

**Electric Vehicle Charge Optimizer**

**Submitted to |** John Kennedy

SDSU ECE Dept. Instructor

**Submitted by |** Jin Choi and Alexander Matthew Reyes

Chief Editors of Reports

**Sponsors |** SDG&E, Alan Dulgeroff, Benjamin Lauer, Daniel Smith

Sherwin Yari, and Omar Zevallos

**Table of Contents**

1. **Abstract** 1
2. **Introduction** 2
3. ***Goals and Requirements*** 3
4. **Body** 4
5. ***Software*** 4

Server Machine 4

Data Storage 5

Main Server 5

Communications 7

Algorithm 8

Web Server 10

1. ***Hardware*** 12

Wall Module 13

*PIC18LF23K22 Microcontroller* 14

*LCD Display* 14

*WIZnet 610WI Wi-Fi Module* 14

*Override Switch* 15

Car Module 16

*PIC18LF23K22 Microcontroller* 16

*Lithium-Polymer Battery* 16

*Coulomb Counter* 17

*Battery Charger* 18

1. **Conclusion and Recommendations** 19
2. **References** 20
3. **Appendix A** 21
4. **Appendix B** 23
5. **Appendix C** 27
6. **Appendix D** 28

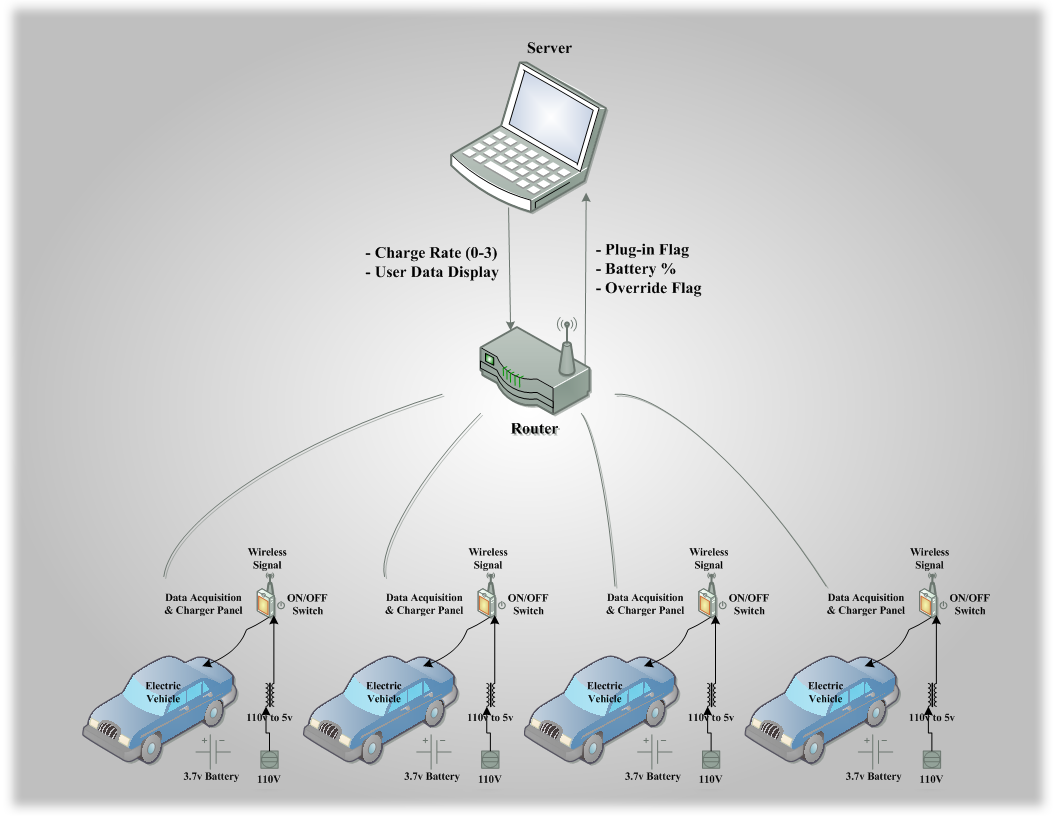
**I. Abstract**

Team eSmart has constructed a system that intelligently manages the charging of electric vehicles (EV). The purpose of this system is to distribute EV loads to off-peak hours where more power is available in order to efficiently allocate supply. We’ve replicated a transformer grid by connecting up to 4 EV load substitutes to simulate real-time monitoring and charging. Each EV communicates wirelessly with our server which calculates and assigns charge rates accordingly based on customer preferences, transformer, and circuit and system capacity constraints. In addition, a web server provides car owners access to charge profiles to meet personal vehicular needs.

**II. Introduction**

The growth of Electric Vehicles (EVs) contributes to the ever-increasing demand for power. When consumers plug in to recharge their EVs, they add significant load to the distribution transformers. Before the eSmart system, the massive power draw caused by charging EVs upset the effectiveness of neighborhood distribution systems due to the fact that charging an EV has a comparable power draw to adding a new house to the transformer. In addition, adding multiple EVs per distribution system increases the likelihood of overloading the transformer causing power outages for consumers. Thus team eSmart has reproduced actual data at a smaller scale in order to construct an optimized charging system. Our system enables multiple EVs to charge simultaneously while satisfying customer needs and abiding by system constraints. It considers system parameters as well as customer preferences to intelligently manage EV charging. By managing the charging schedules, customer satisfaction is ensured, system constraints are met, and the overall system peak load is shaved.

The eSmart system consists of two parts: a physical battery charging device (referred to as “car nodes”) which represents EVs being plugged in and charging and an autonomous smart server that manages the available power supply. When the vehicle is plugged into the station, the server will monitor and manage the charging of the vehicle to respond to the available resource constraints and customer preferences. The chargers will communicate with the server via 802.11 radios while customers can access their personal profiles from a web server through the WLAN. Modeling the current power distribution system in the most realistic way possible and optimizing it using gathered data and techniques will assist our clients, SDG&E, and their clients with a better, more effective power distribution system as EV ownership continues to increase.



*Figure 1 - System Illustration*

***A. Goals and Requirements***

A successful electric vehicle charge optimizer system was realized once the following specifications were met:

1. Software – the Electric Vehicle charge optimizer software should comply with the following specifications:
   * A “controller” implemented using OSI-PI Software to monitor EVs and develop control algorithms for the load profiles of the cars
   * Controller algorithms should schedule EV charge times based on diverse customer preferences, transformer, circuit and system capacity constraints
   * Controller Graphical User Interface to display system conditions
   * Wireless communication between controller and EV charging node via web server
2. Hardware – The hardware components should include:
   * Central server which houses OSI-PI
   * EV charging node module
     1. Monitor EV charge level, instantaneous and total power consumption
     2. Wirelessly communicate data and respond to commands from central computer
     3. User interface embedded system with LCD display and input capability

**III. Body**

***A. Software***

The purpose of the software team is to create a software package capable of managing the power consumption of multiple EVs charging simultaneously. To do so, we’ve constructed multiple program classes each with a specific function to the whole system. The system is coded and run on the Visual Studio 2010 IDE using the C# language; and any new or additional software, such as OSIsoft’s Process Information (PI) technology, have been downloaded and installed as per their user manuals.

**Server Machine**

The server is run from a Lenovo 1143AFU laptop chosen for its raw processing power, high memory volume, and built-in wireless access capability. Also, in order to maximize efficiency, the computer underwent a clean reboot to remove any extraneous programs and then was installed with the minimum required software. A laptop was chosen for this demonstration (as opposed to a desktop) because the laptop’s size combined with wireless capability provided system mobility allowing demonstrations to take place anywhere with an access point.



*Figure 4 - Lenovo 1143AFU laptop*

**Data Storage**

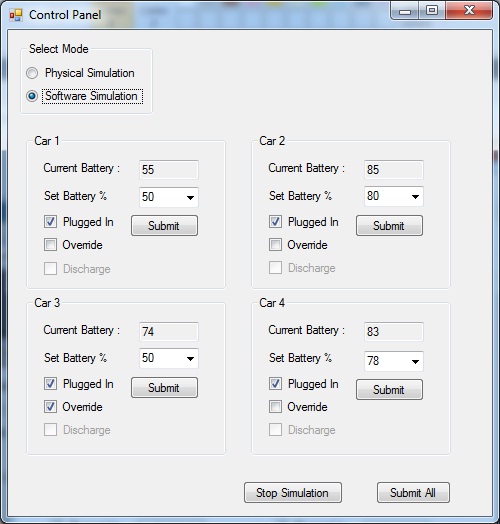
Almost all system data is stored in the web server’s SQL database tables. This is to ensure that the data is not disorganized across multiple sources. In order for any program to read or write to the secured database tables it must first connect to SQL Server, the interface that provides multiple user access. The web server and main server can connect simultaneously through SQL Server and from there manipulate the table data through Language INtegrated Queries (LINQ), allowing the server to search through and retrieve the necessary data values. It should be noted that since the server will be running continuously, the database connection is left open to reduce delay caused by reopening and closing the connection.

While the bulk of the system’s data is stored and accessed through the web server’s SQL database, a copy of the car node’s battery state, plugged-in status, override status, and currently-used charge profile name, along with the transformer grid data, is stored into PI Server for the display GUI. The PI Server itself is a software package provided by OSIsoft that has been optimized to handle and process large amounts of data points. Its database identifies each data type as a unique variable known as a PI Point, which stores data as time-relative instead of object-relative objects, giving it the ability to archive data. This allows the server to retrieve any previous data value to construct trends.

**Main Server**

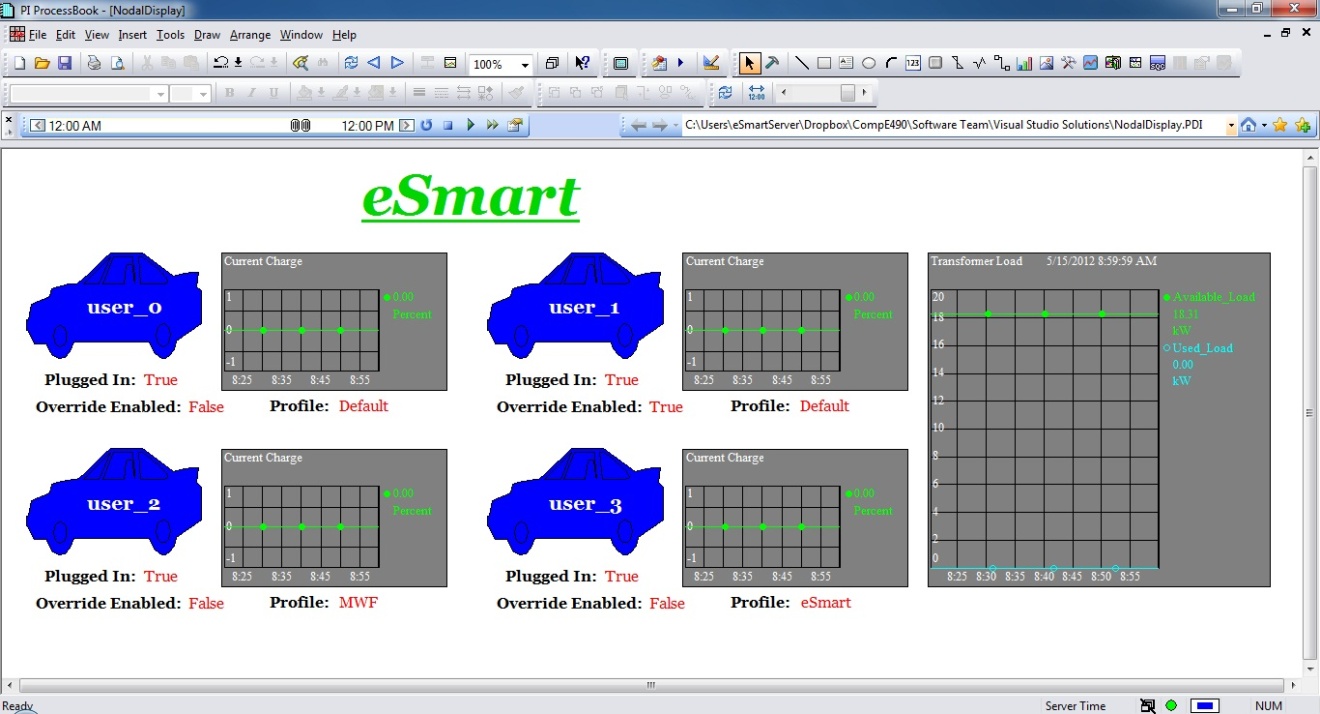
The main server program creates a threaded instance of each hardcoded car node and then routinely calls the communications and algorithm classes during runtime to stimulate communication. It also contains two GUIs: one for simulation and another for display. The purpose of the simulation GUI is for testing and debugging while the display GUI provides real-time updates to the laptop’s operator.

The simulation GUI runs on a Windows Forms application and has two modes: hardware and software. In hardware simulation mode, control is handed to the system and the server machine communicates with the physical car nodes to send/receive updates in real time. In software mode, the server creates its own car objects to simulate a loaded grid and users would be able to manipulate hardware characteristics through the interface; a safer alternative to debug the system without modifying the hardware. The characteristics include setting an initial battery charge state and toggling plugged-in status and override status flags. In addition, both modes display the car battery’s current charge state as well as allow the user to manually discharge the physical (or simulated) battery.



*Figure 5 - Simulation GUI via Windows Forms*

The display GUI provides no user controls to manipulate the hardware and only provides real-time updates to changes made in the system. It displays each car node’s current profile setting, plugged-in status, override status and its current battery state as an expandable[[1]](#footnote-1) line plot. In addition, the GUI provides a graph of the transformer’s available and current load for comparison purposes. The display is built and run using PI Processbook, a development tool used to display data from the PI Server database.



*Figure 6 - Display GUI via PI Processbook*

**Communications**

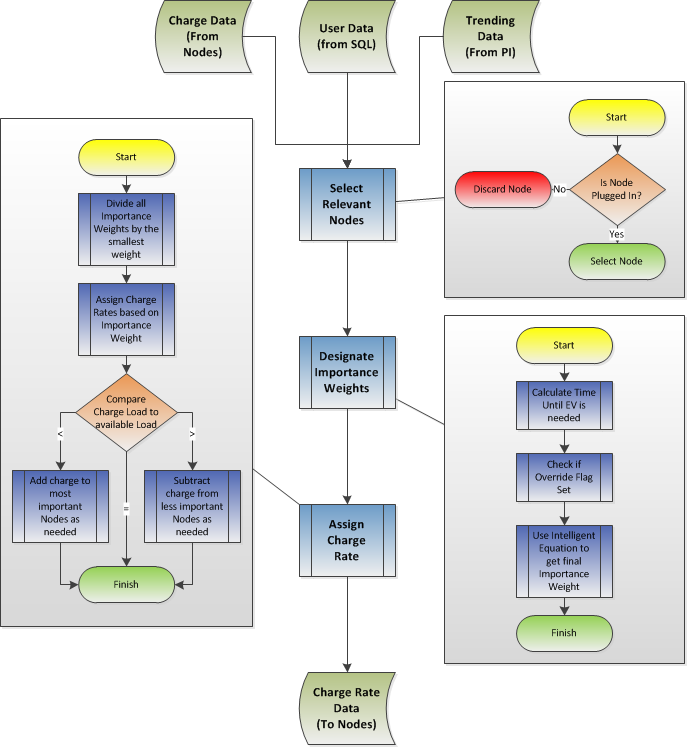
The eSmart system is divided into two wireless networks: the wireless local area network (WLAN) that is connected to a public access point which provides wireless access, and the secure sub network through which the server communicates with the physical car nodes. Both networks communicate using IEEE 802.11b/g and User Datagram Protocol (UDP) and both run simultaneously. Since the subnet runs on the same protocols as the WLAN, it was necessary to ensure web access did not interfere with nodal communication, therefore static IP addresses and port numbers were assigned to each physical car node and also hardcoded into the main server. The communications program initially opens then maintains a bidirectional[[2]](#footnote-2) channel to each of the car nodes without worrying about changing IPs upon restart. Another issue was the server maintaining 1:4 simultaneous connections over the subnet. The most feasible solution implemented was multithreading where each nodal connection would have its own application thread, independent of each other but still capable of running simultaneously.

The communications program consists of a constructor and two primary functions: Receive and Send. The Receive function initially opens the communication channel between the server and car node at the specified port then enters into a loop which constantly checks for data transmission; if data is found, the program reads and parses it into local variables. The code is also embedded in a ‘try-catch’ statement to check if the packet received is in the valid string format (see [Appendix B](#AppendixB)). After the local variables are updated, the SQL and PI databases are updated utilizing customized insert, update, and delete statements. The Send function must be manually called by the main server program and provided with the appropriate data to be sent as well as the destination. Once called, it will merge the separate data types into a string object then a byte array and then transmit byte-by-byte to each destination node.

**Algorithm**

The algorithm reads: the system load, transformer capacity, time of day, each EV’s current state of charge, the customer’s selected charge profile and their usage patterns to calculate the best distribution of power to meet customer needs. These input parameters are obtained through the algorithm’s object factory class which reads the SQL Server database and populates an expendable[[3]](#footnote-3) algorithm object for each node. The transformer data, utilized for available vs. used load comparison, was obtained directly from SDG&E. It contains 5 days’ worth of an average transformer’s load capacity in one hour increments.

An important parameter created and defined in the algorithm object is ‘importance rating’ which calculates a priority value for each specified car node object. This value considers all the previous parameters to determine the car node’s priority level for receiving charge. The algorithm itself assigns each car object an initial charge rate then runs through a loop comparing each car’s importance rating and adjusting their charge rates accordingly. In addition, the charge rates are also checked against the transformer’s available capacity to ensure the transformer is not being overloaded. Once a final charge rate has been determined for each car object the algorithm program updates the databases and then returns the list of the car objects, complete with charge rates, at which the main server calls the communication class to send to each physical car node its appropriate charge rate.

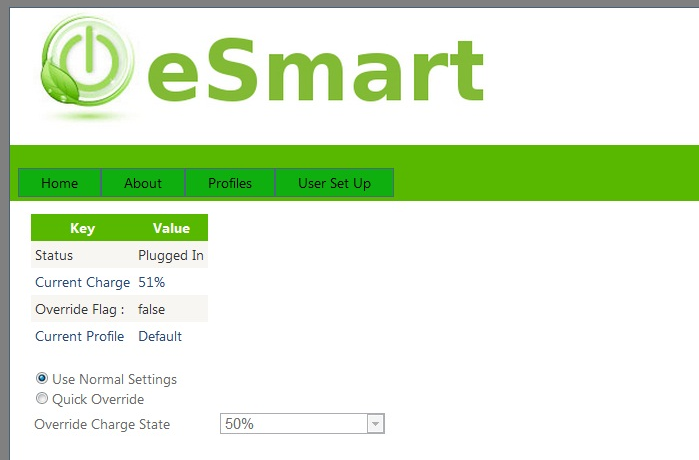


*Figure 7 - Algorithm Flowchart*

**Web Server**

The eSmart website is the primary front-end interface for customers to monitor their vehicle’s charging status as well as change their desired charge settings. The web server is hosted on Microsoft’s Internet Information Services (IIS) with wireless access allowing users to access the site from any web-accessible device through eSmart’s WLAN.

The website contains a number of intuitive pages to ensure easy management of charge profiles. Upon initial access, users must register their vehicle with eSmart to be viable for smart charging. Once logged in, he or she is directed to a personalized home page which displays a quick overview of the vehicle’s current statistics. These stats include the car’s current charge state, plugged-in status, override status, and current profile. Through the website, users have the ability to toggle their charging state by selecting either eSmart’s default profile, the override state, or a personal profile.



*Figure 8 - Personalized Home Page*

If the default profile is enabled, the user’s vehicle will charge at a rate designated by the algorithm and in the process will accumulate marks for good behavior or “goodie points.” These points will proportionally influence the user’s importance rating providing greater priority status for future selections. Should the override profile be selected, the user will have immediate priority until the specified charge level is reached but will obtain no points. Concerning personal profiles, users have the ability to create a new or edit an existing profile. Each profile requires a name and can set up to two types of desired charge states and times. The first (and required) desired charge constraint proffers the system to charge the vehicle to a designated charge level by a designated time. The optional quick charge controls are for urgent or additional requests in cases where the user requires a specified charge on certain occasions. To set profile dates, a calendar feature allows the user to set a specific profile for any day of the year.

|  |  |
| --- | --- |
| *Figure 9 - Creating a Profile* | *Figure 10 - Calendar* |

***B. Hardware***

The purpose of the hardware team is to accurately simulate the influence of the eSmart system on a scaled down model of a neighborhood transformer grid containing multiple EVs. Each EV car node contains a wall module to simulate the home-mounted charging units and a car module which represents an EV with lithium-polymer rechargeable battery. Both modules are physically connected via their microcontrollers’ UART ports and contain the same 3.3V regulator to supply their components. Power supplies are stabilized by connecting a 1µF capacitor to the input power source (either the battery or 5V wall wart) and every device has a 0.1µF capacitor to its regulated supply.

The hardware system is set to be as intuitive as possible. The user plugs in his or her electric vehicle to the wall module which sends a charge request to the server. The server processes that request and returns an initial charge rate to begin charging the vehicle. After the initial transaction, the car node then continuously reads the car’s battery state and updates its status to the server who returns a calculated charge rate until the user’s desired charge state is obtained or the car is fully charged.

**Wall Module**

The purpose of the wall module is to provide a hardware interface to the user to manage the EV’s charge cycle. It contains a Microchip PIC18LF23K22 microcontroller, LCD display panel, WIZnet Wi-Fi radio, and an override pushbutton switch.



*Figure 11 - Wall Module PCB w/ WIZnet module and LCD display*

***PIC18LF23K22 Microcontroller***

We’ve selected Microchip’s PIC18LF23K22 microcontroller because it has the required 5.5V operation specification. Its purpose is to communicate with the different components within (as well as between) the wall and car modules therefore the PIC’s most necessary capability is Universal Synchronous Asynchronous Receiver Transmitter (USART) communication.

In order to enable USART communication, we set the TX1 (pin 17) and RX1 (pin 18) pins and configured the following registers: Transmit Status and Control (TXSTA1), Receive Status and Control (RCSTA1) and Baud Rate Control (BAUDCON1). Since we were working in asynchronous operation, we also enabled the following bits for successful transmission and receiving: TXEN, CREN, SYNC and SPEN.

***LCD Display***

The LCD Display is the 2-line by 8-character Newhaven Display model and its purpose is to provide the user with feedback of the current charging cycle. Changes to the current battery state, plugged-in status, override status and active profile are quickly updated via custom write command functions. It is necessary to initialize the LCD upon startup to enable functionality and to add a delay statement between writes to ensure enough time is allocated for the operation.

***WIZnet610WI Wi-Fi Module***

The Wi-Fi module is the WIZnet 610WI model and is responsible for transmitting and receiving all data to and from the server through the eSmart sub network. As aforementioned, it is set up to communicate using IEEE’s 802.11b/g standard with UDP and is also configured to run in client mode meaning it acts as an end-user able to send and receive data. In receive mode, the module converts wireless data into a serial output which is sent to the wall module’s microcontroller via the SOUT pin; the serial interface is referenced between VDD and VSS. In transmit mode, the module wirelessly transmits byte-by-byte the serial data through the SIN pin. For demonstration purposes, each WIZnet module has been hardcoded with a static IP address and port number through a wizard to avoid confusion with dynamic protocols. When operational, the module’s settings can be changed through its configuration site, accessed by entering the static IP address into a web browser.

***Override Switch***

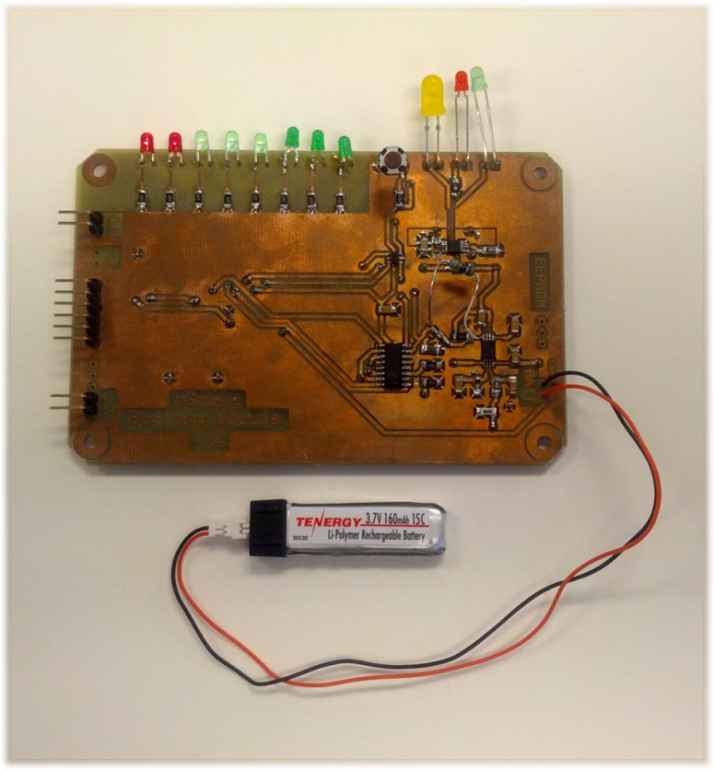
The override switch is a pushbutton hardwired into the wall module, whose purpose is to toggle the hardware system’s override state. The wall module only toggles a flag and sends that to the server, it takes no independent action for it is the server’s responsibility to adjust the car node’s charge rate accordingly. It should also be noted that the override states in the web server and in each car node are linked and toggling one will affect the other; this is to ensure there is no conflict between the software and hardware.

****

*Figure 12 - Override Switch on Wall Module*

**Car Module**

The car module is the EV substitute for our demonstration as purchasing real vehicles were out of the question. It contains its own PIC18LF23K22 microcontroller (the same type housed in the wall module), a lithium-polymer battery with switch, a coulomb counter and a variable battery charger.



*Figure 13 - Car Module PCB w/ LiPo battery*

***PIC18LF23K22 Microcontroller***

The car module’s microcontroller is a Microchip PIC18LF23K22 microcontroller. Since both the wall and car modules need to communicate via UART ports, the best solution was to use the same chip to ensure compatibility. It’s also capable of HDQ communication, which was needed to get the state of charge from the coulomb counter.

***Lithium-Polymer Battery***

To simulate an EV battery we utilized a Tenergy Lithium-Polymer or “LiPo” Rechargeable Battery. The LiPo battery is charged utilizing the battery charger but discharge is accomplished through a chassis mount resistor with a low impedance and high power capacity. Discharge is initiated by a “soft” switch, meaning that it can be stopped when the system recognizes, through the coulomb counter, that the battery is at low capacity. Since the LiPo battery would not function with a trickle charger, we added a Microchip MCP73833chip as a battery manager. In this setup, it acts as a constant-current to constant-voltage charging system whose rate depends on an added external resistance that is set with a switch. For demonstration purposes, only the top 10% of the battery’s total capacity was utilized and the discharge rate was lowered through a pulse-width modulated output with a duty cycle of ~50%.

***Coulomb Counter***

The coulomb counter chip is the Texas Instruments BQ26501 and it measures the battery’s remaining power by sensing current flowing into or out of the battery through a 0.02Ω sense resistor, measuring that current and then accumulating it. The battery’s power (P) is given by:

Max. Capacity- Pout, where P=Vavg (= 3.7V) x I.

When the battery is full, the charger will give a “done” signal by changing one of its output pins from high impedance to low. The microcontroller reads these events and updates the coulomb counter manually

Communicating with the coulomb counter required using the HDQ interface[[4]](#footnote-4) and so a UART port was converted for HDQ by using a transistor to pull the line low, whereas a resistor would usually keep it high. The main difficulty in this was to precisely time a conversion from a UART byte into each of the correct HDQ bits because each bit had two start bits and awkward timing requirements. To solve this: an inverter was placed to buffer the receiver preventing it from influencing the line, and a “reset” command was hardcoded to reset communication after every successful pair of command + data bytes was dealt with.

***Battery Charger***

The charger is a variable rate Li-Po charge manager with the rate of charge controlled by an external resistor. The purpose of the variable rates is to allow the charger to simulate a system where users may be given lower charge rates instead of being left with nothing. It would have been possible to control the rate of charge with a digital potentiometer, but it was much more straightforward to use a digital switch to select between three discrete resistors. This selection is possible because the charger detects the resistance using a very low current through that resistor.

**IV. Conclusion and Recommendations**

Embedded development of the hardware system was not realized in time therefore “plan B” was executed. In “plan B” no real-time communication with the battery occurs and all data utilized was obtained from earlier testing. In addition, three development flaws were discovered: speed, power, and reliability. Regarding speed, bidirectional communication was too slow for reliable packet transmission from the coulomb counter of each node up to the server. If a byte from one UART was erroneous, an overflow error occurred which took additional time to process slowing down the module’s processing system. For power, the numerous LED indicators drew too much current for the slow charge rate to be completed in the expected time frame. And as for reliability, while each individual process could work expectedly, harmonious operation was too unreliable because of the different times it took each component to process a byte of information.

More time should have been allocated for debugging. Bidirectional communication can be stabilized by multiple means: using a higher baud rate between the microcontrollers and WIZnet module, setting UART communication to be turn-based, or lowering the rate of UART transmission. LED power disruption can be fixed by reducing the number of LEDs to display the battery’s charge state. And ensuring reliability requires debugging the embedded code to allow for a larger window for processing. On an additional note, the battery charging rates were too low for complete charges therefore it was necessary to adjust simulation limits to speed up demonstrations by making it appear a small charge of the battery was a full charge in the simulation. And while unimplemented in our project, PI Server’s data archiving feature provides the option to enhance the algorithm’s effectiveness by incorporating trending of the users’ charging history.

**V. References**

“Digi-Key.” Digi-Key Corp. Web. March 2011. <<http://www.digikey.com>>

“Microsoft.” Microsoft Corporation. Web. March 2011. <<http://www.microsoft.com>>

“Mouser Electronics.” Mouser Electronics. Web. March 2011. <<http://www.mouser.com>>

“OSIsoft.” OSIsoft. Web. March 2011. <<http://www.osisoft.com>>

“PIC18F23K22.” MicroChip Technology Inc. Web. March 2011 <<http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en547750>>

“Senior Design Lab.” Senior Design Lab. Web. March 2011. <<http://www.seniordesignlab.com>>

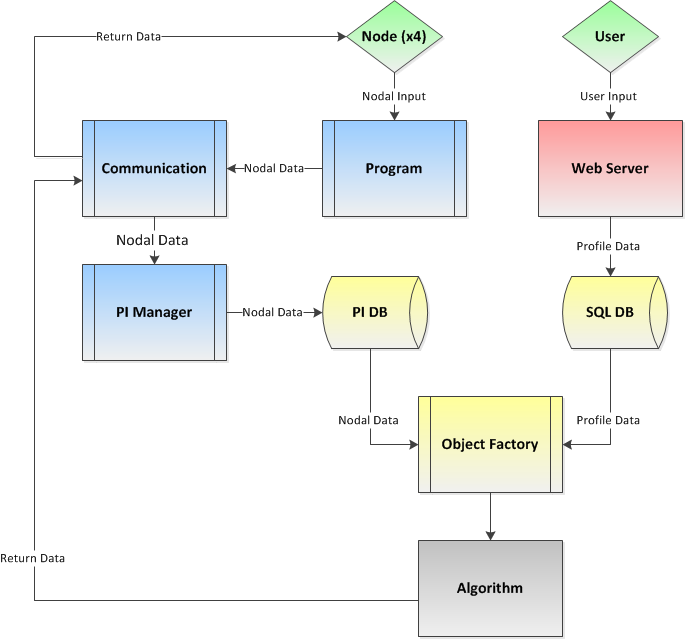
“Sparkfun.” Sparkfun Electronics. Web. March 2011. <<http://www.sparkfun.com>>

“Texas Instruments.” Texas Instruments. Web. March 2011 <<http://www.ti.com>>

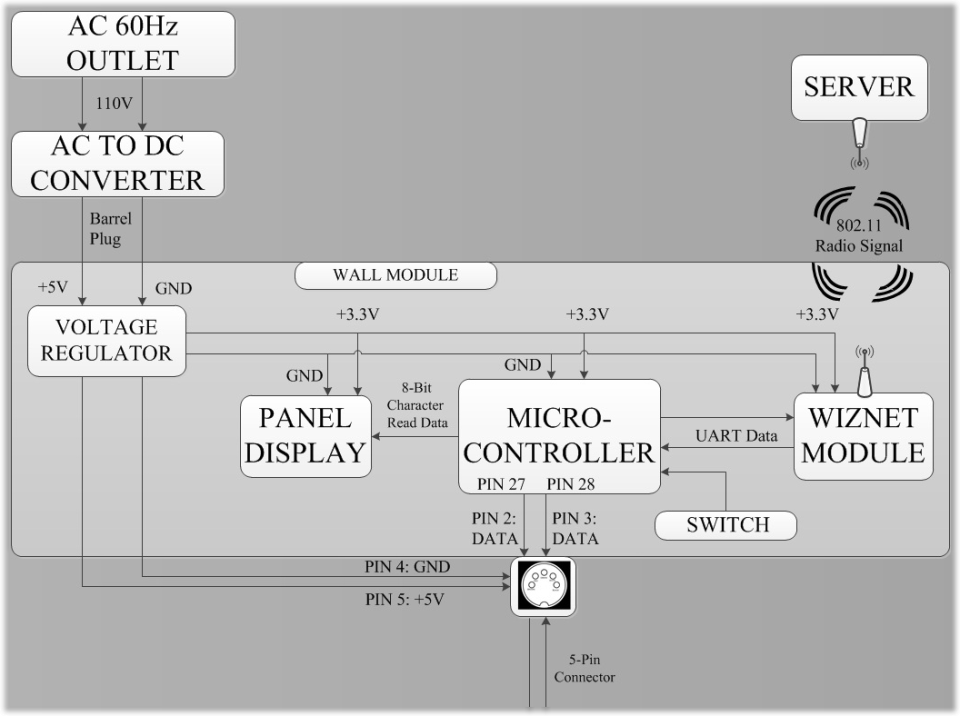
“WIZ610wi.” WIZnet. Web. March 2011. <<http://www.wiznet.co.kr/Sub_Modules/en/product/Product_Detail.asp?cate1=&cate2=&cate3=&pid=1020>>

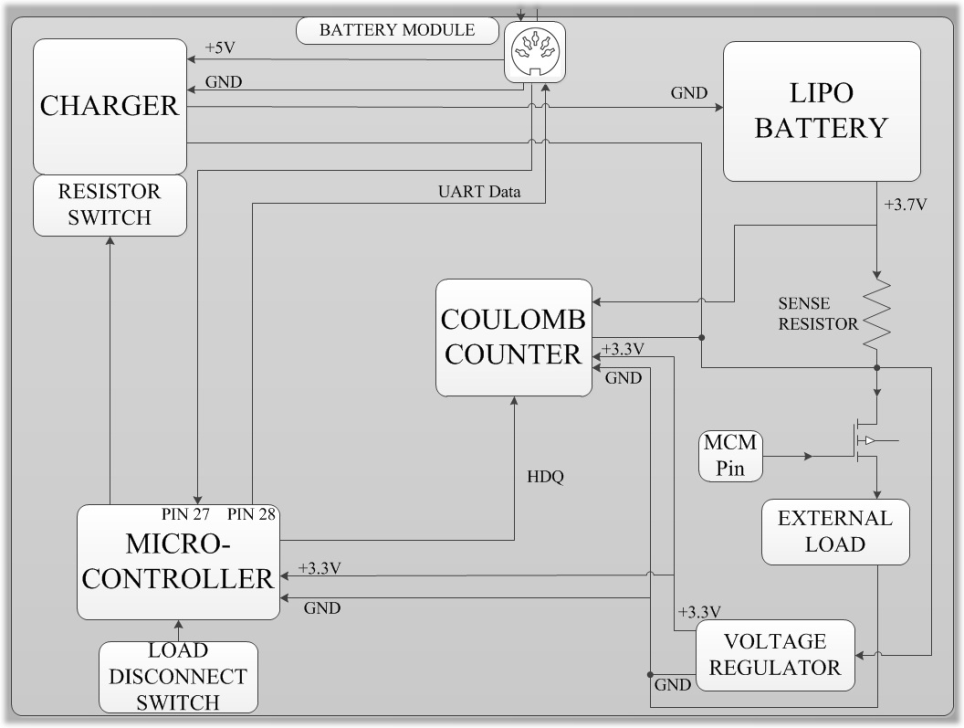
**VI. Appendix A**

***System Block Diagrams***



*Figure 2 - Server System Block Diagram*





*Figure 3 - Hardware System Block Diagram*

**VII. Appendix B**

***Communication Specifications***

Wall Module -> Server

**Description:** Server will expect a wireless packet from the Wall Module only when the charge data has changed. The packet contains information about the current state of charge, whether or not the car is plugged in to the wall module, and whether or not the user has pressed the override button for this charge cycle (the override flag will be cleared each time the user’s car is unplugged).

**Packet Construction:** 7 ASCII Character bytes total – (3) State of Charge, (1) Plugged In, (1) Override, (2) End Code

**Packet Construction Break Down:**

*State of charge* – 3 bytes (ASCII 000-100 representing percentage of charge) sent least significant bit first (e.g. ‘100’ would be sent as ‘001’). If the node is not plugged in, send ‘999’ as state of charge.

*Plugged-In Flag* – either ASCII ‘0’ (ASCII Code 48) or ‘1’ (ASCII Code 49) representing whether the car is plugged in (‘1’) or not (‘0’)

*Override Flag* – either ASCII ‘0’ (ASCII Code 48) or ‘1’ (ASCII Code 49) representing whether the user has pressed the override button on the Wall Module (‘1’) or not (‘0’)

*End Code* – ASCII ‘CR’ (Carriage Return) (ASCII Code 13) followed by ASCII ‘LF’ (Line Feed) (ASCII Code 10)

EXAMPLE: Transmission representing 67% charge, plugged in, override button not pressed will be represented by a the following string: “07610<CR><LF>”, which is represented by the following decimal ASCII Codes: 48,55,54,49,48,13,10

Server -> Wall Module

**Description:** The Wall Module will expect a wireless packet after its own transmission containing the rate at which to charge the car, the updated override flag, as well as the profile name string to display on the Wall Module’s LCD. The charge rate byte will be an ASCII Character from 0 to 3, representing the desired charge rate. Approximate charging times (for the whole battery) are (for codes 0-3): No charge, ~2 hours, ~1 hour 20 minutes, and ~30 minutes, respectively.

**Packet Construction:** 13 ASCII Character bytes total – (1) Charge Rate, (1) Override Flag, (9) Display String, (2) End Code

**Packet Construction Break Down:**

*Charge Rate* – 1 byte (ASCII 0-3) representing the rate of charge. A ‘0’ value will represent no power sent to the car, a ‘1’ will be slow charge rate, and a ‘3’ will represent the maximum charge rate.

*Override Flag* – either ASCII ‘0’ (ASCII Code 48) or ‘1’ (ASCII Code 49) representing whether the user has pressed the override button on the Wall Module (‘1’) or not (‘0’).

*Display String –* 8 bytes of printable ASCII characters (ASCII Codes 32 – 126), followed by <CR> (ASCII Code 13). This will be a string to display on the Wall Module’s LCD Screen to indicate current profile utilized.

*End Code* – ASCII ‘CR’ (Carriage Return) (ASCII Code 13) followed by ASCII ‘LF’ (Line Feed) (ASCII Code 10)

Wall Module -> Battery Module

**Description:** The Battery Module expects information about the charge rate to use that has been accepted from the server.

**Standard:** UART Transmission using 3.3V as “high” and Ground as “low” with Pin 27 on the Wall Module acting as TX and Pin 28 on the Battery Module acting as RX. The transmitter is configured to idle high. Data received is not inverted. The baud rate is 38.4kbps.

**Packet Construction:** 3 ASCII Character bytes total – (1) Charge Rate, (2) End Code

**Packet Construction Break Down:**

*Charge Rate* – 1 byte (ASCII 0-3) representing the rate of charge. A ‘0’ value will represent no power sent to the car, a ‘1’ will be slow charge rate, and a ‘3’ will represent the maximum charge rate.

*End Code* – ASCII ‘CR’ (Carriage Return) (ASCII Code 13) followed by ASCII ‘LF’ (Line Feed) (ASCII Code 10)

Battery Module -> Wall Module

**Description:** The Wall Module expects the current relative state of charge from the Battery Module

**Standard:** See above

**Packet Construction:** 5 ASCII Character bytes total – (3) State of Charge, (2) End Code

**Packet Construction Break Down:**

*State of charge* – 3 bytes (ASCII 000-100 representing percentage of charge) sent least significant bit first (e.g. ‘100’ would be sent as ‘001’)

*End Code* – ASCII ‘CR’ (Carriage Return) (ASCII Code 13) followed by ASCII ‘LF’ (Line Feed) (ASCII Code 10)

Microcontroller <-> WIZnet Module

**Description:** The WIZnet module expects UART Transmission at 3.3V for “high” and Ground for “low” with Pin 17 as TX and Pin 18 as RX from microcontroller to WIZnet. The transmitter is configured to idle high. Data received is not inverted. The baud rate is 38.4kbps. The WIZnet module is designed to act as a data bridge between the Wall Module and the Server through 802.11 Wi-Fi.

Microcontroller <-> Coulomb Counter

**Description:** The Coulomb Counter expects a command from the HDQ input and possibly data from the microcontroller. If a read command is given, then the coulomb counter will send back data over the same HDQ line. The Microcontroller is also configured to transmit and receive on the same line.

**Standard:** The HDQ Transmission at 3.3V for “high” and Ground for “low” with Pin 17 as TX and Pin 18 as RX from the microcontroller to the coulomb counter. HDQ communication works over a single wire which is idle high through a pull up resistor of 8kΩ to 3.3V. TX is inverted and connected to the gate of a transistor switch. When TX is low, the HDQ line is pulled to GND through the drain of the switch. When TX is high, the switch is open drain. HDQ must transmit and receive at separate time intervals so that TX does not overwrite the HDQ data to be received. The microcontroller accepts HDQ data through an inverter connected to RX so GND is “high” and 3.3V is “low”, and then inverts the data again internally.

The baud rate for the microcontroller is set to roughly 50kbps so that each byte of data over its UART is sent in 220µs which coincides with the HDQ timing. When the UART sends 8 bytes of data, the HDQ receiver receives one byte of data. A command will have a 7 bit address from least significant bit first, followed by 1 bit to indicate whether the command is to read or write. If it is a write command, then it must be followed immediately by 8 UART bytes of data. If it is a read command, then the HDQ line will send back a 1.5ms byte that the Microcontroller will receive as 8 bytes.

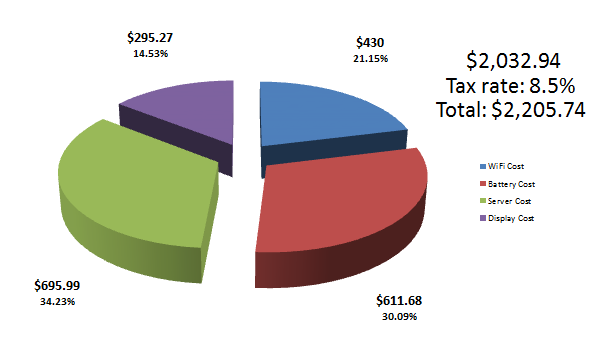
**VII. Appendix C**

***Gantt Chart***

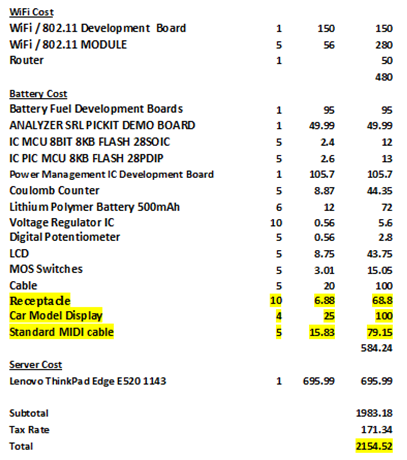


**VIII. Appendix D**

***Budget Analysis***



*Figure 14 - General Costs Pie Chart*



*Figure 15 - Individual Component Costs*

1. Plots the battery’s charge level with respect to time. Allows user to change the time range or enlarge the graph. [↑](#footnote-ref-1)
2. Data can be sent and received from each side of the channel but not simultaneously. While one side sends the other can only receive and wait for the channel to be free again before sending its own data. [↑](#footnote-ref-2)
3. Disappears once the algorithm program is terminated, thereby freeing up program memory [↑](#footnote-ref-3)
4. Single wire, asynchronous, open-drain communication protocol [↑](#footnote-ref-4)