer desire to drive their bigger, faster vehicles more aggressively causes the true fuel use to be even worse than CAFE and targets might suggest, but a mirroring system and application platform could partially address this problem. In addition to proving the economic complications of SUV ownership, data could be used to generate compelling representations of consumption that could cause a shift away from 0% EFRC. Social applications could drive competition to improve economy, and a slew of aggregated data could be used to tweak the big lever arm that is driver behavior. Some devices today – like the Automatic app for iPhone and Android[3] - attempt to do this using accelerometer, but with the richer, real-time, and crowd-sourceable dataset afforded by a digital duplication scheme, it would become possible to drive more consumer behavior change and provide informatics based on hard data rather than gut feeling. With adequate feedback and a compelling application model (perhaps gamificiation of driving habits), CAFE targets – even the more critical targets afforded by improved measurement techniques – would be far more achievable.

Cars are getting ever more efficient, but fuel economy numbers do not reflect these efficiency improvements. Where is all this saved energy being lost?

One way of quantifying potential improvements is the Emphasis on Reducing Fuel Consumption (ERFC), which describes the degree to which improvements are directed toward reducing fuel consumption relative to other factors. ERFC is calculated as in **Equation 1.**

Equation 1:

3

$$ERFC = \frac{Fuel\ consumption\ reduction\ realized\ on\ road}{Fuel\ consumption\ reduction\ possible\ with\ constant\ performance\ and\ size}$$

and

$$ERFC = \frac{FC_{previous} - FC_{realized}}{FC_{previous} - FC_{potential}} \text{ [42]}$$

ERFC will return a value between 0 and 1, corresponding to a 0% or 100% emphasis on technological improvements being directed toward reducing fuel consumption relative to other aspects of the vehicle (*e.g.* improved comfort, safety, or convenience [size]). Put simply, ERFC is the relative importance of fuel economy when compared with performance and luxury. In some respects, this is best modeled as a simple vector math problem. Assume there exist a series of vectors created from component vectors parallel or perpendicular to the "movement" axis or forward direction. These perpendicular vector components point toward "luxury," "power," or, generally, "distraction."

One vector exists entirely in the direction of the movement component. The entire magnitude of this vector forces fuel consumption lower. Another vector exists only in the direction of the "distraction" component, or in the direction of technological improvements that do not directly improve fuel economy. This vector never helps improve economy and instead is focused solely on more palpable user experience, like improving acceleration time. An ERFC of 100% is the former case, while the 0% case is the latter. Combinations of these vectors determine the ERFC and how much an improvement vector contributes to actually moving a vehicle versus simply making it nicer to sit in or drive quickly. Typically, consumers prefer 0% ERFC, while governing bodies prefer 100%.

Supporting the assertion that historical ERFC is near-0%, observe that from 1981 to 2003, fuel economy increased 1%, weight increased 24%, horsepower nearly doubled (93% improvement), and acceleration improved by 29%. Had ERFC instead been 100%, average fuel economy for the entire US fleet would be up 20% between 1985 and 2001 (and presumably better since then, as new technologies such as transmissions with more gearing options and direct injection have since come on-line) [43]. Comparing vehicles within a single class, Knittel, in his AER paper, asserts that improvements in economy could have been 60% instead of 15% realized [35].

If the past 20 years are any indication, US fuel economy would remain constant due to the 0% ERFC consumer preference. That means all the engine improvements go toward offsetting louder stereos, faster acceleration, and carrying around lots of road noise deadening equipment. With a real-time feedback system, consumers may realize the harm ERFC 0% is causing – and take corrective action.

One of the single biggest challenges to CAFE's success is the introduction of additional trucks into the US fleet. Unfortunately, CAFE actively encourages the development and sales of larger vehicles. This problem is often referred to as the "SUV loophole."

CAFE was first implemented at a time when not all automakers offered a full line of vehicles. Full line manufacturers rebelled against the standard, saying CAFE was easier to meet for new automakers that did not yet produce trucks. Thus, the standard was revised, and tiers were put in place. The physical footprint of the vehicle, rather than the intended use or even weight defined these tiers. Larger cars were allowed to achieve lower average fuel economy based on the assumption that larger vehicles had more utility and would be used only if necessary [44]. Auto manufacturers saw an easy option to lower the CAFE target they would have to meet and began to produce larger (and therefore heavier) vehicles [38].

That same incentive exists today, as cars must improve fuel economy 5% per year while trucks are only required to meet a target of 3.5%. With a cloud-based informatics platform, it becomes possible to show users the trust cost to operate an SUV and allow market forces to drive improved fuel economy.

There are other loopholes that increase the likelihood of meeting CAFE targets as well. Alternative fuels and "flex fuel" E85 vehicles are of particular concern. That is because the standard only counts the "perceived energy content," or the fraction of gasoline content as energetic for this fuel, so only 15% of the fuel is considered to be doing work. Thus, fuel economy when running on ethanol blends can be multiplied by a factor of 6.66 (or 1/15%) for CAFE calculation purposes [39]. In an example case, a vehicle gets 28 MPG on gasoline and 25 MPG on E85. The CAFE fuel economy becomes (28 + (6.67)*25)/2=97.4 MPG – even if the car never burns the fuel! Since the alternative-fuel fuel-economy is averaged with the gasoline fuel economy, E85 vehicles have artificially high economies when recorded for CAFE [45] [39] [37]. This incentivizes the creation of mild-dual-fuel vehicles, even if they never run the secondary fuel.

Other stumbling blocks are MPGge calculations, calculations that are still evolving. Combined with credits for mild hybridization and trivialities such as LED lighting, credits appear to be too easy to come by and should be eliminated or significantly revamped [40]. [45] [39] These credits in particular make it easy to continue producing gas-guzzling SUVs, putting a damper on otherwise significant progress. While these credits pose a significant threat to achieving the target goals, one of the biggest factors in CAFE's efficacy and one yet to be discussed is consumer buying habits.

In both cases, the availability of true-use data is useful for the consumer (and any possible next-generation CAFE programs) as it becomes possible to measure the true impact of a technology and figure out a per-vehicle fuel consumption. This sort of data would allow greatly improved metrics for efficiency and consumption – for example, weighting a sports car that gets poor fuel economy but is rarely driven significantly less than a daily-driven SUV receiving similar economy. Manufacturers could even optimize their development and re-tune components to improve efficiency remotely. This would be a great improvement over the dynamometer-test based CAFE metrics today.

3. Development of CloudThink: Hardware Evolution Leading Standardization

In the Field Intelligence Laboratory, projects begin with a complete needs-analysis and use the results to attempt a solution before completing a prior art search. Reinventing the wheel is an excellent exercise, and it allows for unbiased thinking and the viewing of insights otherwise impossible to discern. The same publications will be available after a best-faith effort to develop independently, so taking the time to struggle with problems firsthand is a worthwhile experience.

This tact provides a richer understanding of the depth of the problems faced, and the ability to derive a framework to meet specific wants and needs goals with IoT technology. While it may seem incongruous with a push toward standardization, the reality is that understanding needs leads to understanding what, exactly, standards should define. The following section describes several guiding principles for designing the Avacar creation device, the CANPuter.

3.1 Guiding Principles

To allow successful deployment of the next generation of self-reporting devices, developers must understand how designing within an interoperable framework reduces duplicated effort, extensibility reduces development time by leveraging existing technologies, and convergence utilizes parallel advancements in Cloud and mobile computing to streamline connectivity and drive a positive user experience.

Interoperability – until an interoperable framework is in place, efforts to develop compelling hardware, informatics and applications will be wrought with duplicated efforts.

Extensibility – utilization of existing infrastructure in which investment has been high (for example RFID) for innovative new applications (such as sensing) reduces development time, cost, and complexity

Convergence – leveraging parallel advancements in cloud and mobile computing helps streamline human-device and device-device connectivity by creating a platform for analytics and visualization that performs seamlessly and across digital boundaries

These topics provide a framework for defining methods to *generate*, *capture*, and *analyze* data. These topics are separate, but not freestanding. Without data generation, there would be nothing to analyze; with unreliable data capture, any conclusions could not be trusted reliably, and without analytics, there would be no easy way of drawing conclusions afforded us by these new data. These areas work in concert to provide a complete, end-to-end solution for IoT hardware and software.

Though not the focus of this thesis directly, the author believes there are several key concepts that may help ease IoT development pains and lead to smooth deployment of technology. These beliefs strongly sculpted the author's approach to developing CloudThink as a platform, and guided the vision for digital object mirrors and its drive toward an open standard. Presently, IoT development falls flat due to a series of barriers to rollout. To ameliorate this:

- Open standards support the larger vision of the Internet of Things, and will drive growth and incubate new technologies
- Standards pay off with improved user experience, making the effort worthwhile
- The IoT must serve average people and be usable without requiring specialized knowledge or exceptional skill
- Privacy and security concerns must be addressed sooner rather than later, as they will not go away
- Transparent devices ease data capture and have a butterfly effect when it comes to increasing the volume and reliability of data captured

The first and boldest point contends that open standards are the future. This is sure to be the case. Open standards cut across all aspects of the Internet of Things, from hardware interfaces to communication protocols to privacy and even data storage. With open standards, it becomes possible to develop in an interoperable manner, and the freely available specification allows developers from all backgrounds easy entry. The standardization of data storage allows users to maintain portability of data, creating the opportunity to cross digital borders using freely manipulable data.

The publication of an open standard invites developers to the table, keeping things membership-free and well documented to ensure wide usability. This allows innovative thinkers to create compelling value propositions for consumers and helps get more eyes to pour over rich data sets with hard to discern trends. Neutral accessibility allows hacker culture to participate openly, and open APIs and free data access supports development of analytics, informatics, and visualization - all of which must be present prior to exploring user experience.

This standardization drives rapid adoption and drives costs lower. Standards are expensive, but collaborative development lends itself to lower cost for hardware and software that is interoperable, understood, and customizable. For those interested in data and not hardware development, these lowered barriers to entry are a massive windfall. For those with a focus on hardware, a more compelling app ecosystem means more potential sales, even if margins are smaller.

Standards take time and money to write, but this initial investment is repaid when supporting solutions work out of the box. This positive user experience drives further adoption of the standard, increasing sales of related software, hardware, and services. The process of writing standards fosters better understanding of user needs that have been addressed and those yet to be met, and the process lends itself to the creation of composite metrics such as measures for success for Internet of Things utilization.

Standards also allow a surprising amount of control over user experience, even if they do not directly address the topic. This is especially true for standards that are hard to protect. Reverse engineering is a reality, and it should be viewed as positive that people want to follow this standard as it drives utilization of related products and services. Fighting against openness and requiring competitors to reverse engineer a standard often introduces bugs and creates a clumsy user experience, which reflects poorly upon the standard that was reverse engineered. A great example of this is open-source image editors that claim to open Photoshop files. Any freeze or crash reflects as poorly upon Adobe as it does the creator of the open-source program, and less people will chose to use the file format due to interoperability concerns. The bottom line is, open standards liberate data, and liberated data is boundless. GS1 has shown this to be true with their development of Electronic Product Codes, e-commerce and data synchronization services. The proliferation of services such as Google Fusion Tables serves to further illustrate the demand and utility for interoperble data.

In IoT development today, designers forget that the IoT is about people [46]. People can provide data, people use data, and people support the infrastructure. Too often, devices and standards are designed for the sake of the technology rather than for the stakeholders. Machine to machine is necessary, but so is thinking about how M2M can be made to serve and improve lives rather than simply create work. In a similar vein, data generated must be constantly examined and refined to ensure that it is usable beyond the originally designed application. Millions of data points are meaningless if they are not easy to visualize and manipulate. Standardization helps bring the focus back to the people by providing clearer sets of rules and regulations for IoT hardware, software, and APIs. A well-thought-out standard can ensure the success of IoT technology simply by ensuring a positive user and developer experience.

Security and privacy concerns are core issues to IoT technology. Users should own the data they generate, and should be encouraged, but not required, to opt-in to sharing programs. This may be accomplished with rewards programs or simply asking the user. Security is paramount – SSL encryption is a good start but not a complete solution to protecting sensitive information in transit. The key here is to solicit user input. Some users want to share everything, and other users will want a physical key switch to enable data sharing. These users are potential customers, so not understanding their wants and needs is asking for hardware not to be adopted. Importantly, policies must be clear, honest, and consumer leaning. Here again, standardization can eliminate duplicated efforts and provide reassurance that their concerns have been addressed directly by a standardization group.

The last point concerns transparency of design. In early sentiment testing (conducted by asking users to try the author's hardware, as described in later sections), it became evident that end users did not care to deal directly with the device if it required much intervention. The NEST thermostat is a good example of a device that users feel comfortable with. It is no more complex to use than a typical programmable thermostat, more closely mirroring the decades-old dial type thermostats. Yet, users love the experience – because it just works, and because they do not have to think about their interactions with the device. This approach drives the generation of big data. A set and forget device will continue reporting non-invasively. Devices that require user intervention or remind the user of their presence are likely to be deemed bothersome and unplugged.

These rules for IoT development shaped the design process for the CloudThink project and the thoughts behind the Avacar platform.

3.2 Why Redesign

The author's undergraduate thesis, "Design, Development, and Validation of a Remotely Reconfigurable Vehicle Telemetry System for Consumer and Government Applications" describes the creation of a Real-Time Operating System (RTOS)-based diagnostic tool and supporting applications which were hard-coded, rather than built on a common or extensible platform. This system was based on the LPC2129 ARM7-TDMI processor, a very capable architecture – but the software was not optimized and the user experience was poor. The scheduler was timed poorly and froze, the device was slow to log, and the code suffered from severe memory leaks that caused hard to reproduce problems. The software toolchain was complex and difficult to develop for.

Moving to an easier to develop for model was in the best interest of creating a robust application platform and canonical hardware, allowing developers to focus on their core competencies. Arduino systems enjoyed much success, and the barriers to entry were low from a prototyping perspective. OBDII interface solutions existed in the form of ELM327 modules as well as the "SKPang CANBus shield," which integrated an SD memory card along with provisions for connecting GPS receivers and LCD displays. This configuration could provide excellent vehicle data, and the analog input ports and secondary serial ports were free to be used with accelerometers and cellular interface devices.

Early development for the hardware revolved around the SKPang module paired with an Arduino Duemilanove, followed by an Arduino Uno, and ultimately an Arduino

MEGA 2560.[4] These platform shifts occurred after running into USB driver limitations and subsequent I/O limitation as pins were dedicated to accessories like memory cards, leaving few available for expansion. The transition between the Uno and MEGA form-factor was difficult as it required rewriting the SPI library to properly address the CAN transceiver, and this complication delayed the development for several days prior to the realization that a pin redefinition in software could solve this problem.

Working from software examples provided by SKPang, the author developed a serially operated (sequential execution) program.[5] This type of program looped as quickly as possible to capture data, but the timing was indeterminate and, while it was typically quick to log, it would occasionally lag and miss data despite the integration of multiple prioritized interrupt routines. The utilization of interrupt routines enabled more robust data capture, but again, the logging was unreliable. A restructure of the interrupt priorities improved reliability of capture and logging.

Early testing results were positive, and the system logged OBD and GPS data quickly and reliably to the onboard memory card. Notably missing were the data afforded by an accelerometer and cellular connectivity, though the integration of an LCD made software development and debugging drastically simpler and was deemed an ideal feature for future revisions.

---

[4] http://www.skpang.co.uk/catalog/arduino-canbus-shield-with-usd-card-holder-p-706.html, http://www.arduino.cc/

[5] http://code.google.com/p/skpang/

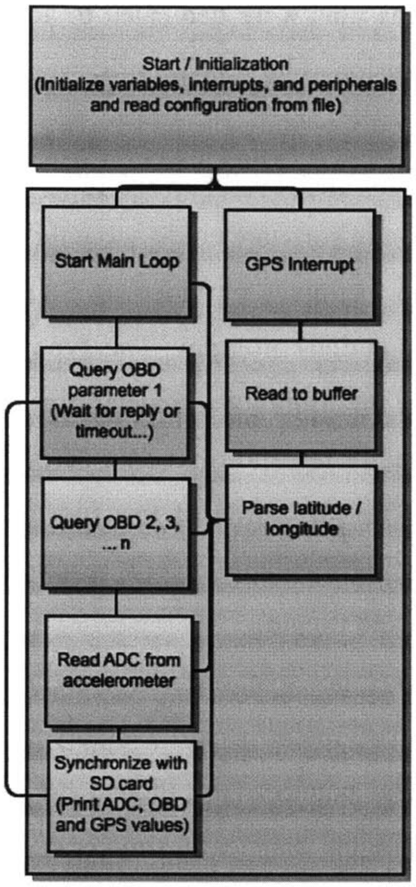

Start / Initialization
(Initialize variables, interrupts, and peripherals and read configuration from file)
Start Main Loop
GPS Interrupt
Query OBD parameter 1 (Wait for reply or timeout...)
Read to buffer
Query OBD 2, 3, ... n
Parse latitude / longitude
Read ADC from accelerometer
Synchronize with SD card (Print ADC, OBD and GPS values)

Figure 9: *This flowchart represents the time-indeterminate Arduino OBD logger capture / recording model*

The next step to address the concern of limited data sources was to develop a custom PCB based around the Arduino test hardware incorporating additional sensors and communication methods. This board added features like battery backup and cellular communication and shared the same ATmega2560 processor as the Arduino MEGA. This allowed the use of the Arduino compiler and Arduino software libraries, as well as inexpensive USB bootloader hardware and free software. The availability of additional UART channels led to the reintegration of a Roving Networks RN41 Bluetooth module to allow for M2M communications when cellular coverage was not available. A second ATmega processor provided USB logic translation to allow debug and programming without special cables, as was the case in the Uno and MEGA prototypes. The SM5100B cellular module was chosen for cellular communication due in part to its free TCP/IP stack and low cost, coming in at nearly one third the price of competitive Telit modules, even when paired with a low-cost GPS receiver like the EM-406.[6] In testing, this module proved to have a poorly written stack with significant bugs in reconnection routines, so the future development reverted to Telit hardware.

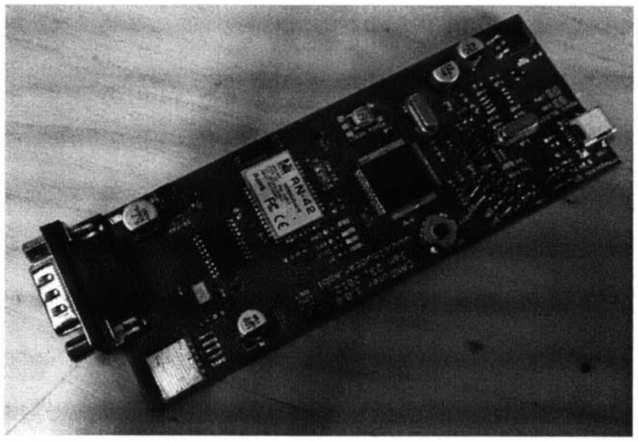


Figure 10: *"Version 4" Arduino board, featuring Arduino MEGA processor, microSD memory, onboard USB, SM5100B cellular module, RN41 Bluetooth transceiver and pin headers to interface with external LCD and GPS modules. The joystick onboard allowed the user to select a program to load, speeding development and debug time.*

The "Version 4" PCB as designed was slightly larger than the previous ARM7 revision, but only required two-layer construction and incorporated significantly more functionality (dual wireless networks and an integral USB transceiver) than the previous boards. The battery backup system worked well to ensure constant power, and the Arduino Processor worked as expected, integrating seamlessly with the Arduino compiler and consequently becoming appreciably easier to program and deploy software updates. Despite this ease of use, the hardware focus led to withering software development and a buggy implementation of logging software. With the software losing focus, there was no scheduler system and data reliability suffered. Further, the new hardware cost as much as the previous ARM7 hardware due to integration of non-essential features.

Another unexpected second-order effect was that making it easy to program over USB with no special tools led to novice developers wanting to run software that had not

---

[6] https://www.sparkfun.com/products/9533

been thoroughly tested. Several drivers testing the hardware "tweaked" features and then complained about poor performance of the device. A slight complication in compiler requirements would raise barriers to entryway sufficiently much that more experienced, "prequalified" software developers could bring about an improved user experience (indeed, far fewer users asked to view the C code running the next version of the hardware). Ultimately, the Arduino model was a move away from the target, though it did provide many opportunities for learning to be integrated into later hardware revisions.

These lessons included the realization that parallelization of tasks is key to reliable data logging, that unreliable bootloaders breed frustration and reduce incentive to develop, and that users want to become involved – potentially to disastrous result. This project also led to better understanding of parts sourcing and supply chain issues. Initially, the author believed the open source nature of the Arduino hardware would lead to low-cost derivative hardware. This was opposite the reality, as Arduino's surge in popularity led to supply chain shortages and ultimately increased lead times and component sale prices. The SM5100B's failures also indicated that paying a premium for well-engineered parts is a worthwhile spend of money, saving untold hours when hard to replicate bugs and poor documentation lead to stalled feature development.

3.3 Failures Lead to Better Understanding: Updated Hardware Goals

The deployment of the Arduino revision was short-lived due to the prevalence of bugs and failure to reduce cost. It was therefore time to lock down the hardware with a needs exploration and through studying the lessons learned by the production of the four previous hardware iterations. This first required asking a few questions, namely "who will use this," "what, exactly is being developed," and "how can the hardware be future proofed, to work with new vehicles and an ever-changing platform?"

The first question is poignant, as application developers differ greatly from embedded developers, the second defines the scope of the development, and the third defines architecture and platform roadmaps. The answers are difficult, but distil down to: the hardware must be designed for embedded engineers supporting application developers; the solution developed is a bridge between OBDII, other sensors, and the cloud, and; the future proofing must be thoughtful and designed-in from the start.

First and foremost, the hardware (now termed the "CANPuter") would have to provide universal access to vehicle data, bringing engine and transmission data from the diagnostic port straight to a secure database. To implement this functionality is non-trivial, and historically has prevented application builders from making use of vehicle diagnostic data. By making access to this data transparent, it would become easier to build applications focusing on innovation rather than security and data capture.

This vision required more than simplification of access to conventional diagnostic data, but also the extension to manufacturer-specific diagnostic and configuration data and CANBus physical layer access. The hardware and embedded software would need to be capable of being open sourced (no highly specialized license requirements or significant fees involved) and functional in providing data capture for a range of applications through the availability of near-real-time communications and onboard data storage. The hardware would have to operate as a thin client (transmitting all data captured), but have sufficient processing available to operate as a thick client (data calculation and aggregation onboard).

The hardware was designed to maximize volume of data captured, which meant a wide install base would be crucial. Therefore, cost would have to be minimized through value-driven design, where each feature is evaluated thoroughly through user surveys and only designed-in if deemed to be a net value add for users and developers. This is different from other low-cost options that focus on optimizing for single applications – instead, this design methodology ensured the broadest possible range of applications would be supported by CloudThink's single hardware platform.

The basic functionality requirements dictated OBDII connectivity and cellular Internet access, though neither of these requirements was specific enough to guide implementation. To reduce cost, the author decided to support only the CANBus networks found in newer vehicles. To ensure future compatibility, the board would have to support all OBDII CANBus variants out of the box (meaning 250 and 500kbps, along with 11-bit standard and 29-bit extended identifiers). The specification for CAN communication would require support for SAE J1979 modes 1, 3, 4, and 9, to allow the reading of live and frozen PIDs, and the reading and resetting of trouble lights, though support of additional modes would be preferred. This would cover all vehicles model year 2008 and newer, and many older vehicles supported CAN prior to legislation dictating the requirement. Examining data generation needs and typical calculations (*e.g.* fuel economy, which requires three parameters) the end result was a target goal of 5 PIDs/second read from the Engine Control Unit (ECU) and Transmission Control Module (TCM).

Developers would only be able to realize the maximum potential for diagnostic networks with support for secondary networks like comfort and convenience of infotainment, either directly or by way of the gateway device. Therefore, support for manufacturer-specific diagnostic data on the HS-CAN network as well as proprietary CANBus (single and dual wire) networks was a high priority goal and ultimately deemed necessary to design but not populate. This would allow users or distributers who value the feature to purchase an additional, installable module and greatly enhance their data generation capacity.

Wireless connectivity was a necessary goal, and Bluetooth and WiFi devices both required separate intermediate devices leading to the requirement for cellular communication. Bluetooth and WiFi had advantages in data throughout and latency so to ensure the cellular connection did not lag far behind, a target was set for a 10-second or better average communication time between sensing a value from a node and reaching the target server for storage and/or processing. This latency demand and the volume of data generated dictated the modem specifications: a "2.5G" network could provide sufficient bandwidth and coverage to allow for this latency in all but the most extreme cases. Where network coverage was not available, buffered storage would offer an opportunity to log locally and upload when connectivity resumes – a worst-case scenario led to a target of one month of OBDII, GPS, and accelerometer driving data logged prior to hitting memory capacity. A removable and/or upgradable memory card would be a highly desirable feature.

Developers had further needs not addressed by these specifications, namely reliability and robustness – in particular for time-sensitive code for data capture and processing, and security, to protect vehicle networks from malicious attacks or data sniffing. "Black boxing" and/or code protecting particular regions of the embedded software could accomplish these needs for the embedded system, but much of this feature development would be server-side to protect the database and API.

The electrical needs were derived from findings learned from the previous hardware. Any design would need true rather than assisted GPS to provide accurate positioning data. Many OBDII power ports in vehicles were current-limited, so there was a soft 2.5A cap to ensure maximum compatibility. The device might require service in the field (especially during the development phase), so an onboard programming port – like JTAG or ICSP – would be essential, as would a way of viewing debug data in the field. A widely compatible and removable memory device would ensure users could debug and share data if the cellular communication failed, and keep costs for supporting hardware like card readers low. Keeping the unique and total parts count low would reduce cost, while support for analog inputs – like accelerometers and battery voltage readers – would not increase cost and significantly increase functionality. Finally, a USB bootloader would ensure redundancy for field reprogramming, while a battery backup might solve problems in some limited-power scenarios by providing buffering capability, while also reducing drain on the vehicle charging system.

To interface with the vehicle, a standard diagnostic link SAE J1962 connector would reduce complexity and provide universal access. Some drivers run double battery systems to help start larger engines in cold weather, so 24VDC support was essential, as was reverse polarity and overvoltage protection, to ensure an internal short would not damage the vehicle or hardware in an irreparable manner. The network interface would build on the previous version's TCP/IP encapsulated serial port, possibly adding support for T(rivial)FTP for file uploads and downloads. The cellular network configuration could be GSM or CDMA, perhaps in conjunction with Bluetooth, with a minimum throughput of 19200bps to prevent a data bottleneck at the designed sample rate.

The previous hardware did not have an enclosure, but for wide distribution later revisions would require a custom-designed case. This case would need to be UV-resistant and water and dust sealed, with a target of an Ingress Protection (IP) 56 rating and a material capable of withstanding temperatures up to 140F for when drivers leave the module out in sunlight regularly [47]. For user feedback, the hardware and case would need to include provisions for viewing power, connectivity, and general purpose LEDs, ideally through directional light tubes to make viewing status at a glance simple.

Finally, the software specifications had evolved from those of the previous embedded device and with lessons learned from optimizing the interrupt routines on the Arduino model. Key features to add included the detection and remote reflashing of software updates, Power-On Self Testing (POST) for all peripherals, and improved security for data capture and transmission (*e.g.* Secure Socket Layer [SSL]). Additionally, tweaks to the existing software could enable faster PID sampling, local parsing of Diagnostic Trouble Codes (DTCs), and VIN reading (not present in the previous version).

3.4 ARM7 Makes a Comeback: Real-time Schedule System Wins Out

As it turns out, most of these needs had been addressed in the previous ARM7 LPC2129 hardware revision, and the majority of the errors had been introduced though the deployment of faulty software in an effort to meet time constraints. Therefore, the next revision of hardware ("Version 5") would base its design off the previous LPC2129 iteration ("Version 3"), with key changes between the revisions being the incorporation of an onboard multiplexer to switch between a Roving Networks (RN41) Bluetooth chipset and a Telit GM862 cellular modem, a reduced-complexity board layout and a transition from SD to microSD memory cards. The inclusion of Bluetooth eliminated the chance of being completely out of cellular coverage, enabling the use of M2M communications (*e.g.* cell phone unlocking in a covered parking garage) in addition to MCM data uploads. The hardware was optimized to fit the same form factor as the previous version while incorporating this new hardware and an improved power supply, and the Bill of Materials (BoM) was optimized so that the cost to manufacture broke even despite the added features. The majority of savings were identified in specifying antennas and power supply components.

The author produced twelve copies of this board and tested extensively prior to designing the final embodiment of the hardware discussed in this document. This section will explore the design decisions made after determining a move back to a scheduler-based OS was necessary, and explore some of the lessons learned throughout the development process.

In testing, the Bluetooth feature was rarely used and required custom software development that would take up significant codespace – the cellular coverage had been better than expected with new antennas, and Bluetooth less reliable than was preferred. Further, the addition of Bluetooth greatly increased the Bill of Materials cost and required additional support hardware like new power supply ICs. Removing this module and related hardware, and shifting from a removable Telit module (GM862) to a Ball Grid Array (BGA) module (GE864) the BoM cost could be reduced by nearly 50% with a similar component count. Additionally, the GE864 would allow further board optimization, as the form factor was smaller in all dimensions.

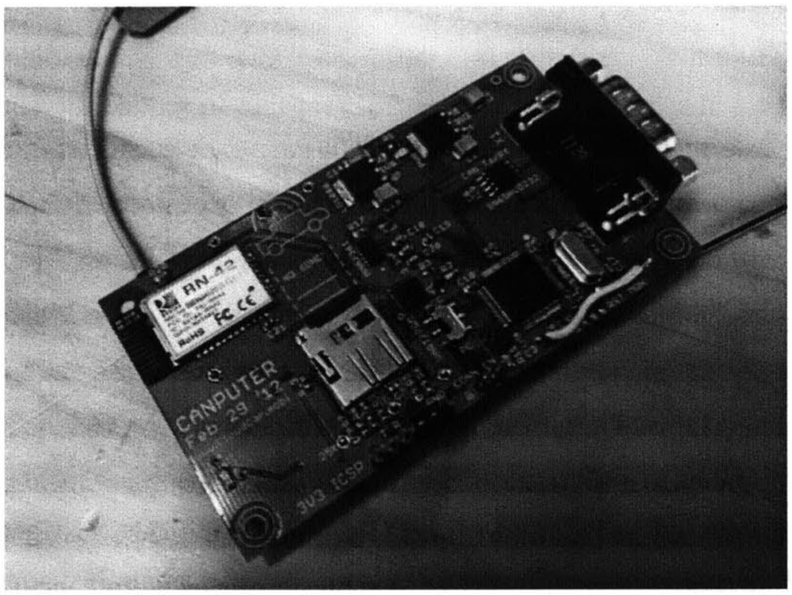


Figure 11: *Version 5 of the CANPuter hardware, featuring both Bluetooth and cellular communication. The yellow jumper wire shorted the bootloader to ground for programming initialization.*

The redesign process was iterative and built upon the rework of existing boards in conjunction with simulation. New hardware architecture emerged that remarkably resembled previous versions, but the new configuration cost less and had better thermal characteristics and improved noise immunity due to more thoughtful trace routing. The resultant hardware was based off of an ARM7TDMI platform running FreeRTOS as a scheduler to manage tasks. It would plug into a standard J1962 diagnostic port and log CANBus OBD, three-axis 1.5g, as with the previous versions, and add a 48 channel GPS receiver and log data to a microSD/microSDHC card. This next segment details several design choices in the context of four key design guidelines:

- **User friendly** – the hardware is plug and forget, to minimize user involvement and therefore maximize collection of untampered data
- **Reliable** – the hardware buffers data to ensure completeness regardless of cellular coverage, adapts to certain in-vehicle networks non-compliant with the OBDII specification, and is remotely updatable should a problem be detected in Power-On Self Test (POST) or a software update be made available. A secondary, SD-card based bootloader provides redundancy in upgrade paths should a flash event fail
- **Protective** – the hardware design is SSL-capable to reduce the chance of over-the-air man-in-the-middle attacks (local attacks may be possible with physical access)
- **Secure** – the addition of external sensors, such as accelerometers, can provide data to mitigate risk of "spoofing." Additionally, periodic VIN reads ensure the device has not been relocated since power-on

3.4.1 Cellular Connectivity

Many vehicle scan tools tout wireless capabilities. However, these scan tools utilize Wi-Fi, Bluetooth, or ZigBee to communicate with other devices (machine to machine, or M2M). These networks work well for visualizing data on a mobile device such as a laptop, tablet, or cellular phone. However, these networked devices all operate at a short range and as such have a fundamental flaw.

For mission-critical metrics, the straightest path between data source and the computation of analytics is ideal. With a short-range protocol, performing analytics at a central location requires the use of a bridge device. Bluetooth devices, for example, require a phone as a bridge to the Internet. If the driver forgets his or her phone, or if the battery dies, the data never makes it to the network and is lost. Further, many short-range scan tools provide real-time data with no means to buffer this information, preventing future auditability. To avoid this pitfall, the author elected to design the diagnostic device with an integrated cellular chipset, mirroring critical vehicle parameters to a central web server.

This design decision turns the vehicle into a self-reporting sensor requiring no user intervention after initial setup. Deciding on a cellular chipset was critical to driving further design decisions. Primary considerations for cost/benefit analysis included power consumption, bandwidth requirements, latency requirements, and long-term network roadmaps. Devices within a similar class (*e.g.*, 2G, 3G, LTE) had similar power requirements, so the next step was to examine network requirements. Latency was not a key factor, but bandwidth use was. Bench top testing of a CAN data logger outputting raw data to a serial terminal demonstrated a bandwidth use of approximately one megabyte per hour, with two megabytes as an upper boundary when reporting VIN, latitude, longitude, speed and distance metrics every five seconds. The 2G network was sufficient to handle this volume of data, had mature coverage, and offered low-cost, low-power cellular modules.

The low volume of production limited access to suppliers. Many suppliers dealing in the 10's, 100's, or even 1000's were Chinese vendors with no online presence and poor documentation. Telit wireless modules were, by contrast, only slightly more expensive and offered excellent documentation along with development kits. In early development, the author decided to develop using the GM862 module due to its low cost and removable module design. mikroElektronika produces an excellent development kit for this module. After validating performance using a series of large file uploads and mapping exercises (driving to ensure connection robustness), the GM862 proved itself competent in both keeping a connection alive and providing GPS positioning data. However, this module was bulky and increasingly hard to source as it neared end of production life. A meeting with a local Telit distributor led the author to test the GE864-GPS, which is largely pin-compatible and offers lower power consumption and an integrated, high-quality SiRF GPS receiver. Sourcing these modules proved to be a challenge due to the lead time, but otherwise proved to be a wise design choice.

Late in the project, AT&T announced it would be sunsetting its 2G M2M platform. While the hardware relies upon 2G technology, Telit offers a software-compatible upgrade path to 3G with new, power-efficient modules.

3.4.2 Memory as a Buffer

As mentioned previously, a downside to wireless connectivity is the intermittent nature of the connection. Typically, this is not an issue – with webpages, packets are requested again and again until they make it through. In the case of a vehicle, however, gaps in coverage – due to cross-country trips, tunnels, or even parking underground – are more significant problems. A further problem is the latency when using a 2G module. As a design decision, the author elected to send a message to the server every 5 seconds. However, this time step is coarse and not much can be done with data of such poor granularity. The hardware on the board is capable of logging at >1Hz. To address both problems, the author added a microSD memory card to the diagnostic hardware to serve as a buffering device. At the end of each drive, as determined by GPS location stagnation, the contents of the most recent buffer file are uploaded to the server. This data fills in the interstitial gaps, providing a richer dataset and backfilling areas with poor connectivity.

An unexpected, additional benefit: the same memory card can be used for software updates, eliminating the need for special programming cables and providing a means of "unbricking" the device in the event of a bad flash event.

3.4.3 Sensor Payload

The sensing hardware provides access for CANBus data as well as GPS data. Additionally, the hardware includes a three-axis accelerometer. This data is of value when determining energy use, and also when attempting to prevent data "spoofing" – by providing reference data about the orientation and location of the device to compare against the CANBus diagnostic data.

Thought not included in the most recent hardware, an additional input that would be of value is battery voltage. This would allow the device to turn off to avoid draining a car's battery entirely. A general purpose input / output is available on the latest board hardware, so such a circuit could be added on if necessary.

3.4.4 Real-Time Operating System

One of the key design decisions for this hardware was the decision to use the Real-Time Operating System (RTOS), FreeRTOS. Typical microcontroller-hosted applications run in a single thread. In many cases, these applications simply loop a series of operations over and over ad infinitum. Slightly more complicated systems may use a priority interrupt, to pause the loop when a certain event occurs, execute a bit of processing code, and return to the main loop. When the code reaches a high level of complexity, this looping execution is no longer sufficient as it can cause delays in execution or even back up so much as to miss incoming data.

To capture the maximum amount of data (by preventing buffering issues as well as operating at a higher rate of speed), the author chose to use an RTOS populated with the following tasks:

Upload – upload data to modem

Accelerometer – capture accelerometer data

GPS – capture GPS data

OBD – capture OBD data

Record – write message to SD card

The RTOS is configured with a time-sharing scheduler. This type of scheduler switches tasks on regularly clocked interrupts, creating the illusion of seamlessly running multiple programs when in reality they alternate sequentially but generally deterministically (finishing in a set amount of time). Incoming data interrupts add an additional layer of complexity to this design, but a thoughtfully crafted scheduler allows the device to do more, in a more repeatable manner, with less hardware resources.

The RTOS allows more predictable steady state operation of the hardware, but it is not without its challenges. The OS requires additional FLASH and RAM space. It also presents many issues with sharing data between tasks. In the design and development of our software, the RTOS was the root cause of a number of issues. Most commonly, these were attributable to memory leaks and unpredictable memory addressing. As the code became more complicated, RAM limitations forced constant optimization of software to avoid stack/heap collisions that would result in unpredictable performance and hard to trace problems.

3.4.5 Processor Selection

The processor onboard is an LPC2129 by NXP, an ARM7TDMI processor. These processors are among the most widely used ARM codes, keeping costs low and offering multiple angles to approach the supply chain. The LPC2129 was an ideal choice as it is natively supported by FreeRTOS and offers sufficient peripherals for all of our interface devices (2x UART, 2x CAN, 3x ADC). All programming for this processor can be done with freeware tools like TextWrangler, ARM-ELF-GCC, and FlashMAGIC. Similar processors offer additional RAM, FLASH, and peripherals, allowing an easy path for future development.

3.4.6 Software Features

The embedded software is primarily a data acquisition and logging system, but has several interesting features. The device is capable of reading configurations from the SD memory card, a feature that greatly reduced development time and allowed easier debugging when testing. These parameters on the SD memory card include SIM card parameters, which OBD parameters to sample, sampling rates for all tasks, and server addresses. A remote update task is implemented (but disabled until a password verification is implemented due to security concerns), as is a "heartbeat" wakeup to allow the device to connect to a server and download an update even if the module is in sleep mode. Basic file system manipulation over the air is included, allowing files to be read remotely, in addition to buffered file recording and uploading on shutdown. VIN reading is included but was a difficult feature to develop, as some manufacturers use different addressing schemes (*e.g.*, GM, who will not respond to a VIN request from functional address 0x7DF on the ECU, and must be directly messaged from functional address 0x7E0). The code automatically adapts in these cases, trying different functional addresses until the module receives a valid reply. Finally, power saving through voltage-based shutdown is possible in this code, but is not implemented, as the most recent hardware does not support voltage reading from the OBD port (instead using the analog input to sample an accelerometer channel). This feature may also not work as expected in (H)EV vehicles due to different voltage regulating technologies, but may be worth exploring. The below list mentions several other features implemented since the author's undergraduate thesis.

- Band configuration and SIM PIN support
- Remote reconfiguration
- Remote software update
- Accelerometer, GPS and OBDII data upload
- Version reporting
- Remote bootloader upgrading
- File sending (completed logs)
- Check engine – read and clear light
- Send/receive packet on secondary network
- VIN reading and reporting
- Automatic reconnection

3.4.7 Security

This document will not discuss the security of in-vehicle networks, as this is a widely researched field and the problems with in-vehicle networks are not addressable with diagnostic hardware. However, it should be noted that remotely updatable hardware can pose a security threat and these concerns must be addressed on both the embedded hardware and on the server to ensure only authorized code makes it to the module. Even local flash events (via SD memory card) are possible to exploit, and these issues must be explored.

For car to server communication, the GE864-GPS cellular module supports SSL encryption. The embedded software provides a framework for this interface; however, the server does not yet support this method of connection.

3.4.8 Bootloader Development

Initially, a USB in-circuit serial programming header loads software. This software downloading routine is reliable and cannot be corrupted as it resides in non-writable memory regions. However, it requires specialized hardware and software to deploy updates, which adds cost, complexity, and time when programming.

In testing, it became apparent that a bootloader would be necessary to recover from flawed updates or flash events resulting in corrupted memory locations. The immediate solution to this problem was to design and deploy a bootloader that reads updates from the microSD card and stores the contents to the code region in memory. This was done through active collaboration with a contractor located on the online freelance site, oDesk. The designed bootloader is invoked by the hard-coded bootloader, checks for the existence of a file, runs an update routine if present, or launches the main application if no update is available. This ensures that, should the software become corrupted, the bootloader will always run and any bad flash event may be recovered in the worst case by removing the memory card and copying a file from a PC to the memory card's FAT partition. The bootloader also stores "backup" images to copy over should a flash attempt fail, but no file validity checking at present.

An extension to this concept is the over the air (OTA) bootloader, which automatically downloads updates from the server to the memory card for invocation upon startup. Updates are triggered upon noticing a change in version number between the installed software and the latest available, invoked directly from the server database, or in response to a failed POST. Updates are sent 512 bytes at a time and emulate the SD card SPI interface, by validating transmission with a CRC16 checksum – reducing required code space by allowing the sharing of libraries. In testing, this method has worked well except in the single case where an update pushed to connected devices had an error in the file download routine. In this case, the local update routine (copying files) avoided a more embarrassing resolution to the problem.

Security in validating updates is critical, but the author was unable to deploy any protection against attacks by the time of this writing. This would ideally involve authenticating firmware images and connection channels. It may also require the cryptographic signing of a file to be decoded with a private key or shared secret key. This hash would encode version, target platform, format, and other metadata.

3.5 Hardware Production / Software Development Stair-steps

The hardware rollout was driven by the VMT application example, and could be broken into four distinct stages of software development. Code for version one provided hardware and software capable of determining vehicle miles travelled using only OBDII diagnostic data, which would ensure that the OBDII portion of the hardware met the specifications set forth earlier for sample rate and that the hardware could connect to the intended networks. Version two expanded upon this hardware with the ability to collect data for fuel economy estimation (essentially a higher-bandwidth capture, often pushing the limits of the vehicle ECU). Version three added cellular communication, which challenged the connection, reconnection, and keep-alive routine design, while version four added the requirement of GPS data logging and retransmission. A reach-goal version 6 added a second CAN transceiver for sensing and actuating on a second network, like General Motors GMLAN single wire BUS (necessitating a fault-tolerant CAN transceiver to avoid the generation of RX passive errors and BUS OFF situations – a fact discovered late in testing). All versions would be tested on an OzenElektronik bench top OBDII simulator and then in a series of vehicles owned by friends of the author.[7] The hardware for all versions remained unchanged, speaking to the successful implementation of a futureproof hardware architecture.

To produce the hardware required sending board design files ("Gerbers") to a fabrication house in Shenzhen, China, who produced a single example PCB to prove the hardware worked prior to sending 99 more devices. To test these devices, the team created a user guide for the hardware and gave devices to end-users without any coaching. The results of this consumer testing will be discussed at length in a later section, and lessons learned in the design process are summarized below.

3.6 Hardware and Software Obstacles and Amelioration

The development of this hardware faced constant barriers to deployment on both the hardware and software sides. These subsections describe the challenges to deployment in detail.

3.6.1 Hardware Design Challenges

Power presented a significant issue in testing. A combination of improperly gauged wire and improperly low ESR capacitors caused brownout conditions where the modem shut off arbitrarily. This problem was addressable through the creation of a watchdog timer to ping the cellular module and reapply power if necessary.

[7] http://www.ozenelektronik.com/

Antenna design was, and remains, a problem. The previous cellular module, the GM862, provided internal LNA amplifiers that were powered by the primary input voltage. The newer module, the GE864, was "command compatible" but not pin compatible and did not power the LNA from the module input voltage directly. This resulted in confusion when an order of 100 active antennas arrived and the board did not have provisions to power them. The antennas were not returnable, so to avoid significantly soldering rework the author had to order passive antennas. These antennas were non-stock items and had to be custom manufactured, adding weeks to the deployment time for the hardware.

The final identified hardware issue was minor, but an annoyance. The author incorrectly specified a part number resulting in the installation of inverted transistors (NPN instead of PNP), resulting in the need for hardware version compiler flags (to invert the power-on command for the modem), and resulting in the inversion of feedback LEDs (turning lights on to indicate sleep mode rather than activity, for example).

3.6.2 Software Complexities

Software development was plagued with similar issues. The most difficult to solve was the implementation of an SPI-based driver for microSD memory cards. The hardware did not have sufficient free pins available to implement SDIO, so SPI became necessary. Evidently, microSD cards are not required to support the SPI specification in the same way that full size SD cards do. Many cards did not support the specification, and others required the use of low-speed interfacing, causing complications with the scheduler system and limiting the maximum data capture rate. SDHC memory cards also presented difficulty in that these cards required automatic identification and a shift from byte to block addressing. 2GB cards were largely not usable due to their use of 1024-byte sectors rather than 512-byte. Ultimately, the solution to these problems required a complete rewrite of both the SPI and SD card drivers, as well as tweaks to the filesystem used (FatFS).

The second software difficulty was equally challenging to debug. As the code grew, the hardware began to act in unexpected and non-repeatable ways, rebooting or printing incorrect values or simply stalling altogether. This was later determined to be a stack overflow. A JTAG debugger would have been incredibly helpful in diagnosing these problems, but printing "high water" marks indicating peak memory use values sped up the development of the software after the first several stack/heap collisions had been identified.

Despite attempts to eliminate bugs, some external factors – like a large number of error frames on the network in conjunction with an unreliable microSD memory card – could in rare cases result in file loss, including the configuration file. This would prevent the hardware from connecting to the server, and remote updates could not be deployed, even though the board was capable of identifying the problem. Though by the time of this writing this error has not been repeated, the author implemented a solution to this problem. The hardware now has a list of default network providers stored to memory, and automatically cycles through them if the configuration file is not found. The device logs in with a special user identification number, and the server knows to send a new configuration file to the device. In the event a bug locks the software up entirely, a watchdog timer reboots the board at 24-hour intervals, during which the hardware may check the server for the availability of updates.

3.7 Novel Development: Interesting Features and Testing Results

The previous version of the ARM7-based hardware was a proof of concept and as such did not deploy many of the more advanced features the author would have liked. The most recent version received several upgrades during the development process, and the following section discusses these features in depth.

3.7.1 Auto Off

Many vehicle diagnostic tools do not automatically shut off. For low power devices, this is not a problem as the vehicle battery can provide for weeks and even months of uninterrupted operation between charging. However, more complicated diagnostic devices consume considerably more power through the use of more advanced processing units, displays, or wireless communications modules and powering these devices without excessive battery drain becomes challenging.

This solution automatically detects when the vehicle is stationary and turned off and enters a low power mode to conserve battery life on vehicles which lack the capability to turn off the diagnostic port (the majority of those tested, as of this writing). Additionally, these data will be useful in determining the stopped state of a vehicle better than existing technology, as it facilitates knowing when the vehicle is simply stopped versus parked. Sample characteristics - such as RPM - combine with other signals - such as electric motor speed, in hybrid vehicles - to determine when the vehicle has been parked sufficiently long to warrant turning off the device. Similarly, a low-power mode may poll for the values from these and other sensors, like accelerometers, to determine when the vehicle is once again moving and re-enable high-power features. The net result is the same volume of data as an always-on system, with the added advantage of consuming significantly less power. Owners of devices with this added functionality do not have to worry about unplugging their diagnostic tool, even during extended parking.

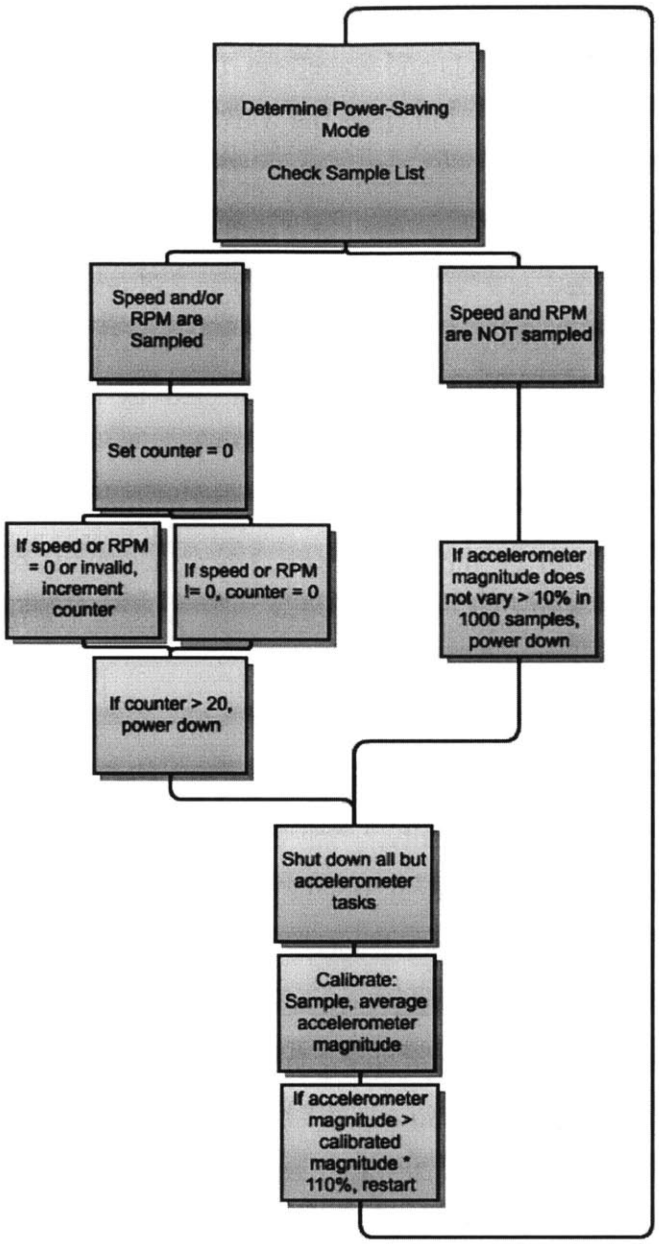

Determine Power-Saving Mode
Check Sample List
Speed and/or RPM are Sampled
Speed and RPM are NOT sampled
Set counter = 0
If speed or RPM = 0 or invalid, increment counter
If speed or RPM != 0, counter = 0
If accelerometer magnitude does not vary > 10% in 1000 samples, power down
If counter > 20, power down
Shut down all but accelerometer tasks
Calibrate: Sample, average accelerometer magnitude
If accelerometer magnitude > calibrated magnitude * 110%, restart

Figure 12: *Flow chart describing power save and wakeup conditions, parameters read*

The author tested several versions of this technology. In the first example case, a sensor monitors engine speed (RPM). Based on engine speed, it may be determined if the engine is on or off. If the engine is off for multiple samples of this datum, it may be reasonable to assume the vehicle is off – and to cut power to the device.

In another example case, RPM is not being sampled, but the device reads input voltage. Examining the voltage levels and waveform, it is possible to see that the voltage drops when the engine is shut off and in cases the waveform becomes smooth. Again, this case would lead to the device being shut off.

The third case requires a parameter not already sampled be added to the sample list. This value may be added as a "low priority" sample and read when the device is not already busy, or at very infrequent intervals. This parameter is not already being requested by the program on the device or connected to the device. It might be "velocity." This parameter, or others like it, may be preferred even if other samples such as RPM are already present. This is because the gasoline engine may shut off in a hybrid vehicle, but the car may still be moving. Again, when the device determines that the vehicle has been stopped for a predetermined number of samples or a predetermined time, the device enters low power mode.

The system may use historic data to determine driver and vehicle trends to optimize on/off thresholds and sampling rates for these data, but in testing, relied on hard-coded values determined empirically.

In low power mode, non-critical functions are shut off. These functions may include wireless connectivity, indicator lights, displays, or additional sensors such as accelerometers. The device may turn the processor off, put it in a low-power mode, or keep the processor running at maximum speed. In cases where the processor remains on, the same technique used to shut the vehicle off may be used to determine the vehicle is on and reactivate the device. In testing, the hardware disabled all extraneous communications and sensing devices.

This provides the diagnostic tool with greatly improved battery life, and adds functionality in the form of stopped/moving and/or on/off detection. These data will facilitate the development of further applications, such as traffic prediction algorithms.

After deploying this method on the fleet of testing vehicles, a "bug" was discovered. "Nagging" PID requests keeps the ECU awake as it expects further incoming requests. One solution is to throttle the sampling rate in sleep mode, but this is not a reliable way to wake up and severely decreases responsiveness – especially seeing as some cars tested had a 30-minute timeout (meaning a 30 minute sample period would be necessary, and could therefore miss capturing data from any drives taking under 30 minutes).

A new approach leverages accelerometer data to wakeup when motion is detected, while still shutting off using OBD. This improves responsiveness, but is not without challenges of its own. Calibration is difficult – there are a number of routines, but a two-point average and a correction factor allowing for noise seems to work well. However, to keep the device sensitive enough to wake up even when in a vehicle driven gently, the threshold for power-on must be set low enough that vibrations like passing trains can wake the module. This required the development of a "NO OBD" recognition algorithm to turn the device off if the car has already shut down and the engine computer is returning invalid data. The advantage to this approach is that data are reported after vibration events, allowing for the creation of "bump and run" detecting applications.

The development of these routines led to the creation of paring and throttling routines to stop requests for invalid parameters, either increasing the time interval between OBD requests or removing parameters from the array of PIDs to be sampled. This raises the number of samples per second for other parameters and reduces filesize that would have been taken by invariant sensor values. A secondary benefit to this throttling routine is that cars with slower ECUs that cannot keep up with the board do not simply overrun and shut off – instead, the sample rate reduces until the board is able to capture data.

3.7.2 Assuring Data Validity

Several applications for OBDII data require reliability of data capture, or in cases, the ability to determine if hardware has been tampered with. The new CANPuter implements many novel features to ensure data reliability, including tamper-proofing and anti-spoofing measures.

One example scenario is the use of automotive diagnostic CANBus for fleet or personal vehicle tracking software. There is not an immediately perceptible benefit to the drivers of the vehicles, but the data are critical for the fleet managers to understand and optimize the utilization of their vehicles. Individual drivers may wish to hide their driving habits, leading vehicle operators to disconnect these devices to prevent a perceived invasion of privacy.

Presently, diagnostic system security metrics are either "security through obscurity," or complicated install procedures that add cost and complexity to deploying a new fleet vehicle, despite the fact that a sufficiently driven person could remove or disable the system – often with basic hand tools. Other systems forego security and rely on an honor system, as they make use of the plug-and-play diagnostic port on the dashboard of cars or under the hood of large trucks. In both cases, a disconnected device does not leave any indication that it has been disconnected – it simply looks as though the vehicle has been turned off. Powering on the device with a bench top power supply at a fixed location, for example, would make it appear as the vehicle was being stored in a garage and not run.

This ease of deception makes relying on telematics and diagnostics systems unfavorable. A tamper-proof system improves the reliability of the data by preventing "spoofing" or removal of the device in its entirety. Detecting that the device has been removed from the vehicle and reporting this change in status, immediately or upon later memory inspection, accomplish this. Invalid data may then be discarded and potential problem users may be identified in order to take appropriate corrective actions.

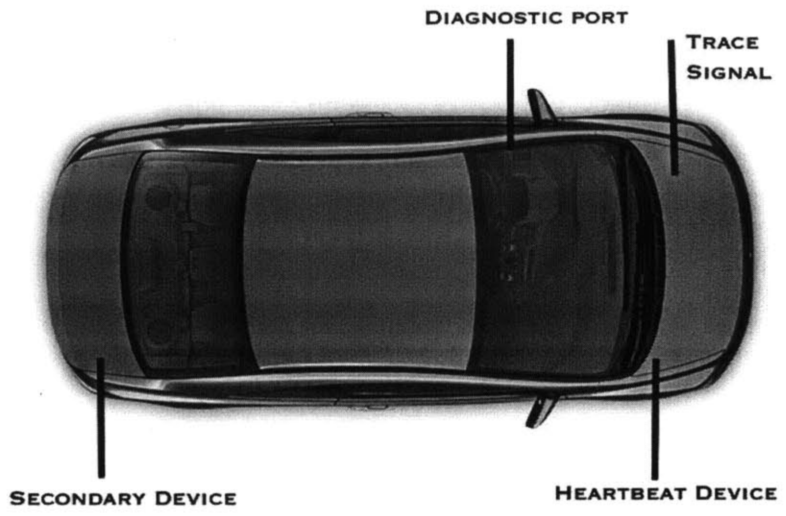


Figure 13: *Visualization of location of "security" nodes in a typical vehicle*

In the above diagram, the CANPuter plugs into the diagnostic port under the dashboard. Depending on the implementation, the device "talks" to one, two, or all three other devices in the vehicle and on the same network to ensure it has not been tampered with. The secondary device responds to "rolling codes" that follow a call and response pattern, the heartbeat device is a device already existing in the network that responds to queries regardless of operational state, and the trace signal is a device already on the network that returns data that can easily be classified (such as signal noise levels, voltage changes over time, or similar). In the above diagram, the terms are defined as follows:

Diagnostic port – OBD2 J1962 connector, where the primary device plugs in.

Secondary device – trunk/under hood/*etc* mounted device for call-and-response purposes. If one device is removed, the other will remain on the network and set a flag in write-once memory

Heartbeat device – a device that is already on the vehicle network (like an airflow sensor) that responds to the primary device in a standardized way, to ensure the device is not removed from the network

Trace signal – a device such as a battery that reports a value that may be indicative of operating state, *e.g*. battery voltage varying with time

There are several ways to determine that the device has been removed from the system. In some networks, simply detecting the removal of the external power source suffices, which could be accomplished by sensing input voltage for level and waveform. In other cases, the device might lose power frequently and power sensing would not be a viable option. A second solution is to install two units in different, possibly hidden, locations, allowing the units to communicate with one another over the network using rolling codes or similar as a form of call and response. This way, if one unit is removed, the other unit must be removed before it senses that a unit is missing from the system. The rolling code adds an additional aspect of security to the system by making it difficult to learn the format of the call/response in a short time. A third option would be to have the unit query other devices on the network, such as sensors or additional computer modules, to ensure that it receives replies and that the replies are of an appropriate format. This type of sampling falls into two categories: predicted response, and known format.

In a resting vehicle, battery voltage is a reliable option, as it can be sensed even after the vehicle is shut off and will change over time, making simulation more difficult. Noise characteristics, waveform, and general trends feed into determining if the predicted response is valid.

In a moving vehicle, RPM or other sensor readings may be more appropriate. These respond with a known format that would require a dedicated simulator to mimic. Depending on the parameter, known format and predicted response may be combined where additional classification is appropriate (for example, RPM in an idling vehicle will rarely report the same value twice – so searching for repeated queries may be helpful in determining that the data is invalid). In cases where tampering is highly likely, multiple sensors, including some that may be located on the tamper-proof device itself, may be compared, for example vehicle speed and GPS speed. This further reduces the likelihood of a simulation device going unnoticed.

As a final tamper-mitigation tactic, multiple heartbeat devices may be used, ensuring that a simulator would have to respond – in the expected manner – to more than one type of message, perhaps in a pseudorandom order.

When the device is unplugged, disconnected from the network, or fed data from a simulator, it will report the change in status to the party attempting to gather data.

Reporting may rely on battery backup to send an SMS or TCP/UDP packet to a server to notify the party that the device has been tampered with. It may set off a light on the device and/or a buzzer, notifying the user of the problem. It may be possible to avoid the use of a backup battery by keeping a variable stored in semi-permanent / permanent memory, in cases including the time of disconnect for auditing purposes. In an example case, the device reads this memory upon startup to determine whether or not the removal was authorized and notifies the server if it was removed inappropriately. Alternatively, the memory may be read with an external reading device to identify tampering without requiring remote network connectivity.

In testing, communicating with other nodes belonging to the vehicle network or checking for the presence of a broadcast message on the conventional OBDII network was effective in determining if the device had been removed from the vehicle while powered on. To determine the status when powered off led to as-yet unresolved battery issues (due to powering multiple nodes and/or keeping ECU's awake), but a rechargeable backup battery in a secondary device is a likely solution to this problem.

3.8 Secondary Network Sensing and Actuation

A feature not included in any existing consumer diagnostic tools is the inclusion of a secondary CAN transceiver, allowing communication on additional vehicle networks without requiring the use of a gateway node. The inclusion of this transceiver allows the device to read or transmit messages on other networks that may include comfort and convenience or infotainment. Messages are sent or received using a special "SEND" command from the server that identifies the network on which the command will be sent, and includes the arbitration ID of the transmitted message, data payload, and return arbitration ID and read timeout. Any portion may be left blank to enable sending without reading or reading without sending. In this way, the server can issue a command to operate an actuator or poll a value from a non-OBD sensor.

Ultimately, this system may be migrated to a token-based approach to improve security and enable a "firewall" approach running on the embedded hardware. A dictionary stored on the microSD card allows for sufficient space to define rules for these tokens, and would reduce the likelihood of malicious communication attempts without first requiring physical access. Additional modification will include a "constant read" mode, allowing parameters which are regularly broadcast to be read.

3.9 Closing Remarks: The CANPuter in the Context of CloudThink

Regardless of this particular hardware implementation, the platform is open and therefore accessible with other hardware versions. The focus here is the data, and the author believes that "black boxing" hardware to create a transparent user experience leads to generation of better data. Despite this, the open source nature of this project drives creativity and avoids potential IP conflicts, while allowing freedom to innovate. The CloudThink standard will allow other hardware manufacturers to enter the field and bring innovation to drive data generation along with user satisfaction.

3.10 Integration with the server

With the hardware on track, designing data capture and application servers became the next focus of the project. This section describes integration of the new CloudThink embedded hardware with the server, though goes less in-depth as Simon Mayer (PhD Candidate, ETH Zurich) took over server development responsibilities midway through the project. This section will focus on the author's contributions and more general themes.

3.11 Server Design Architecture

The server has several responsibilities. Primarily, it must provide a means of collecting data from all CloudThink devices in the field and managing the database. This will provide users and developers alike with a trusted storage location for their data, free from data siloing and with unique access control methods to ensure security and privacy.

Secondarily, the server must serve data to third parties though an API to allow the creation to applications. This handles the difficulties associated with creating a platform to allow developers to focus on their core competencies: building compelling, value-add applications. The server handles security, sharing tools, per-query billing, and a share/subsidy model to allow users to share their data with trusted third parties in exchange for free or discounted applications or bandwidth.

The database server captures incoming data via a Python or Java script and stores entries in a MySQL database. These entries are logged per-VIN, or per-UID (unique [hardware] identifier) if VIN is unavailable. The values stored to the database are human-readable, *e.g.*, HEX is converted to ASCII format. Values may be accessed using a RESTful API and require per-VIN credentials.

The server is based off the Python script from the author's undergraduate thesis, with improvements for more robust connection management, two-way data, and improved error handling. Three entirely new features include the processing of gap data, remote updates, and bidirectional communication to allow for actuation. Gap data processing allows the storage of bulk data after a drive, filling in the interstitial gaps left by the more sparse real-time data stream with contents that had been logged to the microSD card and uploaded after vehicle shutdown. Remote updates push update binary files to connected devices 512 bytes and a checksum at a time, while actuation sends a command to individual connected nodes to send and/or expect values to or from particular arbitration IDs on secondary networks. This was all accomplished using the existing Twisted Framework software and making use of "self.transport.write" function in conjunction with per-vehicle variable storage to ensure replies were sent to the proper receiving node.[8]

Simon Mayer wrote the application server using Java with a Glassfish/Grizzly framework to create a RESTful API capable of outputting both machine and human readable data. This program tracks queries for possible future billing purposes and has provisions to support user accounts and permissions to ensure user security, which is validated by an SSL certificate.

User privacy is an explicit goal for the CloudThink platform. Users must be able to control the entities with access to their data, and to what extent these data are shared. The server was designed with avoiding common social network pitfalls, such as insufficient or poorly visualized access control, in mind. One problem not addressed and a subject for future work is ensuring privacy from the platform owners, either by encrypting data prior to storage in a database and sharing a decryption key, or some other method of ensuring data are obfuscated that may be less computationally intensive.

[8] http://twistedmatrix.com/trac/

Without consumer focus groups, it would be difficult to anticipate user desires at this stage in development.

A basic depiction of data flow to and from the server platform appears below, in **Figure 14**.

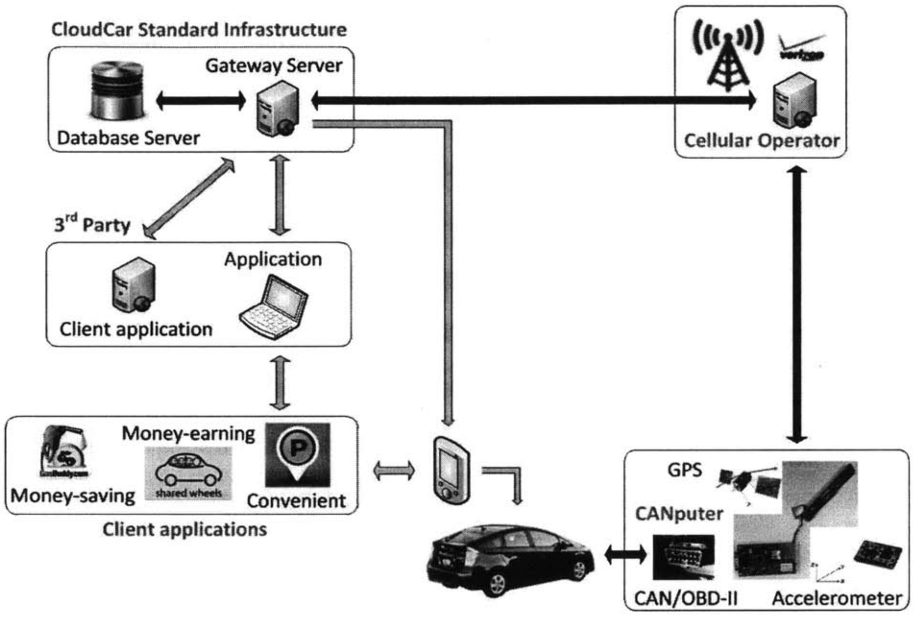


Figure 14: *CloudThink system architecture showing the CANputer and 3rd party servers hosting client applications*

4. Chapter 4: Deployment and Testing Results

This chapter covers the production and deployment of the first 100 units of the version 7 hardware, the state of the server throughout rollout, results from the first users, and the status of CloudThink and testing results for the canonical VMT and fuel metering applications.

4.1 Problems and Resultant Lessons

This section discusses lessons learned throughout the hardware, software, and server development broken down by development group. In many cases, these points that may seem obvious are actually subtle and have cost days or weeks of development time to resolve. From the author's perspective, these notes are the most valuable takeaways from the entire project and may help designers of future systems avoid similar development challenges.

4.1.1 Hardware Production

The role of supply chain on timing and cost was substantial in this project. While all software development was done in house, manufacturing of the electronics was outsourced to MyroPCB in Shenzhen, China.[9] The author ordered integrated circuits and passive components from US-based suppliers and shipped these to China for final assembly. MyroPCB is a high-tech fabrication facility and produces PCBs in house along with stainless steel stencils for solder paste application. Pick and place machines assemble boards, using adhesives to allow for the creation of double-sided boards, and populated PCBs are then reflowed and/or wave soldered for final assembly. MyroPCB is a medium volume production facility, but understood the complexity of the MIT hardware design and offered to produce sample boards prior to the production run of 100 units. This came in useful after a flaw was discovered in the first sample of "Version 6" that prevented power from being applied to the GPS module and led to a revision to "Version 7".

MyroPCB was professional and reasonably priced. In the future, a local supply house for the prototype boards would greatly reduce lead time as each design revision

[9] http://www.myropcb.com/

took at least a week to manufacture and another week in transit. The project's funding was also tied up with MyroPCB for quite some time due to the redesign after the first sample, and import taxes were incredibly high on an order of this size (bare PCBs do not have an import duty when arriving to the United States, but assembled electronic goods do). With the boards produced, the supply chain for support hardware was identified. Alibaba and AliExpress were the most useful suppliers at this stage in procurement, but the components identified were not without problems.[10]

The J1962 connectors and cables from AliExpress were faulty, though a lack of on-arrival testing and long shipping lead times compressed the project timeline to the point that these cables had to be used in final assembly of the "Version 7" hardware for distribution. Further complicating matters, there were at least three distinct types of wires and connectors, some of which were too small in gauge (causing brownout power conditions), some of which arrived with chipped connectors, and some of which became brittle and cracked in cold weather. To properly connect these wires required continuity testing and constant spot-checking, taking a team of four students three full working days to complete and solder to the PCBA. The lesson here is to fully specify each component, no matter how small.

Figure 15: *Failure to specify material properties led to cracked cables and multiple wiring configurations*

As mentioned in the design section of this document, antennas were problematic for sourcing. The original GE864 PCB design required a type of unpowered (passive) GPS antenna that was not produced and sold through common distribution channels, and

[10] http://www.alibaba.com/, http://www.aliexpress.com/

redesigning the PCB was not an option as the setup costs for samples would cost more than even the most expensive bulk antenna purchases. It took weeks to locate a supplier for these antennas, and the end result is a product that works well but is not optimized for signal strength or price. From this failure to identify component sources, the author learned to fully understand component support hardware requirements and identify several possible supply chains prior to placing any board orders. It might have been possible to avoid this problem altogether by reading datasheet errata and taking other steps to reduce design risk earlier in the planning stages, well before starting any production runs. Despite the failure to properly design the antenna interface, performance has not suffered appreciably and the antennas are within OEM design guidelines [48].

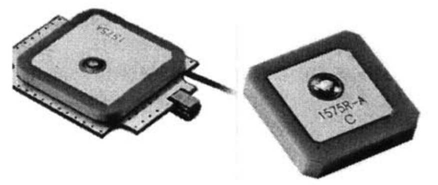


Figure 16: *Comparison of active (left) and passive (right) GPS antenna configurations. Note the active antenna has a cable connection, while typical passive antennas are "pin mount" or directly soldered to a substrate. The passive antennas used on the CANPuter are custom made from active antennas with unpopulated low-noise amplifier PCBs to mate with the IPEX connectors on the PCB.*

With these recent sourcing pitfalls in mind, and as a learning exercise, the author worked with FIL member Dylan Erb to design and manufacture injection-molded cases in the MIT LMP shop. Dylan took initiative in creating the CAD models for these cases, working creatively to design a symmetric snap fit to reduce machining complexity and cost. This involved creating a set of "mirrored" snap tabs and relying on adhesive to ensure a secure seal, as the injection molding machine used would not support undercuts and even a small lip on the mold would cause plastic deformation.

To test the case model, the author printed a test version on a MakerBot Replicator and then a Dimension 3D printer. The case as modeled worked well for 3D printing and would have translated excellently to injection molding. However, limited shop availability and the high cost of tooling meant that the injection molded part would require significant redesign from the "ideal" 3D printed model. The tab features were too small to machine reliably and the severe draft angle required due to the location of the ejector pins made smoothly interconnecting tabs impossible. Further complicating this translation, the MUD used by the injection molding machine was limited in size. After calculating the clamp force and shot size required and feeling satisfied that the machine could deliver such forces, the mold had to be machined down in one corner to avoid colliding with a locating pin, weakening the mold and allowing slight flex which led to significant plastic flash. After tuning the machine parameters, the end result was a case that is close to production ready, but one that requires post machining and is not water tight without filler adhesive due to surface imperfections. Post machining for these cases included trimming the flash off with a razor, inserting the cases in a wooden jig and drilling with a stepper bit, and applying a wiring grommet and adhesive. The process was simple but time consuming for the first 100 cases, and any larger production runs would be better suited to a remachined mold.

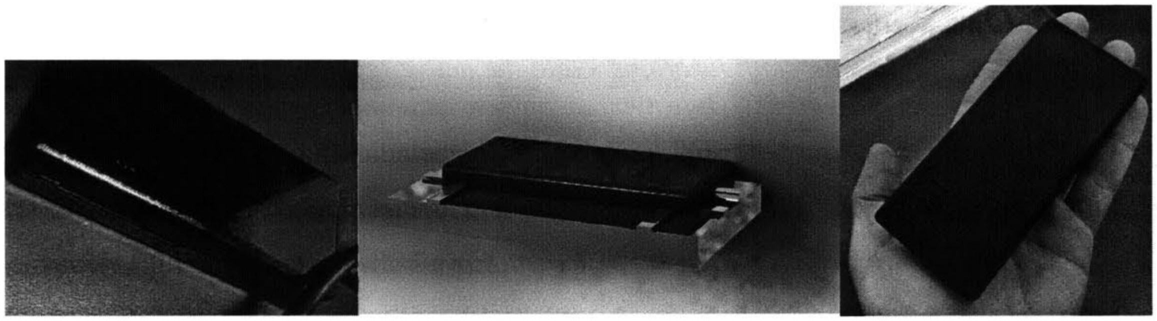


Figure 17: *Left to right –3D printed case, render of case on mold, and injection molded prototype case*

The following section describes many of the systemic problems faced during design and assembly and lessons learned addressing these issues.

4.1.2 Radio Chipset Problems a Result of Convoluted Documentation

The single largest problem in the design of the V7 hardware revolved around the radio chipset. In particular, the use of "pin compatible" in some marketing material appears to have been a misnomer when comparing the GM862 and GE864 modules. In the case of the GM862 to GE864 hardware swap, the pins were named identically but several had dissimilar functionality. Further, pins that were internally bridged in the GM862 required a trace to electrically connect on the GE864 module. This error prevented our GPS from working and required a new revision of the PCB, adding six weeks and hundreds of dollars in development costs.

The author also misread the Telit GE864 documentation, ordering GPS active antennas when in reality GPS passive antennas would have best suited the circuit. This took weeks to correct and required getting custom antennas made in China due to the use of a connector typically reserved for active antennas. In a similar vein, the author learned to test all antennas in the full use scenario, including enclosures. Attenuation dramatically changes signal strength, and antennas that performed well in one case did not work at all in others. This required specifying the GSM antenna after the case design was completed.

Due to a last-minute change in case dimensions, different antennas were sourced from the GaoKe Antenna Company. These antennas did not perform as well as the antennas that had originally been specified and caused problems, notably at lower frequencies. The intended specification appears in **Figure 18:** S11 plot indicates low rejection / strong signal and comparatively wide bandwidth for intended antenna in each of the four bands (approximately 800-900MHz and 1800-1900MHz). Results are based on EAD S-Quad *Datasheet network analyzer results.*, while the return loss problem is depicted in **Figure 19**. The notable omission of a fifth band, 2100MHz, manifested itself in problems in non-US testing, though this is a result of module selection rather than antenna specification.

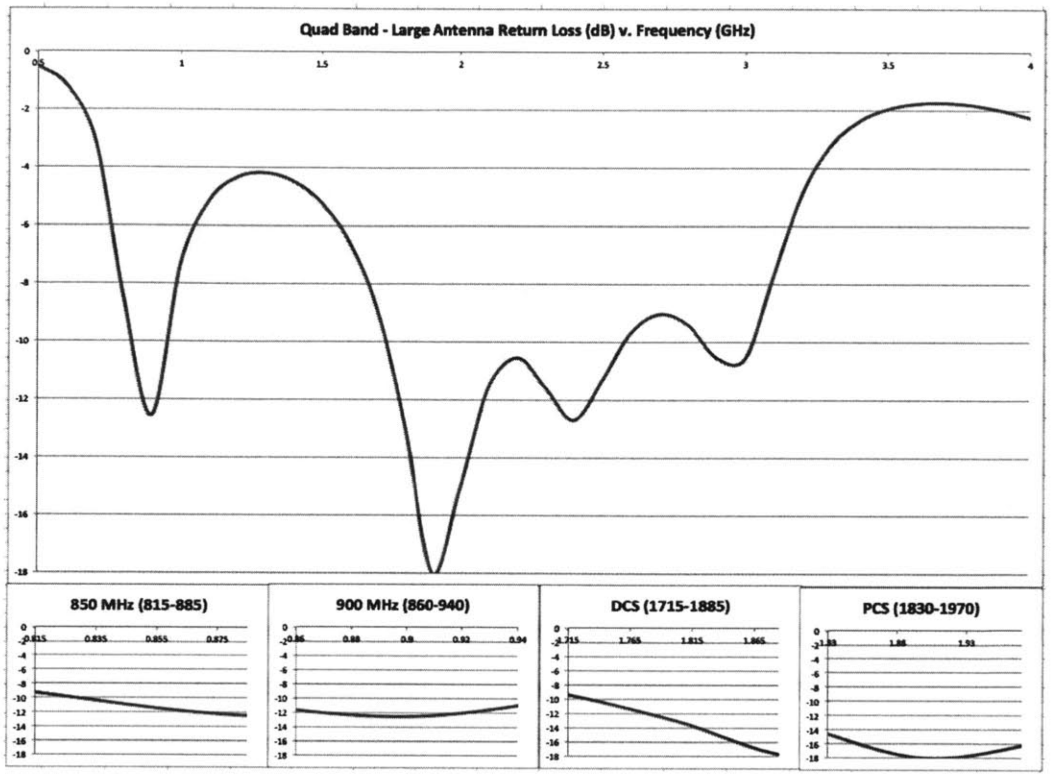


Figure 18: *S11 plot indicates low rejection / strong signal and comparatively wide bandwidth for intended antenna in each of the four bands (approximately 800-900MHz and 1800-1900MHz). Results are based on EAD S-Quad Datasheet network analyzer results. The return loss at the lower frequency band here is -12dB, whereas the antenna with poor performance is -5dB.*

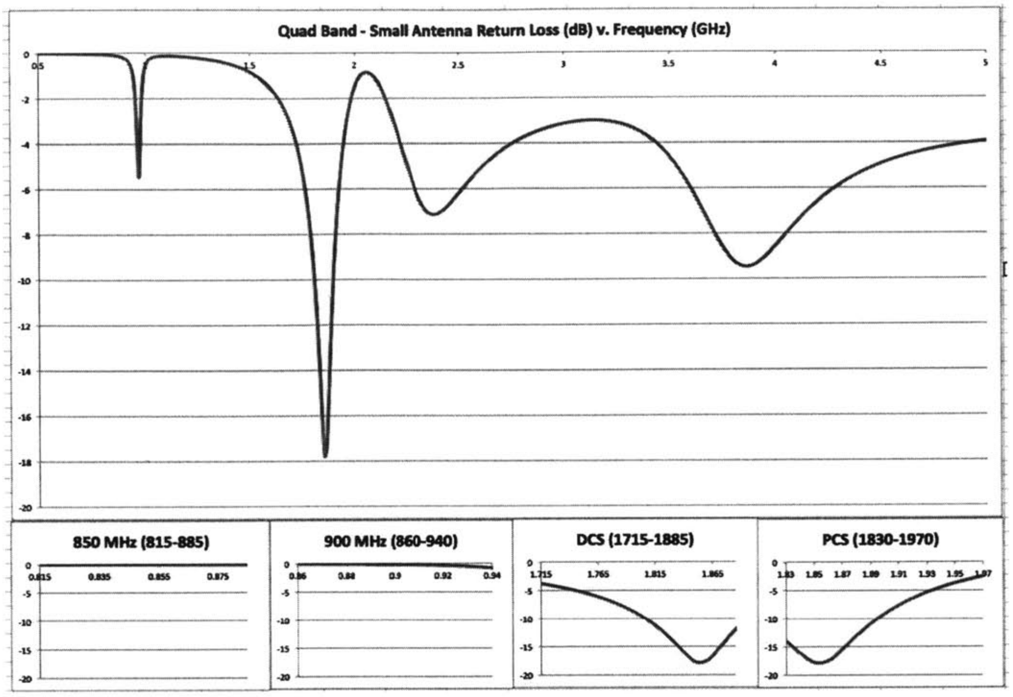


Figure 19: *S11 plot (return loss) – showing high rejection at low frequencies (850 and 900MHz band) and strong performance at higher frequencies (DCS and PCS bands). Simulation results from Ansys HFSS model of antenna, with help from Isaac Ehrenberg. A better-suited antenna would have a similar "dip" in the lower frequency bands as appears in the upper bands, indicating lower return losses and higher signal retention.*

4.1.3 Power Supply and Cabling Cause Hard-to-trace Problems

The power supply presented many challenges during the design and testing of the Version 7 hardware. Many users reported that the GSM status light would stop blinking (indicating failed connectivity), and analysis of server logs confirmed that the boards had non-cleanly disconnected. After tracing the issue with an oscilloscope, two problems were discovered. The first was an issue with the power supply wires being too small to supply the peak 2.5A the J1962 connector is rated for without measurable voltage drop. The second issue discovered was that the ESR on the switching power supply output capacitor was too high, leading to brownouts that could cause the cellular module to shut off unexpectedly. Rewiring individual boards with thicker cabling and replacing the capacitors or adding additional capacitors in series addressed these problems, but required significant effort to rework the 100 boards produced by the time these errors were discovered.

4.1.4 Memory Card Addressing Schemes Vary

Memory card variation was a cause of significant delay in the design process. When addressing (micro)SD memory cards, the SD specification – though costly – may have been a better approach than using SPI. Many microSD cards do not support the SPI specification fully. Similarly, 2GB memory cards were problematic to interface with – as they were developed prior to the SDHC specification, and use 1024 byte blocks instead of 512 byte blocks. In the end, software tweaks expanded compatibility, but 2GB cards are not usable and some microSD card brands may not work in the board at all. These same cards have been verified not to work in other devices that address the memory card using SPI, so the problem is not likely to be a solvable software issue.

4.1.5 Code / Debug Best Practices

This section describes several "hard learned" lessons about code development for embedded systems. The first is to develop with verbose error handling at the start – going back and adding fault tracing took more time than proper coding practices would have. Further, this verbose setting should be switchable by a compile time or runtime flag (ultimately, the embedded software uses a compile-time flag due to code space restrictions, but a runtime flag would be preferable).

Real-time scheduling is complicated. Periodically taking the time to map the scheduler's tasks was immensely useful and resulted in a more efficient program design that wasted fewer processor cycles. Similarly, mapping interrupt priorities and thinking through worst-case scenarios is a valuable exercise. Prior to doing this, the hardware would rarely exhibit unexpected and unrepeatable behavior.

The final point covers hardware and software [What?}. Despite building in verbose debugging over UART, tracing errors remained difficult. Designing in JTAG support or at least building the code in an environment capable of simulation would have saved significant time.

4.1.6 CANBus Complexity Resulted in Difficult-to-Trace Errors

CANBus issues were incredibly difficult to trace. Only after purchasing a neoVI Fire CAN sniffer and installing this device in parallel with the hardware was it possible to analyze why the code was not behaving as expected.[11] Viewing live communication helped determine, for example, that the acceptance filter had been set wrong and that data had been transmitted but the device was discarding the valid packets.

One of the most common problems had to do with functional versus direct addressing. For example, some modules would respond to messages sent by address 0x7DF (the diagnostic tool) while others would not respond unless directly addressed (*e.g.*, module id 0x7E8 would respond to queries from 0x7E0). This prevented reading VINs on GM vehicles until an automatic detection script was integrated into the software.

Another problem was querying too fast, effectively causing a denial of service attack. In this case, the network would appear to go offline, rendering the results useless and possibly distracting the driver by causing other modules to display error codes, or worse, becoming nonfunctional.

Despite the standardization of OBDII, non-OBD parameters are reported uniquely in different vehicles. The VIN, for example, might be reported one way in a Ford vehicle and another in a GM vehicle. When testing outside the US, this became more apparent – vehicles using the same ECU part numbers as their US counterparts would not respond to certain requests at all, despite sharing general OBDII diagnostic parameters (perhaps this software difference is due to fear of reverse engineering and minimizing exposure in nations where this is prevalent).

4.1.7 General Development Lessons Learned

One of the most critical lessons learned from this project was that any component or assembly sourced from a supplier must be fully defined. The author failed to specify the plastic type for cable assemblies, and consequently these cables became brittle and snapped due to in-vehicle vibration in cold weather.

[11] http://www.intrepidcs.com/

Proper ESD packaging and safety is critical in the development phase. Only once did ESD cause a problem with a circuit board, but this problem was difficult to diagnose.

Many cars on the road are driving with poorly maintained charging systems. Despite having a minimal power requirement, the hardware was responsible for discharging at least two car batteries in testing. Upon evaluation, the hardware was determined to be in-specification, but the battery failed load testing when removed from the vehicle. This problem was compounded by early auto-off software which kept the ECUs from sleeping fully and requiring an addition 5-10W of power draw.

4.2 Device Deployment

With the case designed, and 100 units manufactured, the units were deployed to testers. Early testers were selected for possessing only basic knowledge of computer and automotive systems, so that in the event of a failure users would be capable of deploying update files. While this saved the hassle of having to swap boards or visit testers to deploy updates, the testers sometimes took matters into their own hands, editing configuration files and changing settings that would make operation – or repair – difficult. After successfully verifying the functionality of remote update, testing moved to include users with no knowledge of embedded systems or diagnostic tools. These users had fewer complaints, and the remote update recovery scheme worked well in cases where programming errors required reflashing code (primarily, these events were tied to stack overflow issues which would manifest as continual rebooting). As of this writing, 32 boards have been deployed with 26 drivers operating regularly. There has been data transmitted from four countries and 12 states, validating the success of the hardware, software, and server at small scale. The remaining devices will be deployed shortly.

The following sections describe specific responses and results from the device deployment.

4.2.1 Network Testing: Full-Cycle Control is Key

In testing, the cellular network interface presented many challenges. The hardest issues to identify stemmed from RF design, and designing a switchable antenna setup to simulate various types of signal strength fluctuation and forced disconnect / reconnect testing would have saved significant time. Other stumbling blocks included partnering with networks to view SIM card activity, as MIT would not sign a conventional service contract with network providers. The initial AT&T M2M setup did not allow viewing connected devices or bandwidth use, making it hard to diagnose if connection problems were server side, hardware side, or somewhere on the network. AT&T introduced the author to their M2M provider, Jasper Wireless, who provided the author with 10 SIM cards and a portal to view data transmission in real-time and handle provisioning directly. This portal allowed enhanced troubleshooting and improved control over network issues such as bandwidth use.

Additional complications revolved around Access Point Name (APN) selection. Some APNs prohibited bidirectional communication or communication on some ports. This blocking was silent, and made debug of actuation and remote updating difficult. Finally, APN selection is case-sensitive, which caused problems when users were asked to self-update configuration files. AT&T's APN blocked outgoing communication from the server port, so only after starting collaboration with Jasper Wireless did functions like remote update and server-controlled actuation function as intended. With the Jasper portal, the author was able to diagnose the remaining problems quickly and successfully implement bidirectional data transfer, after the use of AT&T's APN server (which was not properly configured for the CANPuter's use) cost the author significant development time.

4.2.2 Consumer Sentiment: Reactions to Using Hardware and Data Provided Valuable Feedback

The author asked the CloudThink testers for feedback frequently throughout the design and testing process. Consumer sentiment was generally positive on the device side of testing, with many users commenting on how easy the device was to set up and use. Consumers liked the black-boxing of the data capture – a user rarely unplugs a silent device, but will often unplug a device that requires attention. With users generating data, they universally enjoyed being able to see the data their vehicle generated on simple web-enabled maps. Some did express privacy concerns, though demonstrating the username/password protection implemented on the server was enough to quell many fears. No names were stored on the server in testing.

The device did not distract from the operation of any vehicles, though some drivers complained about the location of the diagnostic port and suggested the hardware come with right-angle adapters to better route the cable. Many drivers experienced software issues early on, but POST and remote updates addressed several of these issues without user intervention. Developer feedback is still forthcoming, but early responses have been positive with many developers enjoying a one-size approach to their programming, allowing deployment of a single application across multiple web-enabled devices.

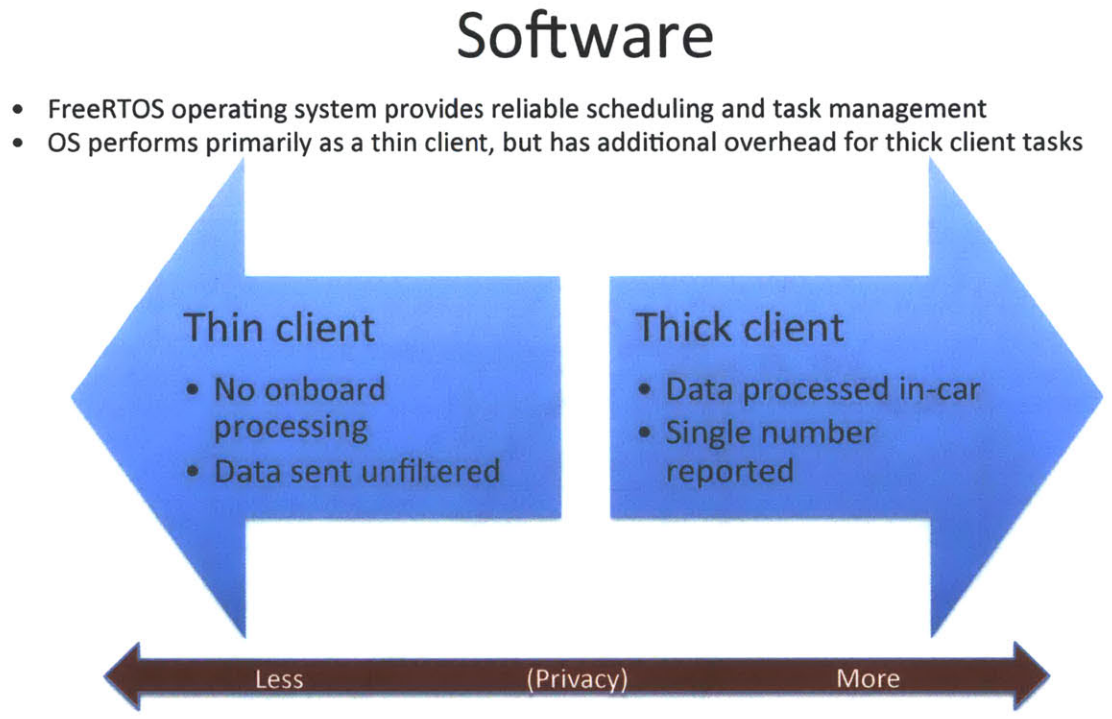
The lessons learned from testing the platform in the context of demonstration applications follow.

4.2.3 Using CloudThink for VMT Distance Monitoring

The decision to use On-Board Diagnostics (OBD) as the primary method for distance monitoring is mentioned earlier in this document. However, this decision fundamentally shaped the development of the embedded hardware and software. Alternatives like GPS, wheel encoders, or other direct-reading sensors would have less stringent hardware requirements than a network-based approach. These sensors would require interrupt-driven sampling and logging rather than a true scheduler-based system. The use of OBD rather than these simpler sensors necessitated complex software to properly translate messages and to deal with timing requirements for long-format messages.

OBD is a complicated specification that takes significant effort to implement. IC-level hardware is capable of providing abstraction for this communication, but these ICs are proprietary and therefore drive cost of supporting hardware significantly higher. Therefore, the decision was made to simplify from a universal OBD interface (SAE J1979) to the new federal diagnostic standard of Controller Area Networks, or CAN. This was deemed appropriate as all vehicles sold since MY2008 are CAN-enabled, and many MY2003 and newer vehicles support the specification due to the non-diagnostic use of CANBus in vehicles since the late 1980's. Today, CAN enabled vehicles make up a large portion of the active US fleet, and nearly all hybrid and electric vehicles support the standard. This decision to build strong CAN support allows for future expansion of network-sensor based VMT taxation, as heavy-duty vehicles follow a similar CAN specification (SAE J1939). These trucks are capable of interfacing with identical transceivers and have a wealth of available data on their networks, making high-GVWR VMT taxation feasible with minimal hardware and software changes.

A major design junction occurred when exploring thin and thick client models. A thin client passes raw data to a server, while a thick client processes data onboard prior to transmission. These models are characterized in **Figure 20**.



Software

- FreeRTOS operating system provides reliable scheduling and task management
- OS performs primarily as a thin client, but has additional overhead for thick client tasks

Thin client
- No onboard processing
- Data sent unfiltered

Thick client
- Data processed in-car
- Single number reported

Figure 20: *Thin clients pass raw data to a server, while thick clients process data onboard – requiring faster processors, but protecting privacy.*

To speed development of the platform and stick within the context of virtual mirroring rather than onboard aggregation and processing, a thin model was the ideal solution. This would allow better testing of the server architecture as well, by providing richer data sets and stress-testing the input parsers.

In the hardware's default configuration, it provides GPS data for localization. However, GPS is not a preferred metric as it can experience dither in poor atmospheric conditions, leading to questionable accuracy. Further, end-users do not like the idea of being "tracked." Therefore, OBD would have to be used to record distances traveled.

Distance travelled is not a metric reported by OBDII. There are other options – like speed, which can be integrated, or other sensors like accelerometers could be used, though such sensors require complex software filters and drift appreciably over time. Fortunately, there are metrics available that can be extended to approximate distance traveled, and these are a legislated part of the OBDII specification. These metrics are 0x21, distance in km since malfunction indicator lamp (MIL) turned on, and 0x31, distance in km since check engine light cleared (both have scaling factors ((A * 256) + B)). By combining these values and reading the state of the MIL from PID 0x01, it becomes possible to define logic to measure distances traveled over OBDII.

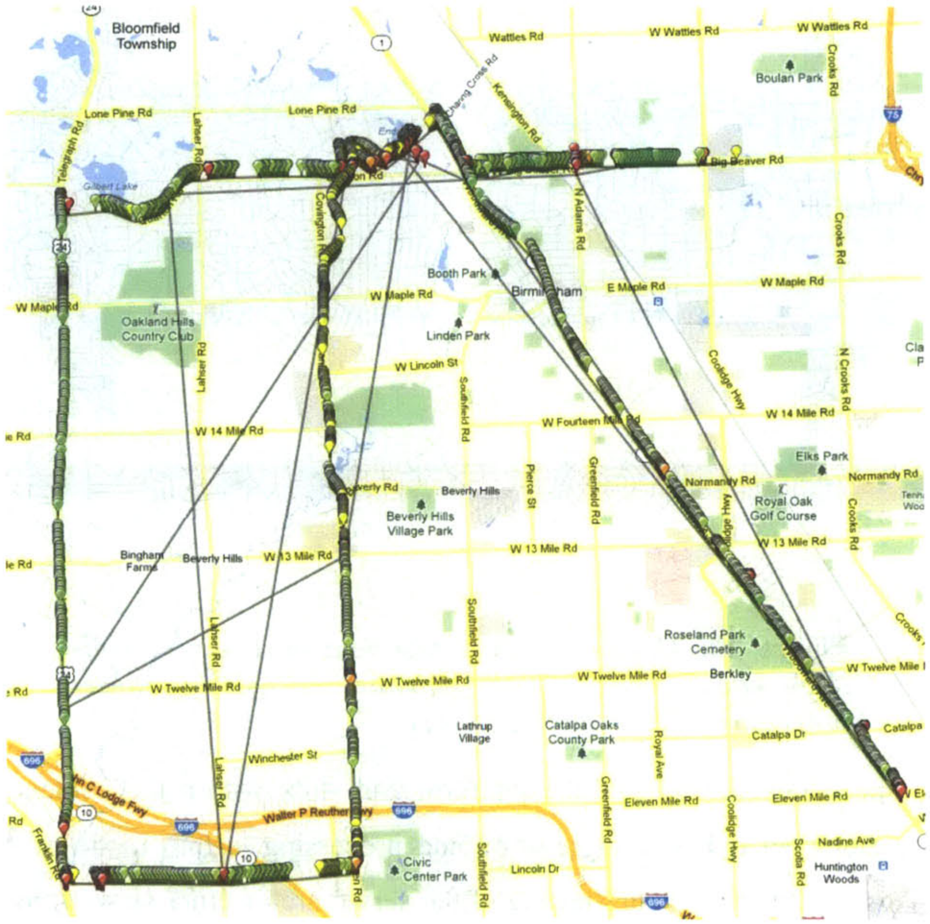


Figure 21: *Map demonstrating location of single vehicle path (from thin client). The lines are artifacts of using a single polyline to connect multiple trips in the Google Maps API V3.*

In cases where users will allow, map-snapping provides a great reality-check for these data, and a quick map-snap routine can be written using the Google Maps routing API and routing each point to the nearest road. In early testing, results were accurate – though every vehicle gathers the distance travelled differently (some manufacturers query the speed sensor from an individual wheel, other manufacturers average all sensors) and each sensor has its own error function.

Though there was not sufficient time to gather and compare distance metrics, early results using GPS data alone have resulted in errors of under 5% on trips 30km or greater. Further research into fusing OBD and GPS data using a Kalman filter is likely to yield even better results. This looks to be an excellent opportunity for continued testing and extension, perhaps to tracking or audit breadcrumb generation for salespeople looking to deduct vehicle-related expenses. This model is also the basis for extension to fuel economy measurements, and ultimately a similar system may be used to validate efficacy of policy changes, like recent updates to CAFE standards.

4.2.4 CloudThink Fuel Economy Estimation – Two Approaches

There are many methods for calculating fuel economy from diagnostic data, all with varying degrees of accuracy and slightly different data input requirements. There is further complication when taking into consideration the fact that not all OBD PIDs must be reported by law, and that even in cases where a PID must be reported, the sensor itself may not be utilized. While some manufacturers provide fuel economy data directly, this data does not address the larger problem statement of universal measurement. There are so many different specifications, even within one automotive make, that it would be infeasible to use these data to record distance traveled.

Provided here are two solutions for the most common vehicle setups – the first case, using the Mass Air Flow (MAF) sensor, should work in nearly every CANBus vehicle and is used commonly in industry and is calibration-free. The second case will work for vehicles without a MAF, but requires a calibration to approximate the Volumetric Efficiency (VE) of a particular engine. It should be possible to build a database of common vehicle VE's, but this is beyond the scope of the project. Please note that much of this discussion is based on an article by Bruce Lightner's 2004 Circuit

Cellar design contest winner, but is corroborated by independent analysis and reverse engineering of common diagnostic tools and software [49]. In both cases, further data (such as long- or short-term fuel trim, injector pulse width, or A/F from O2 sensors) may improve accuracy, but are not commonly available and increase computational complexity. Additionally, manufacturer-specific PIDs may report this value directly, but this is not a universal access method.

4.2.4.1 The MAF Method – Airflow Approximation of Fuel Economy

This method uses the reported flow rate of oxygen along with vehicle speed and a known combustion ratio to approximate fuel economy as shown in **Equation 2**.

Equation 2: $MPG = \frac{14.7*6.17*454*VSS*0.621371}{3600*\frac{MAF}{100}} = 710.7 * \frac{VSS}{MAF}$

$MPG = ideal\frac{A}{F}ratio * density\ of\ gasoline * vehicle\ speed * mass\ air\ flow\ rate * other\ dimension\ conversion\ parameters$

where VSS is in km/hr and MAF is in 100 grams/second.

The calculated results here are approximations, as they assume perfect stoichiometric combustion of gasoline. In modern cars, this should be close to the true value as the closed-loop feedback system utilized by the ECU holds the A/F ratio very close to the ideal 14.7:1. The primary limiting factor of accuracy here is the VSS signal accuracy, as well as temporal resolution (when calculating fuel economy over time). This method is less accurate during startup/warm-up conditions, as the A/F ratio varies here.

4.2.4.2 The MAF-less (IMAP) Method of Fuel Use Approximation

This method simulates the results of the MAF based on other sensors in the vehicle using the formula shown in **Equation 3, Equation 4** and **Equation 5**:

Equation 3: $IMAP = RPM * \frac{MAP}{IAT}$

Equation 4: $MAF = \left(\frac{IMAP}{120}\right) * \left(\frac{VE}{100}\right) * (ED) * \frac{MM}{R}$

Equation 5: $MPG = 710.7 * \frac{VSS}{MAF}$

IMAP is airflow based on the ideal gas law, in grams, RPM is vehicle engine speed, MAP is kPa of manifold absolute pressure, and IAT is the intake air temperature in degrees Kelvin (note that OBD parameters report in degrees Celsius). R is is 8.314 J/°K/mole, MM is the molecular mass of air (28.97g/mol average), ED is the engine displacement in liters, and VE is the volumetric efficiency of the engine (this must be calculated).

This method is not as straightforward and delivers less accurate, but likely still reasonable, results (quantifying the error is beyond the scope of this report). Again, accuracy is best during steady-state operation.

To configure the hardware to operate as a thin client, these parameters must be called-out in the configuration file. The board will then sample PIDs:

- '0D' (VSS) and '10' (MAF) for the MAF Method
- '0D' (VSS), '0C' (RPM), '0B' (MAP), and '0F' (IAT) for the MAF-less method

These parameters may be in addition to other parameters of interest, with the hardware logging up to 10 in total. Keeping to six or less samples will prevent skipped data frames, however, and reduce interpolation errors when integrating the results. In cases where it is uncertain what sensors the vehicle supports, all parameters may be sampled and invariant / unresponsive results stripped from the database.

Conversion factors for these sensors are taken from the J1979 specification and as follows:

0D – Byte A = VSS in km/h
10 – ((A*256)+B) / 100 = MAF in g/sec
0C – ((A*256)+B)/4 = RPM in rpm
0B – (A) = IMAP in kPa
0F – (A-40) = IAT in degrees C (not K!)

4.2.5 Fuel Use: Field Test Results

Field-testing the MAF method yielded rich data. The fuel economy approximations appear similar to the measurements from instantaneous fuel economy presented by vehicle trip computers tested, though the author was unable to gather enough data to generate a concrete accuracy measurement. These data are visualized in **Figure 22**.

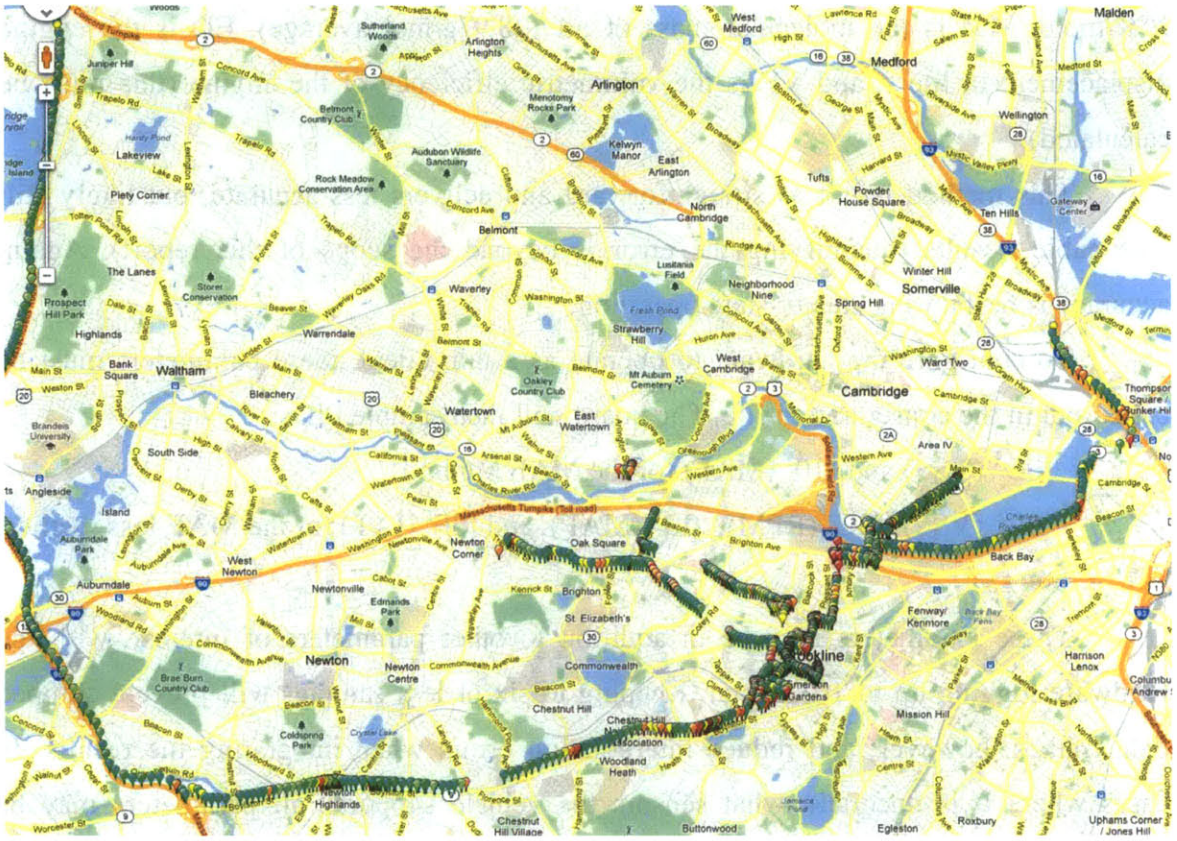


Figure 22: *Fuel economy map demonstrating CloudThink's ability to generate reasonable fuel consumption estimates. Blue/green is more economical than orange/red.*

An interesting note is that the fuel economy measurement appears to suffer the greatest inaccuracies for lack of temporal resolution – that is, the sampling rate may not be fast enough to capture the subtleties of an ever-changing value. Here again, it has not been determined what role the sampling frequency may have in the metric's accuracy, though general trends start to emerge such as improved fuel economy on highways, congestion around traffic lights, and more. With an increased sample rate for OBD (assuming no limitation from the vehicle's engine computer), these data are sure to improve.

Presently, all deployed hardware captures the MAF metric's required parameters to expand the pool of available data, and several drivers are recording fuel use for comparison sake. These data will ultimately be able to provide better informatics, like fuel used on a trip or cost of travel factoring in fuel use, cost of depreciation, and other externalities, or even to validate the success and shape policy decisions for new fuel economy regulations. Paired with a consumer-facing feedback application, these data may ultimately be used to drive changes shaping future consumer behavior. The real-time nature of the platform would allow corrective feedback to take place at a time when it would be constructive, rather than critical. If such a system could shift the Emphasis on Reducing Fuel Consumption (ERFC) away from its present value of 0%, the impact would be massive.

4.3 Beyond OBDII: Secondary Network Findings

After deploying the VMT and CAFE measurement applications, the author began to focus on exploring what data and actuation possibilities lie on networks in typical vehicles. Though detailed coverage is beyond the scope of this document, a brief primer of "sniffing" techniques is valuable at this point.

Most vehicle sensors broadcast at intervals or on event. To determine what is available on a vehicle network, a person may install a CAN interface on a vehicle diagnostic network (the Intrepid Control Systems neoVI fire supports may types of networks, and has an excellent software package for this process called VehicleSpy3).[12] Sitting in the vehicle, the user may watch the flow of data and watch these data for correlation to physical activities. For example, increasing engine speed may increase another parameter – perhaps indicating the presence of an RPM signal – or pressing a button may send a particular packet, indicating that the button transmits across the network being monitored. After reverse-engineering these non-OBD packets, like door locks and steering wheel controls, the user will have a map of sensors and commands to use in the development of applications. Some sensors and actuators may reside on other networks, and in this case it may be possible to utilize a manufacturer diagnostic tool to

[12] http://www.intrepidcs.com/

sniff "testing" commands that can actuate or sense other networks through the gateway node, but this again is outside the scope of this document. The author looked into the data available in several vehicle platforms supporting CANBus secondary networks, and began developing applications to demonstrate the breadth of utility for the Avacar and CloudThink platform.

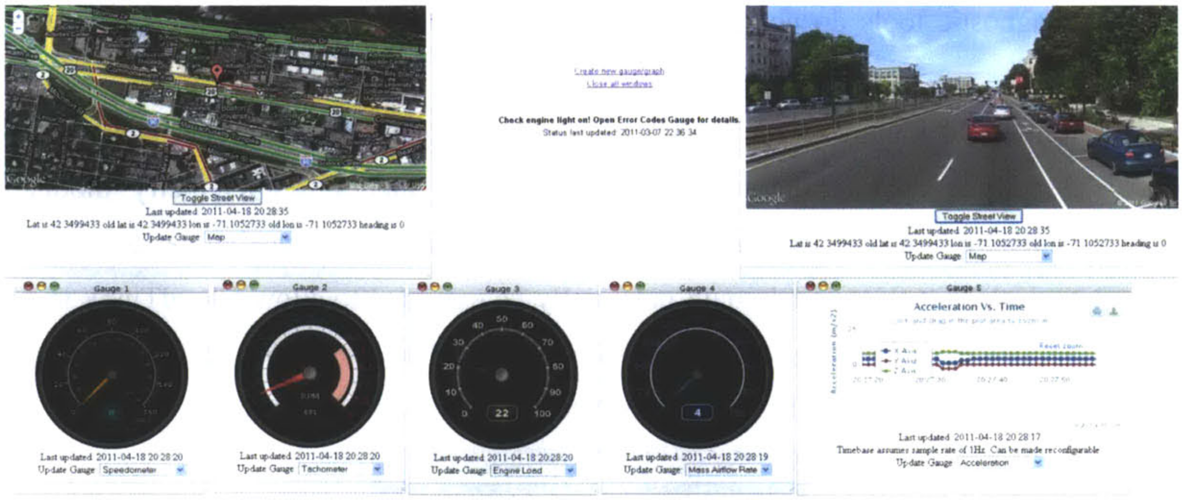
4.3.1 Examples of work beyond OBDII

These applications roughly fall into the categories of visualization, sensing / actuation, and inference, in increasing order of complexity.

4.3.1.1 Visualization

Visualization simply digests and displays processed data to a user, and is a type of application applicable to OBDII parameters as well as extended diagnostics. Examples of this type of application include:

- AudoIt: This software fuses GPS and diagnostic data to log miles traveled for purposes of collecting a miles-traveled tax or providing an accurate metric of miles traveled for corporate audits or personal records. CloudThink's security features ensure the validity and accuracy of these data, operating in a thin client (raw data transmitted) or thick client (aggregate metrics, no raw location data transmitted) model. This model is a location-aware extension of the VMT software.
- Teen tracker: This software provides a near real-time virtual dashboard and map plotting service.



Create new gauge/graph
Close all windows

Check engine light on! Open Error Codes Gauge for details.
Status last updated: 2011-03-07 22:36:34

Toggle Street View
Last updated: 2011-04-18 20:28:35
Lat is 42.3499433 old lat is 42.3499433 lon is -71.1052733 old lon is -71.1052733 heading is 0
Update Gauge: Map

Toggle Street View
Last updated: 2011-04-18 20:28:35
Lat is 42.3499433 old lat is 42.3499433 lon is -71.1052733 old lon is -71.1052733 heading is 0
Update Gauge: Map

Gauge 1
Last updated: 2011-04-18 20:28:20
Update Gauge: Speedometer

Gauge 2
Last updated: 2011-04-18 20:28:20
Update Gauge: Tachometer

Gauge 3
Last updated: 2011-04-18 20:28:20
Update Gauge: Engine Load

Gauge 4
Last updated: 2011-04-18 20:28:19
Update Gauge: Mass Airflow Rate

Gauge 5
Acceleration Vs. Time
Last updated: 2011-04-18 20:28:17
Timebase assumes sample rate of 1Hz. Can be made reconfigurable
Update Gauge: Acceleration

Figure 23: *Teen tracker software displays a reconfigurable virtual dashboard*

Find my car: This software locates the last location the vehicle was operated and renders this point on a map.

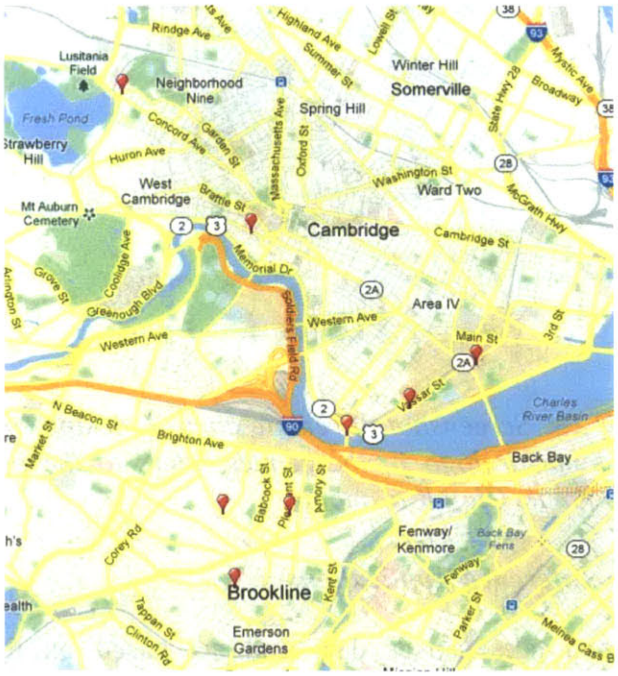


Figure 24: *Find my car shows the last-parked location of any CloudThink-enabled vehicle paired with a user account*

4.3.1.2 Sensing / actuation

Sensing / actuation reads data from the bus as a single point of information, or alternatively trigger actuation. Actuation is not available on most OBDII networks, so current diagnostic tools do not have this capability. Some dealer level tools have this ability.

- Remote lock/unlock: A cellular extension of a typical RF remote. This works by directly emulating the remote control module and/or the telemetry module, depending on the platform. There is limited latency in testing (< 3 seconds typical), but this model could potentially be extended to allow CloudThink to be applied to car-sharing services, like RelayRides. The availability of GPIO's on the Version 7 hardware allow the CANPuter additional control options.

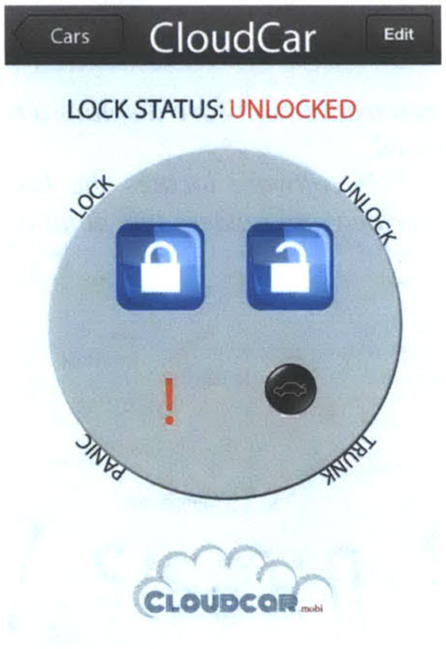


Figure 25: *Screenshot of remote control locking software. The CloudCar logo in the render refers to the project's former name, CloudCar.mobi*

4.3.1.3 Inference Applications

Inference applications take sensor data and act based on that very data. These applications include location-cognizant apps, which use external GPS sensor data to control vehicle functionality. No available diagnostic tool or in-vehicle software package allows for this level of reactive control.

- DealerTrip: For vibrational problems, the author has demonstrated software using the accelerometer to generate Fast Fourier Transforms to determine if the problem is related to wheel, engine, or transmission speed.

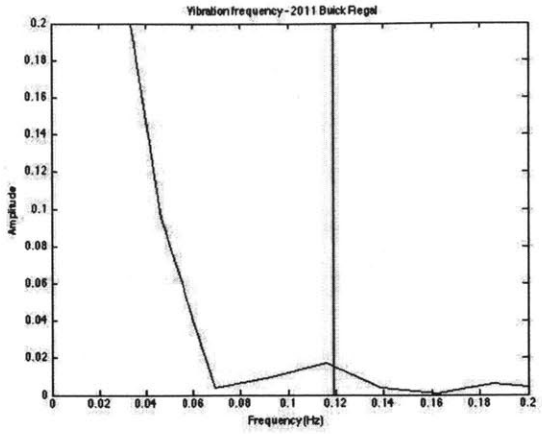
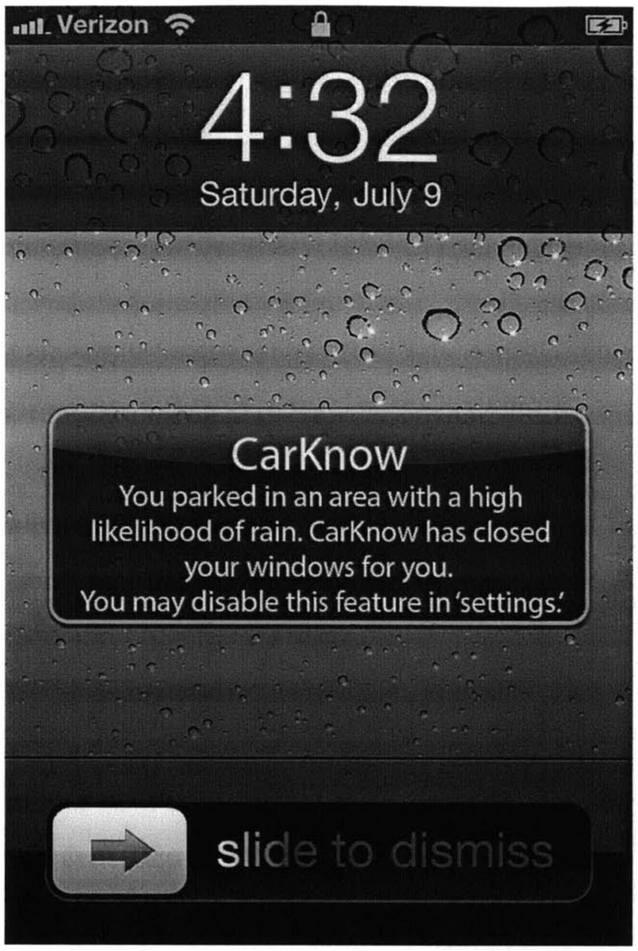


Figure 26: *DealerTrip demonstration application shows wheel speed (red line) vs. vibration frequency distribution (FFT) to indicate that a) a vibration problem exists and b) that problem is related to vehicle speed*

Location-aware apps are a new field of vehicle applications that use vehicle data, run a process in the cloud, and actuate based on location changes. Telematics devices and wirelessly connected diagnostic tools are becoming more and more popular in passenger vehicles. With them comes the added benefit of, in many cases, location data, timestamps, and Internet connectivity.

- Weather watcher: an example location-aware application which has been tested, successfully uses the GPS location to poll weather databases frequently, and if rain is detected, roll up any windows which may be closed.



You parked in an area with a high likelihood of rain. CarKnow has closed your windows for you.
You may disable this feature in 'settings.'

Figure 27: *WeatherWatcher uses GPS location to poll a weather database and close the windows if rain is imminent.*

A future implementation of the location-cognizant vehicle setting adjustor makes uses of these facts to dynamically change the settings of a vehicle or an in-vehicle software application. For example, an application might poll a connected GPS device to ask for altitude data and adjust the air/fuel map based on this information, using locally stored databases. Alternatively, an application might make use of the location data to keep presets "fresh" over a long drive, thereby eliminating the need for satellite radio or streaming music.

In the above example embodiment, the user connects with the diagnostic tool or telemetry device and installs an application like "NPR Anywhere" on the device. The user enters a series of parameters, along the lines of "Update every 10 minutes," "Preset 1 – NPR," "Preset 2 – Rock," "Preset 3 – Pop," and so on. The device then uses the location data and Internet connection to poll a database and keep presets 1, 2, and 3 always set to NPR, Rock, and Pop respectively. This is possible today with databases like FreqSeek and reverse engineering of steering-wheel radio controls, which often reside on a form of CANBus.

5. Conclusion

This thesis described the needs assessment and development of an intelligent, open, and scalable platform for digital vehicle duplication as an inroad toward broader monitoring systems. This platform included hardware and software to create a complete, end-to-end solution for gathering, utilizing, and acting upon collected data. Such a system was possible only when taking a holistic approach to understanding the current vehicle telematics and informatics paradigm, Internet of Things development trends, and the unaddressed demand for vehicle data across sectors.

While important to understand, and a provider of valuable data, vehicle informatics systems were conventionally difficult to design and deploy due to the limitations of OBDII and manufacturer unwillingness to share data. Valuable data exist on the in-vehicle network but are not required to be shared across a common network and are therefore underutilized. A similar lack of standardization plagued IoT development and rollout of data mirroring platforms, despite the clear benefits the Cloud provides in terms of crowdsourcing and real-timing (due in large part to agnosticism regarding data sources and rendering devices). Thus, the author deemed standardization, in particular open standardization, necessary to drive development of the platform and applications by increasing data captured and improving their utility. Further, the cross-manufacturer nature of this open standard would grow the pool of aggregable vehicles relative to single manufacturer, siloed systems in use today. The first section explored this background and presented the solution the author chose as a foundation for development.

In this document, the author described the successful development of the CANPuter hardware, which mirrored vehicle data from OBDII, secondary CANBus, GPS, and accelerometer data in the Cloud. A vehicle-to-cloud standard defined a means of storing and accessing vehicle data on an elastic computing platform along with standard security practices, a communications protocol, and canonical hardware for bridging On-Board Diagnostic data to CloudThink databases, answering the call of collecting and processing data and also creating a platform for others to build unique and compelling applications. The development chapter discussed the challenges faced and resultant solutions as well as novel approaches to problems faced when deploying this new technology.

The author discussed the decision process leading to the selection of an ARM7/FreeRTOS architecture (free and open compiler, excellent feature set and support), and the merits of MCM relative to MMC. In the former, data are mirrored to the Cloud directly, whereas the latter requires a user middleware device that may decrease reliability or increase latency. MCM is also discussed as being more robust when coupled with a data buffer to account for poor cellular coverage.

The development chapter highlighted interesting findings from the design and rollout process, including the proven reduction in latency for some applications using a "buffer and backfill" communication scheme where data are uploaded at intervals and in bulk upon shutdown. A similarly valuable finding pertained to the utility of a real-time operating system in scheduling the system to maximize data capture relative to a conventional, serial execution of tasks. This third section also touched upon unique findings such as the fact that obfuscating development reduces active support time by allowing developers to self-select at a higher threshold than they otherwise might (a simpler Arduino-based system had developers jumping in "too deep" very early, and required a full-time effort to supervise). In discussing the hardware, antenna design and power management arose as key concerns, and solutions to these problems were proposed as the basis of future work. The chapter closed by discussing novel power saving techniques, utilizing an accelerometer to wake the module and OBDII data to shut off, and call and response security to provide better-trusted data for applications where auditability is a concern.

The author successfully proved the design of the CANputer hardware and its interaction with the database server in creating Avacars. Though not the focus of this document, the API tested successfully and allowed the development of applications fully utilizing captured data, and demonstrated the merits of a platform over a set of applications (namely, simplified development, the ability to recreate existing as well as new applications, and improved data sharing).

Early-stage applications proved the utility of the platform by solving real-world problems with demonstrated needs, such as data collection for VMT taxation as described in the second section, or a cross-manufacturer approach to understanding congestion and fuel consumption patterns in a geographic context. Other applications described incorporated actuation, a novel topic relative to other diagnostic scan tools on the market. This actuation was the result of emulating dealership- or manufacturer-proprietary diagnostic tools and spoofing traffic so that the gateway module would retransmit data from the OBDII network to other CAN, LIN, or MOST networks and is a novel approach for accessing controls typically unavailable through the diagnostic port.

With these applications available, the author explored a novel category of application termed "inference applications," which included reactive applications capable of responding so rapidly a user might identify such an application as predictive, or further applications capable of making inferences based on the data trends facilitated by the CloudThink platform. Further development work spans across fields, ranging from integration with additional automation platforms to improved analytics, or improving user interaction by providing some form of feedback. Vehicle efficiency, economy, and comfort may be further improved by leveraging access to the data generated to improve and optimize control strategy for vehicles, both broadly and in specific use cases.

5.1 Future Work

CloudThink allows developers to create their own new and innovative ideas using the hundreds of sensors and actuators in vehicles today. These apps build toward next-generation smart applications, such as car to cloud to car reporting of weather conditions, Pay-As-You-Go (PAYG) insurance, fuel consumption minimization application, and feature unlocking of hidden / extra manufacturer features. To accomplish this would require partnership with auto manufacturers, or community involvement to reverse engineer and map a Wiki-style database of arbitration ideas.

5.1.1 Embedded Hardware Improvements

Plans include creating one revision of hardware to address power problems by adding battery level sensing hardware, and a possible return to Bluetooth multiplexing unless the author finds another way to make GSM/Cloud "Wake on LAN" feasible – as this currently limits realtiming efforts. Further, an integral PCB antenna would serve to lower BOM cost as well as improve signal quality while retaining a compact form-factor. Power supply filtering hardware could be improved at minimal cost and increase the connectivity duty cycle for devices in the field, and switch-mode power supplies optimized for vehicle supply voltages have demonstrated theoretical power savings of 50% relative to the tested embodiment. Integration of additional sensors, such as gyroscopes and temperature and pressure sensors, may yield richer data at minimal cost in terms of power and development complexity, when leveraging networks such as SPI or I2C to simplify multi-device connectivity. These and other changes were designed shortly prior to the submission of this thesis, but remain untested as of this writing.

From an embedded software perspective, it would be ideal to allow variable sampling rates and a more flexible scheduler to allow sampling at frequencies that more realistically mirror actual data change rates.

A near-term goal is to build apps that run on the embedded hardware directly, such as an interrupt-triggered "crash car" black box app is feasible. These apps may work well outside the scope of CANBus networks, utilizing the hardware's GPIO to control relays or similar output devices and improve the range of usable actuators.

5.1.2 Next-Generation Application Development

There exists a massive market potential for this platform, so the creation of an Automotive App Store selling diagnostic or other consumer-facing utilities is likely. The author has proven CloudThink integration with home automation solutions like the Belkin WeMo, so a migration from car-centric mirroring to digital mirrors for everything may not be far off (note: the WeMo does not have an API – tests were cobbled together through the use of a service called "If This Than That" and reading Tweets sent by a Python script).[13] This technology is easily extensible to other objects that have a pressing need for digital duplication, so CloudHome, CloudMe, and CloudCity are realistic next-generation goals.

One cloud-mirroring project of particular interest is the development of a cloud-based monitoring system for electric vehicle battery packs to provide data based on real life use cases, to study cell degradation, and to improve drive cycle simulations.

The lack of battery data is a real problem – today, electric vehicles have proprietary battery management systems and any data logged are stored locally to the

[13] http://www.belkin.com/us/wemo, https://ifttt.com/

vehicle and downloadable only with manufacturer-specific tools [50]. This means it is nearly impossible to monitor battery pack condition in real use cases, and the lack of a standardized method of accessing battery data through On-Board Diagnostics means that users are unable to view their own data at home. This causes problems, as vehicles are not often brought to dealerships for service – and major performance problems could go unnoticed, or at the very least, useful information about cell performance and pack design is lost. By creating a cloud-based platform to visualize battery data, it becomes possible to improve battery designs, to predict failures, and to update battery management system firmware before a problem occurs. This rich dataset may be used in conjunction with novel programming techniques to optimize battery pack configuration and control strategies similar to the dynamic programming techniques described by Dylan Erb's "Optimization of Blended Battery Packs" [50]. With the incorporation of three-dimensional location data and traffic data, it becomes possible to drive innovative control strategies based on vehicle context. The "omniscience" of knowing when high-impact events, such as large swings in charge or discharge current, help vehicle control strategies approach the dynamic solution generated in Erb's research.

The proposed research may drive the development of a standard interface to accessing vehicle battery data, providing useful data in a field where much advancement is yet to be made. There is much value in this field, and the data are directly applicable to solving a number of problems faced by industry.

6. Works Cited

[1] J. E. Siegel, "Design, development, and validation of a remotely reconfigurable vehicle telemetry system for consumer and government applications," Massachusetts Institute of Technology, Cambridge, 2011.

[2] Energy Information Administration, "25th Anniversary of the 1973 Oil Embargo," 03 September 1998. [Online]. Available: http://www.eia.gov/emeu/25opec/anniversary.html. [Accessed 20 March 2013].

[3] D. Barber, "CAFE Standards," 2002. [Online]. Available: http://www.davidbarber.org/research/cafe.html. [Accessed 5 March 2013].

[4] Conjecture Corporation, "What is OBD-II?," 2013. [Online]. Available: http://www.wisegeek.com/what-is-obd-ii.htm. [Accessed 5 March 2013].

[5] B. Wojdyla, "How it Works: The Computer Inside Your Car," 21 February 2012. [Online]. Available: http://www.popularmechanics.com/cars/how-to/repair/how-it-works-the-computer-inside-your-car. [Accessed 2013 3 March].

[6] D. a. E. G. Sosnowski, "Performing Onboard Diagnostic System Checks as Part of a Vehicle Inspection and Maintenance Program," United States Environmental Protection Agency, Ann Arbor, 2001.

[7] Society of Automotive Engineers - Vehicle E E System Diagnostic Standards Committee , *E/E Diagnostic Test Modes,* Society of Automotive Engineers, 2012.

[8] The Equipment and Tool Institute, "The Equipment and Tool Institute," March 2013. [Online]. Available: http://www.etools.org/. [Accessed 28 March 2013].

[9] R. Leale, "Defcon 19 Workshop," 4 June 2011. [Online]. Available: http://www.canbushack.com/defcon19/workshop.pptx. [Accessed 15 January 2013].

[10] United States Environmental Protection Agency, "Control of Air Pollution From New Motor Vehicles and New Motor Vehicle Engines," 20 December 2005. [Online]. Available: http://www.epa.gov/fedrgstr/EPA-AIR/2005/December/Day-20/a23669.htm. [Accessed 15 March 2013].

[11] Logic PD, "Reverse engineering the MINI Cooper automotive CAN bus message format," February 2011. [Online]. Available: http://bobodyne.com/web-docs/robots/MINI/CAN/MINI_CAN.pdf. [Accessed 15 November 2012].

[12] ACDelco, *CAN Communication,* vol. 16, M. DeSander, Ed., Troy, MI, 2009.

[13] D. Hobbs, "Communication is Key: Serial Bus Diagnosis," *MOTOR,* pp. 44-53, September 2011.

[14] S. Garrett, "A Closer Look at Vehicle Data Communications," *GEARS,* pp. 22-27, April 2012.

[15] BOSCH, *CAN Specification 2.0,* Stuttgart: Robert Bosch GmbH, 1991.

[16] S. Corrigan, "Introduction to the Controller Area Network (CAN)," July 2008. [Online]. Available: www.ti.com/lit/an/sloa101a/sloa101a.pdf. [Accessed 15 November 2012].

[17] D. Sharmila, "Bluetooth Man-In-The-Middle attack based on Secure Simple Pairing using Out Of Band Association Model," in *Conference on Control, Automation, Communication and Energy*

Conservation, Perundurai, 2009.

[18] Right to Repair Coalition, "About the Right to Repair Act," 2013. [Online]. Available: http://www.righttorepair.org/about/default.aspx. [Accessed 2013 February 2013].

[19] Hughes Telematics, "Drive Connected," 2012. [Online]. Available: http://www.hughestelematics.com/press/releases/verizon.php. [Accessed 20 October 2012].

[20] Vehicle Service Pros, "Vehicle Service Pros," 29 January 2013. [Online]. Available: http://www.vehicleservicepros.com/press_release/10860055/report-factory-installed-telematics-to-reach-157-percent-in-new-cars-in-2013. [Accessed 4 February 2013].

[21] United States Department of Energy, "One Million Electric Vehicles: February 2011 Status Report," US DOE, 2011.

[22] P. a. T. G. Mell, "The NIST Definition of Cloud Computing," 2011.

[23] W. Voorsluys and J. a. R. B. Broberg, Cloud Computing: Principles and Paradigms, Wiley, 2011.

[24] M. L. M. a. R. R. Chui, "The Internet of Things," March 2010. [Online]. Available: http://www.mckinseyquarterly.com/The_Internet_of_Things_2538 . [Accessed 18 March 2013].

[25] U.S. Department of Transportation: Federal Highway Administration, "Highway History / When did the Federal Government begin collecting the gas tax?," 07 April 2011. [Online]. Available: http://www.fhwa.dot.gov/infrastructure/gastax.cfm. [Accessed 3 November 2012].

[26] Congressional Budget Office, "Alternative Approaches to Funding Highways," The Congress of the United States, Washington, D.C., 2011.

[27] M. Handley, "Eco-Friendly Vehicles Draining State Road Repair Budgets," *US News,* 7 June 2012.

[28] B. S. Z. L. &. N. K. McMullen, "Distributional impacts of changing from a gasoline tax to a vehicle-mile tax for light vehicles: a case study for Oregon.," *Transport Policy,* no. 17, pp. 359-366, 2010.

[29] P. Sorensen, "System Trials to Demonstrate Mileage-Based Road Use Charges," ational Cooperative Highway Research Program, Santa Monica, 2010.

[30] D. D. F. O. R. Z. (. Z. L. W. M. J. A. Z. L. Coyle, "From Fuel Taxes to Mileage-Based User Fees: Rationale, Technology, and Transitional Issues," Center for Transportation Studies, Minneapolis, 2011.

[31] A. Costa, "Vehicle Miles Traveled (VMT) Fee Financing Alternatives: Lessons Learned and Future Opportunities," 2012.

[32] R. a. G. G. Baker, "Exploratory Study: Vehicle Mileage Fees in Texas," National Technical Information Service, College Station, 2010.

[33] C. L. Dudek, Ed., *Papers on Advanced Surface Transportation Systems,* College Station, Texas, 2003.

[34] U.S. Energy Information Administration, "How much gasoline does the United States consume?," 19 4 2012. [Online]. Available: http://205.254.135.7/tools/faqs/faq.cfm?id=23&t=10.

[35] C. R. Knittel, "Automobiles on Steroids: Product Attribute Trade-Offs and Technological Progress in the Automobile Sector," American Economic Review 2012, 2011.

[36] P. Diaz, "An Examination of the CAFE Standards and Mandatory Environmental Regulations,"

Texas State University, 2011.

[37] J. DeCicco and F. Fung, "Global Warming on the Road: The Climate Impact of America's Automobiles," Environmental Defense, 2006.

[38] L. Cheah and J. Heywood, "Meeting U.S. passenger vehicle fuel economy standards in 2016 and beyond," *Energy Policy,* vol. 39, no. 1, pp. 454-466, January 2011.

[39] D. Edmunds, "FAQ: New Corporate Average Fuel Economy Standards," 10 11 2011. [Online]. Available: http://www.edmunds.com/fuel-economy/faq-new-corporate-average-fuel-economy-standards.html.

[40] P. Bastani, J. B. Heywood and C. Hope, "U.S. CAFE Standards: Potential for Meeting Light-duty Vehicle Fuel Economy Targets, 2016-2025," MIT, 2012.

[41] D. Hafemeister, "The CAFE Formula," The American Physical Society, San Luis Obispo, CA, 2007.

[42] L. W. Cheah, A. P. Bandivadekar, K. M. Bodek, E. P. Kasseris and J. B. Heywood, "The Trade-off between Automobile Acceleration Performance,," Sloan Automotive Laboratory, Massachusetts Institute of Technology, 2008.

[43] D. Sperling, E. Abeles, D. Bunch, A. Burke, B. Chen and K. a. T. T. Kurani, "The Price of Regulation," *Access,* no. 25, 2004.

[44] A. Langton, Policy Matters: Rolling Over and Playing Dead, Berkeley: Goldman School of Public Policy, University of California, Berkeley, 2006.

[45] United States Government Accountability Office, "Vehicle Fuel Economy: Reforming Fuel Economy Standards Could Help Reduce Oil Consumption by Cars and Light Trucks, and Other Options Could Complement These Standards," United States Government Accountability Office, 2007.

[46] J. Stolp, "Designing the Internet of Things," 5 December 2012. [Online]. Available: http://build.smartthings.com/blog/designing-the-internet-of-things/. [Accessed 27 March 2013].

[47] "IP - Ingress Protection Ratings," [Online]. Available: http://www.engineeringtoolbox.com/ip-ingress-protection-d_452.html.

[48] Delorme, *GPS Module Antenna and RF Design Guidelines,* Yarmouth, Maine, 2007.

[49] B. D. Lightner, "AVR-Based Fuel Consumption Gauge," *Circuit Cellar,* no. 183, pp. 59-67, October 2005.

[50] D. C. Erb, "Optimization of Blended Battery Packs," Massachusetts Institute of Technology, Cambridge, 2013.