Smart Battery Level Monitor with Remote Access

for IoT devices

A Java based web application

_____

Project Report

Presented to

The Faculty of the Department of Electrical Technology

University of Houston

_____

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

in Engineering Technology – Network Communications

_____

By

Pavan Kumar Reddi

Fall 2016

ACKNOWLEDGEMENT

I wish to express my gratitude to my advisor, Dr. Ricardo Lent for providing me with the opportunity to work under him during the project tenure. His guidance has helped me get a profound understanding of the research question. My sincere thanks to him for providing me with the right resources to fire up my project.

I would also like to thank my loved ones for their constant encouragement and for helping me connect the dots.

ABSTRACT

The Internet of things (IoT) technology is upon us. The ubiquitous nature undeniably makes it a dominant technology. However, this new technology poses many challenges to technology enthusiasts in developing and maintaining the related Do-it-yourself projects. The ability to monitor battery charge level of an IoT device and extend its battery life are among the top requirements currently. When a cluster of such IoT devices are deployed, tracking the battery data and keeping track of every node becomes an even more cumbersome task.

A java based web application was developed using a web framework application and a JavaScript library to display the battery level of a cluster of IoT devices using an interactive frontend design. The backend development of the project includes data procurement from the IoT device, data analysis and manipulation to find the battery State of Charge and other essential parameters.

The project was designed with a research perspective, making a few tweaks to the calculation algorithm by using the existing Smart Battery solutions can make the project even more accurate and industry ready. Future scope and improvements were also explored and discussed.

Table of Contents

Chapter -1 Introduction

The Internet of Things (IoT) is the future. The ubiquitous connectivity has created a new shift in paradigm, and the closed and bounded systems of the past which perform the same tasks as IoT devices will become obsolete. With the connection of low-cost sensors to cloud platforms, it's now possible to track, analyze, and respond to operational data at a large scale. IoT devices are intelligent systems that can communicate with each other and with people in real time.

According to IDC technologies, "the worldwide IoT market spend will grow from $591.7 billion in 2014 to $1.3 trillion in 2019 at a compound annual growth rate (CAGR) of 17%. The installed base of IoT endpoints will grow from 9.7 billion in 2014 to more than 25.6 billion in 2019, hitting 30 billion in 2020." [1]

A white paper from Thinkworx, *Foundational Elements of an IoT Solution: The Edge, The Cloud, and Application Development* by Joe Biron and Jonathan Follett [2] gives a fundamental idea on how to deploy IoT devices and manage them.

Availability of abundant technologies that enable IoT and the advent of IPv6 addressing will definitely allow the deployment of IoT devices on a much larger scale in the future. This report will deal with the planned development of a web application titled "Smart Battery Monitor with Remote Access for IoT devices".

1.1 Motivation

Retrieving electrical data from IoT devices and monitoring them remotely would aide in managing the devices and extend the battery life. An IoT device can be powered by connecting it to a power supply, using battery as a buffer (for power outages) or a rechargeable battery alone.

Selecting the power source for IoT is based on the requirement and availability of a power supply. There is a fair amount of requirement in the present scenario to deploy IoT devices for temporary data acquisition (ex. room temperature, bots, servers) and at remote places that requires timely visits and battery as a power source.

Unlike devices like Laptops or Mobile, IoT devices like Arduino and Raspberry Pi does not have an embedded battery management system. Since, the devices are often not carried around like mobile devices, there is a requirement to monitor battery information remotely.

Since the devices are deployed at remote places, monitoring the data physically would not be a practical solution when deploying a cluster of IoT nodes. The wireless connectivity to the cloud/server permits battery management providing one of the most transparent battery management systems possible since the battery status can be shown on a webpage.

For instance, four Raspberry Pi devices were deployed across the campus for testing for the project. There is major requirement to access the battery electrical data remotely in order to manage the devices and make sure they do not shutdown while performing the everyday tasks.

1.2 Research Objective

Develop a java based web application to enable user to monitor battery charge level of an IoT device. Enable the web application to automatically fetch the electrical data which is hosted on a URL by the Raspberry Pi and store the information in a database.

Study the battery information obtained through data logging from Raspberry Pi or use data stored in the database to study the charge and discharge cycles of the battery connected to Raspberry Pi. Analyze and interpret the data.

Write an algorithm for the web application to calculate the battery charge percentage individually for every device over a charge cycle or a discharge cycle using the existing battery information.

Frontend web development to not only display the results but develop an interactive webpage using a simple-to-use predesigned JavaScript library for the user to view the device information.

Improve the scalability of the whole design so that new nodes can be added easily without changing/modifying the previous computation code.

1.3 Existing Solutions

PiCheckVoltage is a project for the Raspberry Pi which has a voltage regulation unit, a voltage measurement unit using a resistor divider, LEDs to display battery health and a kill switch. A program is run on the Raspberry Pi; only tells if the battery level is low, good or zero. [3]
Most other solutions were done by adding a circuit and display the data on LED or using USB buffer to display current battery voltage and current reading and estimate the battery level.

Chapter – 2 Background and Literature Review

2.1 Raspberry Pi

Raspberry Pi is a fully-functional single-board computer with a Broadcom processor to which a user can plug it into computer peripherals, to have a full graphical user interface. Several physical devices and sensors can be attached to a Raspberry Pi's programmable digital I/O pins to perform operations. A network adapter can also be added on the board to provide connectivity making it an ideal low cost IoT device.

For this project the four Raspberry Pi's that were taken for study have been already deployed in a private network. Each of them performing structurally repetitive tasks and connected to the internet through a Wi-Fi dongle. *Internet of Things with Raspberry Pi* [4] – 1and 2 by RoboCoder clearly explains how Raspberry Pi can be setup. There are several other resources on the Internet on how to perform this. A white paper from Adafruit Industries by Simon Monk explains how to setup Wi-fi and configure the network interfaces on Raspberry Pi 3. [5] *Ultimate Raspberry Pi Configuration Guide* by *scottkildall* gives the complete configuration from scratch on how to setup Raspberry Pi for Internet of Things. [6] Raspberry *Pi with Java: Programming the Internet of Things* by Stephen Chin with James L Weaver explains how Java could be used for

programming if required to use Java. [7] The configurations can however be tweaked as per requirement.

2.2 Using Battery as a Power Source

An IoT device can be powered by connecting it to a power supply, using battery as a buffer (for power outages) or a rechargeable battery alone. Since, the project is aimed at monitoring battery level, basic knowledge of batteries is a must.

Unless specified most of the information on batteries below was retrieved from Battery University™, a free educational website that offers hands-on battery information. The tutorials evaluate the advantages and limitations of battery chemistries, advise on best battery choice and suggest ways to calculate and extend battery life. [8]

2.2.1. IoT devices and the problem of short battery life

It is easy to see that IoT devices perform multiple actions which play a role in discharging the battery. IoT mostly perform structurally repetitive tasks consuming the same load over time for the same tasks. Not having proper powering technology can cause unsatisfactory results and hinder IoT performance. The simplest way to increase the battery life is to use a battery with higher capacity than usage but it is definitely not ideal as it increases the overall cost of the project.

2.2.2. Battery considerations for IoT

Lithium-ion battery having a high energy density is one of the features of that makes it desirable for space-constrained usage over other batteries. Mostly used for applications that require long

life and low maintenance. Li-ion batteries capacity however depreciates with aging and temperature conditions, reducing the amount of charge the battery can hold over time. An alternate battery is Ni-Cd is also durable and able to work better under rough conditions, including low temperatures, currently used to power drones. NiMH is an alternative to Ni-Cd with 30 to 40 percent higher capacity, currently used in cameras. Different types of batteries will be employed based on whether they are the best fit for the application. [9]

2.2.3. Lithium-ion batteries

For the project, rechargeable Li-ion cells 3.7V was considered as it is the most commonly used battery for simple IoT deployments. A DC/DC boost converter module was used to convert battery output to 5.2V DC for running 5V projects. The study is limited to learning about the charge and discharge cycles and some basic characteristics of the Li-ion battery. The chemical compositions and chemistry involved in working of the battery are irrelevant for the scope of the project.

2.2.3.1 Basics about Discharge Cycle

Li-ion batteries discharge to 3.0V/cell with around 95 percent of the energy spent, and the voltage would drop if the discharge cycle continues. To protect the battery from over-discharging, monitoring the discharge level is essential. When done right the voltage of a healthy battery gradually recovers and rises towards the nominal voltage.

Heat increases the battery performance but shortens life by a factor of two for every 10°C increase above 25–30°C (18°F above 77–86°F). Charge and discharge rates of a battery are

determined by C-rates. The capacity of a battery is commonly rated at 1C, meaning that a fully charged battery rated at 1Ah should provide 1A for one hour. The C-Rate varies along with load. The higher C-rate loading will have lower output capacity.

2.2.3.2 How to and When to Charge?

Li-ion is maintenance-free and the battery lasts longer when operating between 30 and 80 percent of State of Charge. This is the most effective working bandwidth range to get a longer life. Some other precautions include keeping the battery at a moderate temperature. The worst combination being high voltage and elevated temperature. Deep cycling should be avoided.

2.2.3.3 Basics about Charge Cycle

Protection circuits are advised to be built into the battery pack to not allow the set voltage to be exceeded. Full charge occurs when the battery reaches the voltage threshold and the current drops to 3 percent of the rated current or cannot go any lower. For a Li-ion, partial charge is always better than a full charge and disrupting the charge cycle also does not cause any loss. Overcharging heats up the battery reducing the overall efficiency. Hence, should be avoided. Lithium-ion self-discharges at 5% in 24h, then 1–2% per month (plus 3% for safety circuit) under ideal conditions.

2.2.4. Smart Battery

A smart battery means that some level of communication occurs between the battery, the device and the user usually to measure the state of charge of the battery. Most smart batteries record battery history, including cycle count, usage pattern, maintenance requirements, etc.

2.2.4.1. Calibration

A smart battery needs to be calibrated by applying a full discharge and charge every 3 months or after every 40 partial cycles since the accuracy falls by more than 10% approximately making the digital readout unreliable. However, smart batteries with impedance tracking provide a fair amount of self-calibration.

2.2.4.2. Data Processing

The IoT devices can be connected to the server/cloud, hence deploying a web application would be ideal method to find the state of charge. Wireless connectivity to the cloud permits collective battery management of a cluster of IoT devices providing one of the most transparent battery management systems possible since the battery status can be shown on a webpage.

2.2.5. Battery Fuel Gauge

The battery voltage decreases linearly as the battery is discharged. Knowing the relationship between battery voltage and SOC allows using voltage translation to estimate the battery voltage. A major limitation with this technique is that the battery voltage is also affected by battery current and temperature.

Coulomb-Counter Fuel Gauges manipulate electrical data to measure the energy in & out of the battery pack. There are algorithms to compensate for the effects of discharge rate, discharge temperature, self-discharge and charging efficiency etc. A white paper, "State of Charge estimate with Li-Ion batteries" by Davide Andrea clearly explains methods to estimate SoC. [10]

For the project Dynamic Voltage translation based SoC estimate is taken to consideration compromising the accuracy as the research project is aimed at creating a web application.

2.3 Adafruit USB Power Gauge

Adafruit USB Power Gauge [11] is an open source programmable USB buffer device used between the power source (Battery Pack) and Raspberry Pi power input to know the electrical data parameters. Using an FTDI cable the readings from the I/O pins can be given directly to the Raspberry Pi itself. The voltage, current and wattage data is obtained as readable text on the TX pin at 9600 baud with an variance of at least 0.1V and 50mA due to noise, thermal changes. The tutorial on how to setup the connections is available on the website. [12] There are several other methods to obtain the electrical data.

2.4 Development Tools

2.4.1. Play

Play Framework [13] is an open-source Java based web framework which follows an MVC (Model View Controller) architecture pattern which separating three types of objects depending on their roles: model objects, view objects, and controller objects.

Model objects represent the data and define the logic to manipulate that data.
View objects play role to display data (model). Controller objects act as the intermediary and are responsible for linking the models to their views and synchronize the intermediary processes. Target package sends the data to the server. This web application framework was very easy to learn and implement.

2.4.2. Arbor.js

Arbor.js [14] library which is a graph visualization library using web workers and the popular jQuery. The libraries were installed in the pubic folder of the web application. Arbor.js is used as the Views object in the web application to display the data with an interactive user interface. Arbor.js is well documented and easy to implement. The basic design and implementation can be quickly learned from thinkist thoughts' blog on "Simple graphs with Arbor.js". [15]

2.4.3. PostgreSQL

PostgreSQL [16] is a powerful, open source object-relational database system with a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

2.4.4. Eclipse IDE

Eclipse IDE [17] is a perfect tool for Java and Web development. Play, PostgreSQL and Arbor.js could be smoothly implemented before deploying the web application in the server. IDE should be selected as per requirement and ease of use.

2.4.5. MobaXterm

MobaXterm [18] is a powerful remote computing tool. Since the web application was deployed in a Linux server, MobaXterm was used to SSH into the server and to run UNIX commands on a windows desktop. SFTP browser will automatically show the remote files for simple navigation, which is very ideal for the managing the web application.

2.5. Non Functional Requirements - Performance Analysis and Tools

2.5.1. MATLAB

MATLAB was extensively used for this project to estimate the SoC. The plot tool was used to plot scatter plots between the voltage and SoC. The Curve Fitting toolbox [19] add-on was used to find the goodness of fit statistics of the curves.

2.5.2. Microsoft Excel

Microsoft Excel was used for data handling and manipulation before using MATLAB which include sorting the RAW ASCII type Electrical data to Rows and columns. The initial analysis was also performed on Excel before using MATLAB.

Chapter 3 – Implementation and Execution

3.1 Methodology

3.1.1. Network Design

The four Raspberry Pi devices (Node 0 through Node 3), considered for this project were already deployed and connected to a private network. The testbed is ideal under the assumption that, since the devices do structurally repetitive tasks, they would have constant load and power requirement over time. However, since multiple devices were taken into study, larger sample size improves the confidence levels and added gravity to the research findings.

3.1.2. Procurement of Electrical Data from The Raspberry Pi

The data was procured from the Raspberry Pi through non-intervention method during the charge and a discharge cycles for every node by connecting it to a Adafruit USB power gauge mini kit. For the project, the FTDI cable was connected back to the Raspberry Pi. Along with the Battery

Voltage, we also acquire Voltage_In and Voltage_Out values from this setup. The electrical data obtained was hosted in URLs. There are however several ways to perform this task and a plenty of resources are available on the internet on how to do this task which include both data logging and pushing the data in a web server using automation scripts in the Raspberry Pi.

Sample data from the URL is "0.004888 3.807429 5.000000 0.019550 0.635386" and can be interpreted as "Voltage_In Battery_Voltage Voltage_Out Current_In Current_Out"

	Charge Mode	**Discharge Mode**
Voltage_In	4v – 5v	0v – 1 v
Voltage_Out	0v – 1 v	4v – 5v
Current_In	Varies	Varies
Current_Out	0 mA	Varies

3.1.3. Initial Data Analysis of Node 0

During initial data analysis, the study was focused not at answering the original research question but to find the quality of the data using descriptive statistics (mean, standard deviation, median). Analyzed the electrical data using Graphical Techniques (scatter plots) and Correlations & Associations for homogeneity (internal consistency), which gave an indication of the reliability.

Since the battery SoC level determination is only based on battery voltage with respective time, and not current and power parameters are not being used there is requirement to maintain

accuracy during the data analysis. The Voltage vs. Time curves were studied using scatter plot and with a precision up to 5 decimal values. The total time of charge and discharge cycles (from Fig1.1 and 1.2) were estimated based on user's requirement and estimation of power requirement. The voltage cut offs can also be estimated using it. The analysis was performed using MATLAB, after importing the data into it.

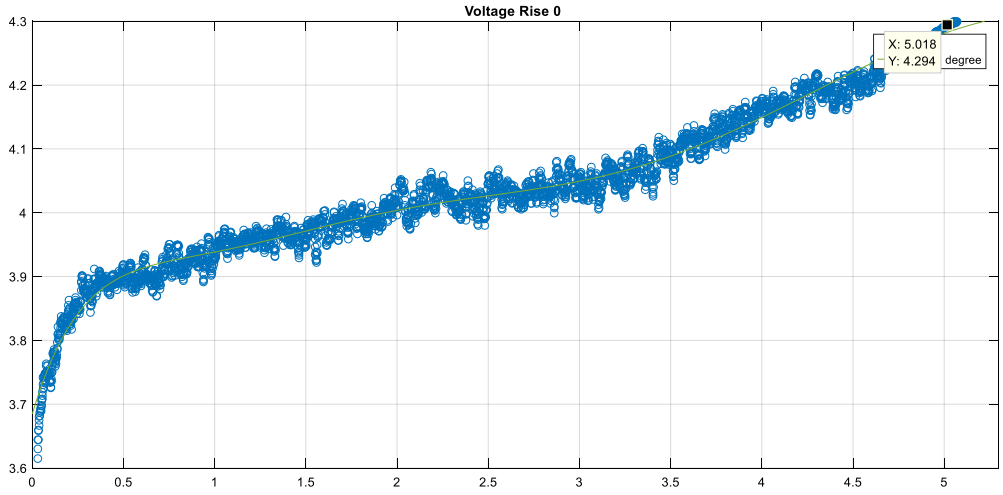


Fig 1.1 Charge Cycle of Node 0

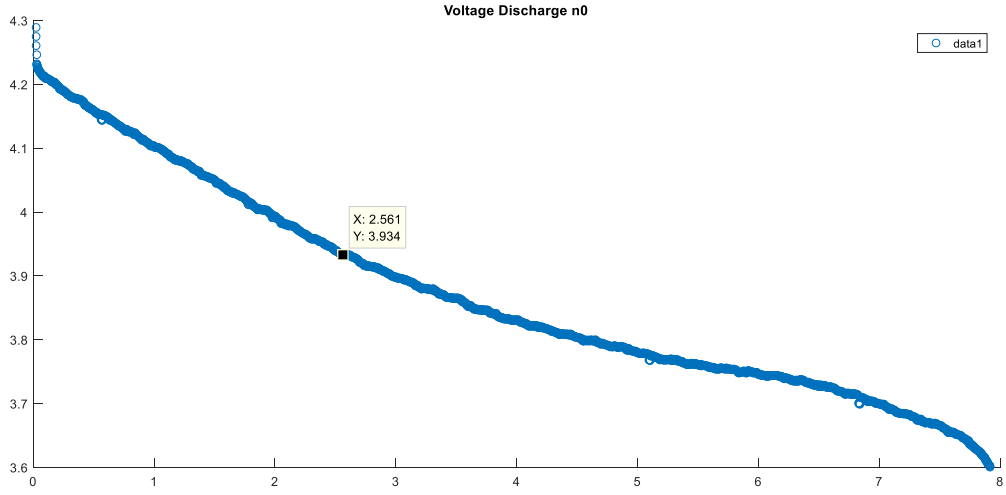


Fig 1.1 Discharge Cycle of Node 0

From figure 1.3 we can see how the overall charge-discharge cycle is. Hence, the overall life span of a battery on a full charge and a discharge cycle. Finding a life span is very important as it gives the overall device and battery activity. It comes handy to find the right battery capacity to use as a power source for any application.

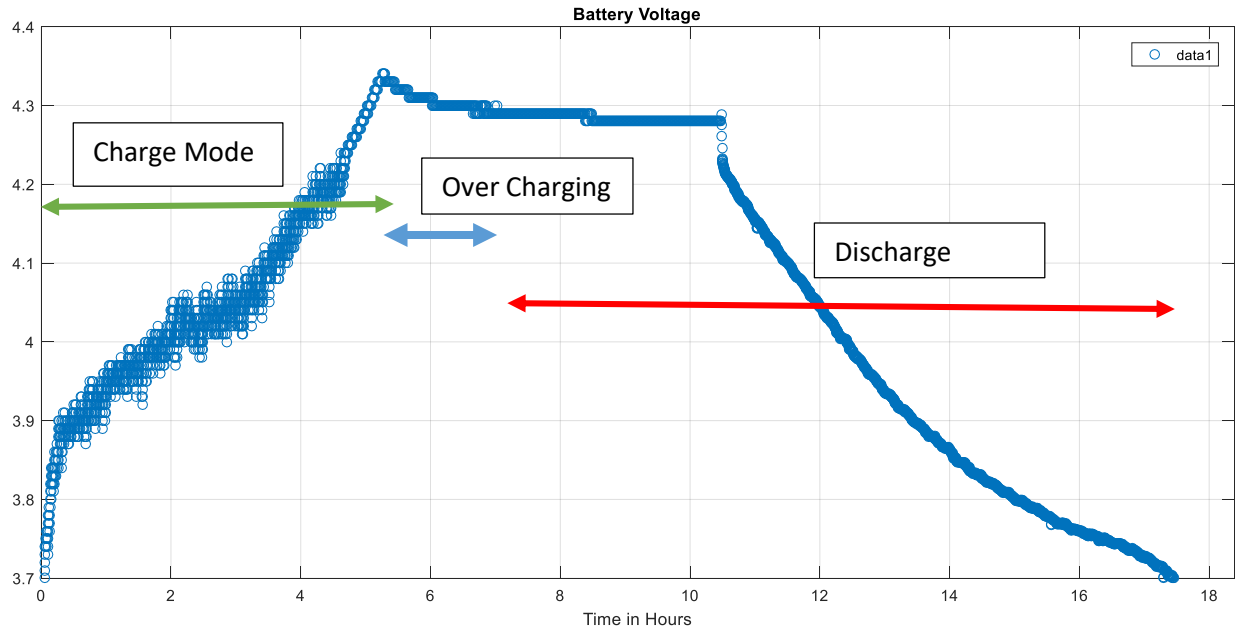


Fig 1.3 Charge – Discharge Cycle

Through correlation and observations, the Voltage cutoff can be improved, again as per requirement and the data was filtered for the accurate results.

3.1.4. Final Analysis with all Four Nodes

The same process is repeated for all the four nodes and with another sample of data for Node0. Data from node0 from the initial and final analysis were compared and analyzed. Using the basic fitting tool from plot in MATLAB the curve properties were compared and found to be almost similar. This ensures consistency proving that readings are valid overtime.

3.1.5. Curve Smoothing

In order to smooth the curves moving averages are taken into consideration. The moving average value of Battery Voltage (VBat) was determined by the total time taken each for a charge and discharge cycle. Since the total time is taken as the SoC determination parameter, 1% of the total time in seconds determines the best method frequency improving both accuracy. For data analysis the values were taken every second, due to a huge data set there was a need for curve smoothing. Moving Averages of the VBat was calculated according for every node and every cycle.

3.1.6. SoC calculation

State of Charge was calculated by converting time in percentages. This is calculated for both charge and discharge cycles (remaining SoC). This was again calculated from the total time – remaining time in percentages.

The equation of the curve was found using a curve fitting tool in MATLAB. It can be even performed using basic fitting tool. But the latter, does not show the goodness of fit statistics like the R-Square value (ideal value:1), Degrees of Freedom (ideal value: 1; non negative) and the root mean square value (RSE, SSE). Using which the goodness of fit can be found. The polynomial equations are found out for each node using later for real time computations.

3.2. Implementation

3.2.1. Application Design

In the early stages of development, Arbor.js interface sketches were created making use of the design method Paper Prototyping. Play, MySQL, Arbor.js packages were installed and using a compatible application Eclipse IDE and a web application was created. The application uses java code to run the scheduler every 10 seconds to scrape the data from URL and manipulate the string data to store as variables in the local database.

It also has code for every node to compute and display the battery percentage and the remaining time left to charge/discharge. This whole setup was deployed in a Linux server after installing the required packages. Since, it is deployed in a server, it is run continuously every 10 seconds until the process is killed by the user.

3.2.2. Code Snippets

Global.java: Main class (global settings) which calls ScheduleWebcrawler.java

ScheduleWebCrawler.java: Utils package in Application directory. Imported Akka from play library for job scheduling. This package calls WebsCrawler.java.

```
public void schedule() {
   Akka.system().scheduler().schedule(Duration.create(0, TimeUnit.MILLISECONDS),
// Initial delay 0 milliseconds
      Duration.create(10, TimeUnit.SECONDS), new Runnable() {
// Create a job scheduler every 10 seconds
          public void run() {
         try {
           WebCrawler.crawl();
         } catch (IOException e) {}
        }
      }, Akka.system().dispatcher());
```

WebCrawler.java: Imports IO packages for scanning input. Scans the URLs and automatically uses space as a delimiter to split the string and send the data to ElectricalDataDAO.java

```
int val = 0;
    ArrayList<String> addr = new ArrayList<String>();
//Created array list to store variable
    addr.add("http://172.26.50.67:8192/0");
    addr.add("http://172.26.50.67:8192/1");
    addr.add("http://172.26.50.67:8192/2");
    addr.add("http://172.26.50.67:8192/3");
    Timestamp timestamp = new Timestamp(new Date().getTime());
 //looping the arraylist until all items in the list are evaluated
    for (val = 0; val < addr.size(); val++) {
//getting value from arraylist position i.e. the URL string at that position
URL my_url = new URL(addr.get(val));
//reading input values from URL into input buffer
      BufferedReader br = new BufferedReader(new InputStreamReader(my_url.openStream()));
//reading line by line
      String temp = br.readLine();
      Scanner sc = new Scanner(temp);
      ElectricalDataDAO dao = new ElectricalDataDAO();
      dao.insert(val, sc.next(), sc.next(), sc.next(), sc.next(), sc.next(), timestamp);
     }
```

ElectricalDataDAO.java: This class inserts the values in to the database from Webcrawler.java and assigns the variables. The database connection is always requested with a Try-catch exception.

```
//assigning variables from the database
//List used to append the values retrieved from DB
  public List<ElectricalData> getLatestElectricalData() {
    List<ElectricalData> list = new ArrayList<>();
    try {
//connect to Database
    Connection connection = DB.getConnection();
//run the select query
    PreparedStatement statement = connection.prepareStatement(selectLatestNodes);
//store retreived results in Result set
    ResultSet rs = statement.executeQuery();
//iterate each line from the results
    while (rs.next()) {
      ElectricalData data = new ElectricalData();
//each row values stored respectively in variables
      data.setNode_id(rs.getString(1));
      data.setVoltage_in(getDoubleData(rs.getString(2)));
```

```
        data.setVoltage_out(getDoubleData(rs.getString(3)));
        data.setVoltage_battery(getDoubleData(rs.getString(4)));
        data.setCurrent_in(getDoubleData(rs.getString(5)));
        data.setCurrent_out(getDoubleData(rs.getString(6)));
        data.setCreation_date(rs.getTimestamp(7));
        data.setBattery_percentage(getBatteryPercentage());
//adding all variables stored results to the List
        list.add(data);
      } //end of try
    } catch (SQLException e) {}
```

ElectricalData.java: This class uses the get-set method to update the variables from the database. It belongs to the model package in the play application.

```
  //variable declarations for methods
  private String node_id;
  private double voltage_in;
  private double voltage_out;
  private double voltage_battery;
  private double current_in;
  private double current_out;
  private Date creation_date;
  private double battery_percentage;

// method to get node id
  public String getNode_id() {
    return node_id;
  }

//method to set node id with a given id. "this" is a pointer.
  public void setNode_id(String node_id) {
    this.node_id = node_id;
  }

//method to get voltage in
  public double getVoltage_in() {
    return voltage_in;
  }
```

NodeDetailsController.java: This is a controller model which is an intermediary for model and view objects.

```
//importing required packages for this program
import play.mvc.Controller;
import play.mvc.Result;
import views.html.index;
import dao.ElectricalDataDAO;
```

```
//extends used to inherit the properties i.e. methods and data members
//of superior class (Controller) into sub-class (NodeDetailsController)
public class NodeDetailsController extends Controller {

// method to get details of node
  public static Result getNodeDetails() {
//creating an object for this method and initializing it
    ElectricalDataDAO dao = new ElectricalDataDAO();
    return ok(index.render(dao.getLatestElectricalData()));
//returning the latest data retrieved through the function
```

Index.scala.html: This is a view object in which arbor.js was implemented. To make the system dynamic a meta command was used to refresh page every 10 seconds. Arbor.js libraries were imported. A legend table is added. Arbor.js particle system parameters were set. Node were created and linked.

```
<head>
!meta command to refresh page every 10 seconds to make arbor.js design dynamic
<meta http-equiv="refresh" content="10">
!Importing arbor.js libraries
<script language="javascript" type="text/javascript"
   src="https://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js"></script>
<script type="text/javascript" src="assets/javascripts/arbor.js"></script>
<script type="text/javascript" src="assets/javascripts/graphics.js"></script>
<script type="text/javascript" src="assets/javascripts/renderer.js"></script>
</head>
<body>
   <script language="javascript" type="text/javascript">
! repulsion = 600, stiffness = 400, friction = 1, gravity is set to be true and these were considered
ideal for the system
          var sys = arbor.ParticleSystem(600, 400, 1);
          sys.parameters({gravity:true});
          sys.renderer = Renderer("#viewport") ;
          var data = {
            nodes:{
! Center node was created
                  centre: {'color':'red','shape':'dot','label':'UH Network'},
!next node0 was created and the same was repeated for rest of the nodes
              @if(nodes!=null && nodes.get(0) != null && nodes.get(0).getMode() != null &&
nodes.get(0).getMode().equalsIgnoreCase("C")){
                 node0:{'color':'green','shape':'dot','label':'node0'},
              }else {
                 @if(nodes!=null && nodes.get(0) != null && nodes.get(0).getMode() != null &&
nodes.get(0).getMode().equalsIgnoreCase("D")){
                    node0:{'color':'red','shape':'dot','label':'node0'},
                 }else{
```

```
                node0:{'color':'lightslategray','shape':'dot','label':'node0'},
            }
        }

 !created a battery percentage node for every node
 n0battery:{'color':'orange','shape':'rectangle','label':@nodes.get(0).getBattery_percentage() +
"%","alpha":1},
 !created another node to display time and the same was repeated for rest of the nodes
 @if(nodes!=null && nodes.get(0) != null && nodes.get(0).getTime() != null){
      n0time:{'color':'blue','shape':'rectangle','label':"@nodes.get(0).getTime()"},
      }else{
            n0time:{'color':'blue','shape':'rectangle','label':"NA"},              }
  !Finally edges were created and linked
  edges:{
            centre:{ node0:{}},
            node0:{node1:{}, n0battery:{}, n0time:{}},
            node1:{node2:{}, n1battery:{}, n1time:{}},
            node2:{node3:{}, n2battery:{}, n2time:{}},
            node3:{centre:{}, n3battery:{}, n3time:{}}
          }
        };
//grafting the data
        sys.graft(data);
</body>
```

Routes and Application Configuration:

Routes: This file defines all application routes (Higher priority routes first)

GET / controllers.NodeDetailsController.getNodeDetails()

Map static resources from the /public folder to the /assets URL path

GET /assets/*file controllers.Assets.at(path="/public", file)

GET /index.html controllers.Assets.at(path="/public/html", file="index.html")

Application Configuration: This is the main configuration file for the application. A secret key is used to secure cryptographic functions.

application.secret="Eqn_m8u]]dRwuv2PrGf<;L9KfjUCo3Gqfo5YH7Z>9s7QLKJMqiA`?x1eoTH]=Tq@"

Set he application languages

application.langs="en"

Global object class - Define the Global object class for this application.

application.global=Global

Define the Router object to use for this application - This router will be looked up first when the application is starting up, already set above.

application.router=my.application.Routes

Database configuration

db.default.driver=org.postgresql.Driver

db.default.url="jdbc:postgresql://localhost:5432/batteryindicator"

db.default.user=pavan

db.default.password="syscon17"

db.default.idleMaxAge=10 minutes

db.default.idleConnectionTestPeriod=30 seconds

db.default.connectionTimeout=20 second

db.default.connectionTestStatement="SELECT 1"

db.default.maxConnectionAge=30 minutes

CalculateBatteryPercentageAndTime.java – This class was created to set global variables for

CalculateBatteryPercentageAndTimeNode0.java,

CalculateBatteryPercentageAndTimeNode1.java,

CalculateBatteryPercentageAndTimeNode2.java,

CalculateBatteryPercentageAndTimeNode3.java,

CalculateBatteryPercentageAndTimeNode0.java: The skeleton of code for the all the nodes is

the same.

```
private void calculate() {
//initialize local variables y and z
   double y = -1;
   double z;
//charge mode
   if (Vin > 4.0 && Vout < 1.0) {
     mode = "C";
     Logger.debug("Node 0 Charge Mode");
     if (Vbat <= 3.80) {
       y = 0;
       time = "5:00";
     } else if (Vbat >= 4.34) {
       y = 100;
       time = "NA";
     } else if (3.80 < Vbat && Vbat < 4.34) {
       z = (Vbat - 4.0542) / 0.1342;
       y =
           0.50336 * Math.pow(z, 5) + 1.1644 * Math.pow(z, 4) - 5.2907 * Math.pow(z, 3) - 5.8996
* Math.pow(z, 2)
               + 37.568 * z + 53.873;
       if (Math.round(y) > 100.0) {
         y = 100;
       } else if (Math.round(y) < 0) {
         y = 0;
```

```
        } else {}
        float t = (float) (5.9 * (1 - y / 100) * 60);
        int hours = (int) (t / 60);
        int minutes = (int) (t % 60);
        time = hours + ":" + minutes;
      }
      batteryPercent = y;
    } // end if
    else if (Vin < 1.0 && Vout > 4.0) {
      Logger.debug("Node 0 Discharge Mode");
      mode = "D";
      // Discharge
      if (Vbat <= 3.63) {
        y = 0;
        time = "NA";
      } else if (Vbat >= 4.23) {
        y = 100;
        time = "8:00";
      } else if (3.63 < Vbat && Vbat < 4.23) {
        z = (Vbat - 3.8726) / 0.15858;
        y =
            -1.4949 * Math.pow(z, 5) + 3.9983 * Math.pow(z, 4) + 2.7806 * Math.pow(z, 3) - 15.006
* Math.pow(z, 2)
                + 30.303 * z + 58.23;
        if (Math.round(y) > 100.0) {
          y = 100;
        } else if (Math.round(y) < 0) {
          y = 0;
        } else {
        }
        float t = (float) (y * 7.9 * 60) / 100;
        int hours = (int) t / 60;
        int minutes = (int) (t % 60);
        time = hours + ":" + minutes;
      }
      batteryPercent = y;
    }// end else if
//When idle the values from the previous entry is taken from the database
 else if (Vin < 1.0 && Vout < 1.0) {
      // idle
      mode = "I";
      Logger.debug("Node 0 Idle mode");
      ElectricalDataDAO dao = new ElectricalDataDAO();
      List<ElectricalData> list = dao.getLatestElectricalData();
      for (ElectricalData data : list) {
        if (data.getNode_id().equalsIgnoreCase("0")) {
```

```
        try {
          batteryPercent = Double.parseDouble(data.getBattery_percentage());
          time = "NA";
        } catch (Exception e) {
          Logger.error(e.getMessage(), e);
          batteryPercent = y;
          time = "NA";
        }
      }
    }
   }else{
    Logger.debug("Node 0 Unknown mode");
   }

 }
}
```

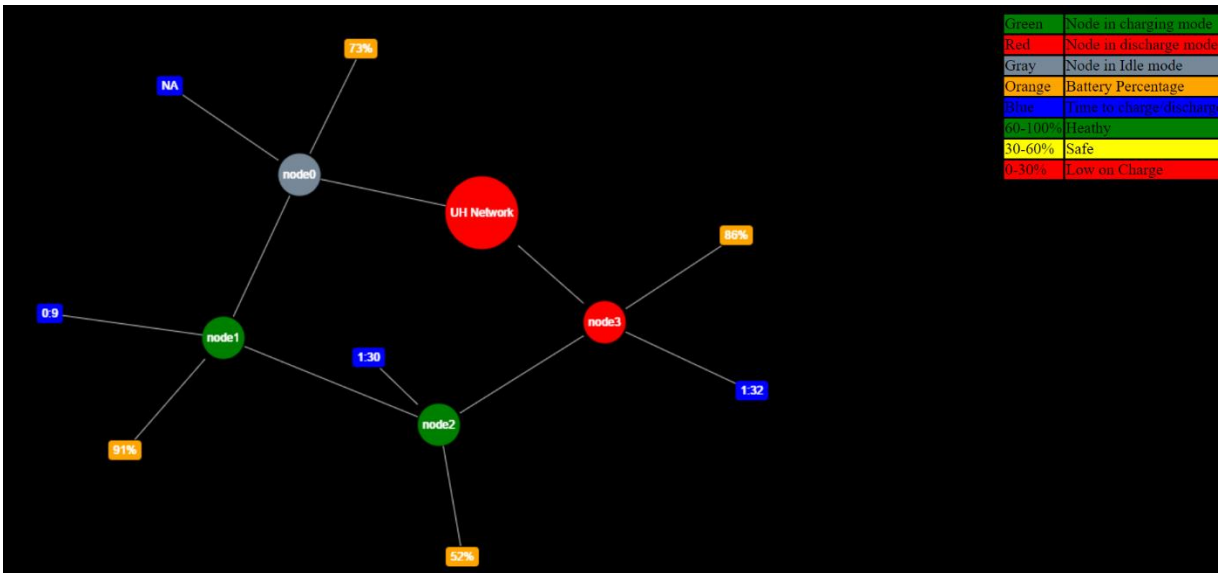
3.4. Maintainability and Scalability

The MVC design pattern was created to enable portability and maintainability. This design separates model, view and controller enabling debugging and makes writing the code easier by breaking it into pieces. Changes in one object does not affect the other. Hence, each of it can be changed as per requirement at any time.

Having different scripts to calculate the battery percentage of each node make it easier if required to add more nodes in future.

4. Results

4.1. Interface Design



4.2. Walkthrough

To start the application:

Go to the application folder

pavan@a1:~/BatteryIndicator$ play start

The webpage can be accessed from http://172.26.50.158:9000/

To kill the process and restart the application:

pavan@a1:~/BatteryIndicator$ ps -ef | grep play

pavan@a1:~/BatteryIndicator$ kill -9 <process id>

pavan@a1:~/BatteryIndicator$ rm RUNNING_PID

pavan@a1:~/BatteryIndicator$ play start

5. Future Scope & Improvements

SoC estimation using current and power: Due to a limited timeline, SoC estimation was performed using SoC estimation using Voltage translation. Since, current and voltage are being already obtained from the device. The Smart battery system can be improved by using coulomb counting and other fuel gauge methods.

Alerts when overcharging: A simple script can be included in the web application to send timely alerts every 15 minutes to the device, in case of overcharging. Alerts can be sent to email to the user to keep track of overcharging as it increases the temperature and hence, making the SoC estimation less accurate.

Landscape Canonical: Landscape [20] is the leading management tool to deploy, monitor and manage your Ubuntu servers. Scripts can be implemented to start the application and manage the Ubuntu server in which the setup was installed.

Apache math library: The apache math library [21] is a powerful math tool in Java that can be used to estimate the SoC estimation directly. This complicates the setup but worth looking into as it reduces the work of user for data analysis. Newton Raphson method, Curve fitting and moving averages trend estimation libraries can be used for automated calculations.

Improve the Frontend Design: Arbor.js is a very powerful predesigned visualization library, it can be further developed to improve the user interface and make it more dynamic.

6. Conclusion

A web application with an interactive user interface is designed and deployed in a server. The arbor.js JavaScript library user interface is used to display the battery level and the battery life remaining in hours using the web application Play to fetch, manipulate, perform computations and send/receive request from the interactive web design. The web application is compatible in most browsers and mobile browser view is also feasible when connected to the private network.

This project had an extensive research phase which include learning about battery basics and SoC estimation, full stack web development, data procurement from Raspberry Pi nodes.
The data analysis was the toughest and the most crucial part of the research as it was a requirement to maintain high accuracy in order to get accurate SoC estimation through a not so accurate SoC estimation method using voltage translation. Curve fitting and comparing goodness of fit statistics of every node for charging and discharging modes was a cumbersome task. The initial analysis took a week to come to an understanding of how the battery cycles work. However, during the final data analysis phase for rest of the nodes took less than hour, since I was well acquainted with the procedure.

The web application development was easy. Coding was performed over a period of two months for each object in application separately. Once the application became stable and error free, it was deployed in the server and thereafter changes were made directly in the server. Arbor.js was a major setback, though easy for a simple setup, it was very complicated to implement the much deeper functions like hovering to show nodes, click to show information. But at the end, the simple setup developed was able to meet the requirements.

8. References

[1] Retrived from "*IDC MaturityScape: Internet of Things*" by Vernon Turner and Carrie MacGillivray

[2] *Foundational Elements of an IoT Solution: The Edge, The Cloud, and Application Development* by Joe Biron and Jonathan Follett

[3] https://github.com/aboudou/picheckvoltage

[4] http://www.instructables.com/id/Simple-IOT-project-for-Beginners/

[5] https://cdn-learn.adafruit.com/downloads/pdf/adafruits-raspberry-pi-lesson-3-network-setup.pdf

[6] http://www.instructables.com/id/Ultimate-Raspberry-Pi-Configuration-Guide

[7] "*Raspberry Pi with Java: Programming the Internet of Things*" by Stephen Chin with James L Weaver

[8] Retrived from http://batteryuniversity.com/learn/

[9] https://www.arrow.com/en/research-and-events/articles/choosing-the-right-battery-for-your-internet-of-things-application

[10] http://liionbms.com/php/wp_soc_estimate.php

[11] https://www.adafruit.com/products/1549

[12] https://learn.adafruit.com/adafruit-usb-power-gauge-mini-kit

[13] https://www.playframework.com/

[14] http://arborjs.org/

[15] http://blog.thinkst.com/2011/06/simple-graphs-with-arborjs.html

[16] https://www.postgresql.org/

[17] https://eclipse.org/

[18] http://mobaxterm.mobatek.net/

[19] https://www.mathworks.com/products/curvefitting.html

[20] https://landscape.canonical.com/

[21] https://commons.apache.org/proper/commons-math/userguide/index.html

9. Appendix

Node 0:

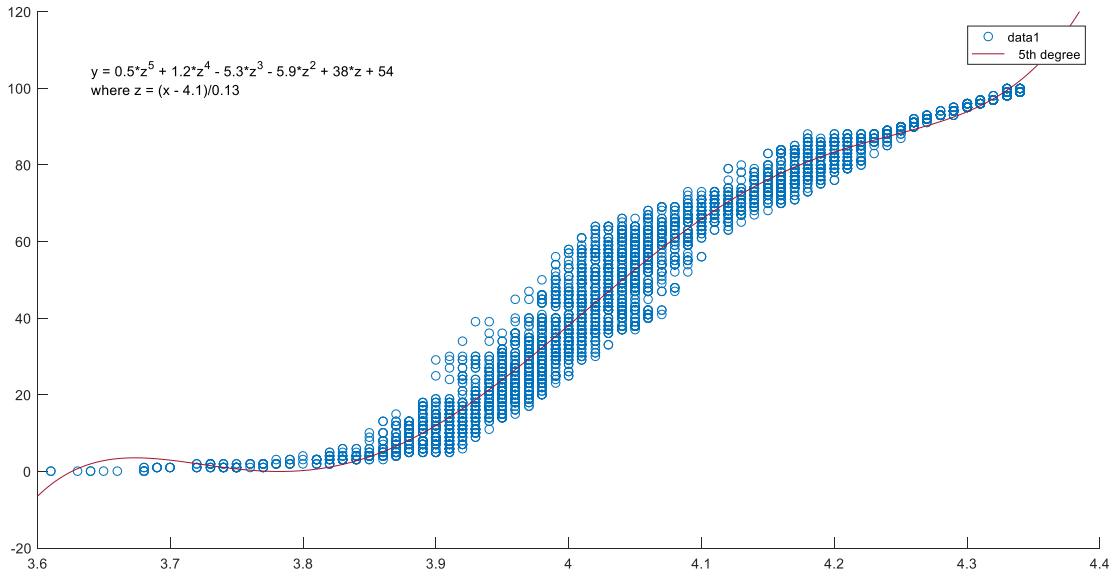


Fig 1.4 Charge Cycle

Voltage Bounds	3.60 – 4.30
Moving Average	18
Time to Charge (Hours)	5.30 hours
R-Square	0.9983

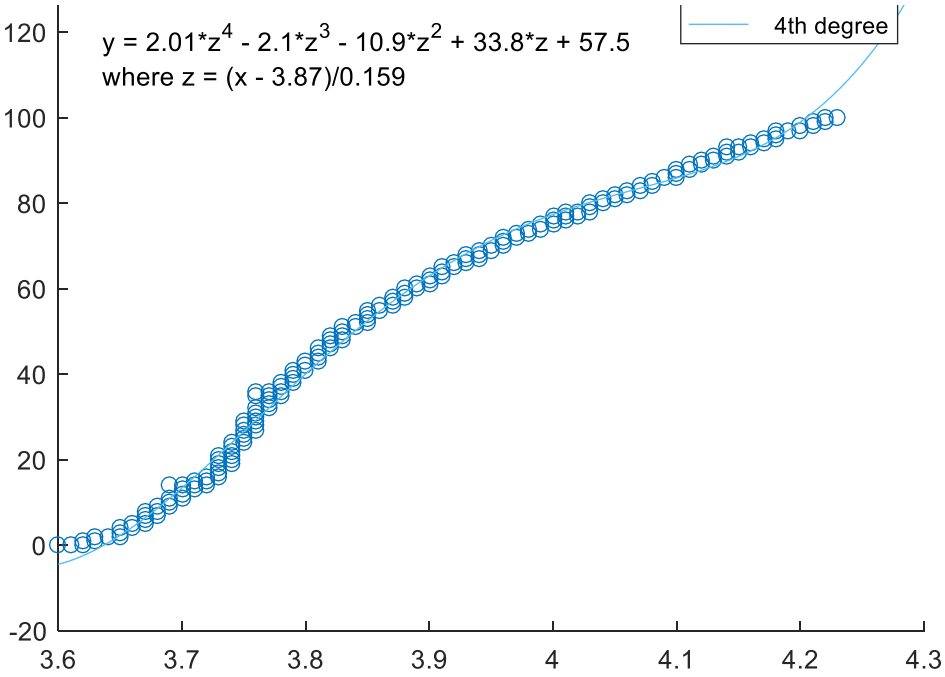


Fig 1.5 Discharge Cycle

Voltage Bounds	4.24 -3.60
Moving Average	28
Time to Charge (Hours)	8
R-Square	0.9994

Node 1:

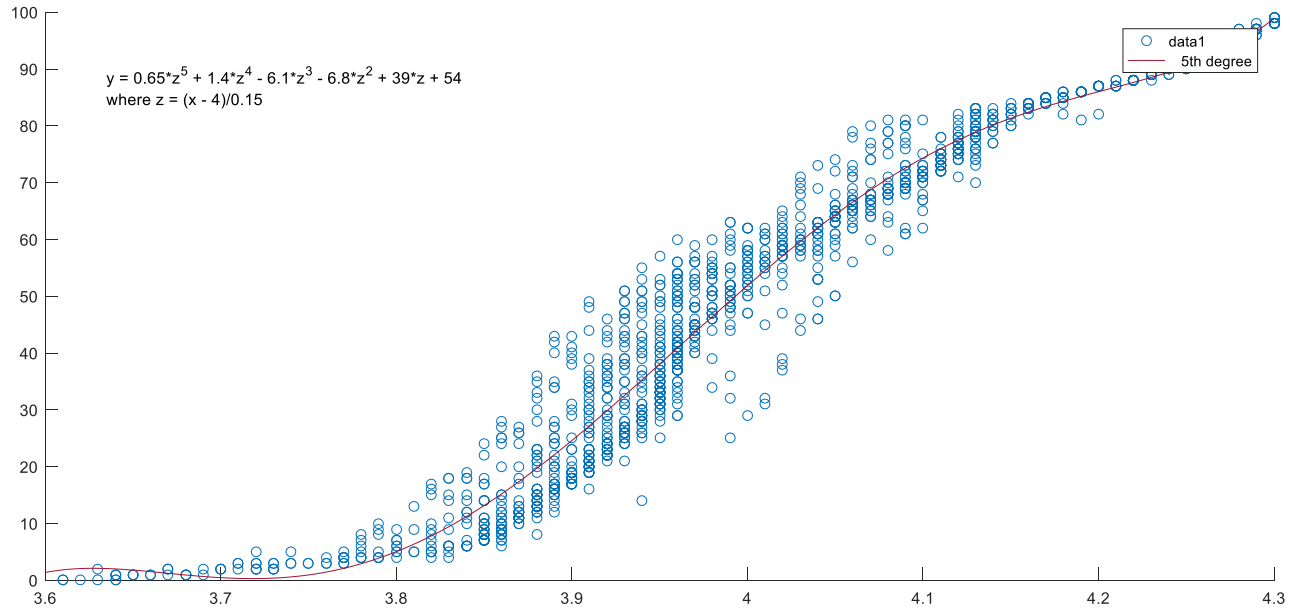


Fig 1.6 Charge Cycle

Voltage Bounds	3.60 - 4.30
Moving Average	6
Time to Charge (Hours)	2
R-Square	0.9987

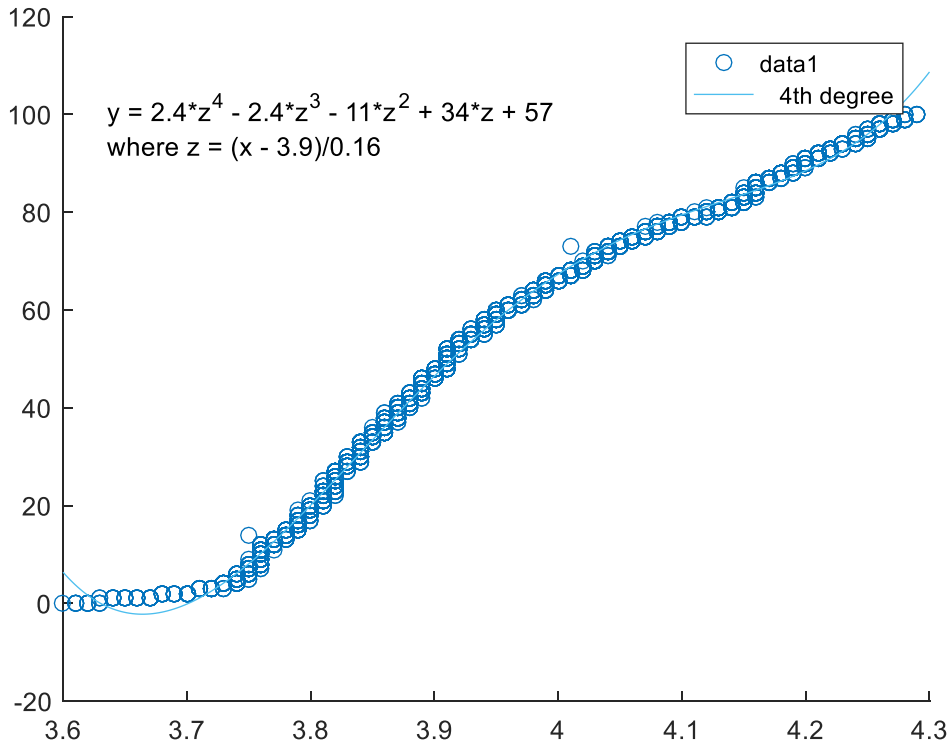


Fig 1.7 Discharge Cycle

Voltage Bounds	3.60 – 4.30
Moving Average	13
Time to Charge (Hours)	3.5
R-Square	0.9977

Node 2:

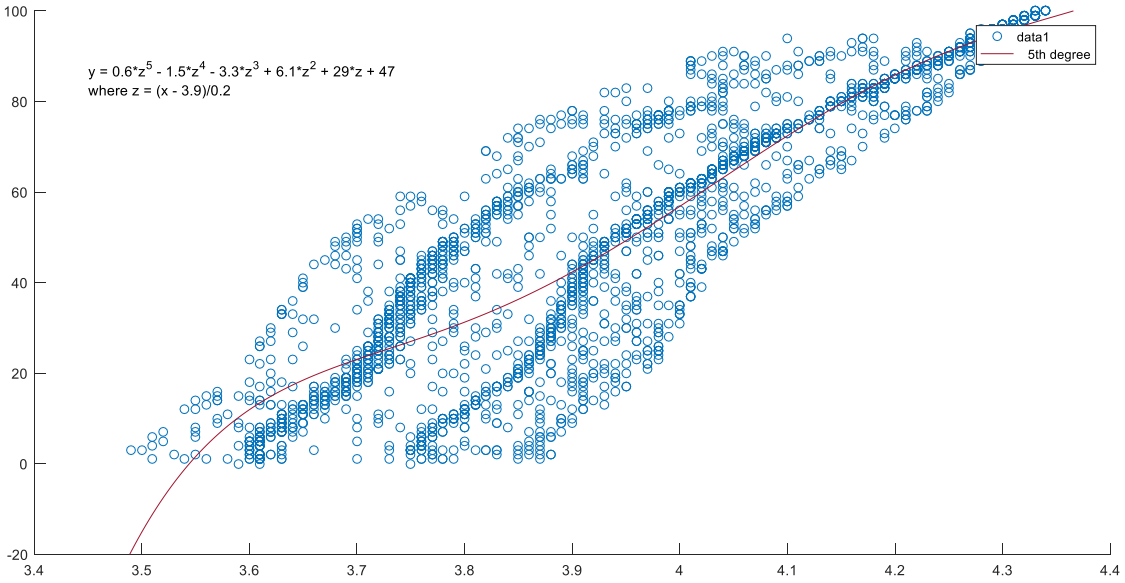


Fig 1.8 Charge Cycle

Voltage Bounds	3.60 - 4.30
Moving Average	11
Time to Charge (Hours)	3
R-Square	0.9980

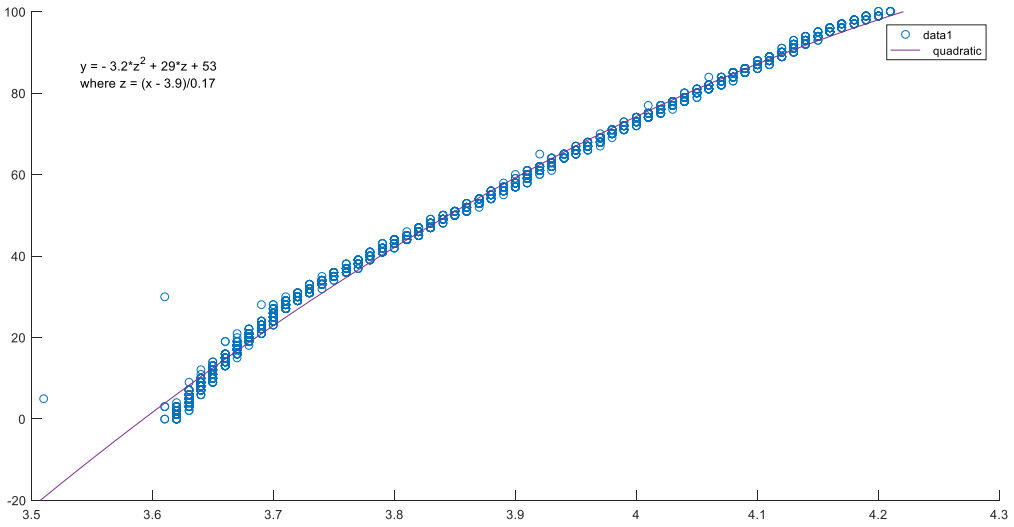


Fig 1.9 Discharge Cycle

Voltage Bounds	4.29 – 3.60
Moving Average	13
Time to Charge (Hours)	3.5
R-Square	0.9987

Node 3:

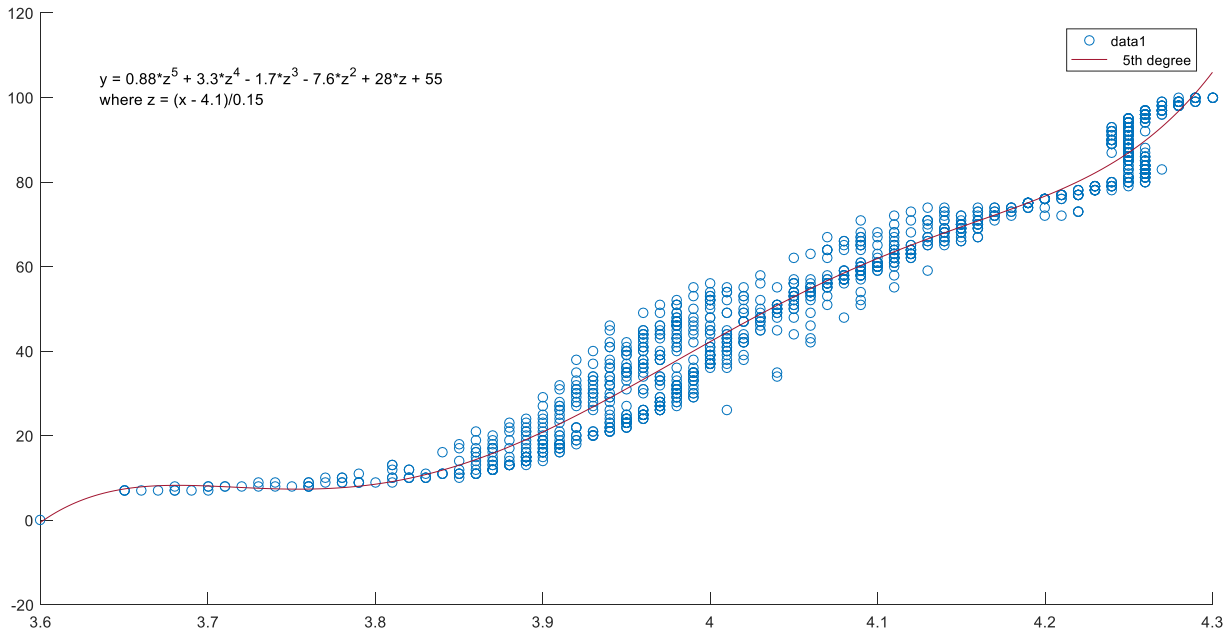


Fig 1.10 Charge Cycle

Voltage Bounds	3.60 - 4.30
Moving Average	6
Time to Charge (Hours)	1.5
R-Square	0.9992

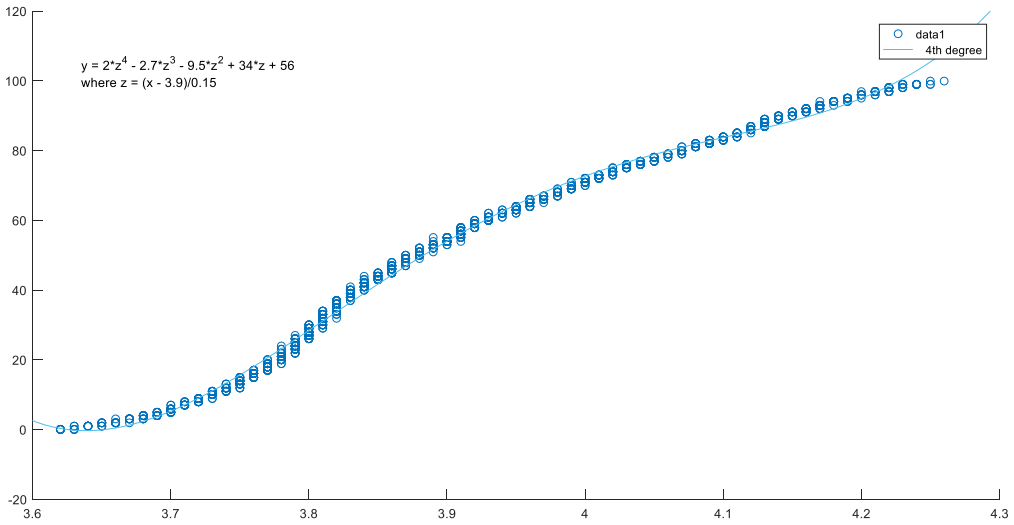


Fig 1.11 Discharge Cycle

Voltage Bounds	3.60 - 4.24
Moving Average	6
Time to Charge (Hours)	1.8
R-Square	0.9987