Development of A Smart Multi-Device SCADA (Supervisory Control and Data Acquisition) System for Household Appliances

by

Muhammad Ali Amin Vellani

B.Eng., NED University of Engineering & Technology, 2012

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Spring 2018

Approval

Name:	Muhammad Ali Amin Vellani
Degree:	Masters in Engineering Science
Title:	Development of A Smart Multi-Device SCADA (Supervisory Control and Data Acquisition) System for Household Appliances.

Examining Committee: **Chair**: Mr. Steve Whitmore
Senior Lecturer

Dr. Ash M. Parameswaran, PhD, P.Eng
Senior Supervisor

Mr. Bob Gill, P.Eng
Supervisory Committee Member

Date Defended/Approved: April 13th, 2018

Abstract

Power consumption and electrical appliances are growing rapidly as technology advances, so does the need to identify which appliances are consuming how much energy for the benefit of both environment and economic point of view. Based on the current smart home systems present in the market that can monitor power and have a proper infrastructure but have poor data analytics at the user end. In this project, we developed a device that measures, logs and represents the measurement to the power consumption data of the Electrical appliances to the user in a proper user-friendly manner by connecting to the same Wi-Fi of the smart plugs. The developed device, acts as a server to communicate with the energy measuring devices to extract the data, records and logs it in a database, and provides it to the user on-demand in a simple Graphic User Interface.

Acknowledgements

I would like to thank Dr. Ash M. Parameswaran for accepting the supervision of the project and always be supportive towards me.

I would also like to thank Mr. Bob Gill for providing me with the idea of the project and encouraged me to bring my own innovative ideas and implement them to make this happen.

This project would just be a concept if it weren’t for Dr. Ash and Mr. Bob.

A special thanks to Simon Fraser Engineering sciences department for making it all happen.

Also, Thankyou Mr. Steve Whitmore for agreeing to chair the committee on such short notice.

Table of Contents

List of Tables

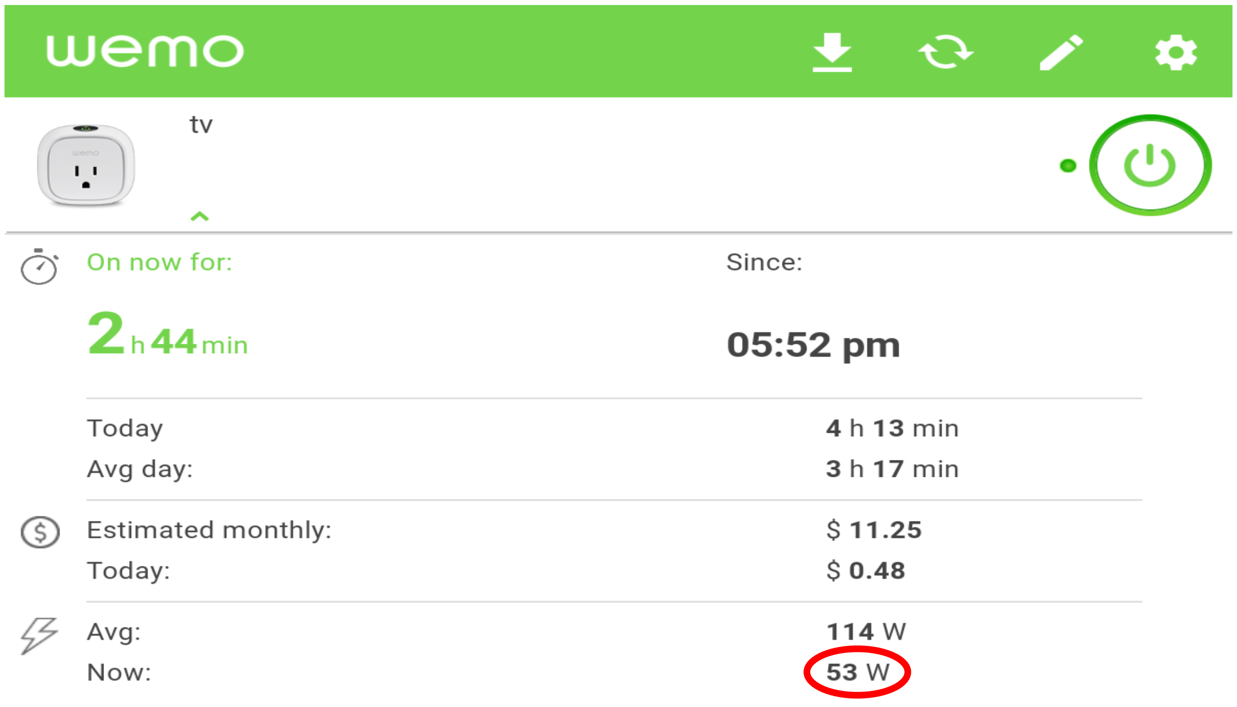
List of Figures

Chapter 1. Introduction

1.1. Background

There are many devices present in the market right now capable of promising a better smart home for the consumers, For example Nest, Alexa, T-P-link Belkin etc. They provide control and automation for lights, thermostats, television, audio systems, security systems which include door locks and cameras, etc. Some of them also have options which include remote control through a distant site and voice commands recognition with a wide variety of different system integration like Google Home and voice assistant, amazon Alexa, Nest, and IFTTT. But they fall short when it comes to energy consumption data management and representation.

Systems developed to measure the energy are surely capable of doing so very accurately but unfortunately do not provide power consumption data to the user if a very sophisticated manner. The data that is presented at the user end is either data in the form of number i.e. how much the device connected to the smart socket is driving right now or in the form of an average of how much it was collected over the whole day.



On now for: 2 h 44 min

Since: 05:52 pm

Today	4 h 13 min
Avg day:	3 h 17 min
Estimated monthly:	$ 11.25
Today:	$ 0.48
Avg:	114 W
Now:	53 W

Figure 1- WeMo app data representation

In the same way, there is no way for the user to compare the data for the whole day minute by minute or device by device. Also that the data protocol is locked by vender and the capability of exporting the data is not even present in most of the devices, this means, that the data collected by the system remains within the system and the user is limited by the options provided by the vendor and if more options for better data representation can only be achieved if and when the vender decides to add the option.

The users require more sophisticated form of visual representation of the data and have features which include visually distinguishing charts representing which device drives what amount of energy at which time and when and where the energy consumption could be minimized.

1.2. Project Summary

Here we have proposed a system which attaches to the same network infrastructure already present and listens to the devices for the data recorded and present it a better manner. The product developed has the following features:

a. Automatic detection of smart plugs (WeMo) present/connected on/to the same local Wi-Fi network capable of calculating the power consumed.
b. Node Server: Creates a local server (which is still on the local Wi-Fi) to access all the smart plugs present on the network.
c. Extract the energy consumption data of the whole household devices attached to the smart plugs using a server fetch.
d. Maintains a continuous fetch cycle that saves the power consumption data in real time from the smart power socket.
e. Saves all the data fetched real-time continuously on a locally maintained database.
f. Visual GUI on a remote device connected to the same wireless system capable of showing the power consumption data to the user in a sophisticated manner where the options include comparing one device to another, finding total data curve of all the devices present, minute by minute real time as well as post collected data.

The project is distributed in 3 parts. The research on the smart plug that is used for our project. This smart plug is the main device that measures the energy consumed by the household device attached to it. The second part of the project is node.js server that is created using the WeMo client libraries, [5]. The data measured by the smart plug is kept to itself until the Node.JS server communicates with the smart plug and extracts it. This server is runs on a JavaScript engine raspberry pi 3 device and saved on the same device in a database form.

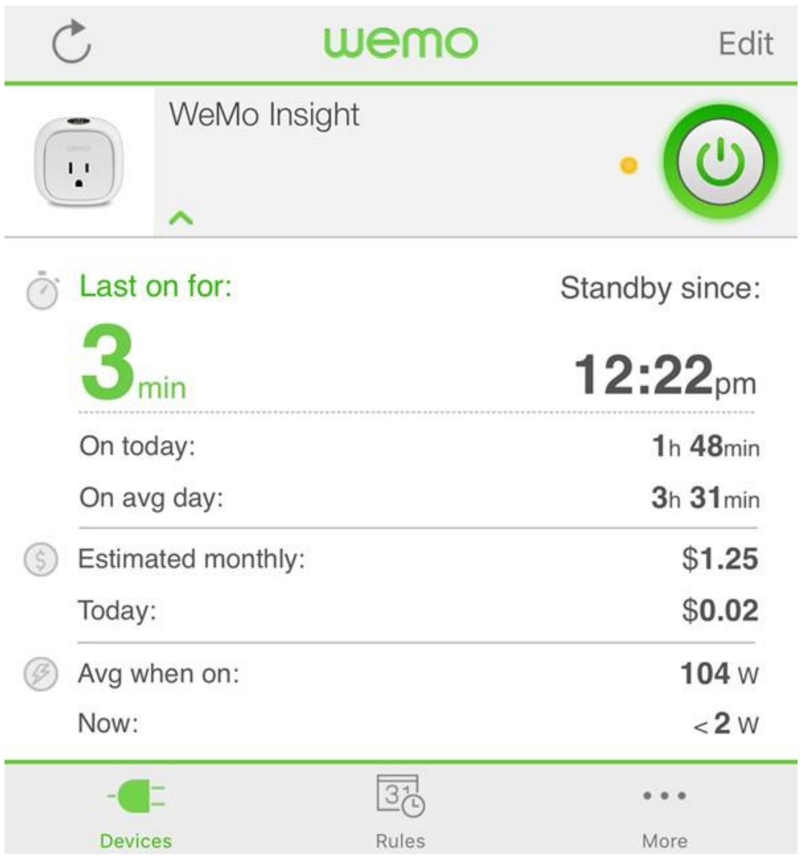
The third part of the project is the visual representation of the data which is also ran on the same raspberry pi and is based on a python. That part takes the data being saved and show to the user in a graphical manner.

The report is distributed in 7 major sections, in the beginning introduction and motivation of the project and explanation of what needed to be done, then the literature review 1 where research is conducted about different devices capable of calculating the energy consumption and selection of the device for this project. After that the secondary literature review conducted on WeMo switches and its software in detail is presented. Followed by the project product development, which includes the background of technology, implementation, and testing scenarios of the product along with data collected, results and graphs which were results of the testing scenarios followed by conclusion and future considerations.

Chapter 2. Energy Measuring Smart Plugs Literature Review

2.1. Belkin WeMo Insight

When compared with other smart plugs, it is one of the best smart plugs out in the market in terms of features and accuracy. It works with many different platforms and can be integrated with Alexa, nest, google home, SmartThings and IFTTT and is decent looking in terms of product design as well. The setup of the device requires a little work in the beginning but once setup, the app works smoothly. App features in1cludes the power being consumed right now; the total power consumed all day and turn on and off features. The app requires a lot of work in terms of the data's visual representation



WeMo Insight

Last on for: 3 min

Standby since: 12:22pm

On today: 1h 48min

On avg day: 3h 31min

Estimated monthly: $1.25

Today: $0.02

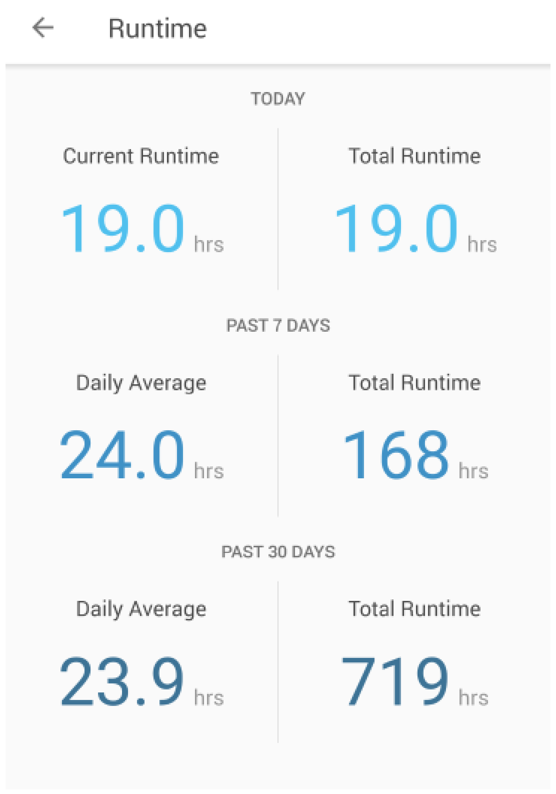
Avg when on: 104 W

Now: < 2 W

Figure 2- WeMo android app interface

2.2. TP-Link Smart-Plug Wi-Fi Hs-110

The setups on iOS and android is quite easy. Feature wise it has most of the features required by common users, but the app protocol is locked to the app Kasa with limited data representation options:


Runtime

TODAY

Current Runtime 19.0 hrs

Total Runtime 19.0 hrs

PAST 7 DAYS

Daily Average 24.0 hrs

Total Runtime 168 hrs

PAST 30 DAYS

Daily Average 23.9 hrs

Total Runtime 719 hrs

Figure 3- T-PLINK android app

The TP-link smart Wi-Fi plug HS-110 is rich in features including

- Remote Access: Control devices connected to the Smart Plug from anywhere, if you are connected to the internet using the Kasa app provided by TP-LINK.
- Smart scheduling feature which enables the user to set timers on turning the devices on and off. Away feature is also available which is an Extension of scheduling feature, devices can be set to turn on and off at desired times to give the appearance that someone is home.
- Easy to Use and Install. the user can connect the Smart Plugs to your Wi-Fi network and control using the free Kasa App on your smartphone.

- Compatible with Amazon Alexa and the Google Assistant, giving you the ability to control devices connected to the Smart Plugs with voice commands.
- Analyze a device's real-time and historical power consumption.

2.3. Ihome ISP8:

iHome has all the features like the other ones but the app gives issues when tried with android 7.0 and above but works well with iOS devices. Its reliable and one of the most good-looking producers out there. The device works with most of the smart home platforms is reliable, but the integration is highly limited. For example, the user in unable to make IFTTT scripts which enables the user to create simple programming basics for example "open home when I reach home".

2.4. D-Link Wi-Fi Smart Plug:

D-LINK smart plug look bulky and have a box like feeling but is feature packed as compared to others. Energy monitoring is also present in the app but as mentioned before, the data is not nicely represented in the app. One of the most important features that's lacking is IFTTT integration which allows users to automate and schedule switches.

2.5. Summary Of All Devices

Table 1- Summary of All Smart Devices

	Dlink Wi-Fi Smart Plug	T-Plink Smart Wi-Fi Plug with Energy Monitoring	Ihome Smart Plug	WeMo Insight Switch
Model Number	Dsp-W215	Hs110	Isp8	F7c029fc
Manufacturer	Dlink	TP-link	Ihome	Belkin
Remote Control	Yes	Yes	Yes	Yes
Energy Monitoring	Yes	Yes	Yes	Yes
Wifi	Yes	Yes	Yes	Yes
Max. Power Rating	1800w	1800w	1800w	1800w
Android App	Yes	Yes	Yes	Yes
Web View	No	No	No	No

Protocol Locked	Yes	Yes	Yes	Yes
IFTTT Support	Yes	No	No	Yes
Price	$30	$30	$60	$70
Expansion Capabilities	No	No	No	Limited
Communication Protocol Lock	Yes	Yes	Yes	Yes
Recommended	No	No	No	Yes
Accuracy	Good	Good	Good	Excellent

2.6. Conclusion:

Looking at the features, accuracy and the protocol used, the device that was chosen was WeMo insight switch.

Chapter 3. Belkin WeMo Insight Switch with Energy Monitoring Literature Review

3.1. Product Features

Belkin WeMo insight switch and app has many features

Some of the features of the product include:

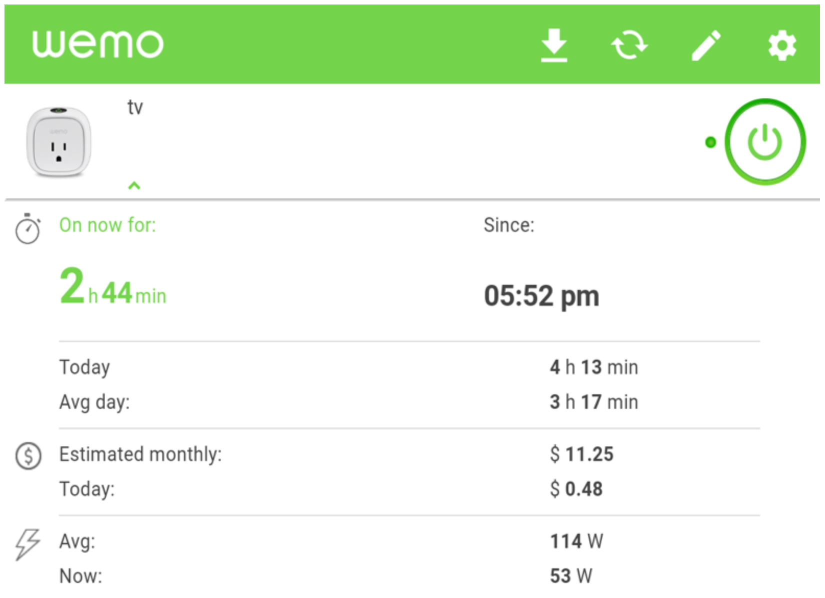
- Turn switch on or off — from anywhere even if you are not on the local network.
- Monitor device energy consumption.
- Calculate the costs of the bill for the appliance.
- Set a schedule for lights/appliances turn on and off.
- Works with Amazon Alexa and Google Voice for hands-free voice control.
- Works with Nest Thermostat for automatic home and away modes.
- Works with your existing home Wi-Fi network and mobile Internet (3G/4G), no hub or subscription required.
- Modular system. You can add additional WeMo Insight smart plugs easily, any time.
- Intuitive, easy set up and easy to use.

- Works with the entire family of WeMo products.
- Works with IFTTT, connecting you to a whole world of Web apps.
- Free WeMo App for Android and iOS operating systems.

But unfortunately, it falls short on the web-interface where the data and control options should be present for the user on the web where the user can log in and view the status, turn device on/off and view and analyse the data.

3.2. App Features

- **Compressive**: Summary of all the data in one place which includes the costs when the switched was turned on, for how long has It been running, estimated monthly and cost.



On now for: 2 h 44 min

Since: 05:52 pm

Today	4 h 13 min
Avg day:	3 h 17 min
Estimated monthly:	$ 11.25
Today:	$ 0.48
Avg:	114 W
Now:	53 W

Figure 4- WeMo app front page

- **Multiple device control:** multiple device can be turned on and off with on the single page.

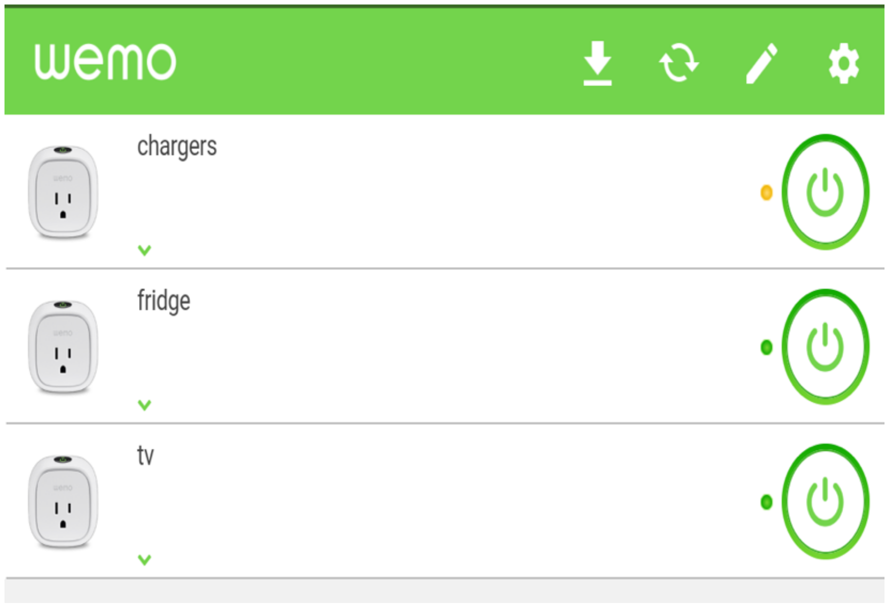


Figure 5- WeMo multiple devices

- **Name and description**: description of the device can be added along with a photo for easier recognition and use.

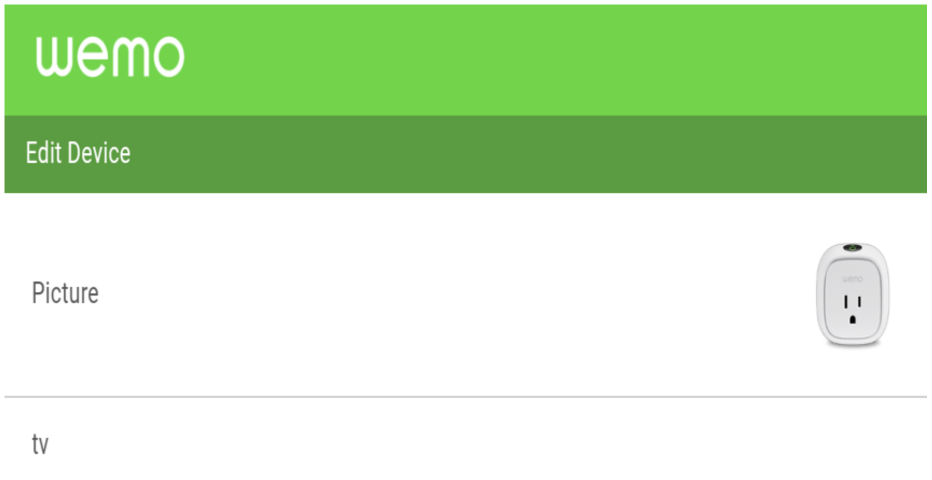
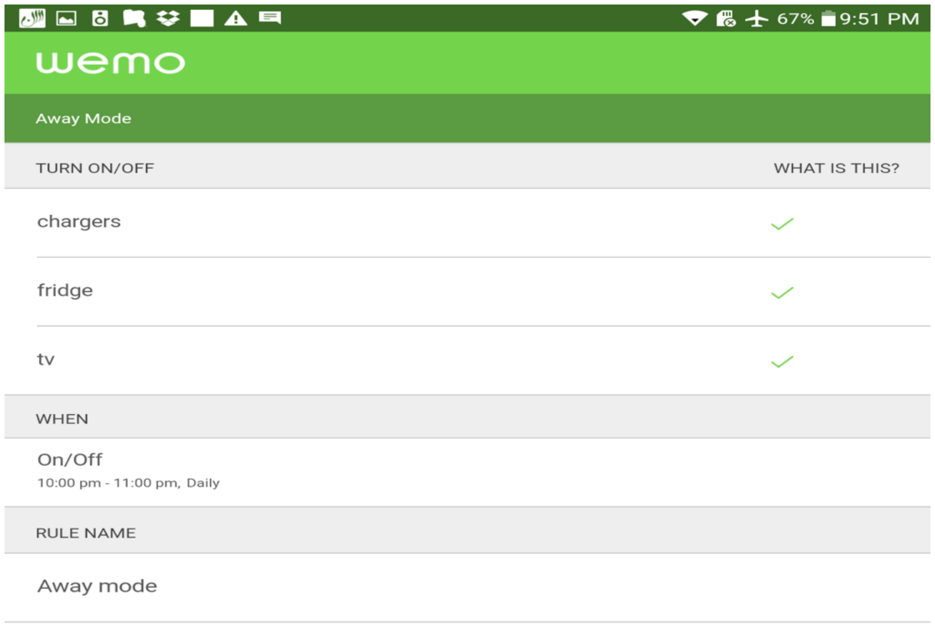


Figure 6- WeMo name and description

- **Away mode:** away mode is present to schedule easy turn on and off to give the look as if someone is living


wemo

Away Mode

TURN ON/OFF | WHAT IS THIS?

chargers ✓

fridge ✓

tv ✓

WHEN

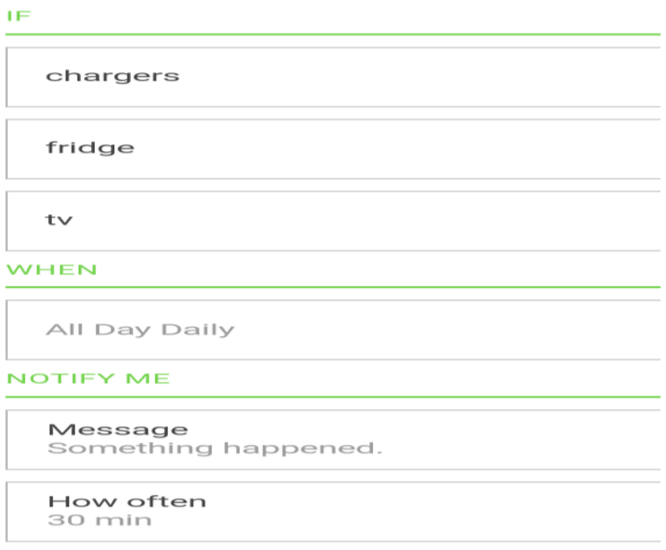
On/Off
10:00 pm - 11:00 pm, Daily

RULE NAME

Away mode

Figure 7- WeMo away mode

- **Rules creation:** rules can be created inside the app without the need of advanced IFTTT


IF

chargers

fridge

tv

WHEN

All Day Daily

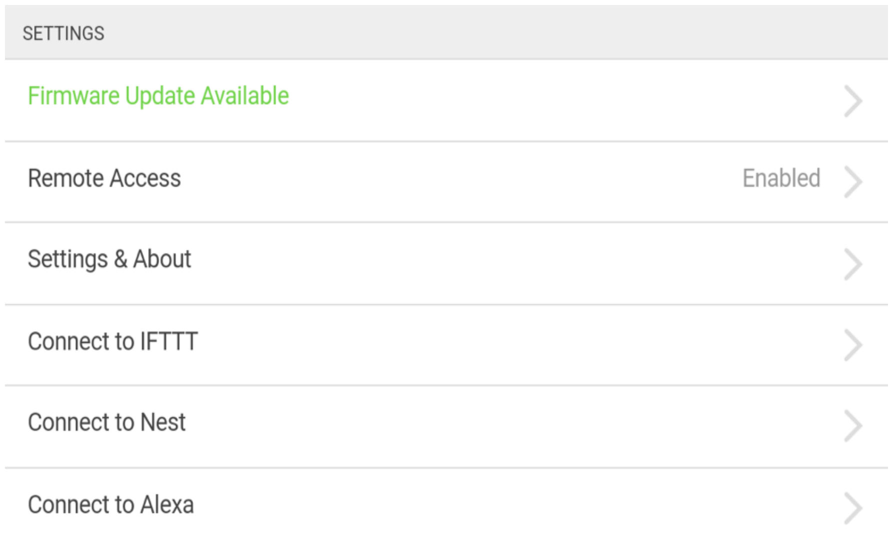
NOTIFY ME

Message
Something happened.

How often
30 min

Figure 8 – WeMo Rules support

- **3rd Party Integration:** easy 3rd party integration options are present in the app options to connect to IFTTT, NEST, Alexa etc.


SETTINGS

Firmware Update Available

Remote Access — Enabled

Settings & About

Connect to IFTTT

Connect to Nest

Connect to Alexa

Figure 9- WeMo 3rd party integration support

Chapter 4. Project Development: Smart Multi-Device SCADA (Supervisory Control and Data Acquisition) System for Household Appliances

4.1. Node.js

Node.js [1] is used to create a webserver that talks to all the smart WeMo plug devices on the Wi-Fi network. just like the devices talk to the android app, the devices also communicate to the custom node.js server that mimics communication protocol.

Node is designed to build scalable network applications, where connections can be handled concurrently [1]. Upon each connection the callback is fired, but if there is no work to be done, Node remains idle. [2]

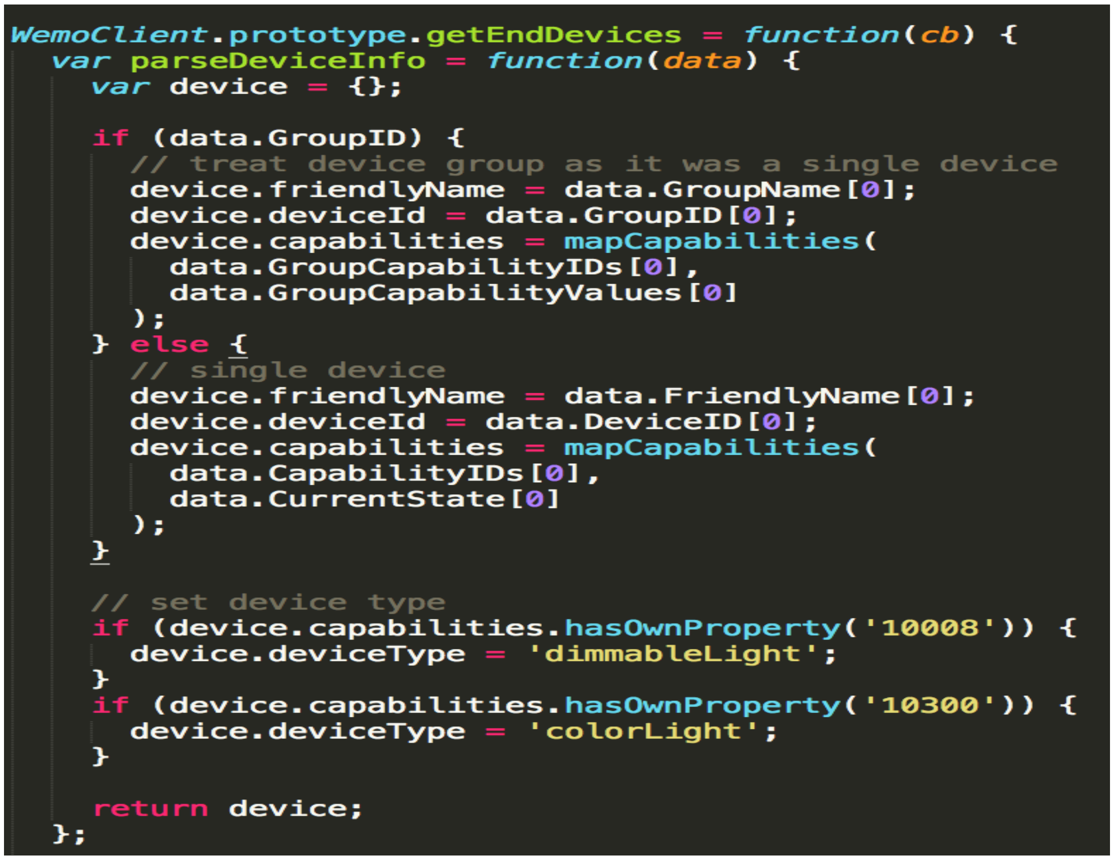
This contrasts with today's more common concurrency model where OS threads are employed [3]. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process, since there are no locks [3]. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, scalable systems are very reasonable to develop in Node.js. [3]

4.2. Data-Fetching Server

All the WeMo devices have a specific MAC address and when connected to a Wi-Fi network, those devices are allotted a unique IP address along with a specific port which acts like a tunnel for the device communication to the server. When a device is turned on and connected to the Wi-Fi, it is discoverable on the network through that port and IP address.

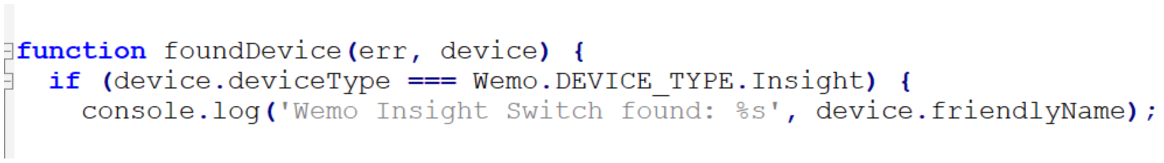
The data-fetching server serves two purposes. First it detects all the smart plug WeMo devices on the local Wi-Fi network. For every device detected, the ID address, friendly name assigned by the user to device, mac address and serial address is received.

Along with the basic network device information that is gathered, it is also detected if the device is capable of measuring energy consumption of the household device attached to it using the function:



```
WemoClient.prototype.getEndDevices = function(cb) {
  var parseDeviceInfo = function(data) {
    var device = {};

    if (data.GroupID) {
      // treat device group as it was a single device
      device.friendlyName = data.GroupName[0];
      device.deviceId = data.GroupID[0];
      device.capabilities = mapCapabilities(
        data.GroupCapabilityIDs[0],
        data.GroupCapabilityValues[0]
      );
    } else {
      // single device
      device.friendlyName = data.FriendlyName[0];
      device.deviceId = data.DeviceID[0];
      device.capabilities = mapCapabilities(
        data.CapabilityIDs[0],
        data.CurrentState[0]
      );
    }

    // set device type
    if (device.capabilities.hasOwnProperty('10008')) {
      device.deviceType = 'dimmableLight';
    }
    if (device.capabilities.hasOwnProperty('10300')) {
      device.deviceType = 'colorLight';
    }

    return device;
  };
```

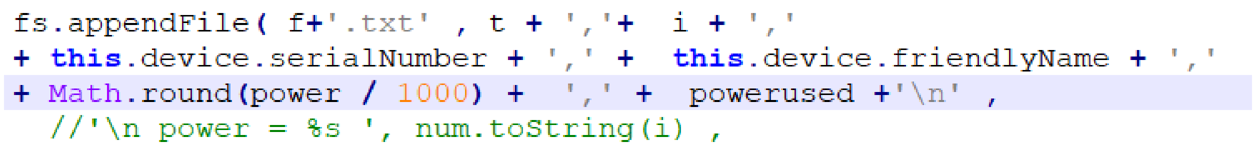
Figure 10- Function for network device detection



```
function foundDevice(err, device) {
  if (device.deviceType === Wemo.DEVICE_TYPE.Insight) {
    console.log('Wemo Insight Switch found: %s', device.friendlyName);
```

Figure 11- Function for device detection

“WeMo Client Library” [4] was used for the basic discovery of the devices present on the network which uses the same protocol used by the android app server built on node.js [2]. Further the extraction of the data, parsing, and saving it in the database is done by the following JavaScript code which uses the variables initialized and loaded by the previous function



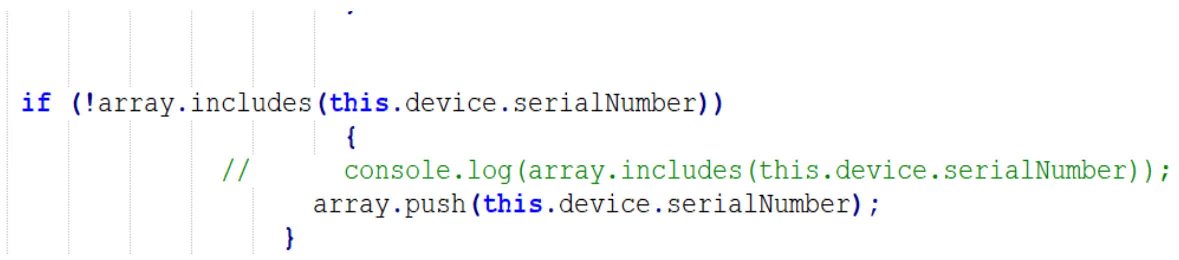
```
fs.appendFile( f+'.txt' , t + ','+  i + ','
+ this.device.serialNumber + ',' +  this.device.friendlyName + ','
+ Math.round(power / 1000) +  ',' +  powerused +'\n' ,
  //'\n power = %s ', num.toString(i) ,
```

Figure 12- Database maintaining script

The JavaScript function saves the following in the database for the graphical code to read:

a) Time stamp,
b) Device ID (0,1,2….),
c) Serial number of the devices,
d) Name of the device set by the user,
e) Power being consumed by the device at that time instance,
f) Total power consumed by the device throughout the day real time updating.

Multiple Device Capability: As soon as a new device is added to the network, the web-server has the capability to automatically adding the device in the database being maintained, with the name assigned to it if any, serial number, and the power being consumed.



```
if (!array.includes(this.device.serialNumber))
                    {
              //      console.log(array.includes(this.device.serialNumber));
                   array.push(this.device.serialNumber);
                 }
```

Figure 13- Expansion of devices script

4.3. Python:

Python is an interpreted high-level programming language for general-purpose programming [6]. Python has a design philosophy that makes code readable, by making use of whitespace [7]. It provides easy and clear programming for small and large scales. [8]

Python features a dynamic type system and automatic memory management [9]. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library [10], with a wide range of functionality which includes:

- Graphical user interfaces
- Web frameworks
- Multimedia

- Databases
- Networking
- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

4.4. Graphical Representation Unit:

The graphical interface unit is built on python. The python code takes the data saved from the server into a database and then is extracted to create a graphical representation for the user.

For the graphing visualization option, the settings can be change between

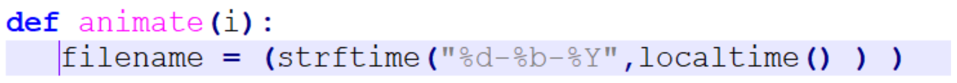
1) Monitoring window: last 5mins/30mins/300mins power consumption of the devices in real time can be selected for view in the GUI.
2) Power consumption of the whole current day and previous days.
3) Bar graph and time graph.
4) View data for any date.

Python libraries used in this project:

1. *Matplotlib* [11]: *Matplotlib* library is a 3rd party library used to create the graphs and figures. It is rich in graphical representation options and supports different styles.
2. *Dates, localtime, strftime [12]:* is time stamp providing library used to import the dates and time data either from computer or the world clock server to provide accurate ticks. These time stamps are saved with the power consumption data in the database.
3. *Animation_api*[13] : animation is the API provided by *Matplotlib* to update the figures in a continuous loop as the data in the server database is changed or updated.
4. *Style_api* [14]: style API is provided by *Matplotlib* to support different styles and formatting options for graphs created.

The data saved in the database using the node.js server is then extracted by secondary python script running which is monitoring data saved in the database in real time and updates the graphs as soon as the database updates.

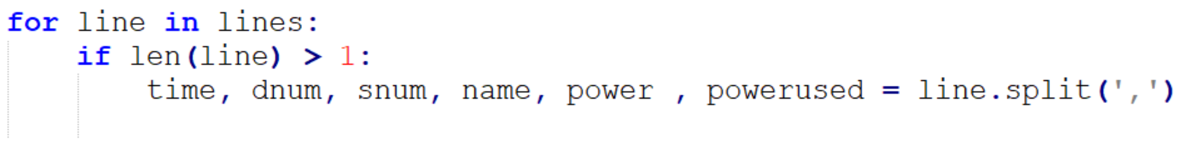
Initially the database is scanned for the current day data and fetched using the date stamps on both the database writer and reader. This feature keeps both database reader and writer on the same part of the database and serves as a synchronization tool.



```python
def animate(i):
    filename = (strftime("%d-%b-%Y",localtime() ) )
```

Figure 14- Code for time sync

While reading the database, every line is processed separately and not as a whole since every line is for one smart plug and has its respective value. The data is read and saved in variables time, ‘*dnum’* is the device number, ‘*snum*’ is the serial number, name is the name given to the device by the user, power is the instantaneous power and ‘*powerused’* is the energy consumed throughout the day till t(the present time).



```python
for line in lines:
    if len(line) > 1:
        time, dnum, snum, name, power , powerused = line.split(',')
```

Figure 15- Data retrieval script

Some of the other features of the graphical unit which are hardcoded are given below:

4.4.1. **Multiple Devices:** If there are more than one device, then the x and the y axis are on different values and they need to be brought on the same data points which is done by the following where it selects the most length of the array most flexible with all the arrays (each array representing a device) to be the dominant one which is used to be the guider for others

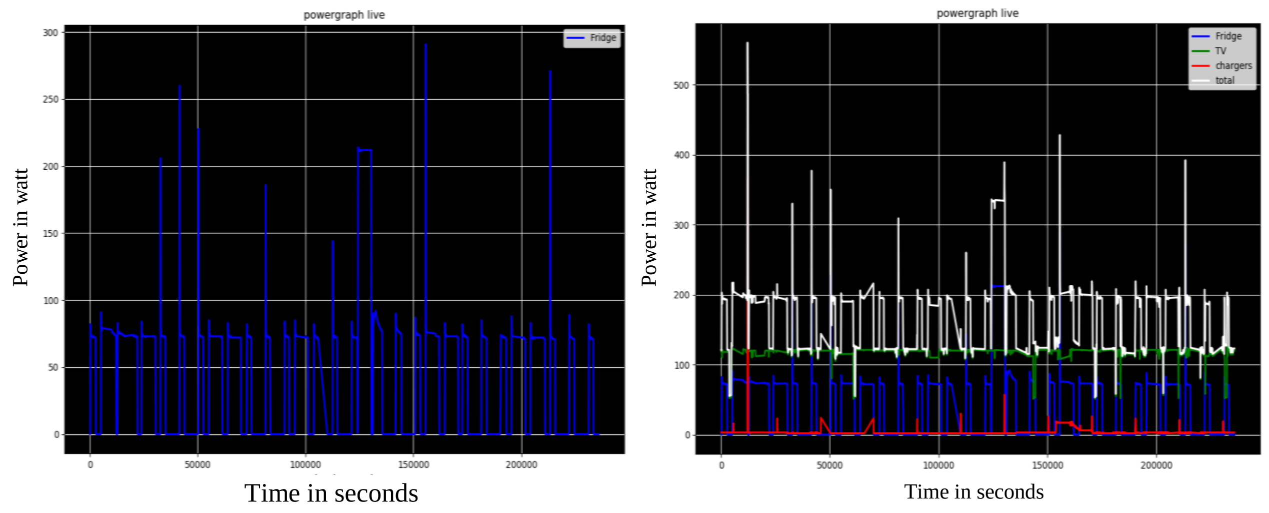
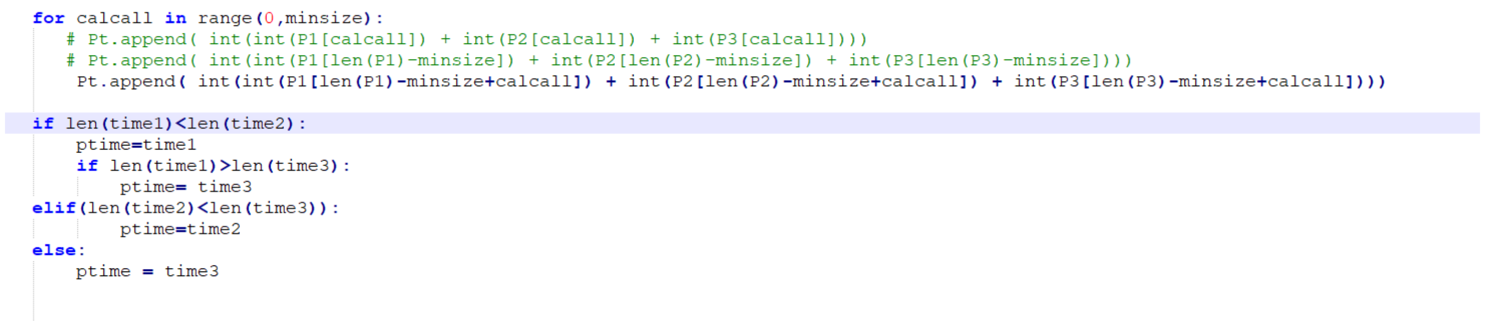


Figure 16- Single device vs Multiple devices

For multiple devices, a white line which represents the total is also added which represents the total amount of power consumed.



```
for calcall in range(0,minsize):
    # Pt.append( int(int(P1[calcall]) + int(P2[calcall]) + int(P3[calcall])))
    # Pt.append( int(int(P1[len(P1)-minsize]) + int(P2[len(P2)-minsize]) + int(P3[len(P3)-minsize])))
     Pt.append( int(int(P1[len(P1)-minsize+calcall]) + int(P2[len(P2)-minsize+calcall]) + int(P3[len(P3)-minsize+calcall])))

if len(time1)<len(time2):
    ptime=time1
    if len(time1)>len(time3):
        ptime= time3
elif(len(time2)<len(time3)):
        ptime=time2
else:
    ptime = time3
```

Figure 17- Axis unifier

4.4.2. **Monitoring Window**: for observation different monitoring window can be selected as mentioned above.

For the case of last T mins of data, a chunk of data from the array is selected to be shown. This reduces the processing power when processing data as well as when showing the graph. Here is shown a picture for a selected window of time.

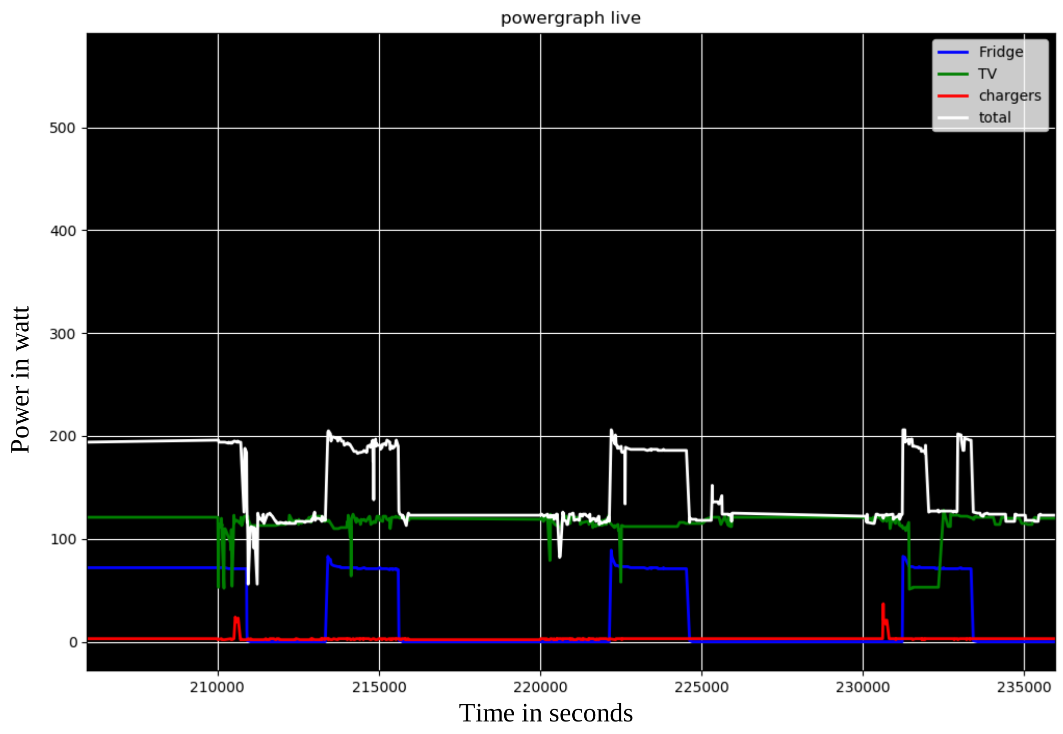
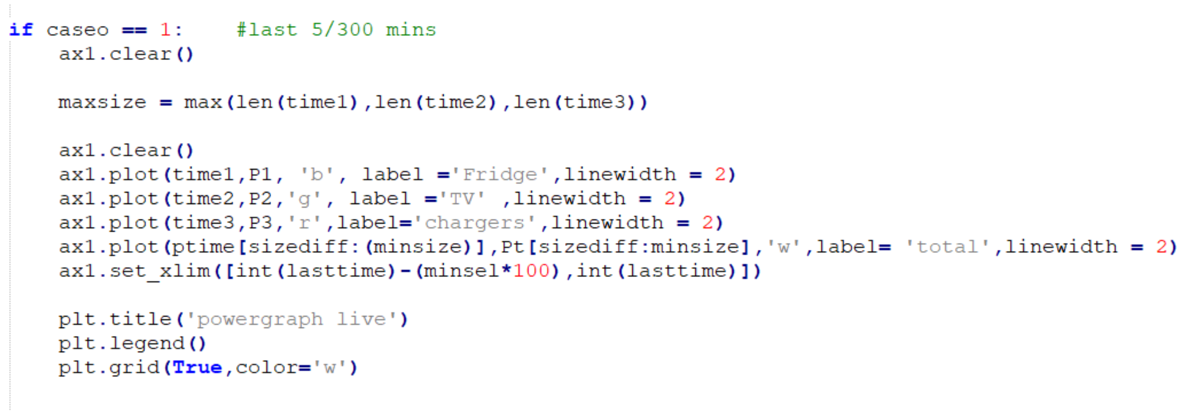


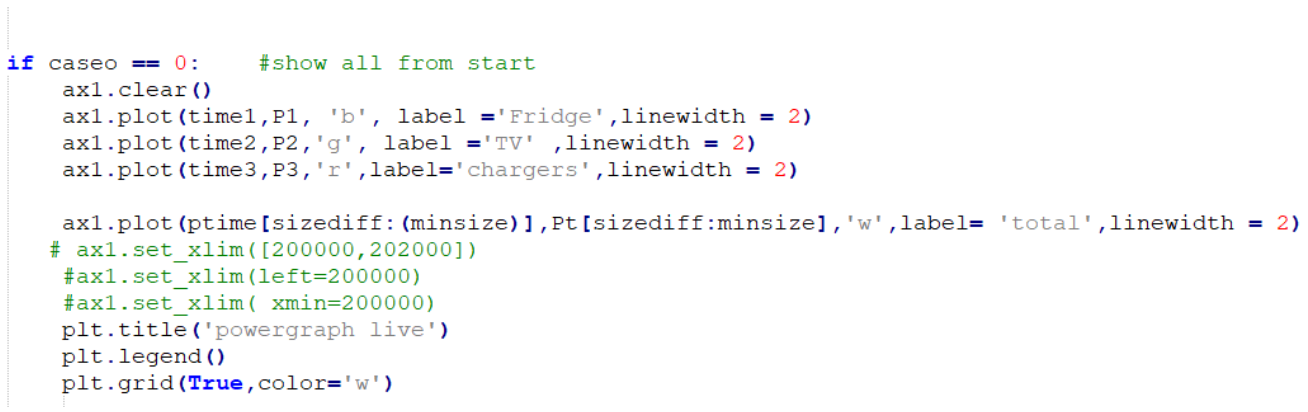
Figure 18- Monitoring window 300 seconds



```
if caseo == 1:     #last 5/300 mins
    ax1.clear()

    maxsize = max(len(time1),len(time2),len(time3))

    ax1.clear()
    ax1.plot(time1,P1, 'b', label ='Fridge',linewidth = 2)
    ax1.plot(time2,P2,'g', label ='TV' ,linewidth = 2)
    ax1.plot(time3,P3,'r',label='chargers',linewidth = 2)
    ax1.plot(ptime[sizediff:(minsize)],Pt[sizediff:minsize],'w',label= 'total',linewidth = 2)
    ax1.set_xlim([int(lasttime)-(minsel*100),int(lasttime)])

    plt.title('powergraph live')
    plt.legend()
    plt.grid(True,color='w')
```

Figure 19- Monitoring window implementation code

When the whole day data is to be shown on the device, case 0 is selected where all the data from the beginning of the data is processed and is played as graph.



```
if caseo == 0:     #show all from start
    ax1.clear()
    ax1.plot(time1,P1, 'b', label ='Fridge',linewidth = 2)
    ax1.plot(time2,P2,'g', label ='TV' ,linewidth = 2)
    ax1.plot(time3,P3,'r',label='chargers',linewidth = 2)

    ax1.plot(ptime[sizediff:(minsize)],Pt[sizediff:minsize],'w',label= 'total',linewidth = 2)
   # ax1.set_xlim([200000,202000])
    #ax1.set_xlim(left=200000)
    #ax1.set_xlim( xmin=200000)
    plt.title('powergraph live')
    plt.legend()
    plt.grid(True,color='w')
```

Figure 20- Whole day monitoring window

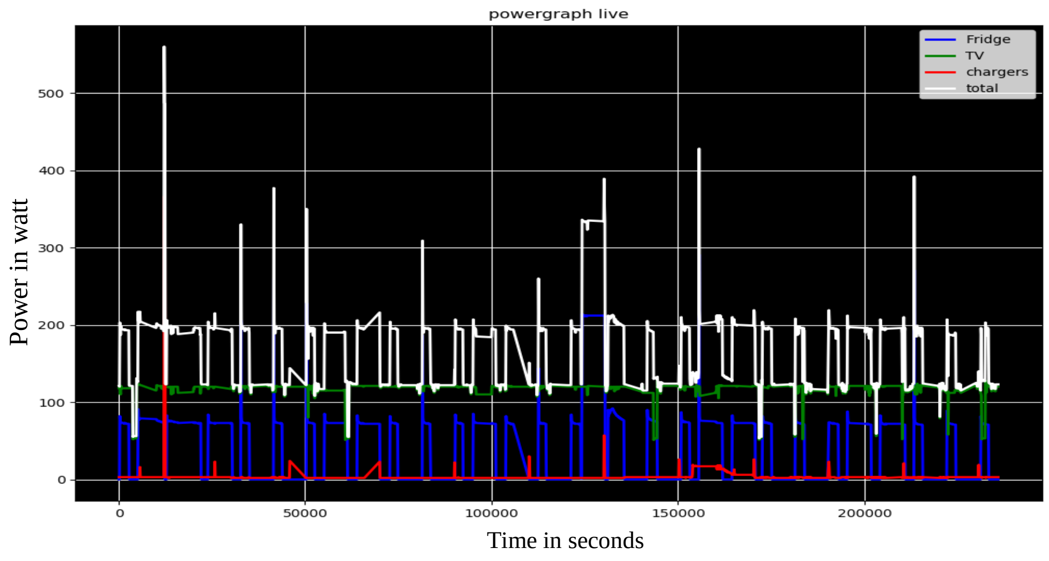


Figure 21- Whole day power graph

4.4.3. Bar Graphs: User is also provided the data in terms of a bar graph. The bar graph includes all the devices detected side by side for the comparison which one

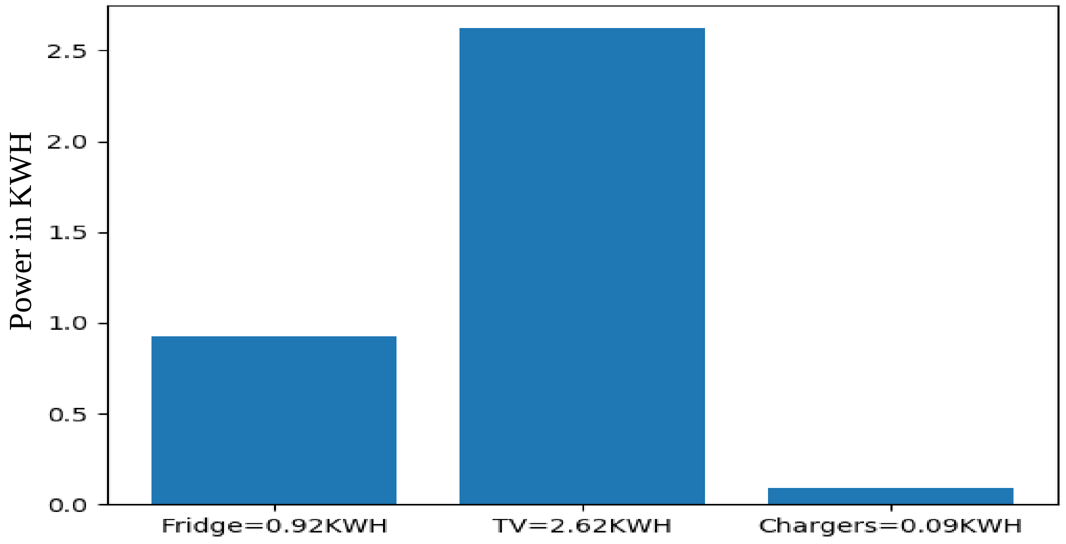


Figure 22- Bar graph representation

4.4.4. Styles And Designs: *Matplotlib [11]* support different styling options which resemble that of CSS styling pattern. Here we can see color change options and background was changed.

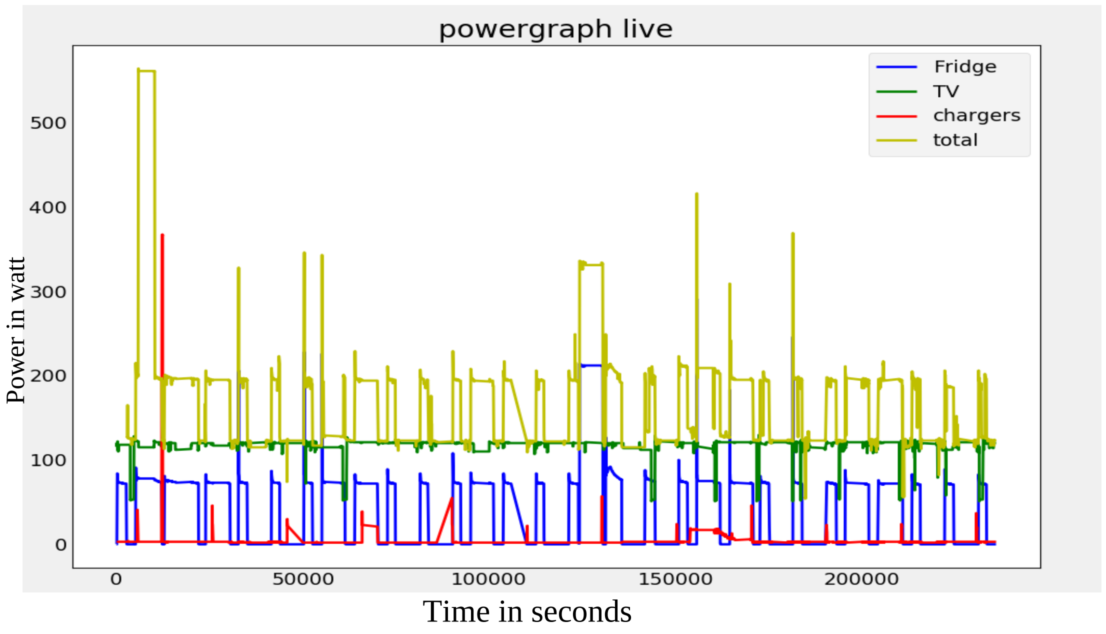


Figure 23- CSS style implementation

4.4.5. Sampling Rate:

Sampling of the data from the database is a very crucial part depending on what accuracy is required of the visual representation. Unlink the bar graph, the simple graphs don't take an average or summation. When taking the data out of database, it selects sampling points after a particular number of dataset. Following is the image of different data rates and its effects.

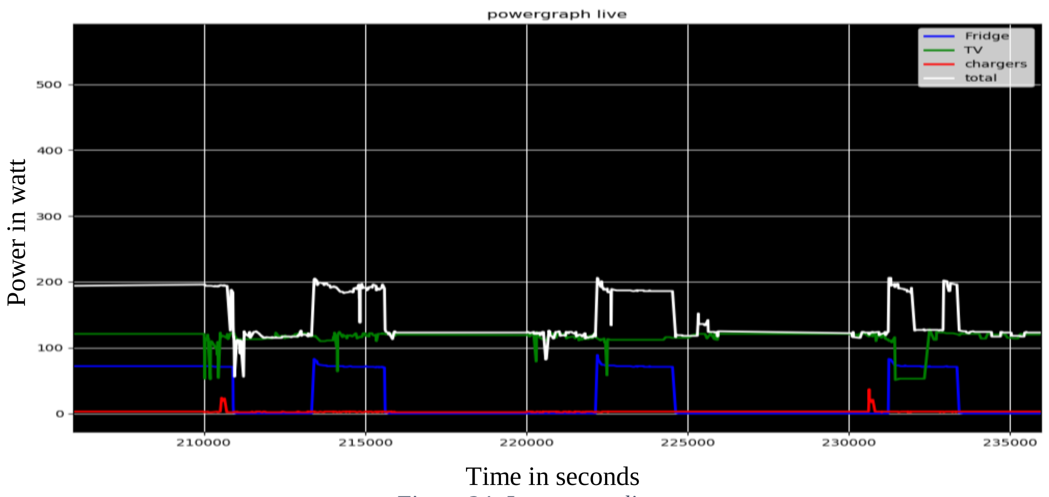


Figure 24- Lower sampling rate

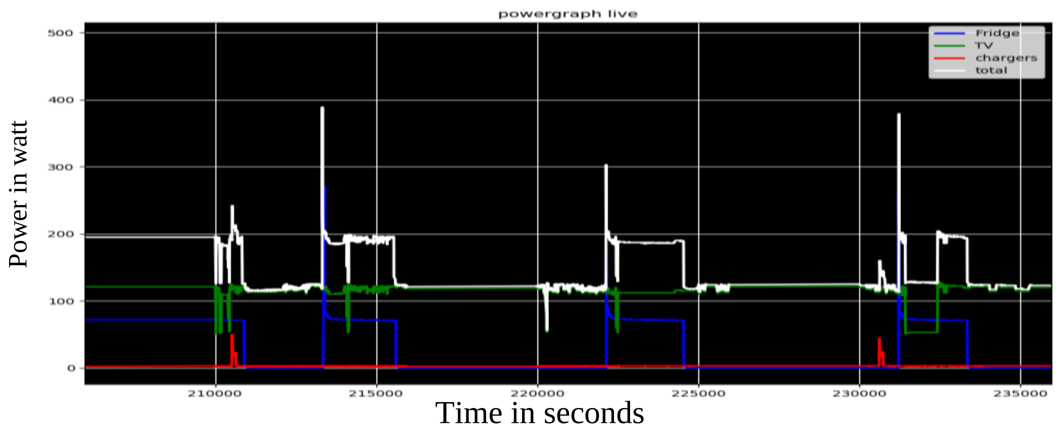


Figure 25- Higher sampling rate

Here we can see the peaks are only visible when the sampling rate is higher, but the time it takes to process higher sampling rates with whole day data points is roughly 10 secs more, which is a lot while running in real-time.

Chapter 5. System Testing

For system response testing purposes, data was collected in different conditions. Most common monitoring windows chosen were 300 seconds/30 mins/3 hours/whole day of energy consumption by 3 different devices attached to the smart plug real time. the devices attached to the smart plug device ID are given by:

1- Device 0 – Fridge.
2- Device 1 – Television.
3- Device 2 – cellphone chargers and hair dryer

For long term testing, the server and the graphing unit has run for a couple of days without and errors besides the ones in which the raspberry pi froze due to some technical issue not connected to the node server or the graphical interface.

During the experimentation window, many different scenarios were chosen in which the graph was being observed for long and then a device was either turned on or off to see the change in the real time graph which was reflected on the graphical interface as well.

5.1. System Response

A scenario where a high-power hairdryer was turned on, a peak representing the energy consumed by the hairdryer can be seen in the graph in figure 23.

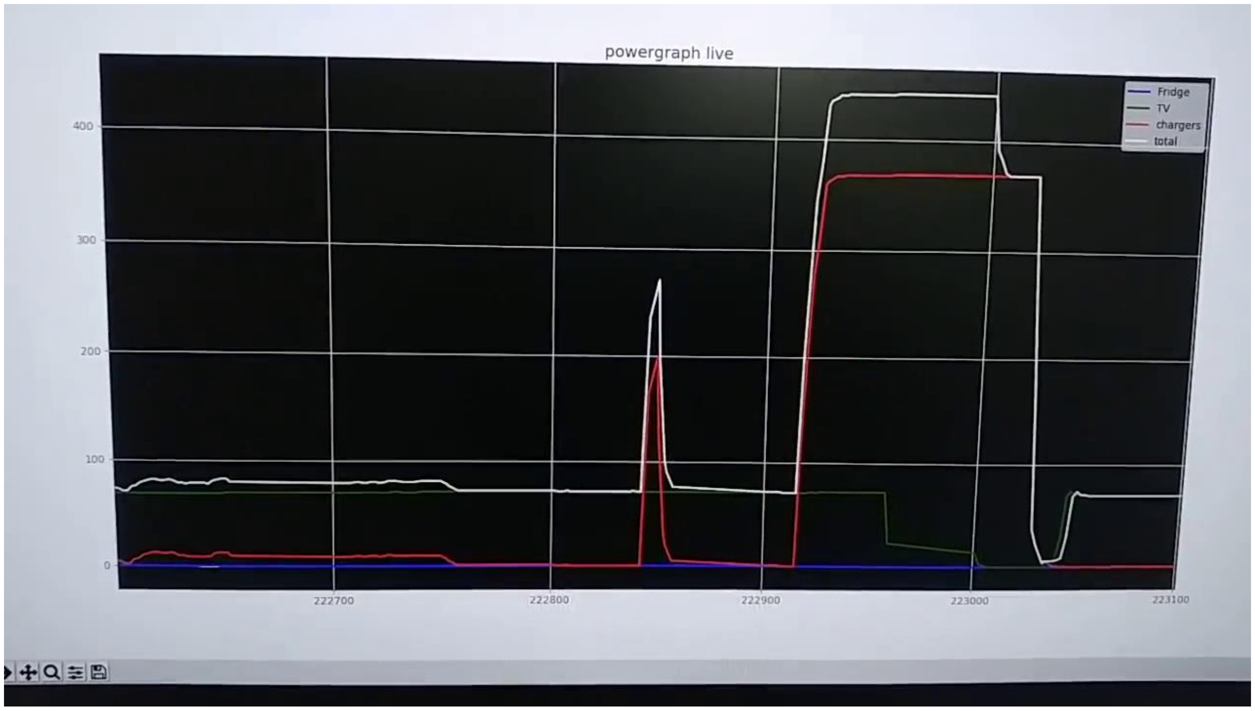


Figure 26- Before device power-on

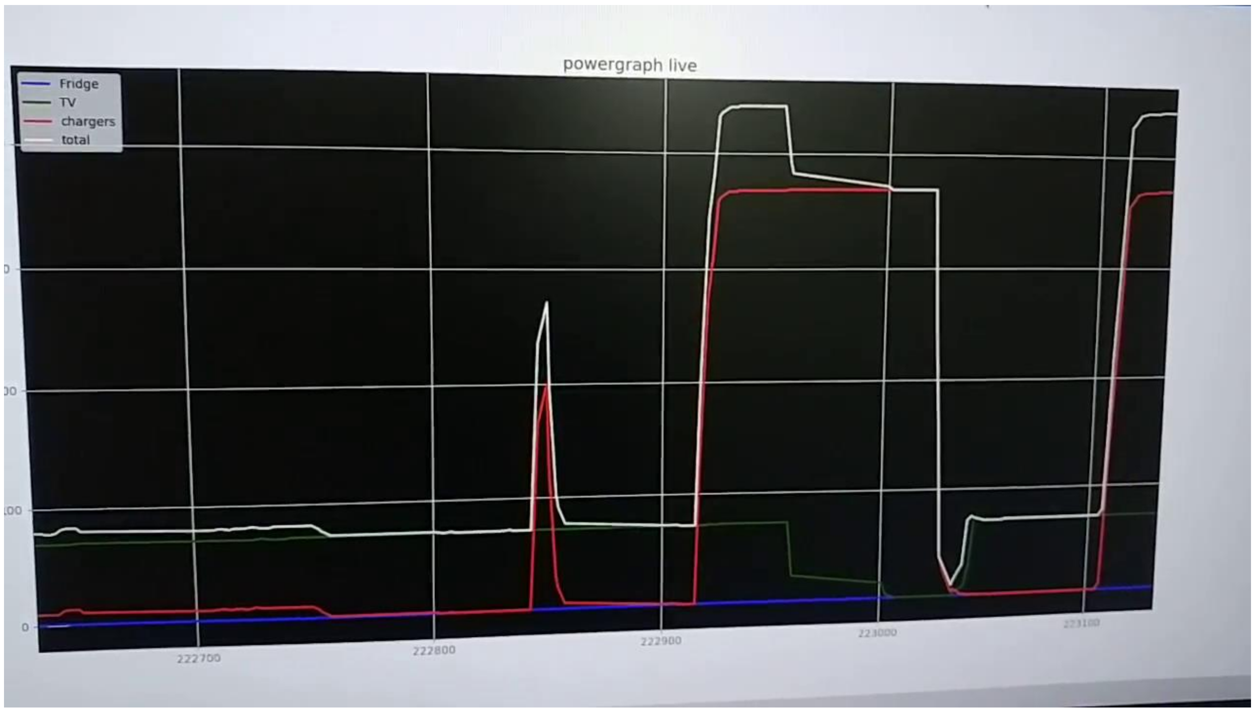


Figure 27- After device power on

5.2. Sampling Rate Response

Different sampling rates options are provided. The different sampling rates that can be chosen from in the code are 1-10 second, 1 min and 5 mins. The data sampling rate has a direct effect on the processing power, but lower sampling rate results in some data to be lost which can result in inaccuracy if the user is looking for power peaks

More data points mean more time taken to process the data. since we are using a raspberry pi 3, it takes substantial amount of time to create the graph, for that particular device, it was recommended to use lower sampling rate.

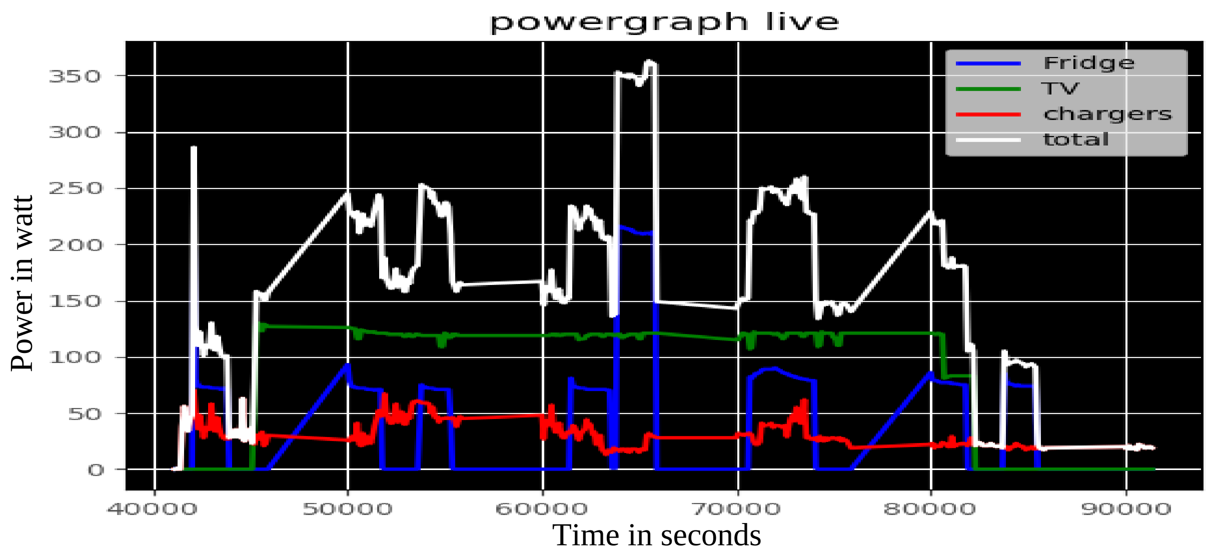


Figure 28- 1 second sampling

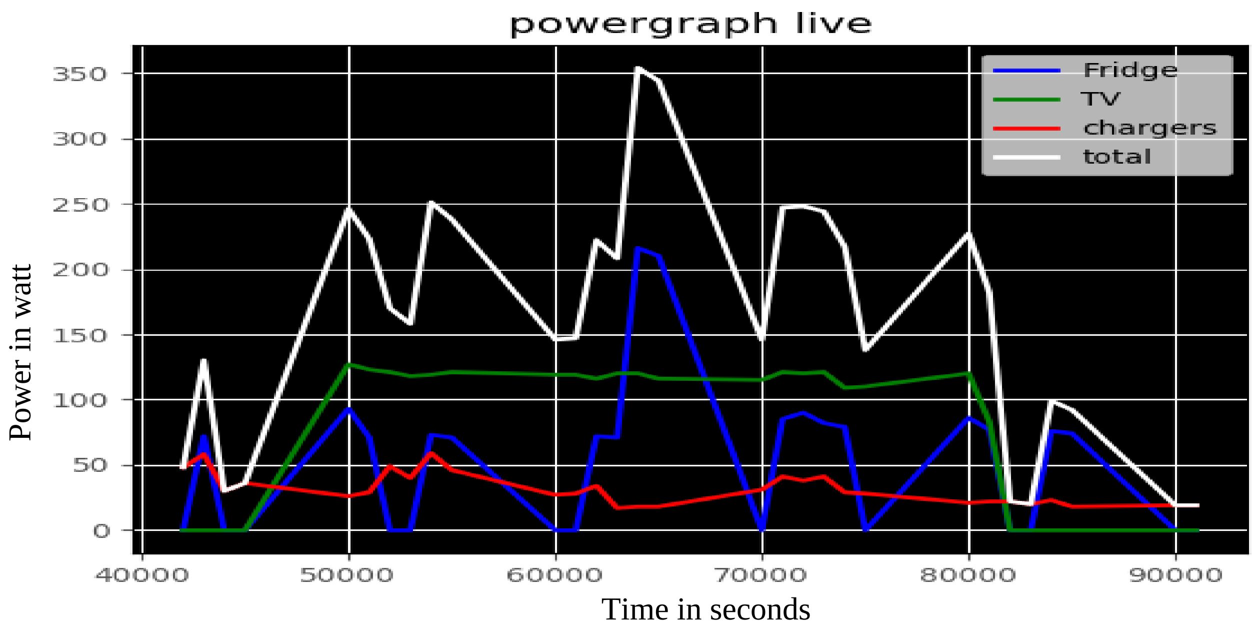


Figure 29- 10 second sampling

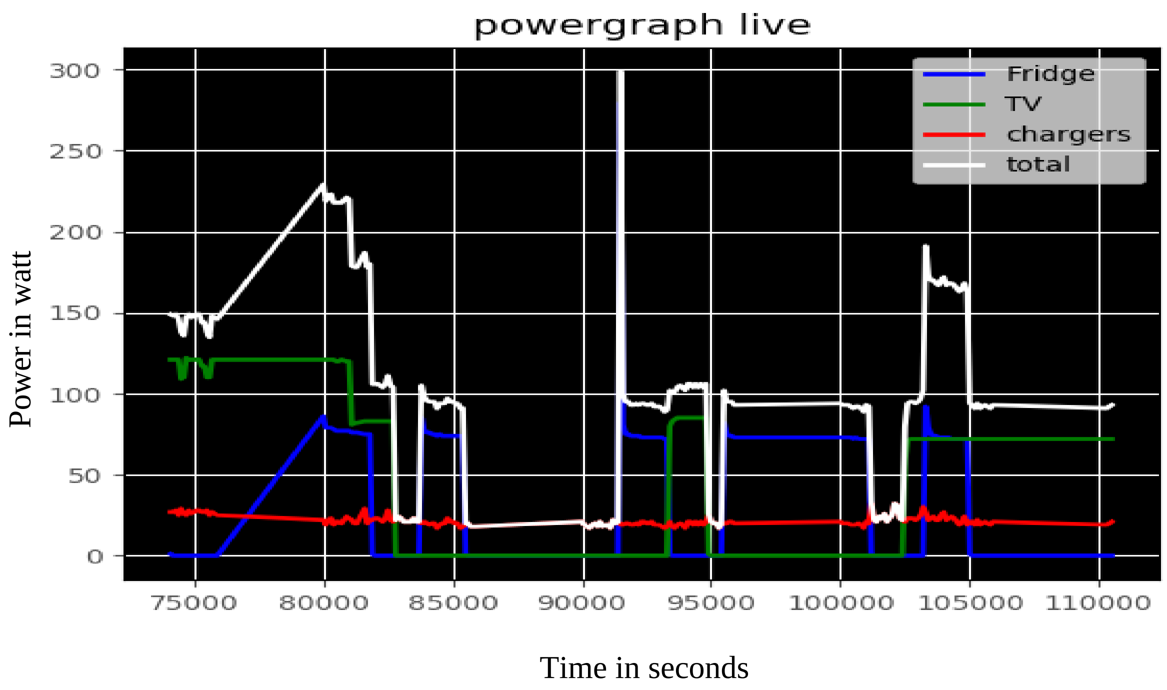


Figure 30- 1 second sampling rate

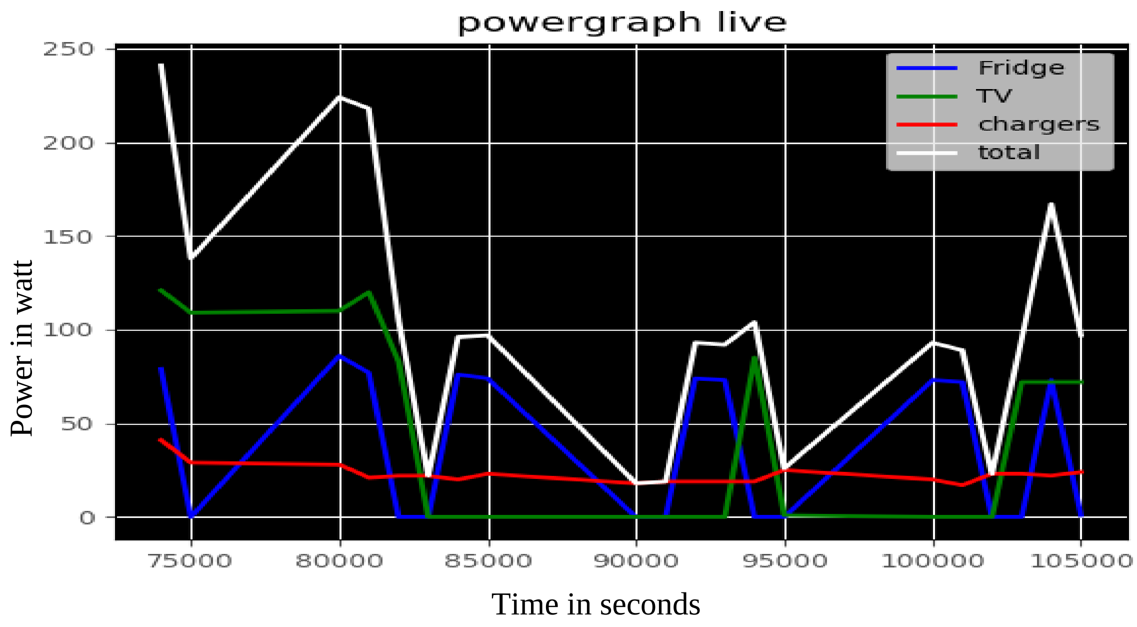


Figure 31- 10 second sampling rate

5.3. Bar Graph Real-Time Response

The bar graph collects data from the database, shows the power consumed by each device throughout the day. The graphs and the labels are real time and the label changes along with the increment in the length of the bars as soon as the more power is consumed by the respective device.

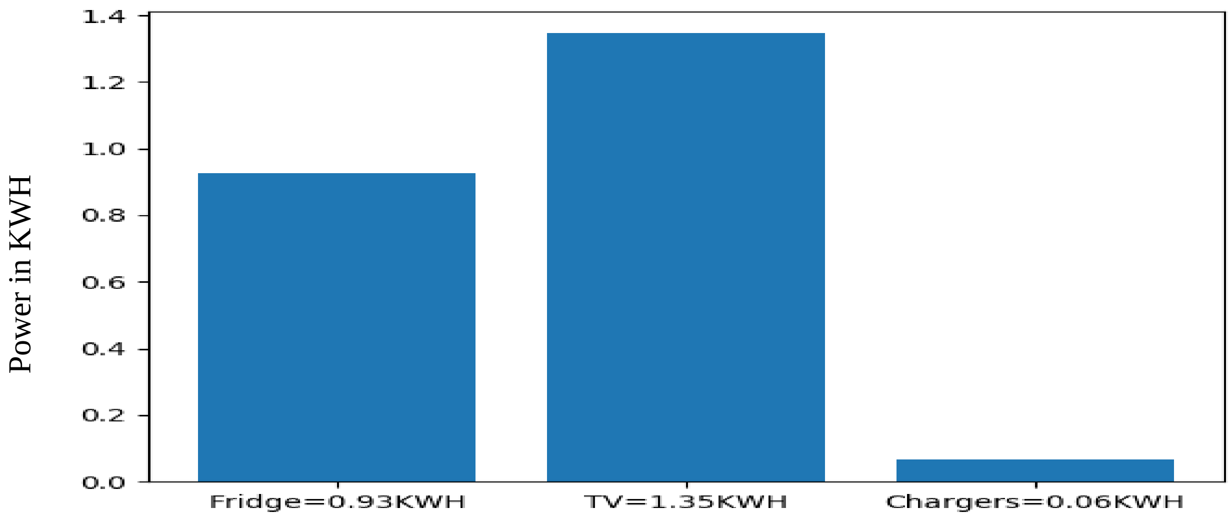


Figure 32- Bar graph of the energy consumed

Chapter 6. Future Work

Future consideration

1. Inclusion of more products other than WeMo smart plugs can be added to the system to remove vender dependency.
2. Feature enhancement: more features can be added in which not only the data is read but also commands to shut down the switch or timers can be added.
3. A better user interface can be added instead of the hard-coded options to give the user more power and flexibility.
4. Household device recognition algorithm can be implemented by making use of device signature matching to detect which appliance is plugged in automatically, which will remove the dependency of user labeling the appliances.

Chapter 7. Conclusions

Home automation and smart home system is every growing. This industry has just started to tap into its potential and has miles and miles to go. Venders locking their protocol and not making things open source is a big hurdle in fast evolution of technology.

If all the vendors would work together and let others also openly communicate with their devices, a great infrastructure can be built from which we all can benefit.

The implementation of the system for WeMo Belkin insight switch up to 3 devices was successfully loaded in the system and the system was able to observe and maintain the real time server running for data logging without any errors. If other vendors also had the capability of their devices able to communicate to their server, more devices can be added.

References

[1] Sanaulla, Mohamed. “Node.js : Reading and Writing File in Node.js.” JavaBeat, 1 July 2015, www.javabeat.net/nodejs-read-write-file/

[2] “About Node.js.” Node.js, www.nodejs.org/en/about/

[3] Burnett, Kristine K. “Node.js – Kristine K Burnett – Medium.” Medium, Medium, 21 Sept. 2016, medium.com/@kristinekburnett/node-js-4fda4c1b0462.

[4] Hardill, Ben. “JavaScript client library for controlling and subscribing to WeMo devices” https://github.com/timonreinhard/wemo-client

[5] Hardill, Ben. “Wemo Client for Node.js” , npm @octoblu/wemo-client , https://www.npmjs.com/package/@octoblu/wemo-client

[6] “What Is Python? Executive Summary.” Python.org, www.python.org/doc/essays/blurb/.

[7] “Topic: Python.” DevHub.io, www.devhub.io/topic/python

[8] Kuhlman, Dave. "A Python Book: Beginning Python, Advanced Python, and Python Exercises", 23 June 2012.

[9] Python Programming Tutorials, www.pythonprogramming.net/natural-language-api-google-cloud-tutorial/

[10] “Python Programming Language - An Overview.” | Vizteams, 31 Aug. 2015, www.vizteams.com/blog/python-programming-language-an-overview/ x

[11] Hunter et al (2007) Matplotlib: A 2D graphics environment, Computing In Science & Engineering , Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems” , IEEE COMPUTER SOC, 2007 , https://matplotlib.org/

[12] Python Software Foundation https://docs.python.org/2/library/datetime.html

[13] Animation API for Matplotlib, A 2D graphics environment, Computing In Science & Engineering , Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems” , https://matplotlib.org/api/animation_api.html

[14] Styling API for Matplotlib, , A 2D graphics environment, Computing In Science & Engineering , Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems” , https://matplotlib.org/api/style_api.html

Appendix A

Python code for graphical representation:

```
import matplotlib
#matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
import matplotlib.dates as mdates
#from matplotlib.backends import *
#from pyplot import *
#import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib import style
from time import localtime , strftime



fig = plt.figure(1)

ax1 = fig.add_subplot(1,1,1)
#fig.patch.set_facebolor('black')
#ax1.plt.gca()
#

#style.use('fivethirtyeight')
#style.use('ggplot')
ax1 = plt.gca()
ax1.set_facecolor('black')

cbar=1
totP1 =0
totP2 =0
totP3 =0

def animate(i):
   # filename = (strftime("%d-%b-%Y",localtime() ) )
   # filename = ("25:2:2018.txt")
   # filename = ("2_3_2018.txt")
    filename = ("maybe.txt")
    graph_data = open(filename,'r').read()
    lines = graph_data.split('\n')
    P1 = []
    P2 = []
    P3 = []
    Pt = []
    time1 = []
    time2 = []
    time3 = []
    timet = []
    snum = []
    lttime = int(0)
    lttime1 = int(0)
    lttime2 = int(0)
    lttime3 = int(0)

    lasttime = 0 # value to give to the time axes so that we can set a maximum

    b1 = 1    #get rid of seconds when sampling - higher the number, lower the samples
    b2 = 1     # min 10 parse by ten b1s (b1 will give us either seconds or minutes or whatever)
    # the sampling will be the miltuolle pf both, b1*b2 so here, b1 makes it minutes and then when its b2 is
ten minutes

    minsel = 300

    #pastT =int(minsel*1.667) #past mins   #0 for unlimited , 1 for 5 mins, 2 for 30 mins , 3 for 300 mins
    pastT = int(minsel*120/((b1)*b2))  #number of samples to create minsel
    #the above formula holds for b1=1, b2 = 1
    caseo = 4


    for line in lines:
       if len(line) > 1:
          time, dnum, snum, name, power , powerused = line.split(',')

          if(        ((int(int(time)/b1))%b2)==0 ):

            # timet.append(time)

             #fridge
             if ((snum == "231625K1200B2C") &  (int(int(time)/b1) > int(int(lttime1)/b1) ) )    :
                         P1.append(int(power))
                         time1.append(int(time))
                         lttime1 = time
                         totP1 = powerused
                         lasttime = time



             #tv
             if ( (snum == "231614K120073E") &  (int(int(time)/b1) > int(int(lttime2)/b1) ) )   :
                         P2.append(int(power))
                         time2.append(int(time))
                         lttime2 = time
                         totP2 = powerused
                         lasttime = time

                         #print(power)
```

```
                #chargers
                if ((snum == "231701K12003E6") &  (int(int(time)/b1) > int(int(lttime3)/b1) ) )  :
                        P3.append(int(power))
                        lttime3 = time
                        time3.append(int(time))
                        totP3 = powerused
                        lasttime = time

            lttime = time

    #print(pastT)
   # print(time2[len(time2)-2],P2[len(P2)-2])
   # print(len(time1),len(time2),len(time3),len(timet))

  #  print(totP1)

  #  print(int(totP1)/(10000*60),int(totP2)/(10000*60),int(totP3)/(10000*60))


    #print(P1)

    #for calcall in range(0,len(P1)):
     #   Pt.append( int(int(P1[calcall]) + int(P2[calcall]) + int(P3[calcall])))
      #  print(time1[calcall],calcall)
    minsize = min(len(time1),len(time2),len(time3))
    maxsize = max(len(time1),len(time2),len(time3))
    sizediff = maxsize-minsize

    for calcall in range(0,minsize):
       # Pt.append( int(int(P1[calcall]) + int(P2[calcall]) + int(P3[calcall])))
       # Pt.append( int(int(P1[len(P1)-minsize]) + int(P2[len(P2)-minsize]) + int(P3[len(P3)-minsize])))
        Pt.append( int(int(P1[len(P1)-minsize+calcall]) + int(P2[len(P2)-minsize+calcall]) + int(P3[len(P3)-
minsize+calcall])))

    if len(time1)<len(time2):
        ptime=time1
        if len(time1)>len(time3):
            ptime= time3
    elif(len(time2)<len(time3)):
            ptime=time2
    else:
        ptime = time3




    print(len(P1),len(P2),len(P3))
    print(len(Pt),minsize,sizediff,maxsize)

        #ax1.plot(time3[((minsize)-int(pastT)):(minsize)],Pt[(minsize-int(pastT)):minsize],'w',label=
'total',linewidth = 2)
```

```
    if caseo == 0:    #show all from start
        ax1.clear()
        ax1.plot(time1,P1, 'b', label ='Fridge',linewidth = 2)
        ax1.plot(time2,P2,'g', label ='TV' ,linewidth = 2)
        ax1.plot(time3,P3,'r',label='chargers',linewidth = 2)

        ax1.plot(ptime[sizediff:(minsize)],Pt[sizediff:minsize],'w',label= 'total',linewidth = 2)
       # ax1.set_xlim([200000,202000])
        #ax1.set_xlim(left=200000)
        #ax1.set_xlim( xmin=200000)
        plt.title('powergraph live')
        plt.legend()
        plt.grid(True,color='w')

     #   ax1.plot(time3,Pt,'w',label='total',linewidth = 2)

    if caseo == 1:    #last 5/300 mins
        ax1.clear()

        maxsize = max(len(time1),len(time2),len(time3))

        ax1.clear()
        ax1.plot(time1,P1, 'b', label ='Fridge',linewidth = 2)
        ax1.plot(time2,P2,'g', label ='TV' ,linewidth = 2)
        ax1.plot(time3,P3,'r',label='chargers',linewidth = 2)
        ax1.plot(ptime[sizediff:(minsize)],Pt[sizediff:minsize],'w',label= 'total',linewidth = 2)
        ax1.set_xlim([int(lasttime)-(minsel*100),int(lasttime)])

        plt.title('powergraph live')
        plt.legend()
        plt.grid(True,color='w')


    if caseo == 2:    #good for nothing
        ax1.clear()
        #                              ax1.plot(time1[(len(time1)-int(pastT*100/(b2*b1))):len(time1)],P1[(len(time1)-
  int(pastT*100/(b2*b1))):len(time1)],'b',label ='Fridge',linewidth = 2)
        #                              ax1.plot(time1[(len(time1)-int(pastT*100/(b2*b1))):len(time1)],P2[(len(time1)-
  int(pastT*100/(b2*b1))):len(time1)],'g',label ='TV' ,linewidth = 2)
        #                              ax1.plot(time2[(len(time1)-int(pastT*100/(b2*b1))):len(time1)],P3[(len(time1)-
  int(pastT*100/(b2*b1))):len(time1)],'r',label= 'chargers',linewidth = 2)
        #                               ax1.plot(time3[(len(time1)-int(pastT*100/(b2*b1))):len(time1)],Pt[(len(time1)-
  int(pastT*100/(b2*b1))):len(time1)],'w',label= 'total',linewidth = 2)

        maxsize = max(len(time1),len(time2),len(time3))
        #print(power)

        #still select the data range that falls in the one under so that you can make it less heavy
        ax1.clear()
        ax1.plot(time1,P1, 'b', label ='Fridge',linewidth = 2)
        ax1.plot(time2,P2,'g', label ='TV' ,linewidth = 2)
        ax1.plot(time3,P3,'r',label='chargers',linewidth = 2)
        ax1.plot(ptime[sizediff:(minsize)],Pt[sizediff:minsize],'w',label= 'total',linewidth = 2)
        ax1.set_xlim([int(lasttime)-(minsel*1000),int(lasttime)])


    if caseo ==4:    #bar graph
        ax1.clear()
        #fig = plt.figure(2)
       # ax2 = fig2.add_subplot(1,1,
        x=[1,2,3]
        #print('eh')
        totP1 = int(totP1)/(1000000*60)
       # print(totP1)
        totP2= int(totP2)/(1000000*60)
        totP3 = int(totP3)/(1000000*60)
        y = [(totP1),(totP2),(totP3)]
       # y = [100,200,300]
        #ax1.bar(x,y,align='center')
        pm,pc,pn = ax1.bar(x,y,align='center')


plt.xticks(x,('Fridge='+str(round(totP1,2))+'KWH','TV='+str(round(totP2,2))+'KWH','Chargers='+str(roun
d(totP3,2))+'KWH'))
        ax1.set_facecolor('white')



    if caseo == 10:    #thresholdiung but old style
        ax1.clear()

        minsize = min(len(time1),len(time2),len(time3))

       #     ax1.plot(time1[(len(time1)-int(pastT)):len(time1)],P1[(len(time1)-int(pastT)):len(time1)],'b',label
='Fridge',linewidth = 2)
       #     ax1.plot(time1[(len(time2)-int(pastT)):len(time2)],P2[(len(time2)-int(pastT)):len(time2)],'g',label
='TV' ,linewidth = 2)
       #    ax1.plot(time3[(len(time3)-int(pastT)):len(time3)],P3[(len(time3)-int(pastT)):len(time3)],'r',label=
'chargers',linewidth = 2)
##                                              ax1.plot(time1[(len(time1)-int(pastT/b2)):len(time1)],Pt[(len(time1)-
int(pastT/b2)):len(time1)],'w',label= 'total',linewidth = 2)

        ax1.plot(time3[((minsize)-int(pastT)):(minsize)],P1[(minsize-int(pastT)):minsize],'b',label
='Fridge',linewidth = 2)
        ax1.plot(time3[((minsize)-int(pastT)):(minsize)],P2[(minsize-int(pastT)):minsize],'g',label         ='TV'
,linewidth = 2)
        ax1.plot(time3[((minsize)-int(pastT)):(minsize)],P3[(minsize-int(pastT)):minsize],'r',label=
'chargers',linewidth = 2)
```

```
      #      ax1.plot(minsize[((minsize)-int(pastT)):(minsize)],Pt[(minsize-int(pastT/b2)):minsize],'w',label=
'total',linewidth = 2)

       #print(maxsize)

##
##       for calcall in range(minsize-int(pastT),len(time1)):
##           Pt.append( int(P1[calcall]) )
##
##       for calcall in range(minsize-int(pastT),len(time2)):
##           Pt.append( Pt[calcall]  + int(P2[calcall]))
##
##       for calcall in range(minsize-int(pastT),len(time3)):
##           Pt.append( Pt[calcall]  + int(P3[calcall]))
##

       for calcall in range(0,minsize):
           Pt.append( int(int(P1[calcall]) + int(P2[calcall]) + int(P3[calcall])))

       ax1.plot(time3[((minsize)-int(pastT)):(minsize)],Pt[(minsize-int(pastT)):minsize],'w',label=
'total',linewidth = 2)




       #= P1[calcall]+P1[calcall]
     #   Pt[calcall] = P1[calcall] + P2[calcall] + P3[calcall]

    #print(len(time1),len(time2),len(time3),len(timet))
   # print(len(P1),len(P2),len(P3),len(Pt))
  #  print(Pt[0:len(P1)])



   # ax1.plot(time1,P2[len(P2)-len(time1):len(P2)])
    #ax1.plot(time1,P3[len(P3)-len(time1):len(P3)])


   # ax1.plot(time1,P1)

    #ax1.plot(time1[(len(time1)-25):len(time1)],P1[(len(time1)-25):len(time1)]);
  #  ax1.plot(time2[(len(time2)-50):len(time2)],P2[(len(time2)-50):len(time2)]);
    #ax1.plot(time3[(len(time3)-500):len(time3)],P3[(len(time3)-500):len(time3)]);

##
##    ax3.clear()
##    ax3.plot(time3,P3)
    #ax3.clear()
    #ax3.plot(time3,P3)
    #axes.set_xlim([20000,80000])
```

```
ani = animation.FuncAnimation(fig, animate, interval=1000)
#print("hi")
##def animate(j):
##
##    ax2.clear()
##    #ax2.plot(time2,P2)
##    ax2.plot(time2[(len(time2)-50):len(time2)],P2[(len(time2)-50):len(time2)]);
##
##
##ani2 = animation.FuncAnimation(fig2, animate, interval=1000)
##
#def animate(j):




##    ax3.clear()
##    #ax2.plot(time2,P2)
##    ax3.plot(time3[(len(time3)-50):len(time3)],P3[(len(time3)-50):len(time2)]);
##
##ani3 = animation.FuncAnimation(fig3, animate, interval=1000)
#ani = animation.FuncAnimation(fig2, animate, interval=1000)
#ani = animation.FuncAnimation(fig3, animate, interval=1000)

#plt.legend()
#plt.grid(True,color='k')


plt.show()
```

JavaScript code for server

node.js code

```
// properly implemented
//1) data taken out from node
//date and time has been split, date is name, time is t
var Wemo = require('wemo-client');
var wemo = new Wemo();
var fs = require('fs');
var array = [];



function foundDevice(err, device) {
  if (device.deviceType === Wemo.DEVICE_TYPE.Insight) {
    console.log('Wemo Insight Switch found: %s', device.friendlyName);

    var client = this.client(device);
    client.on('insightParams', function(state, power,powerused) {

            var now = new Date();
            var dateFormat = require('dateformat');

          // t = dateFormat(now,"dd:m:yyyy H:MM:ss");
            t = dateFormat(now,"HMMss");

           var f = dateFormat(now,"dd:m:yyyy");

               if (array.includes(this.device.serialNumber))
                          {
                            //console.log('HAHAHAHAHAHAHAHAH');
                       //  console.log('%s %d',this.device.friendlyName);
```

```
                                        for(i=0;i<10;i++)
                                         {
                                            if(array[i] == (this.device.serialNumber))
                                            {
                                            console.log('name = %s, device numbeer = %d , power = %s,
powerused = %s',
                                             this.device.friendlyName,
                                              //  this.device.serialNumber,
                                               i,
                                               Math.round(power / 1000),
                                               powerused);

                                             //insert append to file



                                          //fs.appendFile( 'somethignnew.txt', 'name = %s, device numbeer =
%d , power = %s',
                                            //fs.appendFile(  'somethignnew.txt'  ,  t  +  '\t'+   i  +  '\t'  +
this.device.serialNumber + '\t' +  this.device.friendlyName + '\t' + Math.round(power / 1000) + '\n',
                                            //fs.appendFile( f+'.txt' , t + '\t'+  i + '\t' + this.device.serialNumber
+ '\t' +  this.device.friendlyName + '\t' + Math.round(power / 1000) + '\n',
                                            //as requested by the new system
                                            fs.appendFile( f+'.txt' , t + ','+  i + ',' + this.device.serialNumber + ','
+  this.device.friendlyName + ',' + Math.round(power / 1000) +  ',' +  powerused +'\n' ,
                                              //'\n power = %s ', num.toString(i) ,

                                          //this.device.friendlyName, i , Math.round(power / 1000),



                                             function (err) {
                                                       if (err) throw err;
                                        console.log('Updated!');
                                      });

                                      break;
                                      }
                                   }
                              }



              if (!array.includes(this.device.serialNumber))
                              {
                        //      console.log(array.includes(this.device.serialNumber));
                            array.push(this.device.serialNumber);
                           }
//console.log(array);




         // console.log('%s %s %d',this.device.friendlyName,this.device.serialNumber) ;

    });
  }
}

wemo.discover(foundDevice);
```