NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MANAGEMENT OF LARGE-SCALE IOT (INTERNET OF THINGS) NETWORKS

by

Chun Heong Chia

March 2018

Thesis Advisor: Gurminder Singh
Second Reader: John H. Gibson

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

<table>
<tr><td colspan="3" align="center">REPORT DOCUMENTATION PAGE</td><td align="center"><i>Form Approved OMB No.0704–0188</i></td></tr>
</table>

REPORT DOCUMENTATION PAGE			*Form Approved OMB* *No.0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE MANAGEMENT OF LARGE-SCALE IOT (INTERNET OF THINGS) NETWORKS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Chun Heong Chia			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number _____N/A_____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Internet of Things (IoT) networks are often large networks built to maximize the benefits of distributed computing. Management of these networks poses many challenges. IoT devices are often connected wirelessly and are battery powered. In this type of a network, it is important for a network administrator to know which devices may be running low on battery power. Monitoring the battery life of these devices can be difficult and time consuming but is critical to the operation of the network.

This research develops an autonomous Energy Monitoring System that triggers alerts from the battery to the server when battery life reaches a critical level. This enables the network administrator to plan and take appropriate action on a timely basis. The system also enables the network administrator to interrogate an individual device in the network for its energy status. In this thesis, we describe the system design and implementation using a network of Raspberry Pi computers powered by batteries, and report the results of tests conducted to evaluate the design and implementation.

14. SUBJECT TERMS Internet of Things (IoT), large-scale, power constrained networks			**15. NUMBER OF PAGES** 129
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

MANAGEMENT OF LARGE-SCALE IOT (INTERNET OF THINGS) NETWORKS

Chun Heong Chia
Technology Research Officer, Ministry of Defence, Singapore
B.S., Singapore Institute of Management, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2018

Approved by: Gurminder Singh
Thesis Advisor

John H. Gibson
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Internet of Things (IoT) networks are often large networks built to maximize the benefits of distributed computing. Management of these networks poses many challenges. IoT devices are often connected wirelessly and are battery powered. In this type of a network, it is important for a network administrator to know which devices may be running low on battery power. Monitoring the battery life of these devices can be difficult and time consuming but is critical to the operation of the network.

This research develops an autonomous Energy Monitoring System that triggers alerts from the battery to the server when battery life reaches a critical level. This enables the network administrator to plan and take appropriate action on a timely basis. The system also enables the network administrator to interrogate an individual device in the network for its energy status. In this thesis, we describe the system design and implementation using a network of Raspberry Pi computers powered by batteries, and report the results of tests conducted to evaluate the design and implementation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

APM	Auto-Probe Module
BAV	Basic Analytics View
CCM	Client Communication Module
DDV	Detailed Device View
DIY	Do-It-Yourself
DTN	Delay-Tolerant Network
DUT	Device Under Test
DVS	Dynamic Voltage Scaling
EMS	Energy Monitoring System
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
M2M	Machine-to-Machine
MANET	Mobile Ad-hoc Network
MQTT	Message Queue Telemetry Transport
OLED	Organic Light Emitting Diode
OS	Operating System
QDV	Quick Device View
SCM	Server Communication Module
SMBus	System Management Bus
SoC	State of Charge
SoH	State of Health
TTL	Time-To-Live
USB	Universal Serial Bus
WSN	Wireless Sensor Network

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis advisor, Dr. Gurminder Singh, and second reader, Mr. John H. Gibson, for providing me with the opportunity to work under them during the project tenure. Their guidance and domain expertise have helped me to gain a profound understanding on my research question.

I also would like to thank my loved ones and my friends for their constant encouragement and support along the way that helped me complete my research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The Internet of Things (IoT) is designed to allow small, low-powered, Internet-connected devices, not just computers, to communicate with each other and sense real-time data on a large scale. While IoT has enabled new capabilities in the computing world, it has introduced challenges in managing a large number of devices connected on the network. In this chapter, we discuss issues of management of large-scale IoT networks and the research objectives for this thesis.

A. CHALLENGES OF MANAGING LARGE-SCALE IOT NETWORKS

The number of IoT connected devices is increasing exponentially. According to [1], 34.8 billion connected "things" will be deployed worldwide in 2018 (more than the world's population), which represents an increase of 22 percent from 2017. This number is estimated to reach 50.1 billion before the year 2020. (See Figure 1.)
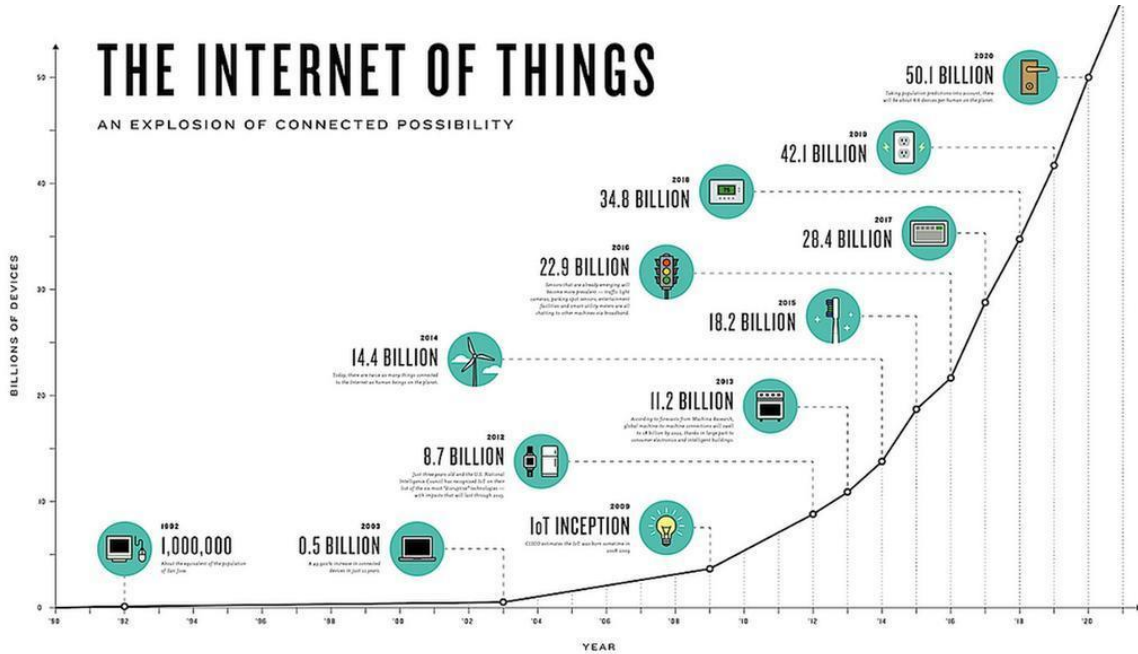
Figure 1.   Estimated Number of IoT Units by Year 2020. Source: [1].

With so many connected devices, administrators face a huge challenge in monitoring the condition of the IoT devices and the networks on which they reside. In particular, remote monitoring of the health levels of these devices, which use an external battery as a power source, poses a constant challenge. In many such scenarios, IoT devices do not have the ability to self-monitor or receive feedback from the external batteries to update their health status. Furthermore, since these IoT device clusters are usually deployed in remote areas, it is not possible to monitor the data physically. As a result, a monitoring system that is able to reflect real-time or up-to-date information would be especially valuable for the management of large-scale IoT networks.

B.   MOTIVATION

IoT devices are suited to many applications in varied implementations; however, they all need energy to operate. Energy is a critical component to the operation of an IoT network made up of battery-powered devices. Typically, IoT devices are low-powered and connected wirelessly, which causes the device to consume high energy at a rapid rate. An IoT device drained of its energy will not be able to function and can cause disruption of service or loss of functionality. Furthermore, communications with other IoT devices will be shut down if an inter-connecting device is down. In the scenario of critical services such as surveillance or health care, the IoT administrator cannot afford to have any downtime. Hence, it is essential for an IoT device to have some form of communication about its energy level with the server in the event of any incidents.

C.   RESEARCH OBJECTIVES

Since power is an essential component that indirectly translates to the health of an IoT network, it is important to manage the power for IoT devices effectively and efficiently. With that in mind, the hypothesis being tested in this research is as follows:

Is it possible to develop a large-scale IoT management system that provides energy monitoring?

To test this hypothesis, this research focuses on developing a solution that allows for communication between an IoT device connected to a battery power source and the

server to which the device is connected. In this solution, the IoT device or IoT cluster will be able to report its energy levels through a web-based graphical user interface (GUI) dashboard that supports easy overview and management. This research covers the following goals:

(1) Enhancements to enable each IoT device to identify itself and report its energy level to the server. This includes devices that cannot monitor their own energy levels.

(2) Development of a reliable communication mechanism between the IoT devices and the server to achieve autonomous monitoring.

(3) Development of a web-based GUI dashboard that provides the administrator with a clear view of an IoT device's energy levels in the large-scale IoT network and to identify problem areas quickly.

(4) Development of a web-based GUI dashboard that allows the administrator to query individual IoTdevices for their energy levels and to receive alerts generated by devices in the network. It also allows the administrator to perform basic trending analytics.

D. THESIS ORGANIZATION

The thesis consists of five chapters. Chapter II discusses the IoT network architecture and the technologies available to measure IoT energy levels in relation to this research. The topics covered include network architecture, energy management, energy measurement, Coulomb Counters assessment, and the Raspberry Pi.

Chapter III covers the system design and experiment setup. It highlights the components for the prototype setup and the other relevant hardware and technologies. A controlled testing environment setup is also reviewed in this chapter.

Chapter IV presents the actual implementation of the system and the experiment results. Challenges surfaced during the field test are presented and analyzed. We also identify some of the improvements that could help to increase the accuracy of the calculations and future enhancements to the current prototype that will make it more robust.

Chapter V summarizes the process and concludes the research. Areas for possible future research are also presented.

II. MANAGEMENT OF LARGE-SCALE IOT NETWORKS

The management of IoT networks encompasses a vast field that spans the domains of multiple disciplines. It includes protocol enhancements, analytical frameworks, and standardization efforts to facilitate interoperability and effective operations [2]–[6]. Both network architecture and power management are parts of the structural layers of IoT and are foundational in protecting and supporting the entire structure. In this chapter, we discuss the architecture and structure behind an IoT network as well as the literature review for the technologies adopted in this research.

A. NETWORK ARCHITECTURE

Network architecture defines the physical and, sometimes, logical structure of the network and can be viewed in terms of its topologies, communication paradigm, and operating mode in the case of wireless technologies. It is also an important design factor in IoT as it affects the devices' ability to communicate and the network's ability to scale.

1. Topologies of IoT

Network topology covers how devices are arranged in any communication network, and it defines the paths or links of data movement [7]. While there are many possible network topologies, our focus is on the common ones used in IoT data communications as shown in Figure 2.
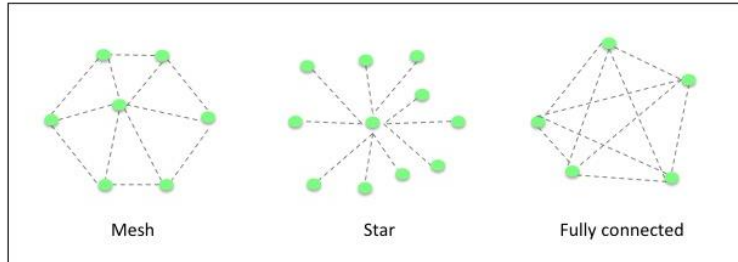
Figure 2.   Different Types of Network Topologies

a.   *Star Topology*

One key characteristic of the star topology is that every end-device has to be connected to a central device. End-devices cannot communicate directly to other end-devices and would need the central-device to function as a server for communicating with each other and possibly with the Internet. This topology is simple to setup and makes it easy to add and remove devices without interrupting the network. However, this topology introduces the central-device as the single point of failure [7].

b.   *Mesh and Fully Connected Topologies*

This topology, as shown in Figure 2, allows each device to be connected to any of the other devices in the network. The key advantage of this topology is the high level of robustness and redundancy established through the interconnectivity among the devices. This means if one link breaks, the devices can easily switch to other links for communication. The tradeoff, however, is the increased network complexity and the cost to establish the redundant links [7]. The fully connected topology is also a type of mesh topology in which each device is connected to every other device directly; this ends up being most expensive and complicated. In a typical mesh, a device may not be connected to all other devices directly.

2.   **Interaction Paradigm of IoT**

While network topology focuses on how the devices are arranged and connected over the network, this section looks into the interaction paradigms by which devices are interacting and cooperating with one another, possibly without human interactions. The interaction paradigm is an important aspect of an IoT architecture because it influences resource utilization in terms of processing, bandwidth, and even power consumption [3]. Common interaction paradigms are described in the following sub-sections.

a.   *Clients-Server Paradigm*

This paradigm is based on a simple interaction between the server and the clients. As shown in Figure 3, it is made up of a cluster of devices connected to a central device that acts as the control system. The main advantage of this setup is its ability to centralize

the management of a large number of devices, making it a suitable model for our experiment. While the paradigm is a logical star construct, the underlying network topology may be a mesh.
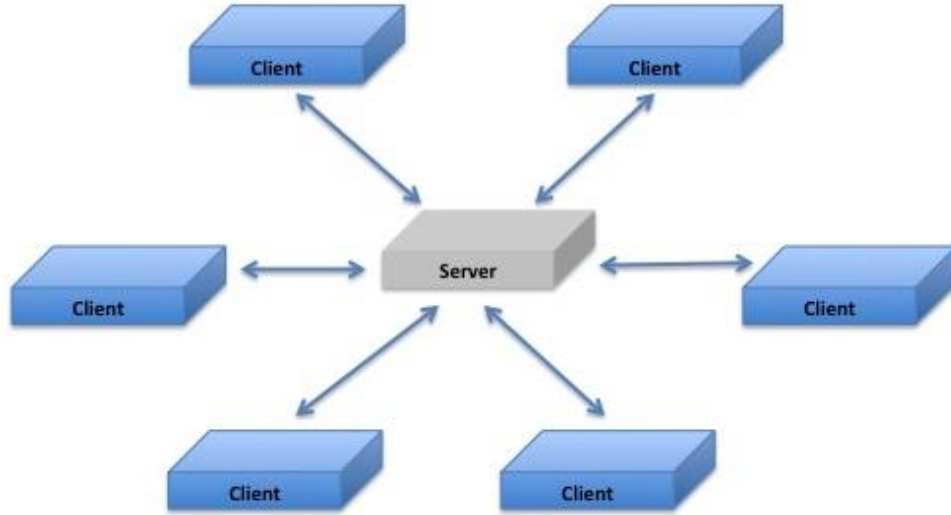
Figure 3.   The Clients-Server Interaction Paradigm

b.   *Message Passing System*

Message passing is built upon a simple interaction in which a sender forwards messages by means of a queue to a recipient. The objective of this model is to send only the required data and thereby improve the network performance [3]. In terms of scaling, large message-passing systems can share common infrastructures made out of Message Brokers, as shown in Figure 4.

One commonly available message passing protocol is the Message Queue Telemetry Transport (MQTT) that is used in this experiment. A notable feature of MQTT, which is applicable in our research, is its Last Will and Testament feature. This feature offers the advantage of notifying the recipient in the event that the sender is disconnected from the message passing system. The sender achieves this by sending a

pre-configured message to the message broker. When MQTT detects a disconnection, this pre-configured message is forwarded to the recipient for notification.
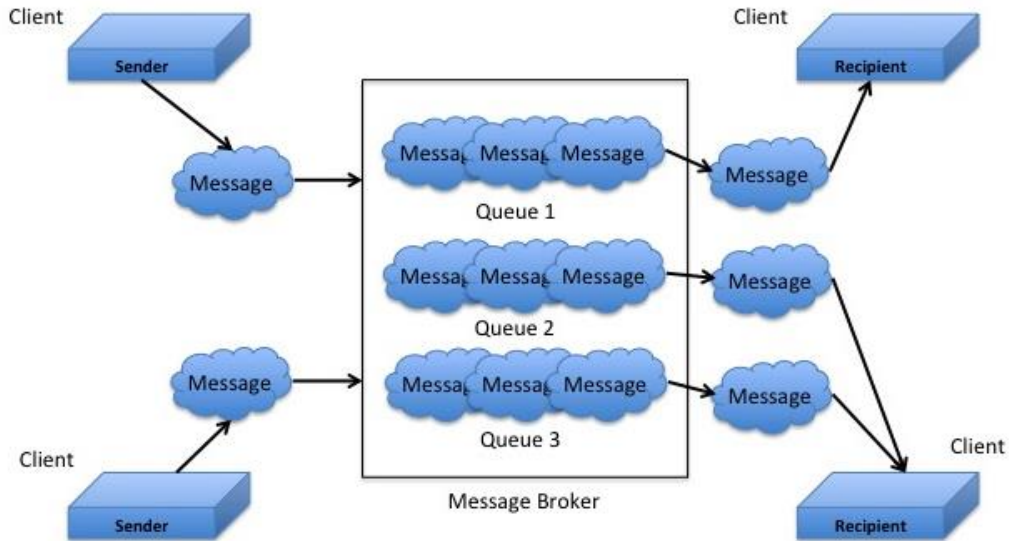
Figure 4.   The Message Passing Model and System

c.   *IoT Gateway*

IoT gateways are designed to support connectivity between devices on different IoT networks and between IoT networks and the Internet. At a minimum, an IoT gateway can be viewed as just a transceiver (forwarder and receiver) of data in a network [8]. However, since many IoT networks frequently use proprietary protocols and often lack interoperability with Internet Protocol (IP)-based networks, such as the Internet, IoT gateways are needed to perform protocol translation and data routing. (See Figure 5.)
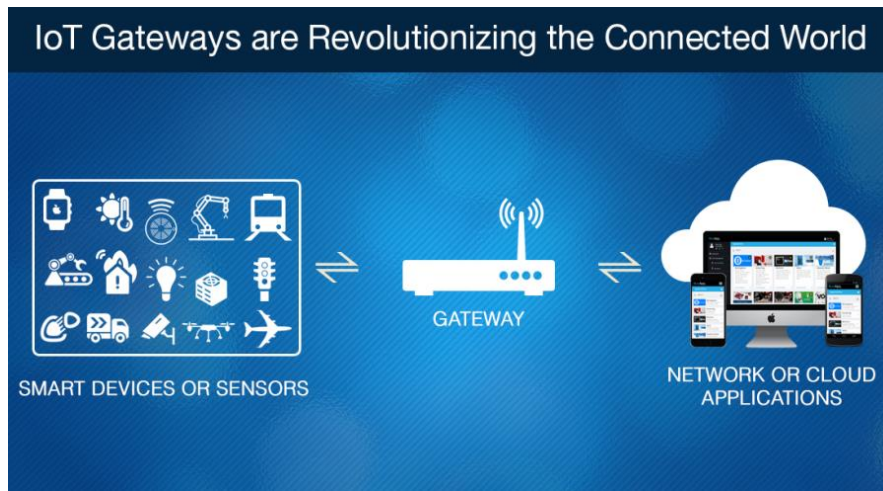
Figure 5.   IoT Gateway, a Centerpiece of the IoT Networks. Source: [9].

3.   Common Network Scenarios

The three common networking scenarios in the current landscape of IoT [10], [11] are as follows.

a.   *Mobile Ad-hoc Network (MANET)*

This is the scenario in which a group of highly mobile IoT devices need to connect without the support of a permanent and stable infrastructure. Devices in this scenario would generally utilize a mesh topology for link robustness and redundancy and can be highly diversified in terms of the type of devices (e.g., mobile phones, laptops, vehicles, etc.) [12]. As such, its interaction paradigms may vary.

9

b. Wireless Sensor Network (WSN)

In this scenario, autonomous sensor devices would be connected to a monitor and gather real world data, such as temperature, motion, pressure, fire detection, voltage and current, etc. [13]–[15]. Typically, these devices are spread out and cooperate in a peer-to-peer manner. The two common roles of the devices in a WSN are the source and sink devices. The source device is any device that provides the collected data, whereas the sink device is any device that needs, or consumes, the collected information. In addition, a source device can be a sensor device within the WSN or an external device such as a Personal Digital Assistant (PDA) or drone that moves around for data collection [16]. Collected data from the sensor devices are commonly forwarded to a central server over the Internet for further processing and storage. This means that a WSN at times may need to be connected to a permanent and stable infrastructure.

c. Delay-Tolerant Network (DTN)

This is the scenario in which end-to-end devices have difficulties trying to reach one another and are subject to intermittent network discontinuities, possibly due to range or masking obstacles. A device in this scenario needs to move data incrementally from the source throughout the network over opportunistic links until the data reach the destination successfully [10]. (See Figure 6.)
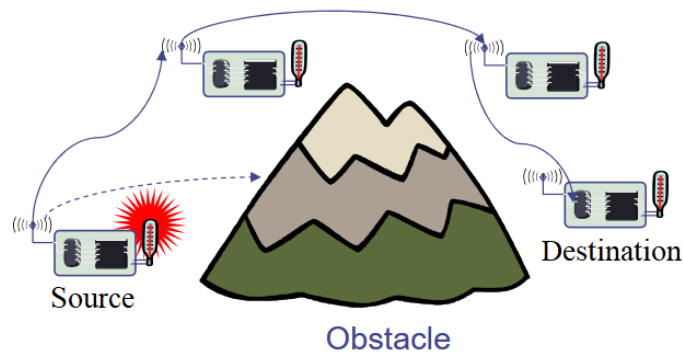
Figure 6.   Store-and-Forward Technique. Adapted from [10].

One common approach used in DTNs is the store-and-forward technique, where several intermediate devices are normally needed to forward the data cooperatively [10] to the destination. Human mobility is also another opportunity for DTN, as explored by Chaintreauet al. in [17].

B.   ENERGY MANAGEMENT

A network environment that is either inconsistent or low in power would affect the network's stability and reliability, as well as limit the devices' capabilities. In this section, we look into the approaches in the various areas of energy management.

1.   Hardware Platforms

The various components of an IoT device are the microprocessor, memory, sensors, actuators, and network adaptor. Over the years, hardware manufacturers have improved the power efficiency of these components through better circuit design that uses lower voltage and techniques such as the Dynamic Voltage Scaling (DVS) [18]–[20]. In addition, IoT devices such as sensors are generally simple, low-power systems, which are often more power-efficient than general purpose hardware.

2.   Software

The two essentials in this area are the operating systems themselves and the technique of in-network processing.

Although an intuitive approach is simply to design a lightweight Operating System (OS) with tighter hardware integration, such a design is not easy to accomplish. A recent study [21] by Hahm et al. showed the complexity in related tradeoffs, requirements, and constraints of current IoT applications and hardware and, hence, did not come to a satisfactory conclusion. Because of its interaction with the hardware, the OS plays an important role in power management.

In-network processing is a popular technique deployed in bandwidth-constrained networks. It works on a simple concept of reducing network traffic through data aggregation and processing by supporting collaboration among the IoT devices, and in

turn, reduces energy consumption [22], [23]. Implementations may use a message passing protocol as seen in [23] or one built into the application level, where Kougianos et al. [16] have proposed minimizing communications load by reducing the image size transmitted.

3. Communication Protocols

Numerous improvements in the various communication protocols have contributed to efforts to reduce power consumption. Some notable ones are as follows:

(1) New Wi-Fi standards for the low-powered environment: IEEE 802.11ah (Wi-Fi HaLow) and IEEE 802.11ax (Wi-Fi Hew) [24]

(2) Bluetooth's focus on low-energy devices: Bluetooth LE and Bluetooth 5 [25], [26]

(3) Zigbee and Thread's approach to tradeoff data-rate for power efficiency [7], [27]

(4) Creation of 6LoWPAN and RPL in both the transport and network layers for use in a low-powered network [28], [29]

4. Environment

In a situation where power is unstable or not available, alternate power may come directly from the environment through power scavenging methods (e.g., solar power, wind power). Although its suitability for IoT has not yet been widely researched, these methods might be a viable approach in the future.

5. Smart Batteries

Smart battery technology was primarily conceived to address the safety concern of battery overcharging and overheating [30]. It allows querying on a battery's energy levels and is commonly found in devices such as portable computers and mobile phones.

The current state-of-art design is typically a two-wire System Management Bus (SMBus) built within the smart battery and uses a standardized protocol (Smart Battery System) to communicate with the device [30]. Data transmitted could include the State of

Charge (SoC) to reflect the amount of energy left in the smart battery or the State of Health (SoH) to measure the condition of the smart battery in-use [30].

A limitation for smart batteries, though, is their dependency on the devices' built-in circuitry to receive battery readings. Without this feedback mechanism, smart batteries would function just like any bare-bones batteries and lose their effectiveness in energy monitoring. As shown at left in Figure 7, smart batteries are generally built to match a vendor or product specific form-factor, making interoperability with other devices a challenge. However, as seen at right, some IoT devices powered by standalone bare-bone batteries normally do not have the ability to report their energy status to the devices.

Figure 7.   Smart Battery in a Laptop (left). Standalone
Bare-Bones Batteries (right).

Smart battery technology will continue to revolutionize the way in which an IoT device gets its portable energy. One company is even taking the technology to a new level by integrating Wi-Fi into its safety products, such as smoke detectors and water-leak detectors [31], for real-time monitoring. Although this method is not widely adopted at this point, it would be an important concept for exploration in the future.

C. ENERGY MEASUREMENT

The first step toward monitoring the devices' uptime and identifying any abnormality in usage behavior is to be able to measure the energy used by the devices. This section outlines the related work done in this area as well as reviews the fundamentals and key approaches to measuring the SoC in a battery.

1. Related Work: PiCheckVoltage

PiCheckVoltage is a hardware and software combination that enables the user to measure the voltage of a battery attached to a Raspberry Pi-based device. It makes use of a voltage measurement unit to tell whether the battery level is low, good, or zero [32]. Figure 8 shows the schematic of PiCheckVoltage implementation. As measurements are estimated based on the simple thresholds, this method lacks the granularity needed for a more precise energy measurement.
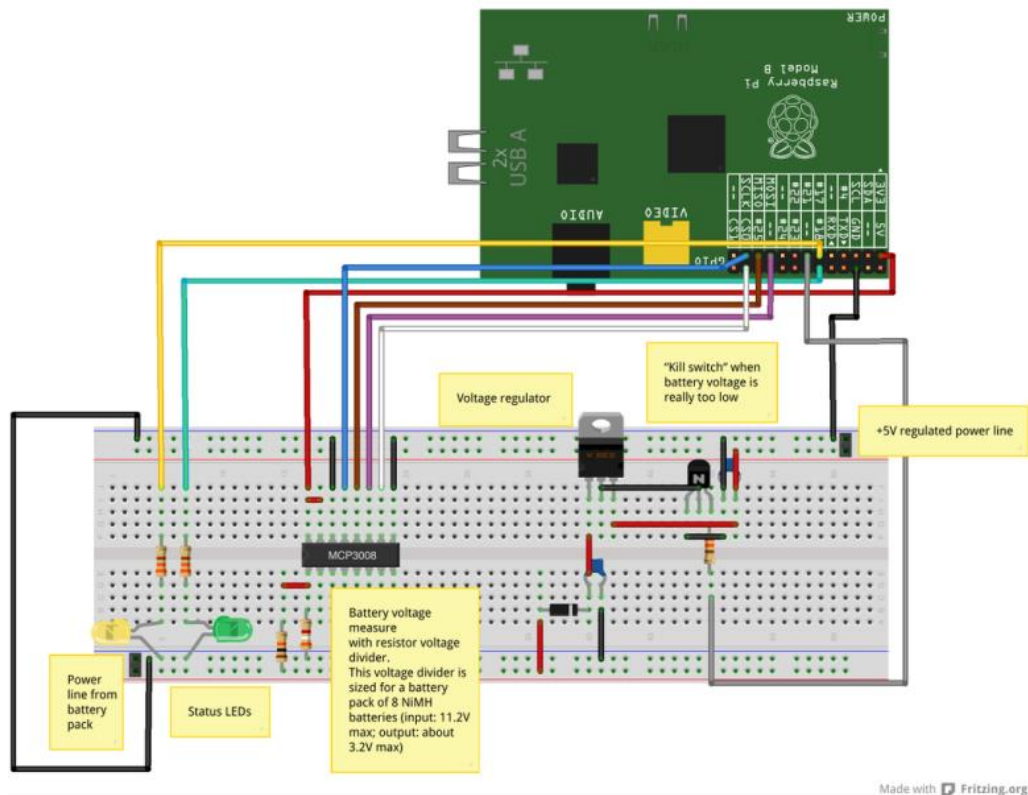
Figure 8.   Schematics of PiCheckVoltage. Source: [32].

2.   Related Work: Amp-O-Meter

This is a project by Felipe Morgan [33] that comprises both hardware and software to measure energy consumption on a third-party device. Morgan makes use of a Raspberry Pi to develop a current meter that is able to measure the current used on third-party devices through his server, using the coulomb-counting approach. (See Figure 9.) We explore the concept of this project for the purpose of our research due to the granularity of the measurement this project has achieved.

Figure 9.   Amp-O-Meter. Source: [33].

3.   State-of-Charge

SoC refers to the energy level currently remaining in or exhausted by a device that uses a battery as an energy source. It is similar to a fuel gauge in a vehicle and is measured in percentage points (0% = empty and 100% = full) [34].

Three key components that determine the SoC are as follows:

(1)   **Capacity**: Refers to the battery's ability to store and retain energy [35] and is commonly measured in milliampere-hour (mAh).

(2)   **Internal resistance**: Refers to the circuit's resistance that would cause the source voltage to drop whenever there is current flowing [35]. In most cases, this would be a non-variable component (fixed value) measured in ohms.

(3)   **Self-discharge**: A self-inflicting chemical reaction within a charged battery that reduces the stored charge levels of the battery over time [35]. Measurement is used to determine the usage duration of a battery as shown in Figure 10 (e.g., a 25 mAh discharge rate will support 48 hours of usage and a 250 mAh discharge rate support get three hours).(See Figure 10.)
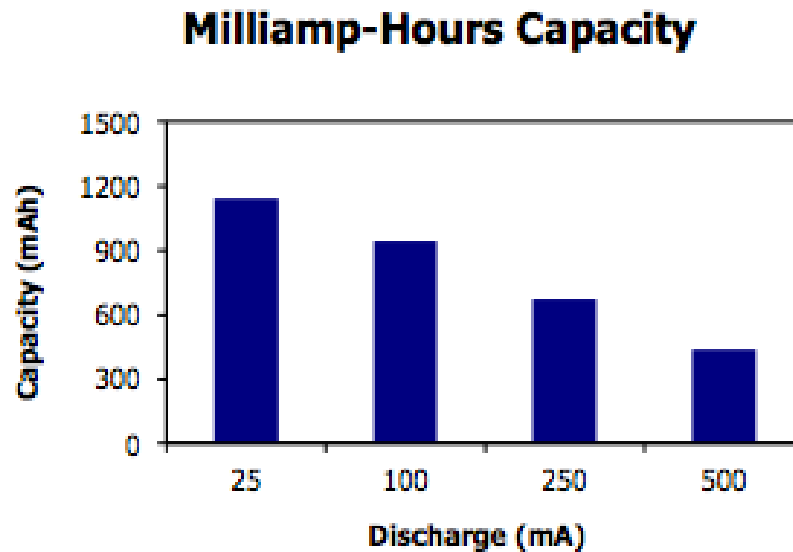
Figure 10.   Datasheet on Energizer Alkaline AAA's Discharge Rate.
Source: [36].

The two predominant methods to measure SoC are the voltage measurement and coulomb counting [33]–[35], [37] and are discussed in Sections C3 and C4 of this chapter.

4. Voltage Measurement

Due to its simplicity, voltage measurement is one of the methods commonly used to measure SoC. It is calculated by converting the battery's voltage reading and the battery's discharge curve. However, this method has its disadvantages, due to the inaccuracy easily caused by external factors such as temperature and disturbance. Voltage measurement requires the battery to supply a constant voltage in order to have an accurate measurement, which makes this method difficult and unfavorable to apply.

Voltage level does not follow a linear relationship against the SoC. As seen in Figure 11, we are only able to gauge the full charge and the low charge. The flatness of the middle segment does not give us an accurate reading regarding the SoC [34], [37].
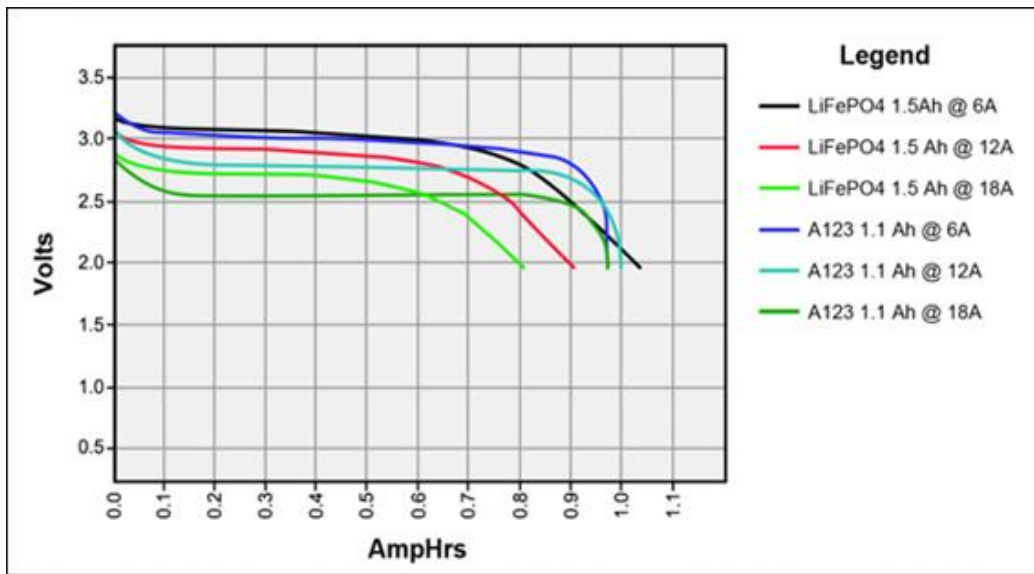
Figure 11.   SoC Estimation by Voltage. Source: [34].

5. Coulomb Counting

Coulomb counting is less common as compared to voltage measurement but is a preferred method to apply if a higher level of accuracy is required. Coulomb counting is an in-line technique that measures a battery's SoC in a constant rate by measuring the flowing current. Assuming that we know the full capacity of the battery, this technique

allows network administrators to yield a more precise and direct estimation of the battery's SoC by knowing how much charge has already been used [34], [35], [37].

A typical setup for this approach would normally require a small electronic device, commonly known as the Coulomb Counter, to be connected between the battery and the device as show in Figure 12.
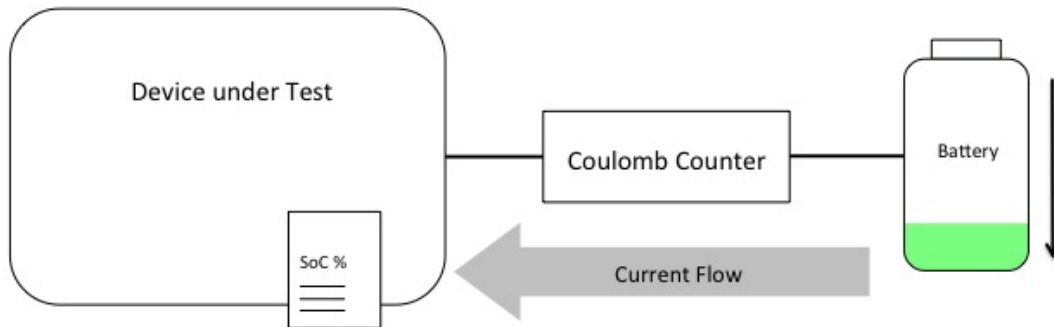
Figure 12. Coulomb Counter Setup

One limitation with this method, though, is that the Coulomb Counter is unable to determine the full capacity of the battery, as the Coulomb Counter's main functionality is only to measure the amount of energy used. Our concept of estimating the battery's full capacity is explained in Section C2 of Chapter III.

Due to the granularity that this approach could achieve, we explored the setup as shown in Figure 12 to measure the energy consumed by the Device Under Test (DUT) and hence derive its SoC.

6.   Calculating Actual Battery Capacity

Although every battery will typically have an advertised capacity, the latter cannot be taken as the actual capacity, due to the voltage conversion within the battery. The difference in actual and advertised capacity occurs because a Universal Serial Bus (USB) interface requires 5V as the input, whereas a typical battery uses 3.7V internally. As such, a conversion circuit is needed in the battery to convert the voltage from 3.7V to

5V. As a result of this conversion, the actual battery capacity is much less than advertised [38]–[40]. An example of converting an advertised capacity to the capacity in 5V is shown in Figure 13.

```
Advertised Capacity x 3.7v / 5v = Actual Capacity in 5V
e.g., 10,000mAh x 3.7/5 = 7,400mAh in 5V
```

Figure 13.   Converting Advertised Battery Capacity to Capacity in 5V

The result shows that a 10,000mAh battery can only supply 7,400mAh at the 5V USB output, illustrating a 26 percent loss from the advertised value. Furthermore, a battery also experiences loss during the energy transfer within the conversion circuit through heat due to internal resistance. This loss can range from 2 percent to 15 percent, based on various industrial standards and the quality of the battery hardware [38]–[40]. Since most vendors do not state how much energy is lost on this aspect, we would need to find out through experimentation with the batteries under test and use the result to calibrate the measurements required in our research.

D.   COULOMB COUNTERS ASSESSMENT

During our literature review, one research gap that we uncovered is that devices cannot determine their battery's energy level without a proper feedback mechanism, regardless of whether a smart battery is in use or not. While there are many coulomb counters available in the industry, we need to be mindful that not all have the appropriate feedback mechanism needed. As such, we have reviewed some of the industrial solutions and have identified the following coulomb counters to be suitable for our research.

1.   Adafruit USB Power Gauge

Adafruit USB Power Gauge is one of the popular solutions on the market. The Power Gauge is a small device that acts as the connector for USB devices, something like a speed gauge. It consists of a programmable microprocessor, an analog output, and a transistor-to-transistor-logic (TTL) serial output [41]. The main feedback mechanism is

via the pin-out at the side of the circuit board. However, the product is unavailable at this point in the research. (See Figure 14.)
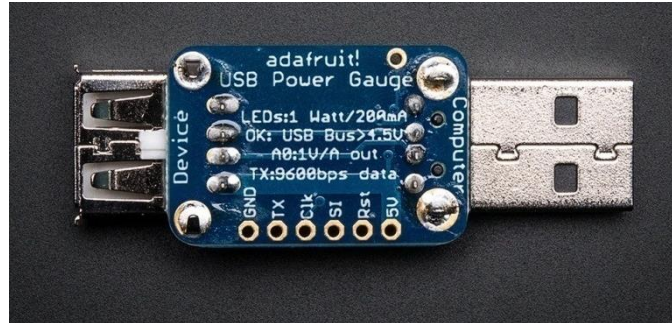
Figure 14.   Adafruit USB Power Gauge. Source: [41].

2.   Electrolabs DIY USB Power Meter

Another feasible solution explored is a do-it-yourself (DIY) USB Power Meter by Electrolabs. It is basically a microcontroller board with a USB interface and a small Organic Light Emitting Diode (OLED) display embedded in it. The microcontroller is programmed to measure the battery's supplied voltage and the current that is being used by the attached USB device. Similar to Adafruit's USB Power Gauge, the feedback mechanism can be via the pin-out at the side of the circuit board. In addition, information can also be displayed on the built-in OLED [42]. (See Figure 15.)

Figure 15.   Electro-Labs DIY USB Power Meter. Source: [42].

3. Fried Circuit USB Tester

This USB tester by Fried Circuit is a similar product to the previous solutions described. It makes use of a microcontroller board with a USB interface to measure the current and voltage of the connected USB device. It has an add-on solution, called the "Backpack," which basically consists of a display unit for easy reading of the measurements and a feature that allows the user to log the readings and import the results into Excel. (See Figure 16.) One unique feature of this solution is the integrated micro-USB for easy feedback from ready-built software [43].

Figure 16. Fried Circuit USB Tester. Source: [43].

4. LTC Coulomb Counter

The LTC Coulomb Counter is a solution that measures current through an external sense resistor. (See Figure 17.) It measures the constant charge flowing between the battery and the DUT through output pulses transformed from the current sense voltage. Each pulse represents a fixed amount of current flowing in or out of the battery. Current polarity indicates whether the battery is discharging or charging. By knowing the full capacity of the battery beforehand, it is possible to gauge the battery usage over time

[44]. In this research, we make use of LTC's Coulomb Counter as the interface between the energy source (battery) and the IoT device, as it is reputable, and readily available.
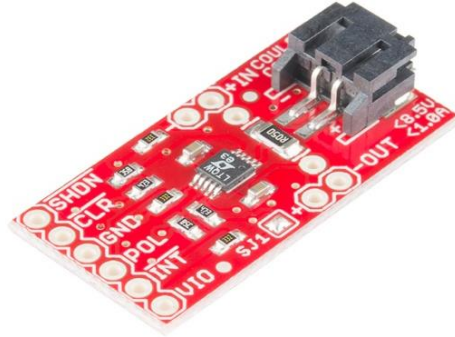
Figure 17.   LTC Coulomb Counter. Source: [44].

E.   RASPBERRY PI

Raspberry Pi is an affordable, versatile microcomputer that is largely used in schools or testing environments.(See Figure 18.) The built-in Broadcom processor provides adequate computing power to run fundamental programs such as Python, MySQL, and PHP. It is Wi-Fi-enabled, which makes it an ideal solution in our test environment as data needs to be transmitted wirelessly. The programmable input/output (I/O) pins provide greater connectivity to other devices, such as sensors or electronic circuits. Raspberry Pi also allows the user to have access to a GUI when it is connected to a computer [45].

Figure 18.   Raspberry Pi

The Raspberry Pi operating system, Raspbian, was developed using the Debian operating system as the base and is customized to run efficiently on the Raspberry Pi [46]. Being open source, it is free and easily configurable, and has a large community of contributors that provides a significant support base. It is also lightweight and reliable. We make use of Raspbian as the basis for our test program to facilitate easy configuration of the IoT devices and test data.

F.　CHAPTER SUMMARY

This chapter provided a detailed study on the various technologies and concepts pertinent to the monitoring of power usage by typical IoT devices. One research gap that we identified was the lack of an autonomous solution that allows remote, real-time review of the devices' SoC in an IoT network. Due to the importance of this requirement, this research focuses on finding a solution that provides autonomous feedback for monitoring the SoC in the battery of an IoT device. It is also imperative for the solution to have a long life span, to have negligible impact on the power usage of the monitored device, to be reliable, and to have the ability to transmit measured data over wireless networks.

Chapter III describes a general overview of the design concept with a detailed explanation of the various components in our experiment, such as the client device, server setup, dashboard design, and communication approach.

THIS PAGE INTENTIONALLY LEFT BLANK

III.   EXPERIMENT SETUP

In this section, we plan the various components of the proposed Energy Monitoring System (EMS) for large-scale IoT networks and discuss the details of the experiment setup, namely the overall design concept, the network architecture, the client device setup, the server setup, the web-GUI dashboard, and the flow diagrams for both the client communication and the server communication modules. All are essential components that contribute to the energy monitoring solution. We also present the feasibility of adapting this solution to a real-life, large-scale IoT environment where manageability and performance are important concerns. A controlled test environment implementation, mimicking an IoT setup, is described using Raspberry Pi as the IoT device and a USB battery as the energy source. The environment described is used to conduct an experiment, detailed in Chapter IV, wherein the IoT devices are connected wirelessly using Wi-Fi and are assumed not to have the ability to monitor their own energy level, thereby requiring the monitoring capability described here.

A. SYSTEM DESIGN CONCEPT OVERVIEW

Figure 19 illustrates the various network topologies in a typical IoT system setup. Devices are connected in both mesh and star topologies, and some devices, serving as limited-capability routers, provide inter-connection between the various networks. These inter-networking devices are critical in maintaining the connectivity of the network.
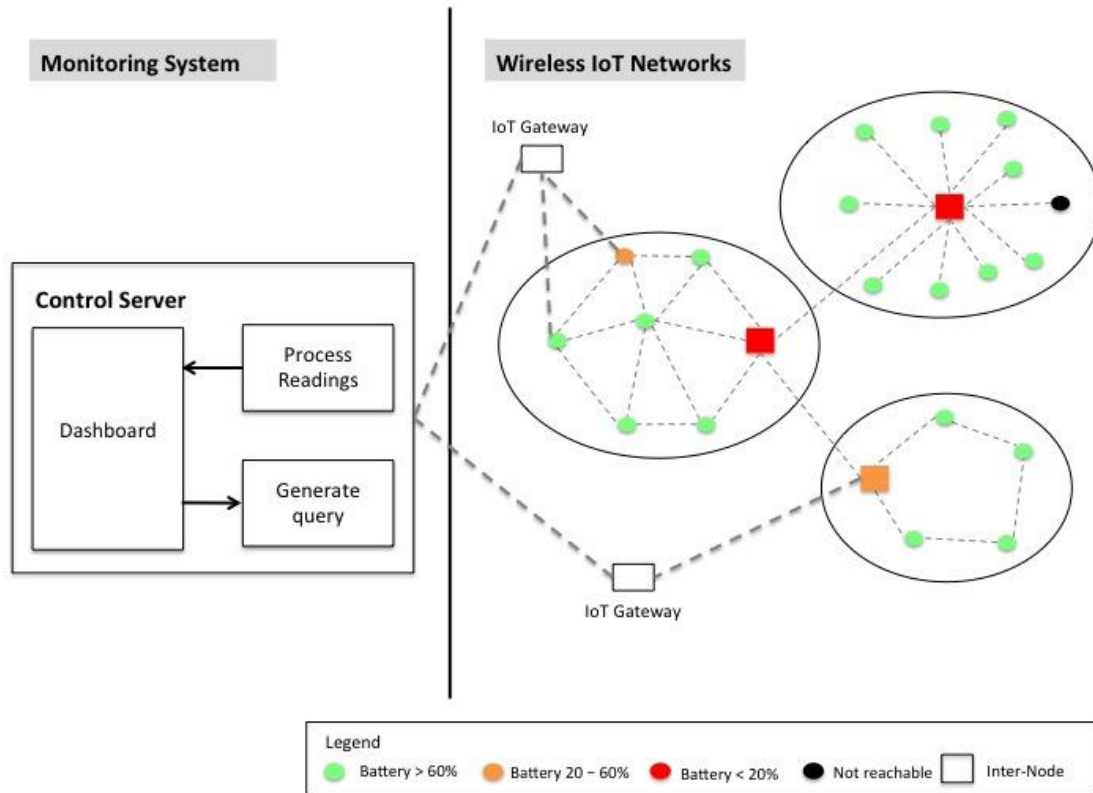
Figure 19. System Design Concept Overview

An Energy Monitoring System for large-scale IoT, leveraging device SoC measurements collected by coulomb counting components, is proposed to oversee the energy status of each device in the network autonomously enabling an administrator to react proactively to IoT devices that require attention.

B. COMMUNICATIONS ARCHITECTURE

A two-way communications protocol is employed between the server and the client devices. The "Push" function from the client devices is achieved via the Message Passing Protocol – MQTT and the "Pull" function from the server is achieved via the Hypertext Transfer Protocol (HTTP). (See Figure 20.)

This design allows the server to query the client devices for information and the client to reply with requested information to the server. The client device regularly updates the server with its battery health status ("push"). The server can also probe the client for real-time status through the web-based GUI ("pull").
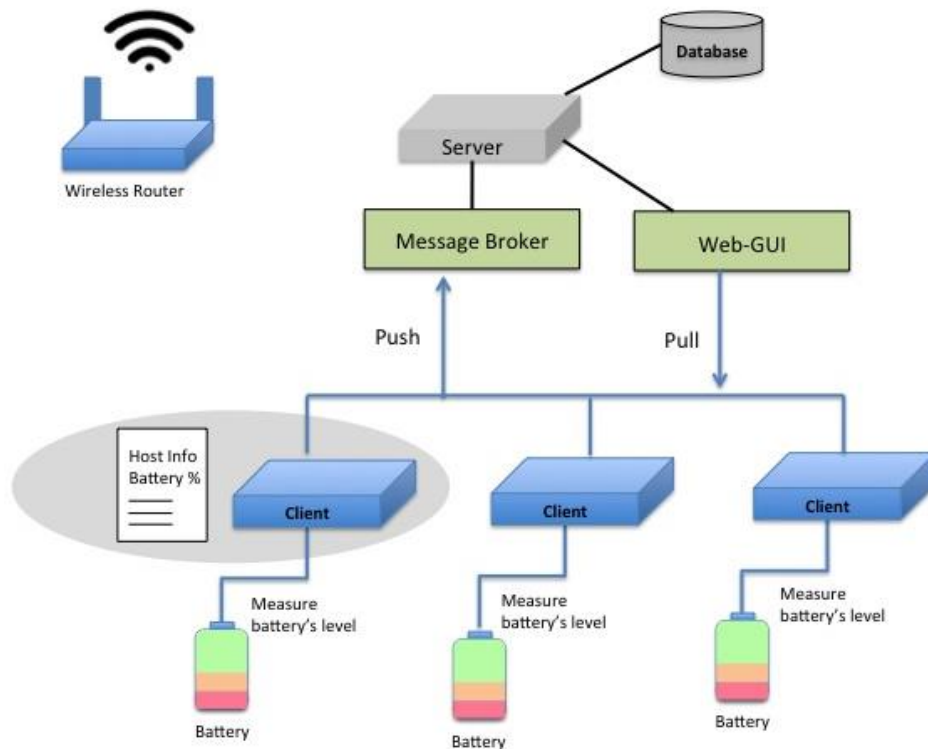
Figure 20.   Server-Clients Connectivity Diagram

C.   CLIENT DEVICE SETUP

Each client must have the ability to monitor its own battery status and send the information to the server. This section outlines the underlying concept and the various aspects of setting up the client device.

1.   Converting Charge Capacity from Coulomb to Ampere

The current way for representing the charge capacity of a battery is commonly expressed in units of milliampere-hours (mAh), reflecting the duration a current may be sustained, and thus the electric charge. As such, it is necessary to convert the coulomb measured to milliampere. Table 1 shows the conversion.

Table 1. Conversion of Coulomb to Milliampere.

1 Coulomb	= 1 Ampere per second (As)
3600 Coulombs	= 1 Ampere-hour (Ah)
3.6 Coulombs	= 1 milliampere-hour (mAh)

By directly measuring the coulombs that pass through the hardware via the Coulomb Counter, we can derive the amount of the current used in mAh (e.g.,a measurement of 36,000C would mean that 10,000mAh of energy have been used). This conversion also allows for a more standardized way of expressing discharge rate that is commonly obtained by using the amount of milliampere-hour used over time.

2.   Determining the Actual Battery Capacity

As discussed in Section C6 of Chapter II, it is possible to estimate the actual battery capacity through both the voltage conversion equation and the inference of conversion loss through experimentations. For this experiment, we have conducted multiple full discharge of the battery to assess the actual capacity of the battery; therefore, subsequent measurements reference this capacity as the actual capacity of the battery. However, it is important to note that the battery's SoH would deteriorate over

time due to routine wear from charging and discharging, and it is not practical to perform a full discharge test every time before measurement. Due to these constraints, our EMS is unable to determine the batteries' deterioration over time.

3. Device Data Pertinent to the System Monitoring

Following is the client device data tracked by the server and their definition.

(1)   Inter-network device: A configurable setting to help identify a device as being inter-connected between networks. A check on this setting would imply a higher level of priority in tracking the devices' energy level.

(2)   Hostname: A human readable identifier that corresponds to the IP address of the device.

(3)   MAC address: A unique identifier of the device for validation with the monitoring system.

(4)   Location: The current position of the device.

(5)   Average discharge rate: The average amount of energy used per hour and is measured in mAh.

(6)   Battery left: The amount of energy level remaining in the device. It is calculated through the difference between the full battery capacity and the amount of current measured since the last calculation.

(7)   Battery run-time: The entire amount of time that the battery has been deployed, not counting the duration when the device is powered down.

(8)   Estimated Battery life left: The estimated amount of energy left in the battery.

(9)   Device online/offline indicator: A real-time indicator that shows whether the device is online or offline.

(10)   Alerts: Notifications to the server when a device reaches a different energy state (e.g., Amber or Red) or gets disconnected from the server.

4. Client Device Components

Figure 21 illustrates the various components of a device setup. There are five components in this setup: client device, Coulomb Counter, battery, web-based GUI, and the communication module. The client device, Coulomb Counter, and battery form the

hardware component of this device, whereas the web-based GUI and communication module form the software component. Information collected from the battery by the Coulomb Counter is pushed down to the server via the Client Communication Module (CCM). The data is encoded to minimize its size and sent via the message-passing technique as described in Chapter II. A web-based GUI then allows the administrator to query the client device for information such as the hostname, uptime, system load, location, and the estimated battery life balance of the client device. In this regard, a lightweight approach is adopted where the client device would only need to host a text file where the file would be constantly updated with the latest client device information.
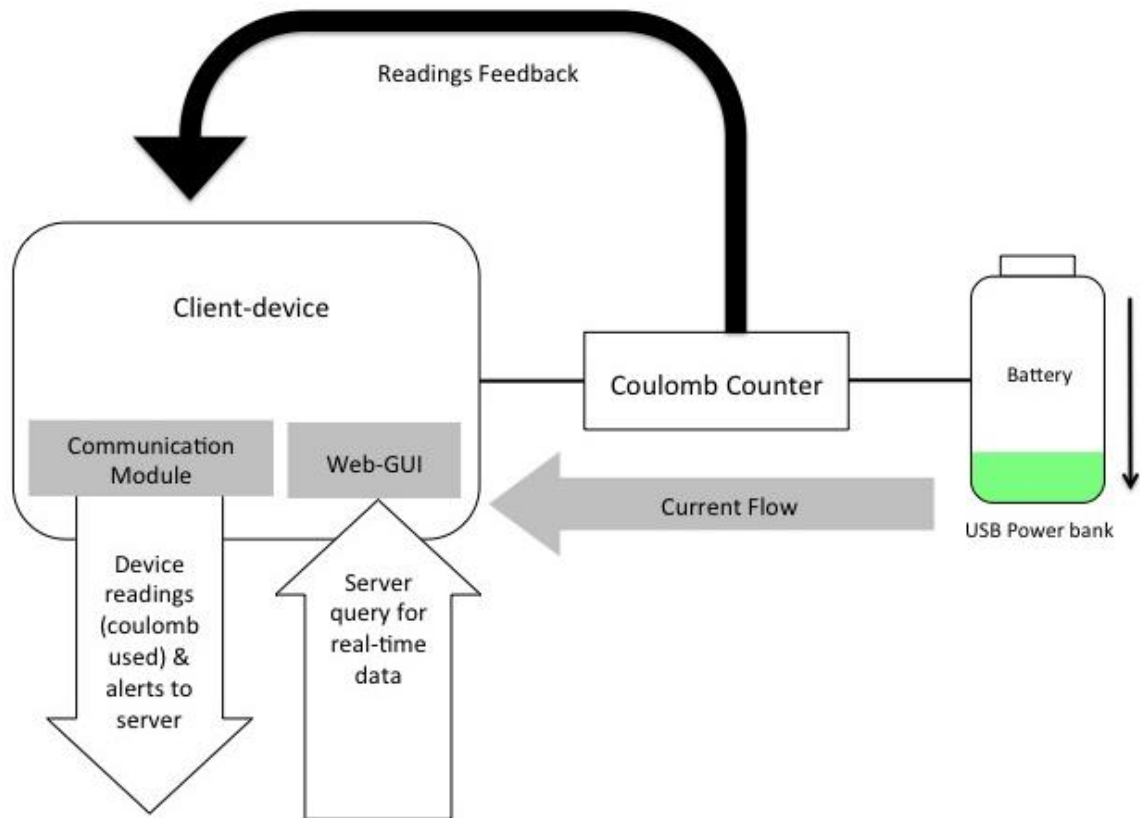
Figure 21.   Client Device Concept Diagram

D.   SERVER SETUP

The server tasked with the central management of data provides a platform for the administrator to manage the associated client device energy levels. This section outlines the various components of the server as shown in Figure 22, namely the database, the Server Communication Module (SCM), Auto-Probe Module (APM), and the dashboard.
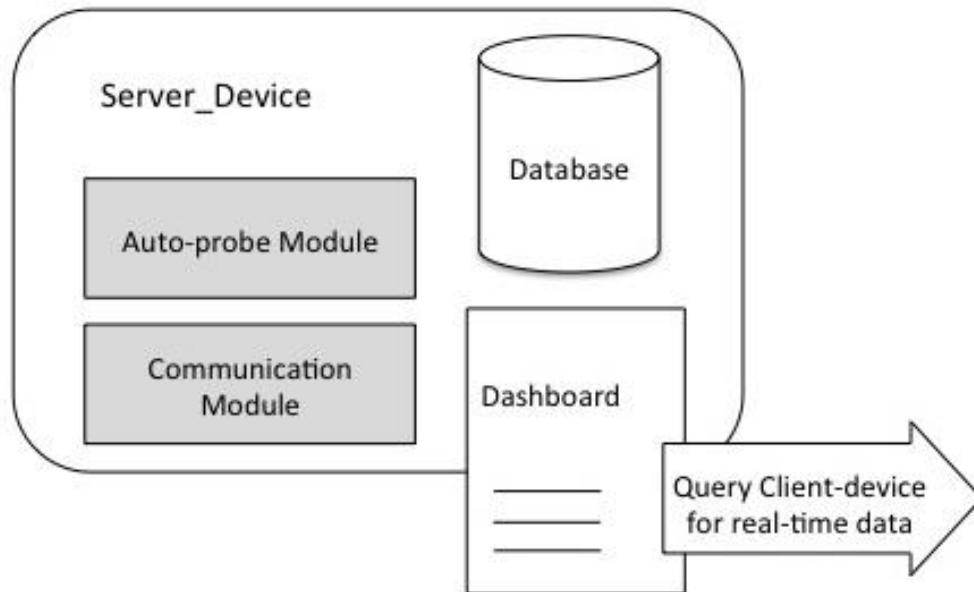
Figure 22.   Server Concept Diagram

1.   Database

The server is setup to have a database for two primary purposes. The first is to maintain important device information such as the pertinent device location, MAC address, and energy used as discussed in Section C3. The second is to keep records of the energy consumption for each monitored device.

2.   Server Communication Module

This communication module allows the server to receive routine updates from the client devices on their location and energy used, and is also able to distinguish alerts from

routine updates for separate notifications. All information is updated to the database upon receipt of data.

3.   Dashboard—Web-based GUI

The dashboard is an essential component of any monitoring system. It is the main interface between the administrator and the client devices. Important device information is displayed on the main page. The administrator can perform a number of functions and achieve specific objectives through this interface. This aids in the overall management of the entire IoT network.

The administrator can monitor device health by using simple color-coded cues, as well as interact with client devices to check their status in real time. The dashboard automatically refreshes itself in regular intervals to ensure that the data is always updated.

a.   *Detailed Device View*

Figure 23 illustrates the difference between a typical device cluster view and a Detailed Device View (DDV). The DDV displays device information in itemized rows. This allows for a structured method of viewing data. Structured data allows the administrator to easily query and analyze the information collected. The dashboard also allows the administrator to query the client for real-time information by clicking on each device's hostname under the DDV to access real-time information such as the device's current energy level and location.

Device ID	Inter-network	Hostname	Location	Battery Status
1.	☑	Host 1	A	10%
2.	☐	Host 2	B	100%
3.	☐	Host 3	C	85%
4.	☐	Host 4	D	87%
5.	☐	Host 5	E	80%
6.	☐	Host 6	F	50%
7.	☐	Host 7	G	100%

Figure 23.   Cluster View versus Detailed Device View

b.   *Quick Device View*

The Quick Device View (QDV) is intended to provide an efficient method of displaying devices according to their device ID, together with basic information such as battery status. This information is displayed in each cell, enabling the administrator to pinpoint a problem device in an efficient manner. Administrators can also click on each cell to interrogate the associated device for additional information such as its location and MAC address. Each cell represents a client device. New devices added to the network fill in the empty slots, as shown in Figure 24. There is no practical limit to the number of devices that can be added.
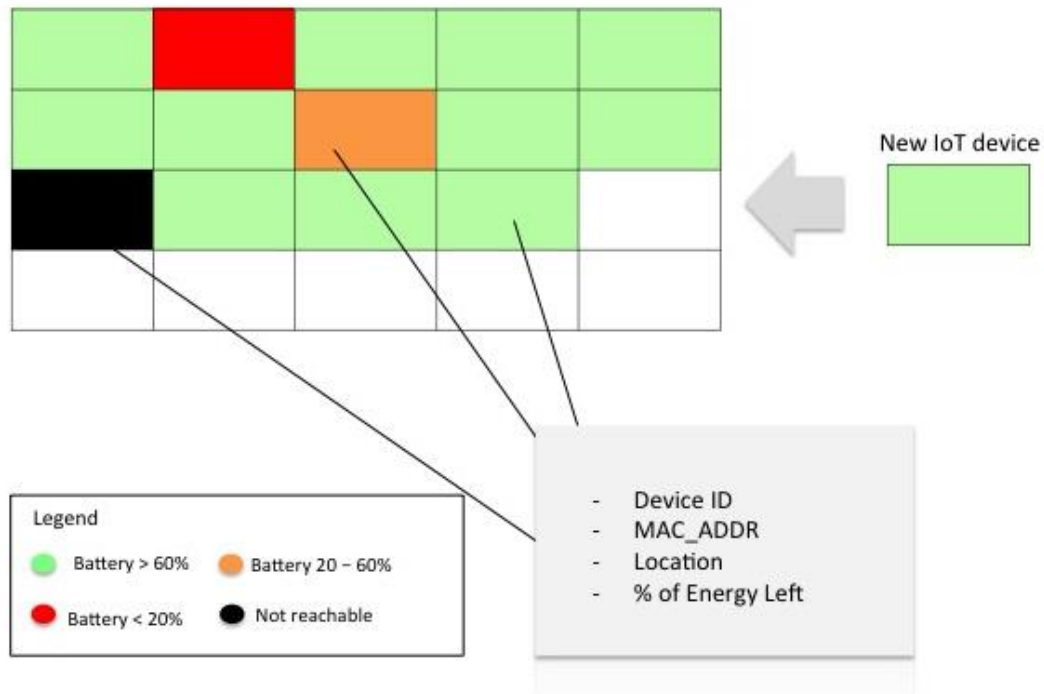
Figure 24.   Quick Device View of Dashboard

c.   *Basic Analytics View*

The dashboard facilitates applying analytics to the information collected. As illustrated in Figure 25, trending of discharge rate over time can reflect whether a device is idling without any load (e.g., Node 1), or suddenly being overloaded by events (e.g.,

Node 2), or just simply undergoing a routine cycle of events (e.g., Node 3). This enables the administrator to call for a better load balancing strategy or to re-examine the deployment strategy.

The dashboard is also helpful in predicting the battery's runtime by keeping track of the average discharge rate. A sudden death could translate to important data loss or services downtime. Therefore, having access to this information adds to the predictability of a battery's runtime for maintainability.
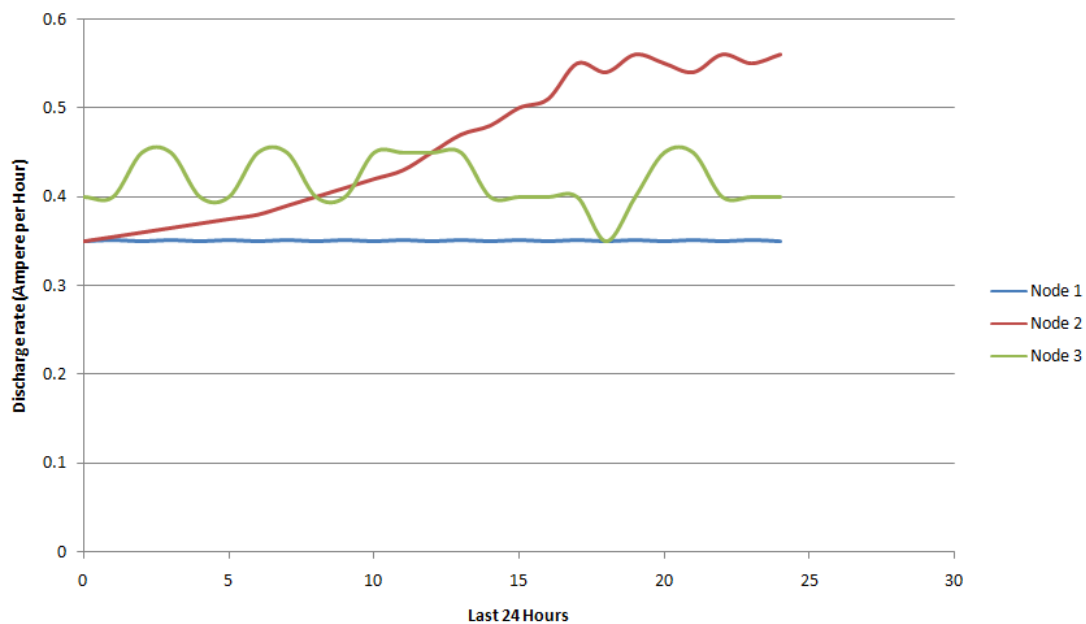
Figure 25. Simulated Trending of Device Discharge Rate

E. PROGRAM FLOW DIAGRAMS

This section illustrates the three program modules developed for this experiment. They provide a clear overview of the program flow.

The CCM, shown in Figure 26, is connected to the SCM to allow the client device to send alerts and feedback to the server. It will update the server if the state of the client device is ONLINE or ON RESUME. Information about the energy level and location of the client device is also sent to the server. In the event of low energy levels, alerts are generated and sent to the server as well.
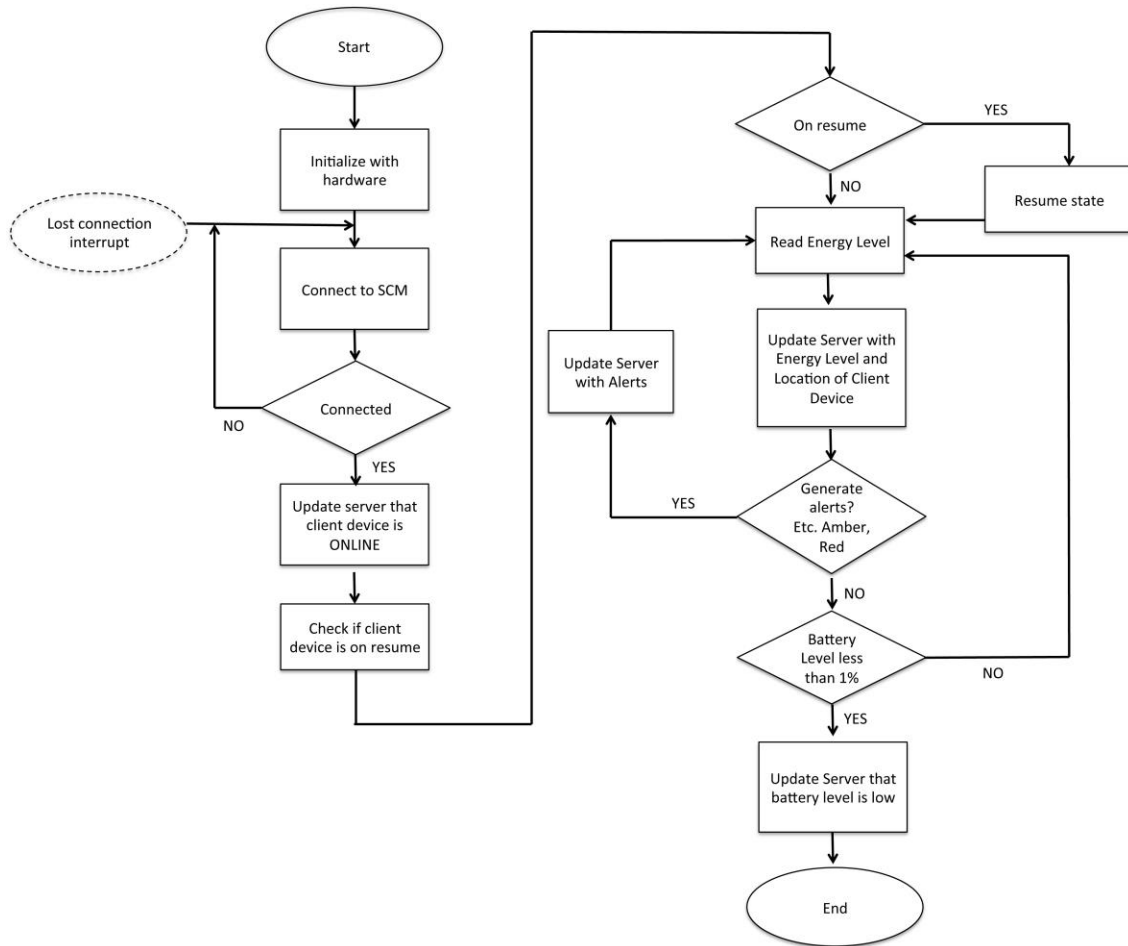
Figure 26. Client Communication Module

The SCM, shown in Figure 27, receives feedback and readings from the CCM and sends probes if it detects that communication with the client device is lost. It generates alerts to inform the administrator when it detects the lost connection and updates the database accordingly.

Alerts are generated in the event of low energy levels. The SCM also updates the server database regarding the client device's energy level and location.
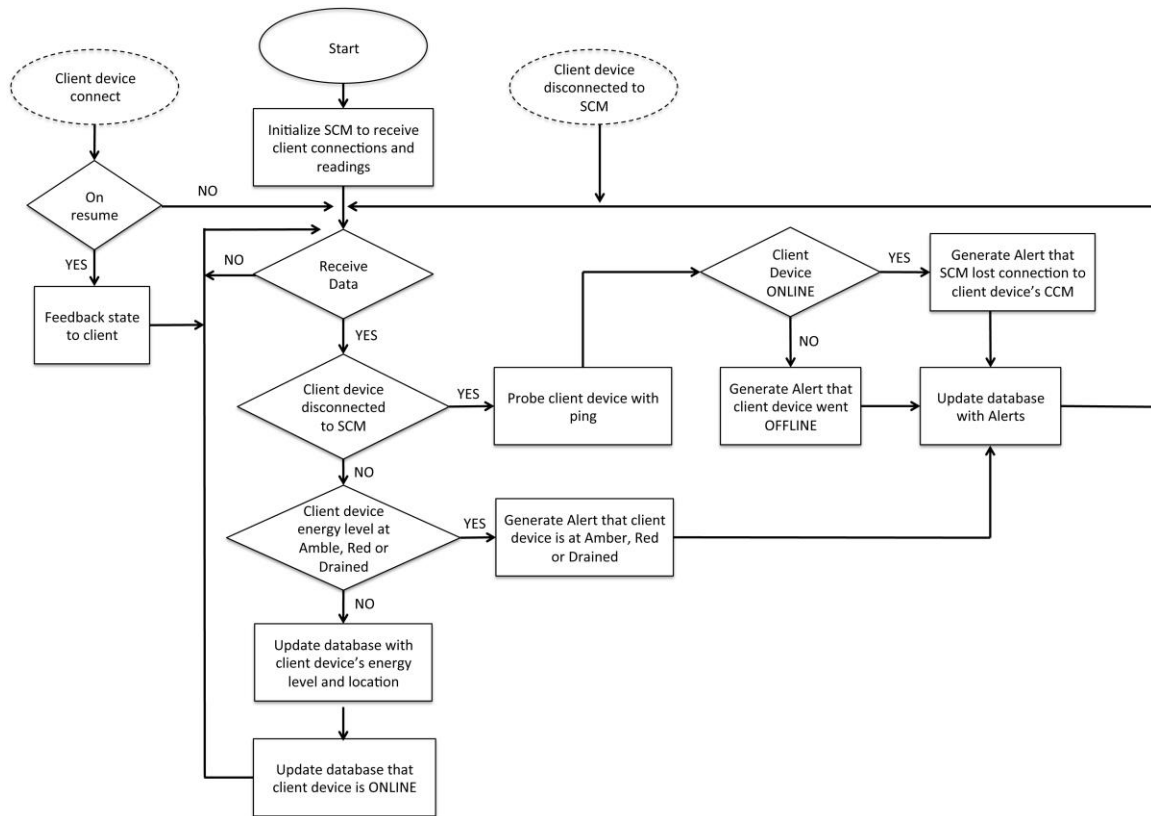
Figure 27.   Server Communication Module

The APM, shown in Figure 28, routinely checks for the ONLINE/OFFLINE indicator of the client devices based on the time schedule set by the administrator. It gets the latest list of client devices and their IP addresses, and systematically sends a probe to each client device in the list. The APM then updates the database regarding whether the device is online or offline.
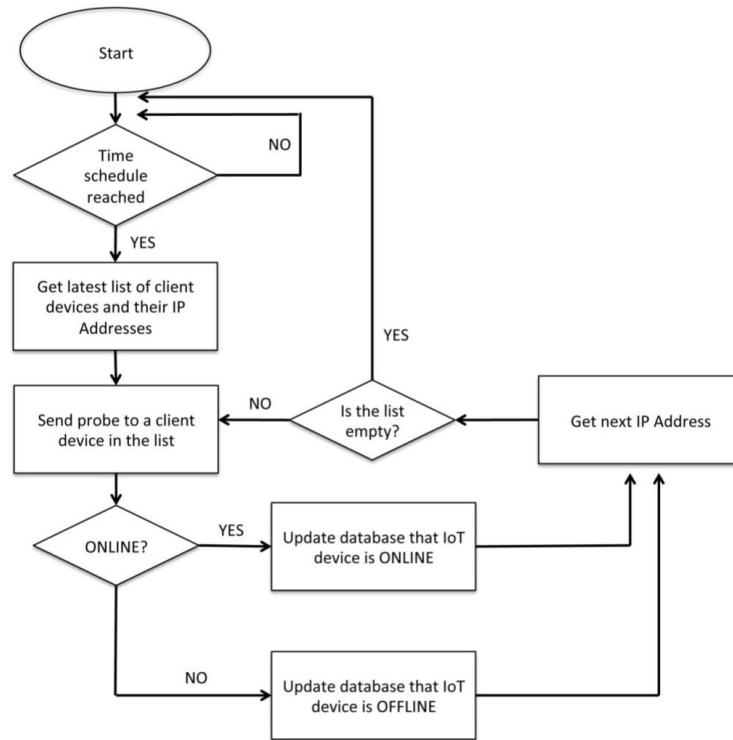
Figure 28.   Auto-Probe Module

F.   CHAPTER SUMMARY

In Chapter III, we have covered the various design concepts for the EMS, such as the overall system design, communications architecture, energy calculation, client device design concept, server design concept, dashboard, and the program flow diagrams.

In Chapter IV, we showcase the actual implementation setup and the test results as illustrated in our research objectives.

THIS PAGE INTENTIONALLY LEFT BLANK

IV.   IMPLEMENTATION AND TESTING

This chapter outlines the implementation process and the various tests done against our hypotheses in Chapter I. Screenshots of the results gathered are also provided. We discuss our implementation of the EMS, starting with implementation considerations for the various components used to achieve the results in this experiment.

A.   IMPLEMENTATION CONSIDERATIONS

Our primary implementation considerations were as follows.

(1)   Lightweight implementation: Applications developed should not affect the client device's normal functioning.

(2)   Accurate measurement: Monitoring system developed should be accurate and consistent for reliability.

(3)   Responsive to energy consumption: Monitoring system developed should be able to detect the different levels of energy consumption due to changes in system load.

B.   THE APPROACH

We made use of coulomb counting as the approach for energy measurement in this research as it yields more granular data as compared to voltage measurement. The Coulomb Counter was incorporated onto the Raspberry Pi, and then attached to a USB battery. As the Coulomb Counter was required to be in line between the battery and the client device, modification was done to the USB cable to allow a customized connection.

Initial testing was done using an open source Python script, Amp-O-Meter [33]. However, we encountered some challenges with it as the script was not meant to self-monitor, and it crashed at times after long hours of measurement. Hence, the final implementation was accomplished using our own scripted Python program.

C.   OVERVIEW OF ENERGY MONITORING SYSTEM

The solution is made up of four main components, as depicted in Figure 29: the wireless router, client devices, server, and dashboard. We now look into the details of how each component was setup in this experiment as well as its functionality.
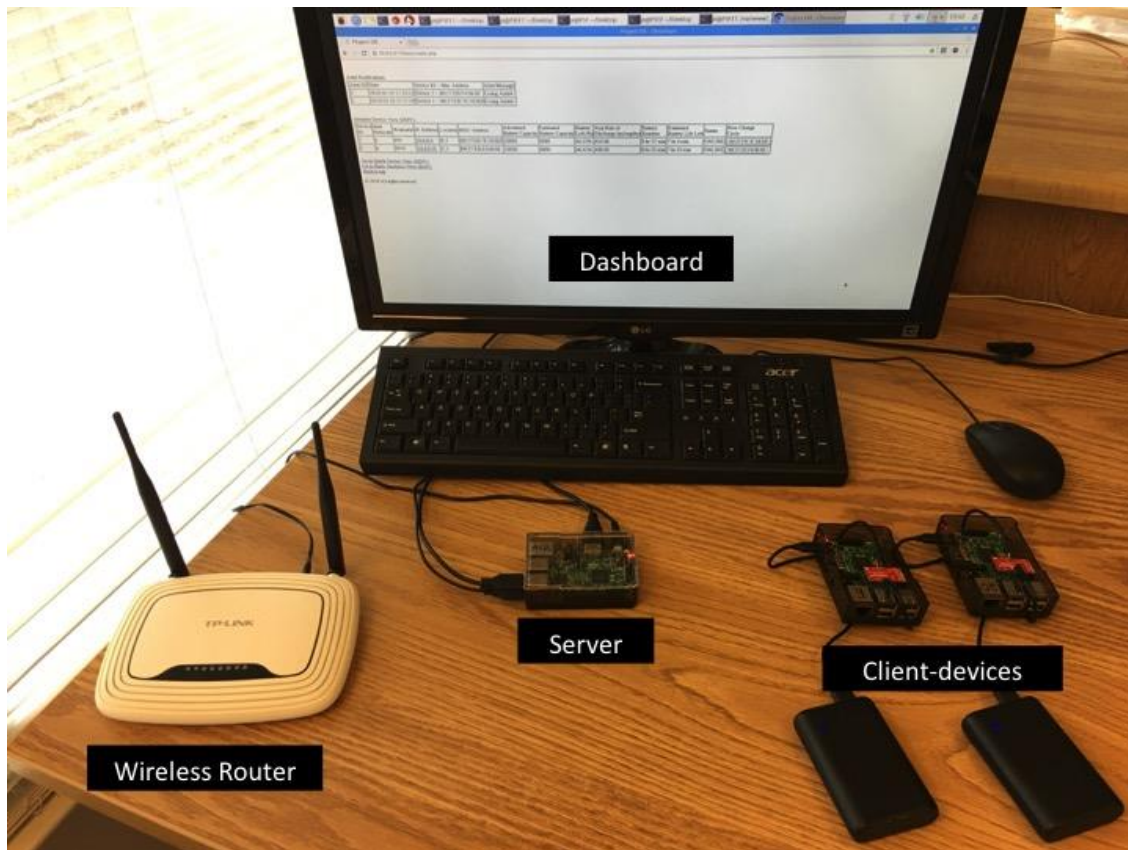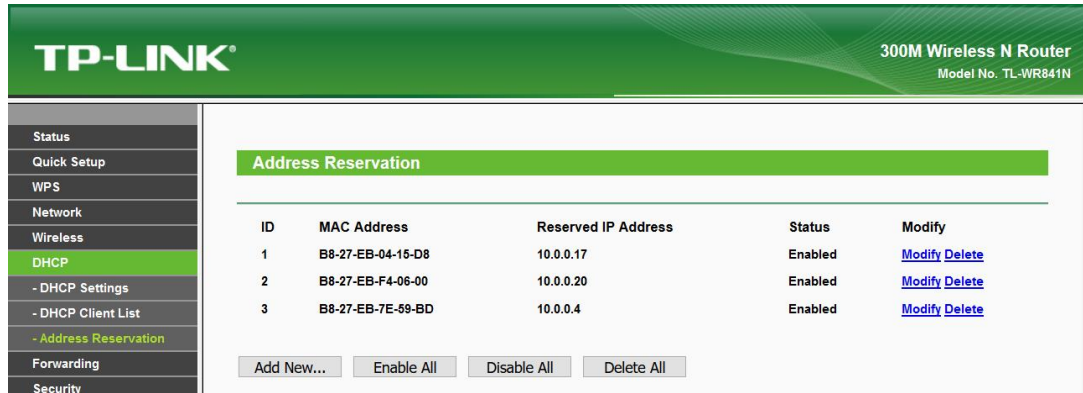
Figure 29.   Energy Monitoring System

1.   Wireless Router Setup

In this experiment, we made use of TP-Link's TL-WR841N as the wireless router for the wireless network setup. The wireless router allows the devices and server to communicate with each other. For better access control of the devices connecting to the network, IP addresses were assigned by the wireless router, according to reservations made with respect to each device's MAC address. (See Figure 30.)

Figure 30.   Wireless Router Configurations

2.   Server Implementation

The server is the host for the dashboard and also the point of central management for the data collected. For this experiment, we used a Raspberry Pi 3B for the server. Unlike the client device, the server required no modification. The main requirements were the need for the database to store device information, for energy readings, and for web programs such as PHP and Apache to support the dashboard.

3.   Client Device Implementation

The client device acts as a typical IoT device in an IoT network, which does not have the ability to self-monitor its energy level. In our implementation, we added the feedback mechanism by fitting a Coulomb Counter (LTC 4150 to Raspberry Pi 3B) to the device. In addition, we also built a CCM for communication with the server and a web

interface to display real-time information. The following description provides more details on how the client device was implemented.

For this experiment, we used a Raspberry Pi 3B, otherwise referred to as the device under test (DUT); a Coulomb Counter (LTC 4150); USB cable; and the female header for the client device. The pin-out details for the DUT and LTC 4150 are shown in Figures 31 and 32.

Figure 31 shows the general purpose input/output (GPIO) pin-out used for the DUT. For this experiment, we used the pins 16, 20, 21, and one ground pin as circled in red to interface with the LTC 4150.
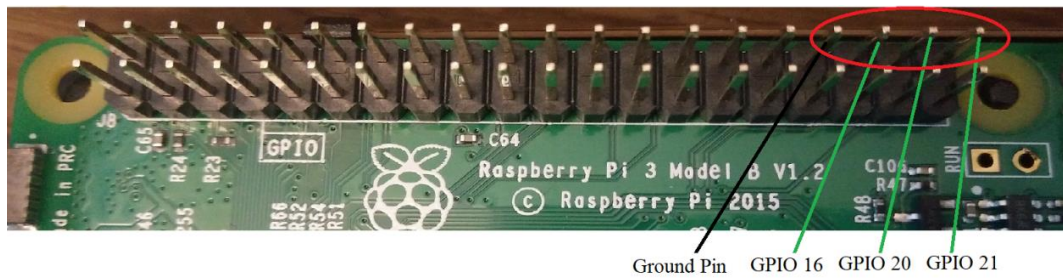
Figure 31. Raspberry Pi 3B GPIO Pin-Out Used

Figure 32 shows the pin-out for LTC 4150. We used the GND, POL, INT, and VIO pins as circled in RED to interface with the DUT. Both the IN and OUT pins, as circled in GREEN, were connected to both the battery and the DUT, respectively. Pin-out mapping of the LTC 4150 and DUT is illustrated in Table 2.
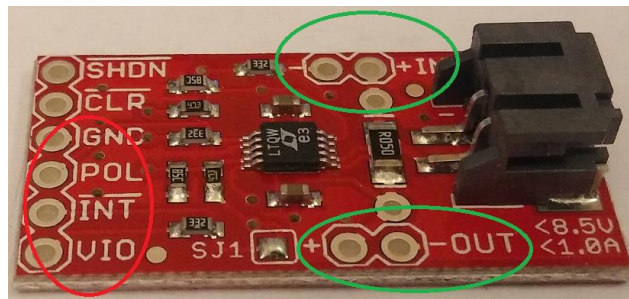
Figure 32. LTC 4150 Pin-Out Used

Table 2.   Pin-Out Mapping of LTC 4150 and Raspberry Pi 3B. Source: [44].

LTC 4150	Raspberry Pi 3B	Purpose
IN	-	Input power from Battery
OUT	DC 5V	Output power to Raspberry Pi
VIO	GPIO 21	Voltage reference for output pins
INT	GPIO 20	Interrupt pin that will flip to low when 0.1707 milliamp-hours have passed through the LTC 4150
POL	GPIO 16	Polarity pin to indicate charging or discharging of current.
GND	Ground Pin	Ground pin.

Figure 33 depicts the modification of the USB cable (cut in halves and wires stripped) for fitting into LTC 4150 to allow inline measurement: Black wirings (for -ve connection) and red wirings (for +ve connection) were both soldered to the LTC 4150's IN and OUT pin-outs, as shown in Figure 34. A female header was also soldered to LTC 4150 for easy fitting to the DUT.
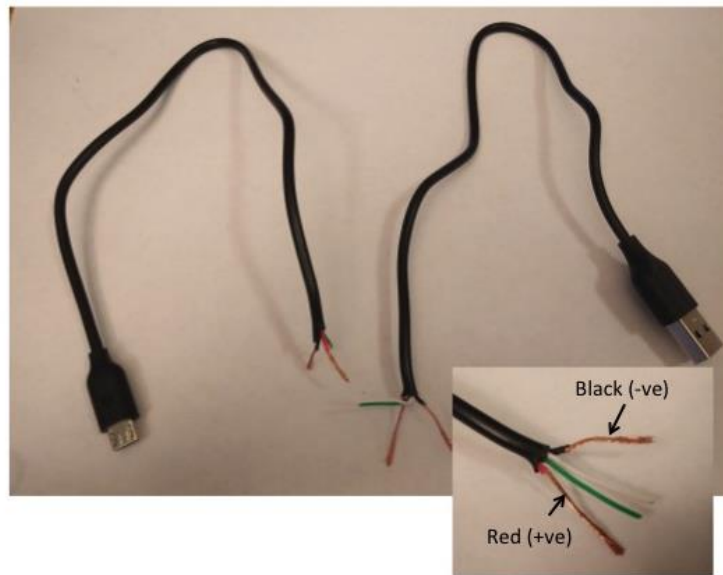
Figure 33.   Modified USB Cable
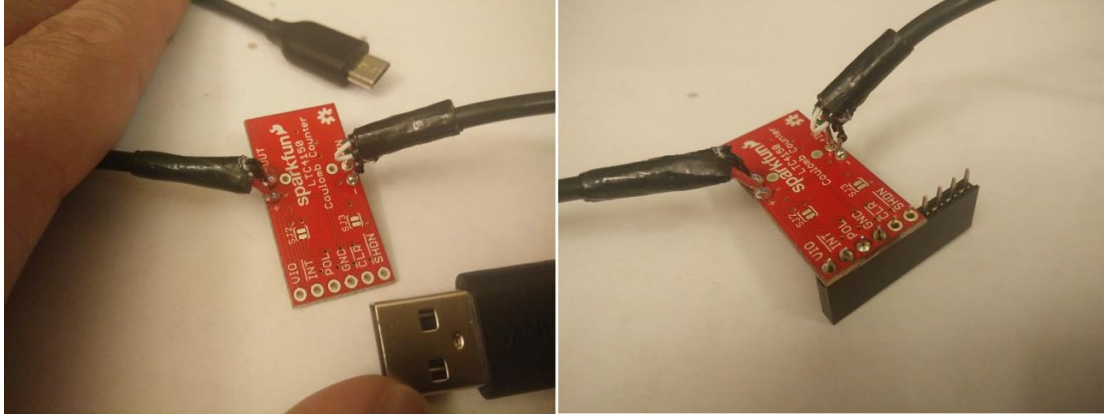
Figure 34.   Fitting the Modified USB Cable into LTC 4150 with
a Female Header

Figure 35 shows the actual fitting of the LTC 4150 to the Raspberry Pi 3B, for which the connection diagram is provided in Figure 36.

Figure 35.   Actual Fitting of LTC 4150 into Raspberry Pi 3B with Anker
PowerCore 10000
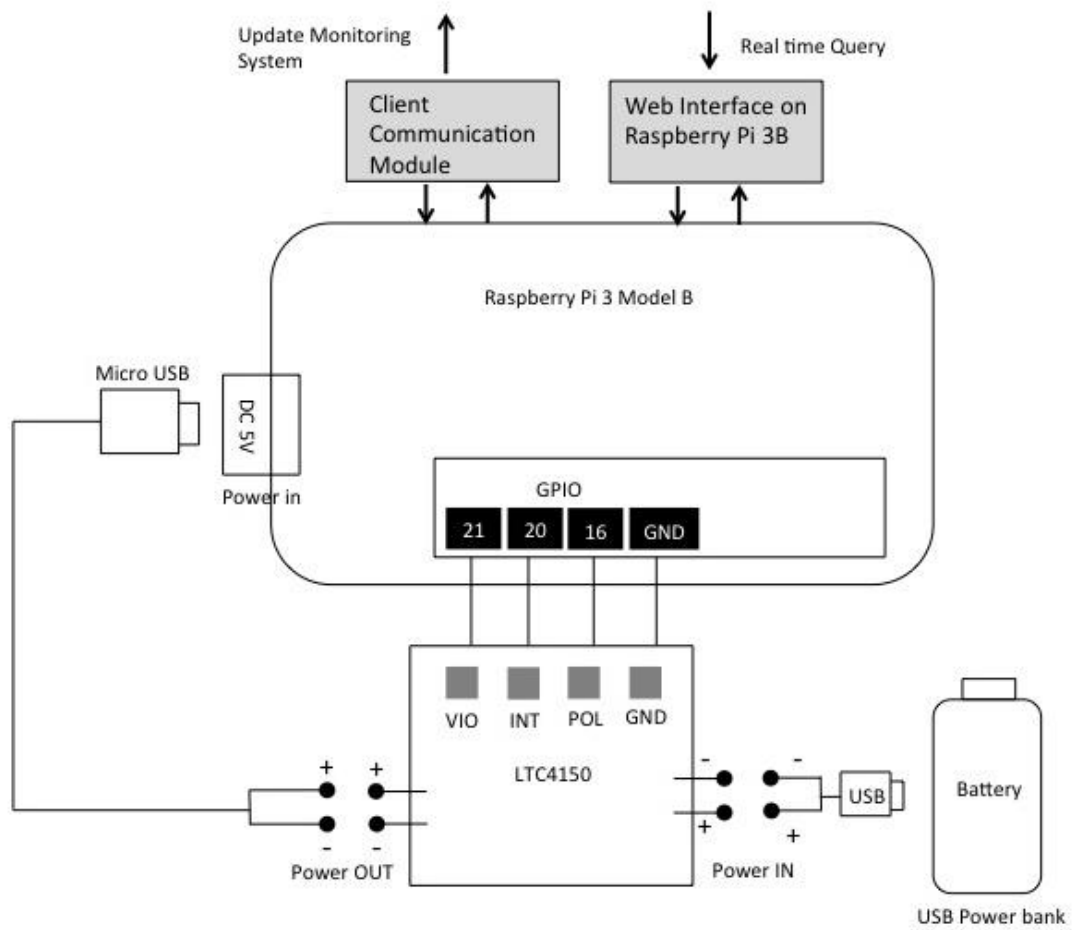
Figure 36.   Connection Diagram of LTC 4150 and Raspberry Pi 3B

D. COMMUNICATIONS MODULES

Figure 37 illustrates communications flow between the client device and the server through the Communication Modules.
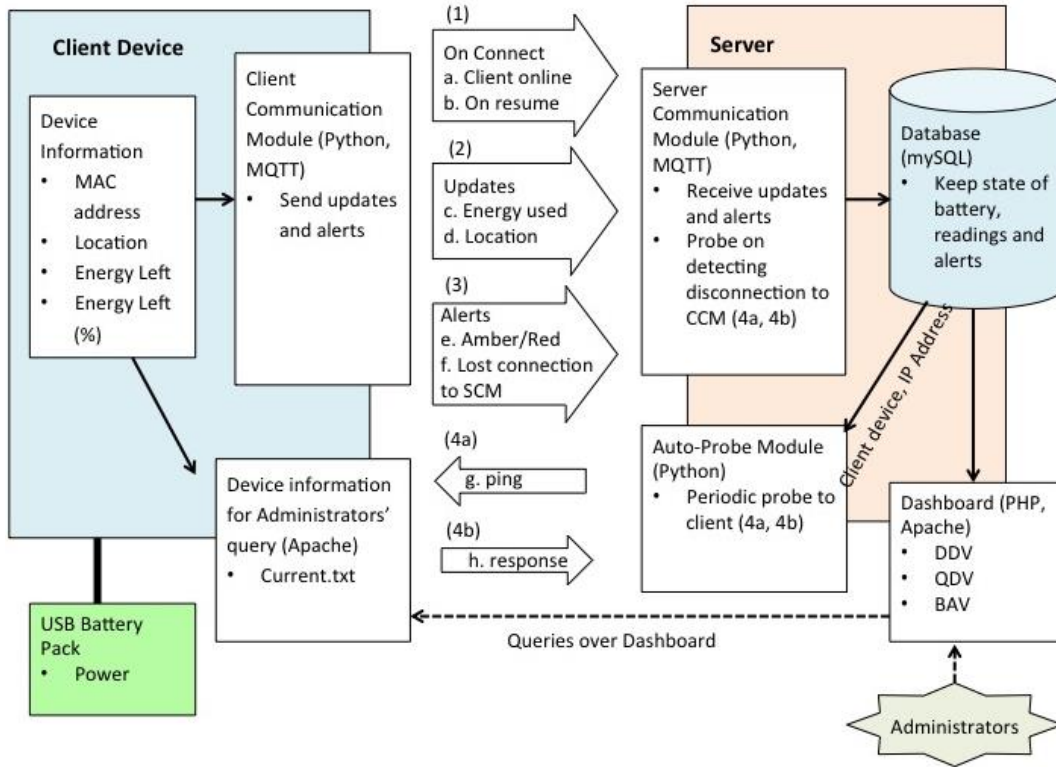
Figure 37. Communications Diagram

In the client device, important information such as the MAC address, location, and energy remaining are captured as an energy reading or an energy message by the CCM for updating the server. The energy readings, with other client device information such as system load, are also available through the client device's web interface for direct querying. There are three types of updates from the CCM. The first type (1) is to update the server that the client device is connected to and to check whether the connection is to resume a previous measurement. The second type (2) is to update the server with the energy readings. The third type (3) is to update the server on the change of energy states (e.g., GREEN to AMBER) or disconnection from the server.

In the server, the SCM is tasked with receiving the updates from the CCM and updating the database for data management. An APM checks the ONLINE/ OFFLINE status of the client devices using the Ping and Response (4a, 4b in Figure 37). For this implementation, APM is set to run at every five-minute interval. Finally, the dashboard is the main platform to display useful information to the network administrator for device energy status monitoring. The following sub-sections describe the building of the CCM, SCM, and APM.

1.   Building the Client Communication Module

In this sub-section, we explain how the CCM was built. The CCM provides the communication channel for the client device to send data to the server, allowing the server to collect updates. The three components for building CCM include the Python3 for the main program, an MYSQL connector for querying the server database, and MQTT for sending energy readings to the server. We also upgraded Raspbian to the latest version (dated November 29, 2017) to ensure support of the various sub-dependencies needed. This module's flow diagram is illustrated in Figure 26 of Chapter III.

The autonomous functionalities built into the CCM were implemented using Python3 and full coding of the CCM can be found in Appendix A. Key functionalities are described here.

(1)   Measurement of Energy Used

The LTC 4150 measures the energy used by counting ticks. Each tick triggers the INT pin to go low when 0.1707 milliamp-hours have passed through the Coulomb Counter. This means that a full capacity battery of 1,000 mAh will generate approximately 5,858 ticks until the energy in the battery is depleted [44]. Sending of measurement reports was tuned to 50 ticks per interval to limit network traffic and yet remain effective for monitoring.

The following code snippet, shown in Figure 38, was used to setup the GPIO of the Raspberry Pi for the INT, POL, and VIO pins, and to trigger an event when detecting

a LOW from the INT. Guidance on how to configure the pins for Python can be found in [33] and [47].

```
# GPIO initialized
GPIO.setmode(GPIO.BCM)
GPIO.setup(INTERRUPT, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(POLARITY, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(VIO, GPIO.OUT)
GPIO.output(VIO, GPIO.HIGH)

# Event trigger when INTERRUPT goes LOW
GPIO.add_event_detect(INTERRUPT, GPIO.FALLING, callback=send_tick)
```

Figure 38.   Python Code Snippet to Setup the GPIO Needed.
Source: [33], [47].

(2)   Reading of MAC Address

Each energy reading was sent together with the device's unique identifier to the server for identification. Implementation was via the Python3's UUID library [48].

(3)   Reading of Location Information

In addition to the MAC address, location information is also an essential piece of information sent to update the server during each energy update. Implementation was through reading a file that contains a pre-defined location of the client device.

(4)   Message Passing to the Server

To achieve communications between the client devices and the server, we adopted the message passing approach as the underlying mechanism to deliver both the energy updates and the alerts. Implementation is via the MQTT protocol, where the client device pushes the energy updates and alerts as messages to the Server via the MQTT's publish function. Leveraging MQTT's advantage of pre-configuring a disconnection message as described in Section 2B of Chapter II, our CCM is equipped with the ability to notify the server even when the client devices are abruptly disconnected.

(5)   Generating Energy Alerts

In addition to generating the routine updates as described in (1) to (4), it is important to alert the administrator immediately when a client device is changing its energy state to either Amber or Red. We achieved this by auto-computing the threshold level of each energy state and updating the server once the client device reaches these thresholds.

(6)   Generation of Connection Alerts

At the same time, the administrator should also be immediately notified of the client device connectivity status when it is disconnected or goes offline. Implementation of this functionality is via the MQTT's Last Will and Testament feature where a pre-configured offline message is automatically sent when MQTT detects a disconnection.

(7)   CCM as a Service

As shown in Figure 26 of Chapter III, CCM is built to continuously loop for energy readings until the client device goes offline or is drained of battery energy. Together with auto-start on boot, CCM can fully function as a service from boot-up without any administrator intervention.

(8)   Resumable Module

In the situation where the client device goes offline and loses connection to the server, the CCM is able to recall its last measurement by doing a quick query to the server via the MYSQL connector. This design allows for a more lightweight client implementation where the client device does not need to maintain the previous readings through additional programs.

2.   Server Communication Module

The SCM allows the server to receive data from the client device. Similar to the CCM, the three components for building the SCM were Python3, MYSQL, and MQTT. The module's flow diagram is shown in Figure 27 of Chapter III.

The functionalities built into the SCM are described in the following sub-sections. Implementation was accomplished using Python3. Full coding of the SCM can be found in Appendix B. The database schema can be found in Appendix C.

(1)   Receiving Readings of the Energy Used

Energy readings are periodically sent to the server together with the client devices' MAC address and location. Implementation is via the MQTT protocol for interoperability with the CCM. In this experiment, the server was configured to receive the energy readings via the MQTT's subscribe function. Upon receiving the readings, they were updated directly to the server's database. Figure 39 shows the message format of the energy readings received.

Figure 39.   Message Format of Energy Readings Received

(2)   Receiving Alerts

This functionality makes use of the same MQTT messaging mechanism. The main difference is the SCM's ability to distinguish the alerts from the routine energy updates. We achieved this by tagging the various alerts at the client device with simple encodings for filtering at the SCM. Once received, alerts also updated the database for display on the dashboard as alert notifications. Figure 40 shows the message format of the alerts received.

Figure 40.   Message Format of Alerts Received

(3)   Auto-Ping Client Device on Disconnection

This is a proactive functionality upon detection of disconnection from the client device's MQTT connection. The SCM will autonomously ping the client device to determine whether it is alive, and report the status accordingly to the Alerts Notifications.

(4)   SCM as a Service

Similar to CCM, SCM is implemented as a service by auto-starting upon boot and constantly monitoring for any energy events without any administrator intervention.

3.   Auto-Probe Module

The APM is necessary to monitor the online status of client devices in the network. The probe frequency is implemented by using Raspbian's CROND scheduler and can be fine-tuned to the administrator's requirements. The module updates the server on the online/offline status of the client device after the probe. The program flow of this module is as shown in Figure 28. Implementation is in Python3. Full coding of the APM can be found in Appendix D.

E.   WEB-BASED GUI DASHBOARD

The web-based GUI is the main component of this experiment; it enables the administrator to have access to multiple management functions through various view screens. The following sub-sections describe the building of the dashboard to display information for the CCM, SCM, and APM.
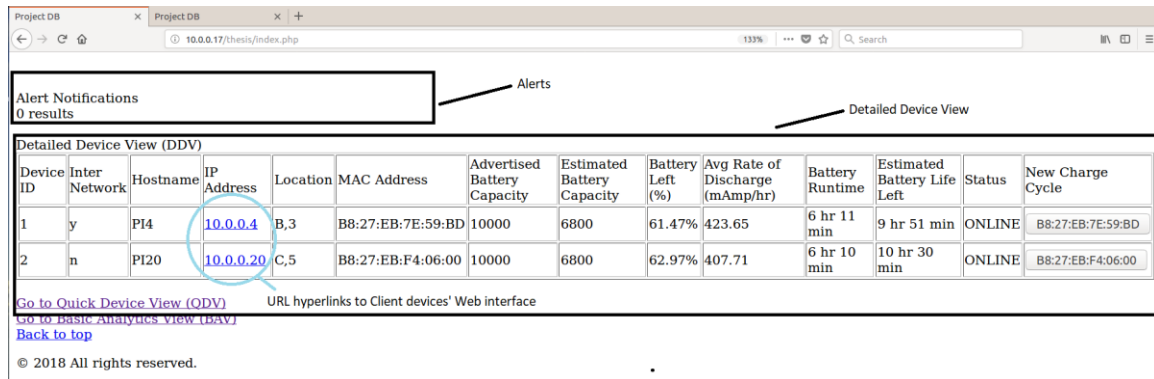
1.   Web Interface

The web interface in this concept demonstrator is designed to allow the administrator to have a clear overview of the energy level of each of the IoT devices in the large-scale IoT network and be able to identify any problem area quickly. The core features of this interface include the organization of IoT devices into a structured view for overall monitoring, display of alerts regarding any energy event or connectivity issues, a simplified view of devices for pinpointing problem areas quickly, and display of the trending of discharge rates for analytical purposes.

The main component for building the web interface is the Apache web program in the server. Development of the dashboard was performed entirely using the PHP scripting language. Full coding of the dashboard implementation in PHP can be found in Appendix E. The following sub-sections provide more information regarding the implementation of the various views of the dashboard.

(1)   Main Page: Detailed Device View and Alerts

Figure 41 shows the main page of the dashboard. It consists of two areas as highlighted; the DDV and Alerts. They are configured to auto-refresh at periodic intervals for autonomous monitoring.

Figure 41.   Main Page: Detailed Device View and Alerts

Under the DDV, important information such as the amount of energy left and connectivity status is reflected. It also contains other essential information such as the device's location and whether it is an inter-network device. This information further assists the administrator in terms of prioritizing and identifying any problem area. The hyperlink implementation in the IP address column also allows the administrator quick access to the client device's web interface to query for real-time readings. On the extreme right of each row, there is a button for the administrator to reset the battery's reading during every new charge cycle. This is a method to overcome the limitation of the Coulomb Counter, which does not have the ability to determine the full capacity of the

52

battery (as explained in Section C5 of Chapter II) and hence makes the Coulomb Counter unable to recognize whether the measurement was the result of a new charge cycle.

The other area of alerts notification displays real-time messages for notifying the administrator of incidents that require immediate attention (e.g., Device 2's energy level is critically low). It enables the administrator to be more responsive. The alert types included Amber (device has used up more than 40% of its energy); Red (device has used up more than 80% of its energy); and Black (device has lost connection to server or is offline).

(2)   Quick Device View

The other view on the dashboard is the QDV. This view displays devices in a tile manner as shown in Figure 42. Each device displays essential information such as the percentage of energy left and the location of the device. The simple color cues provide the administrator with a quick overview of the device's health status as shown in Table 3. Inter-network client devices are also marked with a red border as depicted in Figure 42. This allows the administrator to identify and prioritize problem areas efficiently. The QDV also auto-refreshes at periodic intervals for autonomous monitoring.
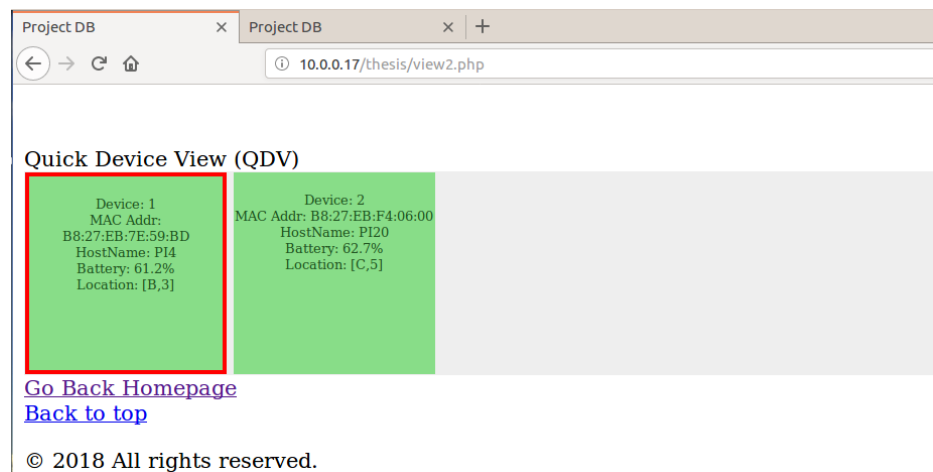
Figure 42. Quick Device View

53

Table 3.   Color Cues to Indicate Battery Level and Device Offline.

Color	Battery Level left
Green	61% - 100%
Amber	21% - 60%
Red	1% - 20%
Black	Offline

(3)   Basic Analytics View—Discharge Rates Line Chart

In addition to both DDV and QDV, the administrator also has the option to perform some basic analytics from the dashboard's Basic Analytics View (BAV). As seen in Figure 43, the BAV tracked the discharge rate of the two client devices in the line chart. Discharge rate is expressed in milliampere-hours. A higher discharge rate implies a higher system load on the device and vice-versa. This capability is implemented using an open source JavaScript graphing tool, Chart.js [49], [50].
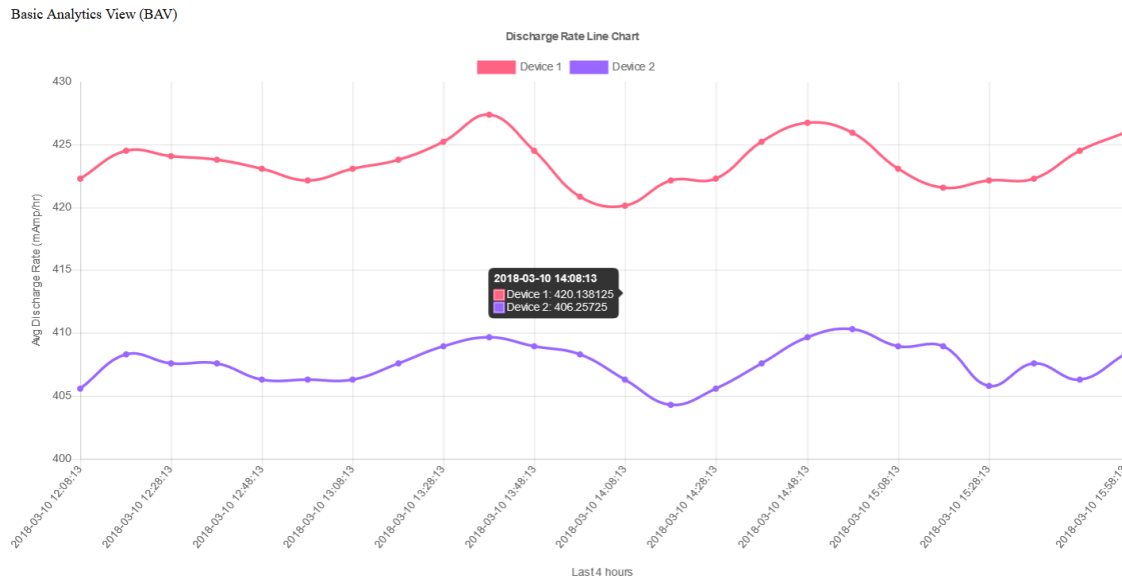
Figure 43.   Discharge Rate Line Chart

The duration of monitoring and interval length can be fine-tuned by adjusting the two variables in the PHP script, view3.php, as shown in Figure 44. The meanings of these two variables are self-explanatory.

```
$TotalIntervals=48;
$minsPerInterval='-10 minutes';
```

Figure 44.   Configurable Intervals and Interval Duration for Line Chart

F.   SYSTEM TESTING

In this experiment, we tested the various hypotheses mentioned in Chapter I. The implementation was tested for accuracy, stability, and its functionalities to ensure that our research objectives were met. The following sub-sections describe the various tests conducted after the implementation of our EMS.

1.   Determining the Actual Capacity of the Batteries

There were two objectives in determining for the actual capacity of the batteries used for the evaluation. The first objective was to determine the ability of the EMS to calculate the amount of energy left in the client devices based on measuring the energy consumed. The second objective was to assess the accuracy of our EMS through repeated testing.

To determine the actual capacity of the batteries, we measured the number of ticks generated by the LTC 4150 during the batteries' full discharge. This test was run multiple times to test for consistency and to verify whether the readings were within the theoretical range. We also made use of the battery capacity at 5V, as shown in Figure 13, to estimate the conversion loss of the batteries tested. Table 4provides the test results for the two (Anker PowerCore 10000) batteries used. Each full discharge test was done using the same Raspberry Pi without additional load and lasted about 16 hours.

Table 4. Full Discharge Test on the Two Batteries Used.

	Battery 1				Battery 2		
Test	# of ticks	Equivalent to	% Avg. to mean	Test	# of ticks	Equivalent to	% to Avg. mean
1	39918	6814.00 mAh	100.04%	5	40400	6896.28 mAh	100.56%
2	39850	6802.40 mAh	99.87%	6	40115	6847.63 mAh	99.85%
3	39987	6825.78 mAh	100.21%	7	39870	6805.81 mAh	99.24%
4	39855	6803.25 mAh	99.88%	8	40312	6881.26 mAh	100.34%

Avg. Mean	39902.5	6811.36 mAh		Avg. Mean	40174.25	6857.75 mAh

(7400 - 6811.36) / 10000 = 5.89% Conversion loss

(7400 - 6857.75) / 10000 = 5.42% Conversion loss

As may be concluded from the results in Table 4, the number of ticks in each measurement was consistent with less than one percent variance from the average mean. The advertised capacity of the Anker batteries is 10,000 mAh. As such, voltage conversion, as described in Chapter II, 5V yields 7400 mAh. Estimated conversion loss was around 6 percent of the advertised capacity for both batteries used. This value was well within the range of the industry standard of 2 percent to 15 percent as explained in Section C6 of Chapter II. As such, the actual capacity was assessed to be 68 percent of the advertised battery capacity throughout the experimentation.

2. Verifying the Granularity of the Tested Coulomb Counters

One key factor in achieving a more precise and reliable monitoring system, lies in the granularity of energy measurement. To verify the granularity of the tested Coulomb Counters, two client devices were measured for the number of coulombs remaining over time until each battery was fully discharged. Figure 45 illustrates the outcome of this test.
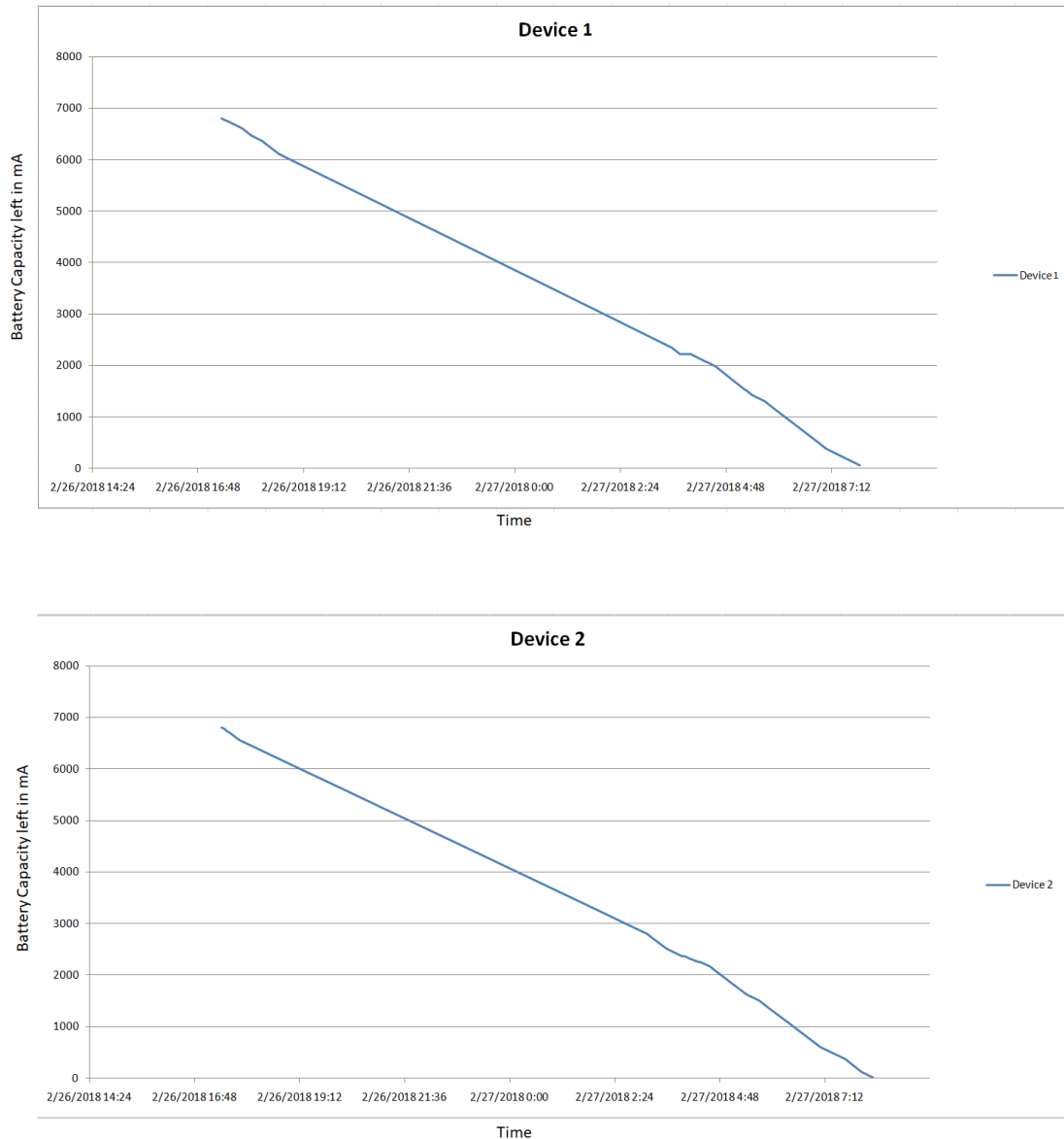
Figure 45. Discharging of Energy Over Time in Both Tested Client Devices

As seen in Figure 45, we were able to measure the energy used over time throughout both client devices' entire discharging cycle. This is unlike the voltage measurement approach in which the voltage level does not follow a linear relationship against the SoC, as seen in Figure 11.

3. Accuracy of Measurement

Another experiment goal was establishing the accuracy of measurements. After calibrating the monitoring system to the calculated capacity of the tested batteries, two client devices were setup to be powered by two fully charged batteries. The objective was to measure the error rate between the calculated capacity and the measured capacity. Figure 46 shows the result captured by our EMS.

View 1 :: Nodes Info

Node ID	InterNetwork	Hostname	I.P.	Location	Mac ID	Advertised Batt Cap	Estimated Batt Cap	Battery Left (%)	Avg Rate of Discharge (mAmp/hr)	Battery Runtime	Estimated Battery Life left	Status
1	y	PI4	10.0.0.4	B,2	B8:27:EB:7E:59:BD	10000	6800	0.97%	464.37	14 hr 30 min	0 hr 8 min	OFFLINE
2	n	PI20	10.0.0.20	C,5	B8:27:EB:F4:06:00	10000	6800	0.34%	456.07	14 hr 51 min	0 hr 3 min	OFFLINE

Figure 46.   Error Rate of the Energy Monitoring System

As seen in Figure 46, both devices had an error rate of less than one percent. This result reflects that the calculated capacity settings were, in fact, very well calibrated and close to the energy capacity in the tested batteries.

4. Stability of Communication Modules and the Ability to Resume after Disconnection

In a typical IoT network, IoT devices go offline due to the failure of hardware and/or depletion of energy. In addition, wireless IoT networks are often sporadic in terms of connectivity, resulting in frequent disconnection of the IoT devices. It is therefore important to ensure the reliability of the communications modules to keep the server updated on any disconnection from the client devices. In the event that a client device is temporarily taken offline, it is also important that the client device resumes its energy monitoring upon reconnection.

We also had two objectives for this assessment. The first was to ensure that the communication mechanism developed is stable without intermittent disconnection between the client devices and the server. Any disconnection impacting the message passing protocol would be reflected via the alerts displayed in the dashboard. The second objective was to test whether communication could be resumed after a disconnection from the server.

This test was conducted by connecting two client devices to the server. The power supply was disconnected abruptly after ten hours to simulate a hardware failure. The power was restored and the implementation was expected to resume the energy readings from where the device left off. Figure 47 shows the outcome from this test.

Figure 47. Resuming Energy Monitoring after Disconnection

As seen in Figure 47, the client device connection remained stable for the entire ten hours. The abrupt disconnections were displayed as alerts and the energy readings were not disrupted by the temporary disconnections.

5. Autonomous Monitoring of Client Device Energy Level

One important objective of the EMS is to reduce the administrator's workload in monitoring a large number of IoT devices. As such, this EMS was built to enable autonomous monitoring through the various functionalities implemented. This portion of the test encompassed the monitoring of two client devices in terms of their energy level through both the dashboard's DDV and QDV without the need for the administrator to intervene. Figure 48 illustrates the corresponding view between the DDV and QDV with respect to monitoring of energy levels.

Figure 48.   Autonomous Monitoring of Energy Level in Both DDV and QDV

Key observations made during the test are as follows:

(1)   Both the DDV and QDV were able to consistently track and display the depleting energy level of the devices in real time.

(2)   In the DDV, both the Battery Runtime and estimated time left gave the administrator a good estimate on the performance of the battery.

(3)   In the QDV, the administrator was able to quickly differentiate which devices are low on energy and which devices have gone offline.

(4)   Overall, both DDB and QDV were able to autonomously provide the administrator with a clear overview on the energy status of the devices as seen in Figure 48.

6.   Rapid Identification of Problem Areas

Other than providing a clear overview of the client devices' energy levels, it is also essential that administrators be able to identity problem areas quickly. As discussed in Section E2 of this chapter, the QDV serves this purpose. This test evaluated the different color cues implemented by setting two client devices to transit through the different energy states. Figure 49 showed that the various color cues would give the administrator a quick way to identify client devices that require attention. Combined with the location information embedded in each device, the administrator was able to pinpoint the problem area quickly.
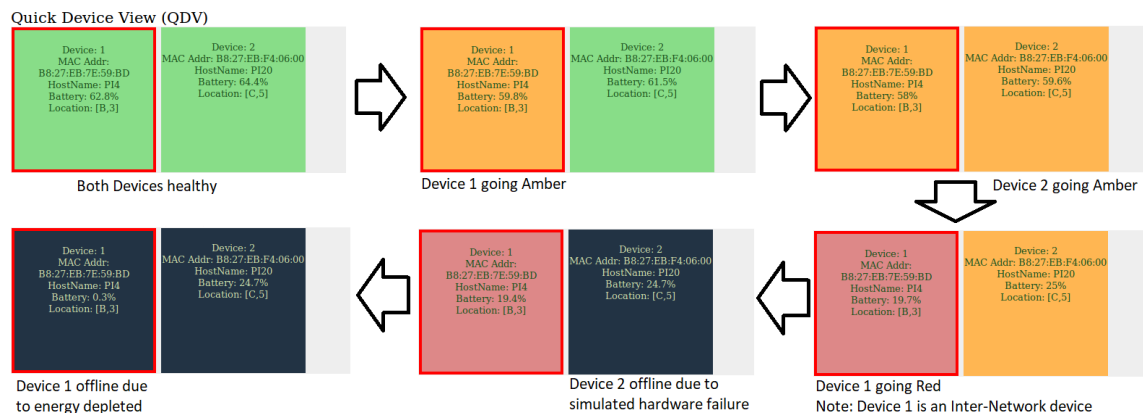
Figure 49.   Color Cues and Location Information

7.   Alerts Monitoring

Alerts monitoring provides the administrator with a quick overview of the events at-hand and hence enables the administrator to respond proactively to any IoT device that requires attention. This assessment was conducted by setting up one client device to test all the built-in alerts available. As seen in Figure 50, our EMS was able to detect all state changes, generate appropriate alerts, and display them on the dashboard.

Alert Notifications

Alert ID	Time	Device ID :: Mac Address	Alert Message
29	2018-03-07 22:48:29	Device 1 :: B8:27:EB:7E:59:BD	Battery Drained
28	2018-03-07 20:45:10	Device 1 :: B8:27:EB:7E:59:BD	Auto Probe update: Device 1 is OFFLINE
27	2018-03-07 20:37:05	Device 1 :: B8:27:EB:7E:59:BD	Going Red
26	2018-03-07 12:04:30	Device 1 :: B8:27:EB:7E:59:BD	Lost connection with Device: B8:27:EB:7E:59:BD
25	2018-03-07 12:01:10	Device 1 :: B8:27:EB:7E:59:BD	Lost connection with MQTT server
24	2018-03-07 14:30:04	Device 1 :: B8:27:EB:7E:59:BD	Going Amber

Figure 50.   Testing of Built-In Alerts

8.   Direct Query to Client Device from the Dashboard

A hyperlink is available on the dashboard to access the client device directly if the administrator would like to query the client device for real-time data; Figure 51 shows the location of the hyperlink in the dashboard and a resulting web display of the real-time data such as the Hostname, MAC address, Uptime, System load, Location and Energy left.

Detailed Device View (DDV)

Device ID	Inter Network	Hostname	IP Address	Location	MAC Address
1	y	PI4	10.0.0.4	B,3	B8:27:EB:7E:59:BD
2	n	PI20	10.0.0.20	C,5	B8:27:EB:F4:06:00

```
← → C ⌂                    ⓘ 10.0.0.4/current.txt

Hostname: PI4
MAC address: B8:27:EB:7E:59:BD
Current Time: 2018/03/11 18:02:51
Uptime: 1 min,
System average load (1 min, 5 min, 15 min): 0.85, 0.32, 0.12
Current Location: B3
Energy left: 6794.7083 mAh
Energy left(%): 99.92 %
```

Figure 51.   Hyperlink to Client Device's Web-based Interface

9.   Basic Analytics View

It is important to detect the different workload experienced by a client device. This improves the usability of the EMS and allows the administrator to take pre-emptive actions for better energy management. The workload of a client device can be observed from its discharge rate. A higher discharge rate would reflect a higher workload while a lower discharge rate would correspond to a reduced workload. In this test, we generated three different levels of system load by using the Python2's multiprocessing library and tested our EMS's ability to detect the changes from the (1) baseline system load to the (2) half system load to the (3) full system load and (4) normalized back to the baseline. Full coding of the Python load generating program can be found in Appendix F. Figure 52 illustrates the four levels of system load tested, and Figure 53 shows the corresponding load detected in the BAV during the 30-minute test.
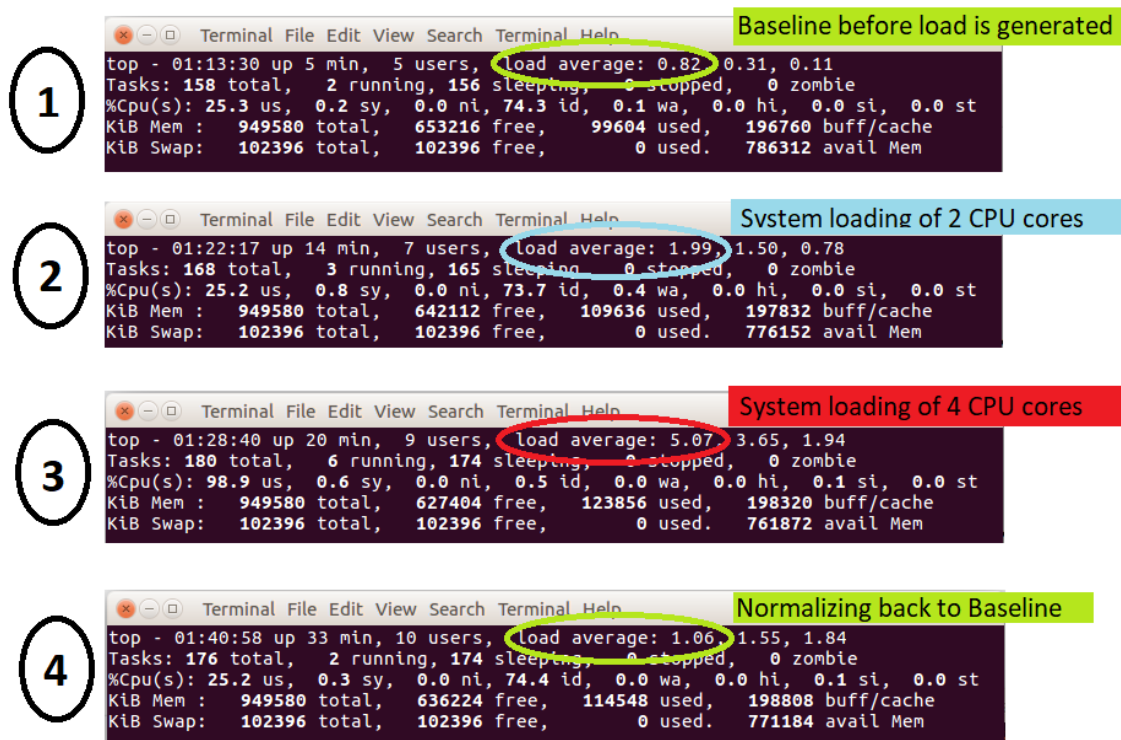
Figure 52.   Four Different Level of System Loads Tested

63
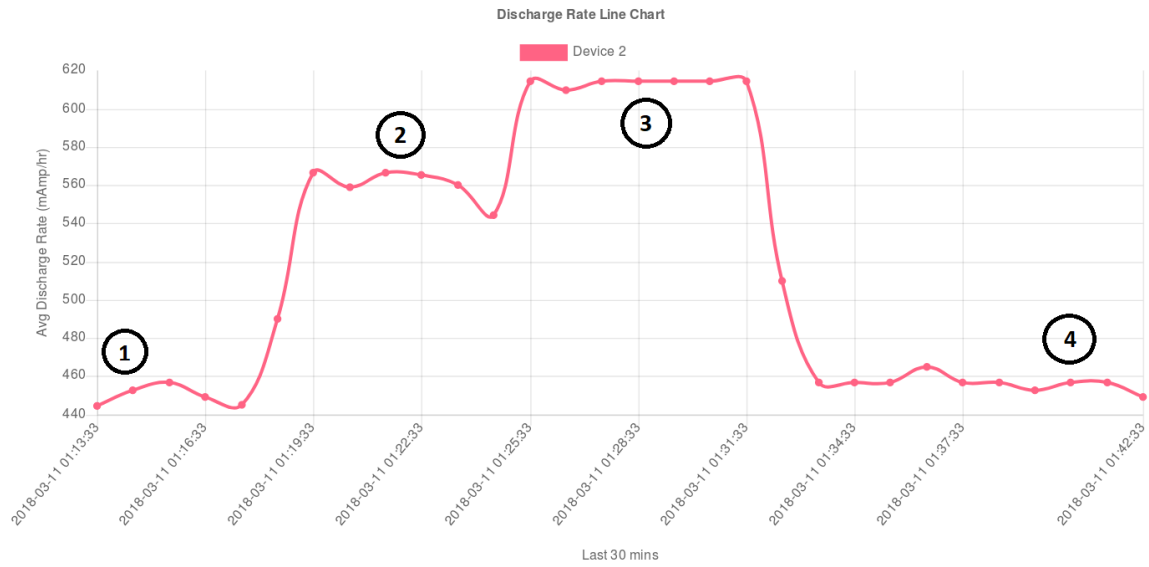
Figure 53.   Corresponding Load Detected in the BAV

As seen from Figure 53, our EMS was able to track the different system loads in the client device from the BAV. This allows the administrator to perform basic trend analysis regarding the correlation of system load and energy usage by the IoT devices.

10.   Auto-Probing Client Devices

One important function of the server is the ability to periodically check the availability of the client devices and update the database with the results. The APM is responsible for this function and was tested against two client devices. The results are presented in Figure 54.
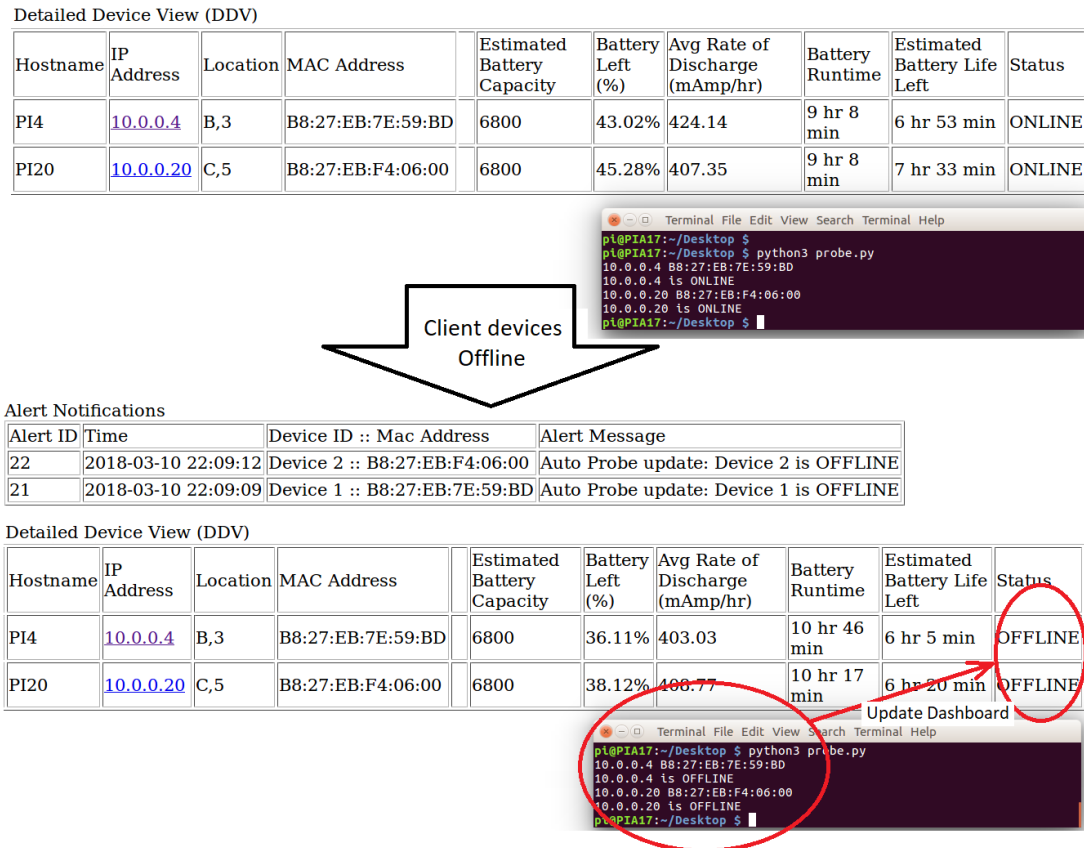
Figure 54.   Test Results of the APM on Two Client Devices

As shown, the APM was able to accurately detect the availability status of the client devices and the dashboard was updated accordingly.

11. Stability and System Load of the Communication Modules

An important consideration of the EMS is that the applications must not crash or cause the devices and server to crash. The tests conducted in Sections F1, F2, and F3 of this chapter demonstrated the stability of the communication modules, as both communication modules did not crash during the long discharging test of the batteries.

Another consideration was the system load produced by the communication modules themselves. In this test, the system load in both the client device and server were measured before and after the running of the communication modules. Figure 55 showed the screenshot of the load readings.

A. Client Device System Load - Baseline

A-1. Client Device System Load - With CCM

B. Server System Load - Baseline

B-1. Server System Load - With SCM
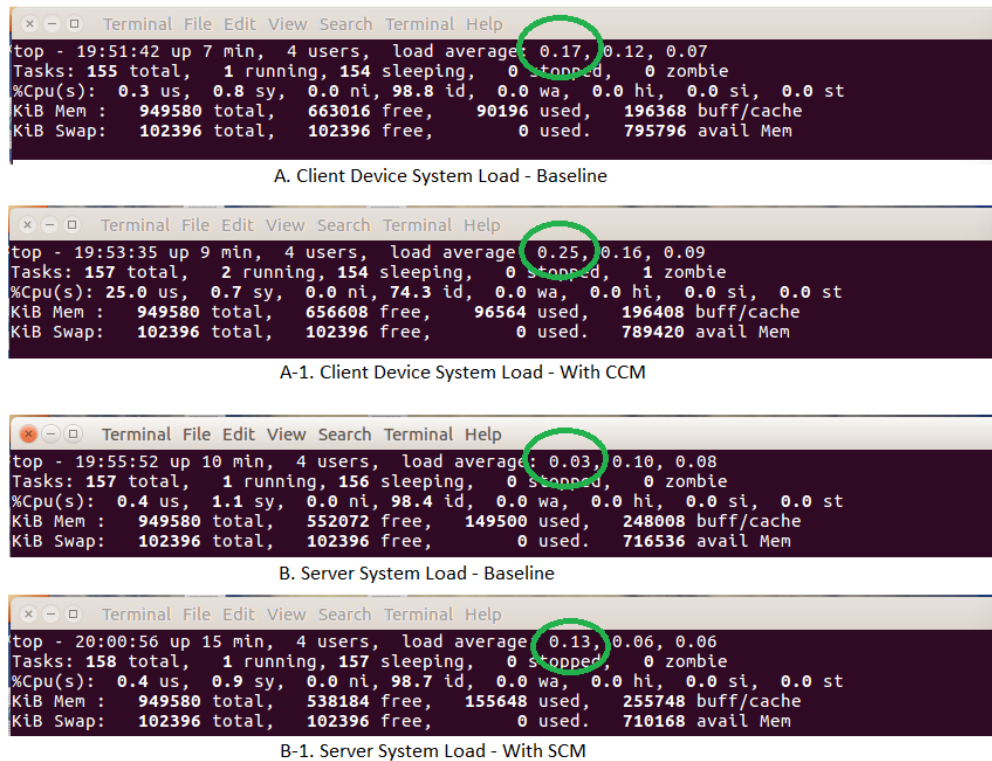
Figure 55. System Load of CCM and SCM in Both Client Device and Server

As seen in Figure 55, both CCM and SCM utilized only a small amount of system resources.

G.   CHAPTER SUMMARY

In this chapter, we have described the entire implementation and testing of the system. We have managed to complete the testing process with our implementation considerations in mind and achieved the results desired.

In Chapter V, we summarize the experiments done and the results achieved. We also discuss some of the future improvements that can build upon this research.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. RESEARCH SUMMARY

In large IoT networks, it is often challenging to ensure the uptime of each IoT device due to the scale and diversity of the network. In this research, we aimed to build an autonomous Energy Monitoring System (EMS) for such networks to enable an administrator to have a clear overview of the energy levels of IoT devices in the network. The developed monitoring system provided for a clear overview of the energy levels, as well as the location and essential information associated with each individual IoT device. Our EMS supports two-way interaction between an administrator and the IoT device by enabling the administrator to send queries to the devices and the IoT devices to send autonomous feedback to the server.

In this research, we set up a concept demonstrator using Raspberry Pi extended with a Coulomb Counter microprocessor. This system is powered by a USB Battery Pack. We have also designed a web-based GUI, which is hosted on the server for easy energy monitoring on a large IoT network.

B. PERFORMANCE

Our testing has demonstrated that it is feasible to build an EMS to support management of large IoT networks.

In this implementation, we have developed an application that allows an IoT device to identify itself and report its energy level to a centralized server. A hardware enhancement is also implemented by using a coulomb counter to enable the feedback mechanism for IoT devices that cannot monitor their own energy level.

In terms of communications between the IoT device and the server, we have developed a stable and resumable communication module, known as the Client Communication Module, and a Server Communication Module using the MQTT's message passing protocol as the underlying transport mechanism. In our experiment, we found that the MQTT's library in Python is well- built, allowing for good integration

between retrieving the energy readings from the hardware and pushing these readings over the network to the server.

Numerous functionalities were also designed to enable autonomous monitoring, thereby reducing the monitoring efforts of the administrator. For example, one function allows for the determination of the device's information and energy readings without the need of the administrator's intervention.

A web-based GUI dashboard was developed to give the administrator a clear overview of the IoT devices' energy level, as well as the ability to identify any problems quickly by using location and color cues. The dashboard also allows the administrator to prioritize problems based on the criticality of client devices. The communication function of the web-based GUI allows the administrator to directly query the IoT devices for real-time data and perform basic analytics based on the discharge-rate trends.

Furthermore, as depicted in Chapter IV, the implementation was tested over a long duration to ensure the stability and reliability of the results.

C.   LESSONS LEARNED

Energy monitoring is essential to maintaining an IoT network. At the start of this research, though, we realized that the heterogeneity of IoT devices makes it difficult to apply a single all-in-one solution to this task. Different devices have different methods of monitoring their energy levels, and in most cases, the devices themselves are unable to self-monitor. An efficient EMS solution that is able to cater to the heterogeneity of the devices is hence important in a large IoT network. We explored available solutions in the market, but none of them fully catered to the needs of our system. Hence, we decided to code our own program, which can provide the desired functionalities. Modifications to the hardware were also made to connect the devices.

During the experimental process, we discovered that the advertised battery capacity is often not the actual capacity in the USB battery pack. We had to calibrate this result so that the Energy Monitoring System would be accurate.

In the earlier stages of our experiment, we encountered several failures with open source applications. These failures not only impeded our progress of the development of our EMS, they also affected the system's overall stability. One key factor to stability is the lightweight approach adopted in our implementation whereby energy readings are crafted in small message format and properly tuned in the update intervals to reduce system and network load.

Another important lesson learned is that a good EMS needs to have a user-friendly interface. The interface must allow the administrator to perform a number of functions and give a clear overview of essential information such as the energy levels, location, and MAC addresses of devices to enable the administrator to work efficiently.

D.   RECOMMENDATIONS FOR FUTURE WORK

As the number of connected IoT devices continues to grow, we foresee that a more complex and comprehensive monitoring solution will be required. While this research has provided the basis for an EMS, further enhancements can be added to provide more functionality and features. Here we highlight some of the future enhancements that can be added to this solution.

(1)   Enable the Energy Monitoring System to have the ability to recognize a battery's new charge cycle, its actual capacity and its deterioration rate over time.

(2)   Enable the Energy Monitoring System to have the ability to identify each battery as a unique entity so that the batteries can be interchanged between devices. The current solution links each battery unit to one IoT device.

(3)   Equip the client device with temperature sensor, as ambient temperature is known to influence a battery's discharge rate. This will enable the administrator to co-relate the discharge rate with ambient temperature.

(4)   Equip the client device with Global Positioning System capability so that the administrator can identify the actual location of the device. The current solution uses file-based location information.

(5)   Optimize the accuracy of the Energy Monitoring System by further minimizing the error rate.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. PYTHON CODE FOR CCM

Filename: Client_device.py

```python
### Code references ###
## https://github.com/openwsn-berkeley/amp-o-meter
## http://www.steves-internet-guide.com/into-mqtt-python-client
## http://helloraspberrypi.blogspot.com/2016/02/python-to-get-mac-address-
using-uuid.html
## http://www.steves-internet-guide.com/mqtt-last-will-example

import sys
from uuid import getnode as get_mac
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import time
import mysql.connector as mariadb
import os
import socket

if sys.version_info[0] <3:
print('Python 3 needed')
    sys.exit(1)

# Hardware connections
interrupt = 20
polarity = 16
vio = 21

# Auto generate mac address
mac = get_mac()
macString = ':'.join(("%012X"% mac)[i:i+2] for i in range(0, 12, 2))
clientID = macString

# Auto get location info (Simulated)
#Location_Row = "B" #Simulate GPS location
#Location_Col = "2"
fh = open("location.txt,""r")
lines = fh.readlines()
Location_Row = lines[0].strip()
Location_Col = lines[1].strip()
fh.close()

# Per tick
tick_mAh = 0.1707 # in mAh

# Current counters with resumable ability
db = mariadb.connect(host='10.0.0.17', user='root', passwd='passw0rd',
db='projectdb')
cursor = db.cursor()
online_sql = 'UPDATE nodes SET status="%s" WHERE node_macaddr="%s"'% ('ONLINE',
clientID)
prev_sql = 'SELECT ticks_count, read_time FROM readings where node_macaddr="%s"
ORDER BY reading_id DESC LIMIT 1'% (clientID)
cursor.execute(online_sql)
cursor.execute(prev_sql)
row = cursor.fetchone()
if row is None:
    count = 0
```

73

```python
else:
    prev_tick = int(row[0])
    prev_time = int(row[1])
    count = prev_tick

# Generate effective capacity in mAh
SoH_sql = 'SELECT capacity FROM soh_readings where node_macaddr="%s" ORDER BY
soh_reading_id DESC LIMIT 1'% (clientID)
cursor.execute(SoH_sql)
SoH_row = cursor.fetchone()
if SoH_row is None:
    advertised_capacity = 10000
else:
    advertised_capacity = int(SoH_row[0])

db.commit()
db.close()

actual_capacity = 3.7/5 * advertised_capacity
conversion_loss =  0.06 * advertised_capacity #Tested that this battery's
conversion loss is around 6% on average
effective_capacity = actual_capacity - conversion_loss
full_capacity_count = int(round(effective_capacity / tick_mAh))
threshold_drained = int(round(0.99 * full_capacity_count)) #Used 99%
threshold_amble = int(round(0.4 * full_capacity_count)) #Used 40%
threshold_red = int(round(0.8 * full_capacity_count)) #Used 80%
threshold_drained = int(round(0.99 * full_capacity_count)) #Used 99%

# Other settings
broker = "10.0.0.17"
port = 1883
topic = "Energy/ticks"
last_will = clientID + "|"+ Location_Row + "|"+ Location_Col + "|"+ "L"
filename = "current.txt"
wd = "/var/www/html/"
hostnm = socket.gethostname()

def send_tick(_):
global count
    count += 1

# Set up the mqtt publisher. Only necessary arguments is the client name
## Client(client_id=","  clean_session=True, userdata=None, protocol=MQTTv311,
transport="tcp")
mqttp = mqtt.Client(clientID)

# Will setup. Must be behind client connect
mqttp.will_set(topic, last_will, 1, False)

# Server to publish to. Port is optional and by default: 1883
##connect(host, port=1883, keepalive=60, bind_address="")
mqttp.connect(broker, port)

# GPIO setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(interrupt, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(polarity, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(vio, GPIO.OUT)
GPIO.output(vio, GPIO.HIGH)

# Event trigger when INTERRUPT goes LOW
GPIO.add_event_detect(interrupt, GPIO.FALLING, callback=send_tick)
```

```python
# Put a timestamp to indicate when starting
print (time.strftime('%Y/%m/%d %H:%M:%S'))

# Running main program
try:
while True:
if GPIO.event_detected(interrupt):

if (count % 50) == 0:
                string_count=str(count)

# The only mandatory are topic and payload (aka message)
                ##publish(topic, payload=None, qos=0, retain=False)
msg = clientID + "|"+ Location_Row + "|"+ Location_Col + "|"+ string_count
                mqttp.publish(topic, msg)

if count == threshold_amble:
                msg = clientID + "|"+ Location_Row + "|"+ Location_Col + "|"+
"A"
mqttp.publish(topic, msg)

elif count == threshold_red:
                msg = clientID + "|"+ Location_Row + "|"+ Location_Col + "|"+
"R"
mqttp.publish(topic, msg)

elif count == threshold_drained:
                msg = clientID + "|"+ Location_Row + "|"+ Location_Col + "|"+
"D"
mqttp.publish(topic, msg)

else:
pass

            if (count % 10) == 0:
                uptime_cmd = 'uptime | cut -d"" -f4-6'
uptime = os.popen(uptime_cmd).read()
                sys_load_cmd = 'uptime | cut -d"" -f12-15'
sys_load = os.popen(sys_load_cmd).read()

                current_time = time.strftime('%Y/%m/%d %H:%M:%S')
                ampere_used = count * tick_mAh
                ampere_left = effective_capacity - ampere_used
                ampere_left_percent = ampere_left / effective_capacity * 100
ampere_left_percent_round = str(round(ampere_left_percent, 2))
                cmd = 'echo '+ '"Hostname: '+ hostnm + '\n'+ 'MAC address: '+
clientID + '\n'+ 'Current Time: '+ current_time + '\n'\
                    + 'Uptime: '+ uptime+ 'System average load (1 min, 5 min,
15 min): '+ sys_load + 'Current Location: '+ Location_Row + Location_Col\
                    + '\n'+ 'Energy left: '+ str(ampere_left) + ' mAh'+ '\n'+
'Energy left(%): '+ ampere_left_percent_round + ' %"'+ '>'+ wd + filename
#print(cmd)
os.system(cmd)

except KeyboardInterrupt:
    GPIO.cleanup()
print('\n\nKeyboard Interrupt. Script ended normally!')
## End of Client_device.py
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. PYTHON CODE FOR SCM

Filename: Server.py

```python
### Code references ###
## https://github.com/openwsn-berkeley/amp-o-meter
## http://www.steves-internet-guide.com/into-mqtt-python-client
## http://helloraspberrypi.blogspot.com/2016/02/python-to-get-mac-address-
using-uuid.html
## http://www.steves-internet-guide.com/mqtt-last-will-example

import paho.mqtt.client as mqtt
import mysql.connector as mariadb
import time
import subprocess
from uuid import getnode as get_mac

# Auto generate mac address
mac = get_mac()
macString = ':'.join(("%012X"% mac)[i:i+2] for i in range(0, 12, 2))
clientID = macString

# Server config
broker = "10.0.0.17"
port = 1883
topic = "Energy/ticks"

# Per tick
tick_mAh = 0.1707 # in mAh
tick_mC = 614.4393241 # in mC

# Auto ping function
def ping(ip_addr):
    time.sleep(2)
try:
        subprocess.check_output(
            ['ping', '-c', '2', ip_addr], stderr=subprocess.STDOUT,
universal_newlines=True
)
return True
    except subprocess.CalledProcessError:
return False

## on_message will provide 3 output
def call_back(client, userdata, message):
global cursor

    curr_time = int(time.time())
    payload = str(message.payload.decode("utf-8"))
    mac_addr = payload.split('|')[0]
    loc_R = payload.split('|')[1]
    loc_C = int(payload.split('|')[2])

    msg = payload.split('|')[3]
print(msg)
if msg not in ('A', 'R', 'L', 'D'):
        curr_tick = int(msg)

        prev_sql = 'SELECT ticks_count, read_time FROM readings where
```

77

```python
node_macaddr="%s" ORDER BY reading_id DESC LIMIT 1'% (mac_addr)
        cursor.execute(prev_sql)
        row = cursor.fetchone()
if row is None:
            prev_tick = 0
prev_time = int(time.time())
else:
            prev_tick = int(row[0])
            prev_time = int(row[1])

        difftick = curr_tick - prev_tick
        difftime = curr_time - prev_time
# discharge_rate = (difftick * tick_mAh) / (difftime * 60 * 60)
if difftime == 0: # To handle the divisble by zero limitation
difftime = 1
discharge_rate = (difftick * tick_mC) / (difftime) # Calculated based on mC per
sec. Just for reference. Not using this in Dashboard
mAmpUsed = curr_tick * tick_mAh

# DB functions here
sql = 'INSERT INTO readings(node_macaddr, read_time, discharge_rate,
mAmpUsedAccum, location_col, location_row, ticks_count) VALUES
("%s,""%s,""%s,""%s,""%s,""%s,""%s")'% (mac_addr, curr_time, discharge_rate,
mAmpUsed, loc_C, loc_R, curr_tick)

else:
if msg is 'A':
            alert_msg = "Going Amber"
elif msg is 'R':
            alert_msg = "Going Red"
elif msg is 'L':
            alert_msg = "Lost connection with MQTT server"
#One node assumed to have one active IP
ip_sql = 'SELECT ip FROM nodes where node_macaddr="%s"'% (mac_addr)
            cursor.execute(ip_sql)
            row = cursor.fetchone()
            ip_addr = row[0]
            response = ping(ip_addr)

if response is False:
                alert_msg = "Lost connection with Device: "+mac_addr
print(ip_addr + " is OFFLINE")
                status_sql = 'UPDATE nodes SET status="%s" WHERE
node_macaddr="%s"'% ('OFFLINE', mac_addr)
                cursor.execute(status_sql)
                db.commit()
elif msg is 'D':
            alert_msg = "Battery Drained"
else:
            alert_msg = "Unrecognized data"

sql = 'INSERT INTO alert(read_time, node_macaddr, alertrmsg) VALUES
("%s,""%s,""%s")'% (curr_time, mac_addr, alert_msg)
        online_sql = 'UPDATE nodes SET status="ONLINE" WHERE
node_macaddr="%s"'% (clientID)
        cursor.execute(online_sql)
        db.commit()

try:
        cursor.execute(sql)
        db.commit()
except:
```

```python
        db.rollback()

try:
    #set up the DB connections
    db = mariadb.connect(host='10.0.0.17', user='root', passwd='passw0rd',
db='projectdb')
    cursor = db.cursor()

    #set up the mqtt subscriber
    mqtts = mqtt.Client(clientID)

    #server to subscribe to. Port is optional and by default: 1883
    mqtts.connect(broker, port)
    mqtts.subscribe(topic)
    mqtts.on_message = call_back

    #let subscriber run forever
    mqtts.loop_forever()

except KeyboardInterrupt:
    db.close()
print('\n\nKeyboard Interrupt. Script ended normally!')

## End of Server.py
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. DATABASE SCHEMA

[Database Table 1 - readings]
reading_id int NOT NULL AUTO_INCREMENT,
node_macaddr varchar(100) NOT NULL,
read_time int NOT NULL,
discharge_rate float NOT NULL,
mAmpUsedAccum float NOT NULL,
location_col int NOT NULL,
location_row char(2) NOT NULL,
ticks_count int NOT NULL,
PRIMARY KEY (reading_id)

[Database Table 2 - alert]
alert_id int NOT NULL AUTO_INCREMENT,
read_time int NOT NULL,
node_macaddr varchar(17) NOT NULL,
alertrmsg varchar(100) NOT NULL,
PRIMARY KEY(alert_id)

[Database Table 3 - nodes]
node_id int NOT NULL AUTO_INCREMENT,
interNTK varchar(1) NOT NULL,
hostname varchar(100) NOT NULL,
ip varchar(15) NOT NULL,
node_macaddr varchar(17) NOT NULL,
adv_cap int NOT NULL,
status varchar(15),
PRIMARY KEY(node_id)

[Database Table 4 - soh_readings]
soh_reading_id int NOT NULL AUTO_INCREMENT,
node_macaddr varchar(100) NOT NULL,
read_time int NOT NULL,
capacity float NOT NULL,
PRIMARY KEY (soh_reading_id)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. PYTHON CODE FOR APM

Filename: Prober.py

```python
import subprocess
import mysql.connector as mariadb
import time

# Auto ping function
def ping(ip_addr):
try:
        subprocess.check_output(
            ['ping', '-c', '2', ip_addr], stderr=subprocess.STDOUT,
universal_newlines=True
)
return True
    except subprocess.CalledProcessError:
return False

db = mariadb.connect(host='10.0.0.17', user='root', passwd='passw0rd',
db='projectdb')
cursor = db.cursor(buffered=True)

ip_sql = 'SELECT ip, node_macaddr, node_id FROM nodes WHERE status="ONLINE"'
cursor.execute(ip_sql)
row = cursor.fetchone()

while row is not None:
    ip_addr = row[0]
    mac_addr = row[1]
    device_id = row[2]
    row = cursor.fetchone()
print(ip_addr + ""+ mac_addr)
    response = ping(ip_addr)
if response is False:
print(ip_addr + " is OFFLINE")
        offline_sql = 'UPDATE nodes SET status="%s" WHERE node_macaddr="%s"'%
('OFFLINE', mac_addr)
        cursor.execute(offline_sql)
        db.commit()

        curr_time = int(time.time())
        alert_msg = "Auto Probe update: Device "+ str(device_id) + " is
OFFLINE"
alert_sql = 'INSERT INTO alert(read_time, node_macaddr, alertrmsg) VALUES
("%s,""%s,""%s")'% (curr_time, mac_addr, alert_msg)
        cursor.execute(alert_sql)
        db.commit()

else:
print(ip_addr + " is ONLINE")

db.close()
## End of Prober.py
```

APPENDIX E. DASHBOARD IMPLEMENTATION IN PHP

Filename: index.php

```php
<?php
include("db.php");
include("db-functions.php");
include("printables.php");
?>

<!DOCTYPE html>
<html lang="en">
<head>
<title>Project DB</title>
<meta http-equiv="refresh" content="10">
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body id="myPage">

<?php

if(isset($_POST['submit'])) {
      echo "Battery refresh for  ."$_POST['submit'];
      deletefromReadings($conn, $_POST['submit']);
}



/******  View 1 *************************************************/
echo "<br><br>Alert Notifications <br>";
printView1_alert($conn);


echo "<br><br>Detailed Device View (DDV) <br>";
printView1($conn);

?>

<br>
<a href="view2.php"> Go to Quick Device View (QDV) </a><br>
<a href="view3.php"> Go to Basic Analytics View (BAV) </a><br>

</body>
<footer class="container-fluid text-center">
<a href="#myPage" title="To Top">Back to top</a>
<p>© 2018 All rights reserved.</p>
</footer>
//End of index.php
```

Filename: view2.php

```php
<?php
include("db.php");
include("db-functions.php");
include("printables.php");

$result = getdataforTable2($conn);
$numofNodes = mysqli_num_rows($result);
/*display in max of 6 columns and X rows depend in number of nodes */
$COLS = 6;
$ROWS = (int) ( $numofNodes / 6 )  + 1;

?>

<!DOCTYPE html>
<html lang="en">
<head>
<title>Project DB</title>
<meta http-equiv="refresh" content="10">
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
.grid-container {
display: grid;
grid-template-columns: <?php for ($i=0; $i< $COLS; $i++) echo "150px "; ?> ;
grid-template-rows: <?php for ($i=0; $i< $ROWS; $i++) echo "150px "; ?> ;

grid-gap: 5px;
background-color: #EEEEEE;
padding: 1px;
}
.grid-container > div {
background-color: rgba(255, 255, 255, 0.7);
text-align: center;
padding: 3px 0;
font-size: 10px;
}
</style>
</head>
<body id="myPage">

<?php
/******  View 1 **************************************************/
echo "<br><br>Quick Device View (QDV) <br>";
printView4($conn, $COLS , $ROWS );

?>

<a href="index.php"> Go Back Homepage</a>

</body>
<footer class="container-fluid text-center">
<a href="#myPage" title="To Top">Back to top</a>
<p>Â© 2018 All rights reserved.</p>
</footer>
//End of view2.php
```

85

Filename: view3.php

```php
//Note the dependency of view3.php on Chartjs
//Available at URL: http://www.chartjs.org/docs/latest
<?php
include("db.php");
include("db-functions.php");
include("array-functions.php");
include("printables.php");
?>

<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="refresh" content="10">
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Line Chart</title>
        <script src="dist/Chart.bundle.js"></script>
        <script src="dist/utils.js"></script>
        <style>
        canvas{
            -moz-user-select: none;
            -webkit-user-select: none;
            -ms-user-select: none;
        }
        </style>
</head>
<body id="myPage">

<?php
/******  View 3 ** Since this is pure graph, I will just do the printable here.
THIS PAGE IS GONNA GET MESSY AS IT COMBINES JAVA AND PHP TO PRESENT GRAPH
PROPERLY ****************************/
echo "<br><br>Basic Analytics View (BAV)<br>";

$colorarr = getcolorArray();

?>
<div style="width:75%;">
<canvas id="canvas"></canvas>
</div>
<br>
<br>
<script>
var config = {
type: 'line',
data: {

<?php
    // HERE NEED TO Set 48 Time Intervals  and each interval 10 min ////////
    // Change when necessary
    // $minsPerInterval must be in format '-XXX minutes' where XXX si number of
Minutes.
    //$TotalIntervals=48;
    $TotalIntervals=30;
    //$minsPerInterval='-10 minutes';
    $minsPerInterval='-1 minutes';

        $timearr =
get_X_No_timeFmReading($conn,$TotalIntervals,$minsPerInterval);
```

```php
?>
labels: [
<?php
      for ($i=0; $i<$TotalIntervals; $i++){
            if ($i > 0)
                  echo ,"";
            echo "\"$timearr[$i]\"";
      }
                ?>
                            ],

<?php
    // HERE NEED TO CHECK NUMBER OF NODES IN DATABASE
      // BY RIGHT NEED TO VERIFY WITH LATEST TIMING FOR NODE......
                            $result = getdataforTable2($conn);
                            $nodecnt = mysqli_num_rows($result);
                            $nodetravs = 0;
                            $outtemp ="";
?>
datasets: [
<?php
                  while($row = mysqli_fetch_assoc($result)) {
                        $nodeid = $row['node_id'] ;
                  $node_macaddr = $row['node_macaddr'] ;
                        if ($nodetravs  > 0)
                              echo ,"";
                        $nodetravs += 1;
?>
                {
                  label: "Device <?php echo "$nodeid"; ?>,"
backgroundColor: window.chartColors.<?php echo $colorarr[$nodetravs-1];?>,
borderColor: window.chartColors.<?php echo $colorarr[$nodetravs-1];?>,
data: [

                                    <?php
      // FOPR EACH NODE, PRINT THE last 40 VALUES
/****
Changing how we read values, Here need to average all the times that falls
within the interval
fixed for at 10 mins/inteval for 40 intervals
***/
                  $secsPerInterval= $minsPerInterval * 60 * 60;

                              for ($j=0; $j < $TotalIntervals ; $j++){
//                                $outtemp.=$timearr[$j]."->";;
                                    if ($j>0)
                                          echo ,"";

//  SHOULD do function call getAvgReading($conn, $timearr[$j],
$minsPerInterval, $node_macaddr);
                        $avgDischargeRate=getAvgReading($conn, $timearr[$j],
$minsPerInterval, $node_macaddr);
echo $avgDischargeRate;


      } // End 2nd loop for gettting data in each node
                  ?>
                  ],
fill: false,
            }
<?php
                        }// END BRACKET FOR NODE WHILE LOOP
```

87

```
?>
                    ]
                },
options: {
responsive: true,
title:{
display:true,
text:' Discharge Rate Line Chart'
                },
tooltips: {
mode: 'index',
intersect: false,
                },
hover: {
mode: 'nearest',
intersect: true
                },
scales: {
xAxes: [{
display: true,
scaleLabel: {
display: true,
labelString: 'Last 30 mins'
                    }
                }],
yAxes: [{
display: true,
scaleLabel: {
display: true,
labelString: 'Avg Discharge Rate (mAmp/hr)'
                    }
                }]
            }
        }
    };

        window.onload = function() {
var ctx = document.getElementById("canvas").getContext("2d");
        window.myLine = new Chart(ctx, config);
    };

var colorNames = Object.keys(window.chartColors);

</script>


<?php
//echo $outtemp;
//print_r($timearr);
?>
<br>
<a href="index.php"> Go Back Homepage</a>

</body>
<footer class="container-fluid text-center">
<a href="#myPage" title="To Top">Back to top</a>
<p>© 2018 All rights reserved.</p>
</footer>
//End of view3.php
```

Filename: db-functions.php

```php
<?php

function getdataforTable1($conn){ // Ed

    $query = "select r.reading_id, r.node_macaddr, ";
    $query.= "r.read_time, r.discharge_rate, r.mAmpUsedAccum, r.location_row,
r.location_col from ";
        $query.= "readings as r ";

        $result = mysqli_query($conn, $query);
        return $result;
}

function getAlerts($conn){ // Ed
        $query = "select a.alert_id, a.read_time, a.node_macaddr, n.node_id, ";
        $query.= "a.alertrmsg from alert as a left join nodes as n on ";
        $query.= "a.node_macaddr like n.node_macaddr order by a.alert_id desc
limit 6";
        //$query.= "a.node_macaddr like n.node_macaddr limit 10";

        $result = mysqli_query($conn, $query);
        return $result;
}

function getdataforTable2($conn){ // Ed

    $query = "select n.node_id, n.interNTK, n.hostname, n.ip, ";
        $query.= "n.node_macaddr, n.adv_cap, n.status from ";
        $query.= "nodes as n ";

        $result = mysqli_query($conn, $query);
        return $result;
}

function getnodedata($conn ,$node_macaddr){ // Ed

        $query = "select r.reading_id, r.read_time, r.discharge_rate,
r.mAmpUsedAccum, r.location_row, r.location_col from ";
        $query.= "readings as r where node_macaddr like \"$node_macaddr\" order
by r.read_time";

        $result = mysqli_query($conn, $query);
        return $result;
}


function getnodedataearliest($conn ,$node_macaddr){ // Ed

        $query = "select r.reading_id, r.read_time, r.discharge_rate,
r.mAmpUsedAccum, r.location_row, r.location_col from ";
        $query.= "readings as r where node_macaddr like \"$node_macaddr\" order
by r.read_time limit 1";

        $result = mysqli_query($conn, $query);
        return $result;
}

function getnodedatalatest($conn ,$node_macaddr){ // Ed
```

```php
        $query = "select r.reading_id, r.read_time, r.discharge_rate,
r.mAmpUsedAccum, r.location_row, r.location_col from ";
        $query.= "readings as r where node_macaddr like \"$node_macaddr\" order
by r.read_time desc limit 1";

        $result = mysqli_query($conn, $query);
        return $result;
}


function deletefromReadings($conn, $node_macaddr){
        $pre_query = "select read_time, mAmpUsedAccum from readings where
node_macaddr like \"$node_macaddr\" order by reading_id desc limit 1";
         $pre_result = mysqli_query($conn, $pre_query);

        if (mysqli_num_rows($pre_result) > 0) {
                $pre_row = mysqli_fetch_assoc($pre_result);
                $read_time = $pre_row['read_time'];
                $capacity = $pre_row['mAmpUsedAccum'];
        }

        $add_query = "insert into soh_readings(node_macaddr, read_time, capacity)
values (\"$node_macaddr\," $read_time, $capacity)";
        $result = mysqli_query($conn, $add_query);

        $query = "delete from readings where node_macaddr like
\"$node_macaddr\"";
        $result = mysqli_query($conn, $query);
}

function getlatesttimeFmReading($conn){
        $query = "select read_time from readings order by read_time desc limit
1";
        $result = mysqli_query($conn, $query);
        $latestdatephp="2016-01-01 00:00:00" ;
        if (mysqli_num_rows($result) > 0) {
                $row2 = mysqli_fetch_assoc($result);
                date_default_timezone_set("UTC");
                $latestdatephp = date("Y-m-d H:i:s,"$row2['read_time']);
        }
        return $latestdatephp;
}

function get_X_No_timeFmReading($conn,$TotalIntervals,$minsPerInterval){
        $timelatest=getlatesttimeFmReading($conn)." UTC";
        $currentDate = new DateTime($timelatest);
        $timearr = array();
        $SStart=$TotalIntervals-1;
        for ($i=$SStart; $i>=0; $i--) {
                $timearr[$i]= $currentDate -> format('Y-m-d H:i:s');
                $currentDate ->modify($minsPerInterval);
        }
        return $timearr ;
}


function getAvgReading($conn, $starttime, $XminsB4, $node_macaddr){
        $tempX = new DateTime($starttime." UTC") ;
        $endtime = $tempX->format('U');
        $tempX ->modify($XminsB4);
        $starttime = $tempX->format('U');
```

90

```php
        $query = "select node_macaddr, read_time, discharge_rate, mAmpUsedAccum
";
        $query.= "from readings where node_macaddr like \"$node_macaddr\"";
        $query.= "and read_time > $starttime and read_time < $endtime ";
        $result = mysqli_query($conn, $query);

        $match = 0;
        $disRateAdd = 0 ;
        if (mysqli_num_rows($result) > 0) {
                while($row = mysqli_fetch_assoc($result)) {
                        #$start_time = $row3['read_time'];
                        $match+=1;
                        $disRateAdd += $row['discharge_rate'];
                }
        }

        if ($match > 0)
                return $disRateAdd/$match ;
        else
                return 0;
}
//End of db-functions.php
?>
```

Filename: db.php

```php
//Credentials to accessing Database here
<?php
$servername = "localhost";
$username = "root";
$password = 'passw0rd';
$dbname = "projectdb";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
die("Connection failed: " . mysqli_connect_error());
}

?>
//End of db.php
```

Filename: printables.php

```php
<?php

function printTableOne($conn){
        $result = getdataforTable1($conn);
        if (mysqli_num_rows($result) > 0) {
            // output data of each row
        echo "<table border=\"1\"><tr>
                <td>Reading_ID</td>
                <td>Mac_ID/Node_ID</td>
                <td>Read Time</td>
                <td>Discharge Rate</td>
                <td>mAmp used<br>Accum</td>
                <td>location</td>
                </tr>";
        while($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
                    echo "<td>."$row['reading_id']."</td>";
                    echo "<td>."$row['node_macaddr']."</td>";
                    echo "<td>."$row['read_time']."</td>";
                    echo "<td>."$row['discharge_rate']."</td>";
                    echo "<td>."$row['mAmpUsedAccum']."</td>";
                    echo
"<td>."$row['location_row'].,"."$row['location_col']."</td>";
        echo "</tr>";
            }
        echo "</table>";
        } else {
        echo "0 results";
        }

}

function printTableTwo($conn){
        $result = getdataforTable2($conn);
        if (mysqli_num_rows($result) > 0) {
            // output data of each row
        echo "<table border=\"1\"><tr>
                <td>Device<br>ID</td>
                <td>interNTK</td>
                <td>Hostname</td>
                <td>I.P.</td>
                <td>Mac Address</td>
                <td>Advertised<br>Batt-Cap</td>
                <td>status</td>
                </tr>";
        while($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
                    echo "<td>."$row['node_id']."</td>";
                    echo "<td>."$row['interNTK']."</td>";
                    echo "<td>."$row['hostname']."</td>";
                    echo "<td>."$row['ip']."</td>";
                    echo "<td>."$row['node_macaddr']."</td>";
                    echo "<td>."$row['adv_cap']."</td>";
                    echo "<td>."$row['status']."</td>";
        echo "</tr>";
            }
        echo "</table>";
        } else {
        echo "0 results";
```

```
        }

}

function printView1_alert($conn){
/*
node_id | interNTK | hostname | IP (hhtp://<IP>/current.txt) | Loc | Mac ID |
Adv_Bat_Cap | Est_Bat_Cap | batt left % | Avg dischage | status | New Batt
*/

        $result = getAlerts($conn);
        if (mysqli_num_rows($result) > 0) {
            // output data of each row
                echo "<table border=\"1\"><tr>
                <td>Alert ID</td>
                <td>Time</td>
                <td>Device ID :: Mac Address</td>
                <td>Alert Message</td>
                </tr>";

        while($row = mysqli_fetch_assoc($result)) {
                    $node_macaddr = $row['node_macaddr'];
                    $node_id = $row['node_id'];
                    $alert_id = $row['alert_id'];
                    $read_time =  $row['read_time'];
                    $alertrmsg = $row['alertrmsg'];

        echo "<tr>";
                    echo "<td>."$alert_id."</td>";
                    date_default_timezone_set("UTC");
                    $printTime = date("Y-m-d H:i:s,"$read_time);
                    echo "<td>."$printTime."</td>";
                    echo "<td>Device ."$node_id." :: ."$node_macaddr."</td>";
                    echo "<td>."$alertrmsg."</td>";
        echo "</tr>";
            }
        echo "</table>";
        } else {
        echo "0 results";
        }

}


function printView1($conn){
/*
node_id | interNTK | hostname | IP (hhtp://<IP>/current.txt) | Loc | Mac ID |
Adv_Bat_Cap | Est_Bat_Cap | batt left % | Avg dischage | status | New Batt
*/

        $result = getdataforTable2($conn);
        if (mysqli_num_rows($result) > 0) {
            // output data of each row
                echo "<form action=\"index.php\" method=\"post\" id=\"form1\">
                <table border=\"1\"><tr>
                <td>Device<br>ID</td>
                <td>Inter<br>Network</td>
                <td>Hostname</td>
                <td>IP Address</td>
                <td>Location</td>
                <td>MAC Address</td>
```

93

```php
            <td>Advertised<br>Battery Capacity</td>
            <td>Estimated<br>Battery Capacity</td>
            <td>Battery<br>Left (%)</td>
            <td>Avg Rate of<br>Discharge (mAmp/hr)</td>
            <td>Battery<br>Runtime</td>
            <td>Estimated<br>Battery Life Left</td>
            <td>Status</td>
            <td>New Charge<br>Cycle</td>

            </tr>";

    while($row = mysqli_fetch_assoc($result)) {
                $node_macaddr = $row['node_macaddr'];
                $adv_cap = $row['adv_cap'];
                $status =  $row['status'];
                $hostname = $row['hostname'];
                $interNTK = $row['interNTK'];
                $ip = $row['ip'];
                $node_id = $row['node_id'];

                $r2=getnodedatalatest($conn, $node_macaddr);
                $row2 = mysqli_fetch_assoc($r2);
                $location =
$row2['location_row'].,"."$row2['location_col'];
                $discharge_rate = $row2['discharge_rate'];
                $mAmpUsedAccum = $row2['mAmpUsedAccum'];
                $read_time = $row2['read_time'];

                $r3=getnodedataearliest($conn, $node_macaddr);
                $row3 = mysqli_fetch_assoc($r3);
                $start_time = $row3['read_time'];

    echo "<tr>";
                echo "<td>."$node_id."</td>";
                echo "<td>."$interNTK."</td>";
                echo "<td>."$hostname."</td>";
                echo "<td><a
href=\"http://."$ip."/current.txt\">."$ip."</a></td>";
                echo "<td>."$location."</td>";
                echo "<td>."$node_macaddr."</td>";
                echo "<td>."$adv_cap."</td>";
                /*Computing Estimated Batt Cap**********************/
                $est_cap = (3.7 / 5) * $adv_cap - ($adv_cap * 0.06) ;
                echo "<td>."$est_cap."</td>";
                /*Computing Battery Left*********************/
                echo "<td>";
                $battleft = ( $est_cap - $mAmpUsedAccum ) / $est_cap * 100
;
                echo round($battleft,2) . "%</td>" ;

                /*Computing Avg Rate of Discharge***************/
                echo "<td>";
                $discharerate = 0 ;
                if ($read_time - $start_time > 0)
                        $discharerate = $mAmpUsedAccum / ($read_time -
$start_time) * 3600 ;
                echo  round($discharerate,2)."</td>";

                /*Get Battery runtime **************/
                $runtime = ($read_time - $start_time);
                $runtimeHr = (int) ($runtime / 3600 ) ;
```

```php
                    $runtimeMin = (int) (($runtime - ($runtimeHr * 3600) ) / 60
) ;
                    echo "<td>."$runtimeHr." hr ."$runtimeMin." min</td>" ;

                    /*Get Est Battery Batt Left ***************/
                    $discharerate_secs = $mAmpUsedAccum / ($read_time -
$start_time) ;
                    $estlifespansecs = 0;
                    if ($discharerate > 0)
                            $estlifespansecs = (int) $est_cap /
$discharerate_secs ;
                    $estleftsecs = $estlifespansecs - $runtime ;
                    if ($estleftsecs < 0 ) $estleftsecs = 0 ;

                    $runtimeHr = (int) ($estleftsecs / 3600 ) ;
                    $runtimeMin = (int) (($estleftsecs - ($runtimeHr * 3600) )
/ 60 ) ;
                    echo "<td>."$runtimeHr." hr ."$runtimeMin." min</td>" ;

                    echo "<td>."$status."</td>";
                    /* Reset Button ***************/
                    echo "<td>";
                    echo"<input type=\"submit\" name=\"submit\"
value=\"$node_macaddr\">";
        echo "</tr>";
              }
        echo "</table></form>";
        } else {
        echo "0 results";
        }

}




function printView2($conn, $grid_cols, $grid_rows){
        $nodearr=[];
        $result = getdataforTable2($conn);
        if (mysqli_num_rows($result) > 0) {
                while($row = mysqli_fetch_assoc($result)) {
                        $nodearr[] = $row;
                }
        }
// Converting Location of NODE to Position in Table
// Each Row has $grid_cols number of Column
// A => Plus 0 to position
// B => Plus $$grid_cols to position
// C => Plus 2x$grid_cols  to position ...

// Drawing the Table here =========================
        echo "<div class=\"grid-container\">";
        $sizeofgrid = $grid_cols * $grid_rows;

        for ($i=0; $i < $sizeofgrid ; $i++){
                $node_detect = 0;

                // getting locROW and locCOL from $i to print
                $locROW = chr(floor($i/$grid_cols)+65) ;
                $locCOL = $i % $grid_cols + 1 ;

                foreach($nodearr as $node) {
```

```php
                    $result2 = getnodedatalatest($conn ,$node['node_macaddr']);
                    $row2 = mysqli_fetch_assoc($result2);
                    $upperRow = strtoupper ( $row2['location_row'] );
                    $addrowpos = ord ( $upperRow ) - 65 ;
                    $gridpos = ( $addrowpos * $grid_cols) +
$row2['location_col'] - 1;

                    if ($gridpos == $i){
                        $node_detect = 1 ;
                        echo "<div class=\"item."$i."\" style=\"";
//Change format/color of box depending on node attribute
                        $est_cap = (3.7 / 5) * $node['adv_cap'] -
($node['adv_cap'] * 0.06) ;
                        $battleft = ( $est_cap - $row2['mAmpUsedAccum'] ) /
$est_cap * 100 ;

                        if ($node['status'] == "ONLINE" ){
                            if ($battleft > 60)
                                    echo "background-color: #88DD88;
color:#225522; ";
                                else{
                                    if ($battleft >20)
                                            echo "background-color: #FFB652;
color:#225522;";
                                        else
                                            echo "background-color: #DD8888;
color:#225522;";
                                }
                        }
                        else{ // Assume Offline
                                echo "background-color: #223344;
color:#CCDDAA; ";
                        }

                        if ($node['interNTK'] == 'y')
                                echo " border:3px solid red;";

                        echo "\">."$locROW.,."$locCOL."<br>";
//Entering Details of Node
                        echo $node['node_id']."<br>";
                        echo round($battleft,1)."%<br>";
                        //echo "HostName: ."$node['hostname']."<br>";

                        echo "</div>";
                    }

                }
                if ($node_detect == 0 ){
                        echo "<div class=\"item."$i."\">."$locROW.,."$locCOL;
                        echo "</div>";
                }
            }
        }
        echo "</div>";
// END Drawing the Table here ===========================
}


function printView4($conn, $grid_cols, $grid_rows){
        $nodearr=[];
        $result = getdataforTable2($conn);
        if (mysqli_num_rows($result) > 0) {
```
96

```php
            while($row = mysqli_fetch_assoc($result)) {
                    $nodearr[] = $row;
            }
        }
// Drawing the Table here ==========================
        echo "<div class=\"grid-container\">";
        $sizeofgrid = $grid_cols * $grid_rows;

            $i = 0 ;
            foreach($nodearr as $node) {
                    $result2 = getnodedatalatest($conn ,$node['node_macaddr']);
                    $row2 = mysqli_fetch_assoc($result2);

                        echo "<div class=\"item."$i."\" style=\"";
//Change format/color of box depending on node attribute
                        $est_cap = (3.7 / 5) * $node['adv_cap'] -
($node['adv_cap'] * 0.06) ;
                        $battleft = ( $est_cap - $row2['mAmpUsedAccum'] ) /
$est_cap * 100 ;

                        if ($node['status'] == "ONLINE" ){
                            if ($battleft > 60)
                                    echo "background-color: #88DD88;
color:#225522; ";
                                else{
                                    if ($battleft >20)
                                            echo "background-color: #FFB652;
color:#225522;";
                                        else
                                                echo "background-color: #DD8888;
color:#225522;";
                                }
                        }
                        else{ // Assume Offline
                            echo "background-color: #223344;
color:#CCDDAA; ";
                        }

                        if ($node['interNTK'] == 'y')
                            echo " border:3px solid red;";

                        echo "\"><br>";
//Entering Details of Node
                        echo "Device: ."$node['node_id']."<br>";
                        echo "MAC Addr: ."$node['node_macaddr']."<br>";
                        echo "HostName: ."$node['hostname']."<br>";
                        echo "Battery: ."round($battleft,1)."%<br>";
                        echo "Location:
[."$row2['location_row'].,"."$row2['location_col']."]<br>" ;
                        echo "</div>";
                        $i ++ ;
            }
        echo "</div>";
// END Drawing the Table here ==========================
}


?>
//End of printables.php
```

Filename: array-functions.php

```php
<?php
function getcolorArray(){
      $colorarr = [];
      $colorarr[] = "red";
      $colorarr[] = "purple";
      $colorarr[] = "blue";
      $colorarr[] = "green";
      $colorarr[] = "yellow";
      $colorarr[] = "orange";
      $colorarr[] = "gray";

      $colorarr[] = "red";
      $colorarr[] = "purple";
      $colorarr[] = "blue";
      $colorarr[] = "green";
      $colorarr[] = "yellow";
      $colorarr[] = "orange";
      $colorarr[] = "gray";

      $colorarr[] = "red";
      $colorarr[] = "purple";
      $colorarr[] = "blue";
      $colorarr[] = "green";
      $colorarr[] = "yellow";
      $colorarr[] = "orange";
      $colorarr[] = "gray";

      $colorarr[] = "red";
      $colorarr[] = "purple";
      $colorarr[] = "blue";
      $colorarr[] = "green";
      $colorarr[] = "yellow";
      $colorarr[] = "orange";
      $colorarr[] = "gray";

      $colorarr[] = "red";
      $colorarr[] = "purple";
      $colorarr[] = "blue";
      $colorarr[] = "green";
      $colorarr[] = "yellow";
      $colorarr[] = "orange";
      $colorarr[] = "gray";
      return $colorarr;

}

?>
//End of array-functions.php
```

APPENDIX F. PYTHON CODE FOR GENERATING LOAD

Filename: load.py

```python
from multiprocessing import Pool
from multiprocessing import cpu_count
import sys

#processes = sys.argv[1]

def f(x):
while True:
        x*x

if __name__ == '__main__':
    processes = cpu_count()
print 'utilizing %d cores\n' % processes
    pool = Pool(processes)
pool.map(f, range(processes))

## pi@PI4:~/Desktop $ python load.py
## utilizing 4 cores
## End of load.py
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1]   Andflywrite, "The fourth Industrial Revolution," October 7, 2017. [Online].
        Available: https://publicservicesalliance.org/2017/10/07/the-fourth-industrial-
        revolution

[2]   "IEEE-SA Internet of Things ecosystem study," Institute of Electrical and
        Electronics Engineers, January 2015. [Online]. Available:
        http://standards.ieee.org/innovate/iot/study.html

[3]   "Towards a definition of the Internet of Things (IoT)," Institute of Electrical and
        Electronics Engineers, May 2015. [Online]. Available:
        https://iot.ieee.org/definition.html

[4]   AsmaHaroon et al.,"Constraints in the IoT: The world in 2020 and beyond,"
        International Journal of Advanced Computer Science and Applications, vol. 7, no.
        11, pp. 252–271, November 2016. [Online]. Available:
        http://thesai.org/Downloads/Volume7No11/Paper_33-
        Constraints_in_the_IoT_The_World_in_2020.pdf

[5]   Yaqoob et al.,"Internet of Things architecture: Recent advances, taxonomy,
        requirements, and open challenges," *IEEE Wireless Communications*, vol. 24, no.
        3, pp. 10–16, Jun. 2017. [Online]. doi: 10.1109/MWC.2017.1600421

[6]   A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash,
        "Internet of Things: A survey on enabling technologies, protocols, and
        applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp.
        2347–2376, Fourthquarter 2015. [Online]. doi: 10.1109/COMST.2015.2444095

[7]   Rberlia, "Communication protocols for Internet of Things," January 5, 2015.
        [Online]. Available: https://learn.sparkfun.com/tutorials/connectivity-of-the-
        internet-of-things/all

[8]   Q. Zhu, R. Wang, Q. Chen, Y. Liu and W. Qin, "IoT Gateway: Bridging Wireless
        Sensor Networks into Internet of Things," *2010 IEEE/IFIP International
        Conference on Embedded and Ubiquitous Computing*, Hong Kong, 2010, pp.
        347–352, Dec. 2010. [Online]. doi: 10.1109/EUC.2010.58

[9]   "IoT Gateways are revolutionizing the connected world," VensiInc. Accessed
        February 20, 2018. [Online]. Available: http://blueapp.io/wp-
        content/uploads/2016/10/Gateway.jpg

[10] Holger Karl, "Ad Hoc and Sensor Networks chapter 3: Network architecture," University Paderborn - Computer Networks Group. [Online]. Available: http://www.uta.edu/utari/acs/ee5369/Karl%20slides/sensys-ch3-network-architecture.pdf

[11] Alessandro Bassi et al.,"Internet of Things - architecture - deliverable D3.4 - guidelines on next generation protocols," September 2013. [Online]. Available: http://www.meet-iot.eu/deliverables-IOTA/D3_4.pdf

[12] "MANET... mobile ad hoc networking (a.k.a. Mesh)," Silvus Technologies. Accessed February 20, 2018. [Online]. Available: http://silvustechnologies.com/technology/introduction-to-manet

[13] L. Mainetti, L. Patrono and A. Vilei, "Evolution of Wireless Sensor Networks towards the Internet of Things: A survey," *SoftCOM 2011, 19th International Conference on Software, Telecommunications and Computer Networks*, 2011, pp. 1–6, September 2011. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6064380&isnumber=6064354

[14] S. Distefano, G. Merlino and A. Puliafito, "Sensing and actuation as a service: A new development for Clouds," *2012 IEEE 11th International Symposium on Network Computing and Applications*, Cambridge, MA, August 2012, pp. 272–275, Aug. 2012. [Online]. doi: 10.1109/NCA.2012.38

[15] N. Khalil, M. R. Abid, D. Benhaddou and M. Gerndt, "Wireless Sensors Networks for Internet of Things," *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, 2014, pp. 1–6, Apr. 2014. [Online]. doi: 10.1109/ISSNIP.2014.6827681

[16] E. Kougianos, S. P. Mohanty, G. Coelho, U. Albalawi and P. Sundaravadivel, "Design of a high-performance system for secure image communication in the Internet of Things," *IEEE Access*, vol. 4, pp. 1222–1242, Mar. 2016. [Online]. doi: 10.1109/ACCESS.2016.2542800

[17] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass and J. Scott, "Impact of human mobility on Opportunistic Forwarding Algorithms," *IEEE Transactions on Mobile Computing*, vol. 6, no. 6, pp. 606–620, June. 2007. [Online]. doi: 10.1109/TMC.2007.1060

[18] Aaron Arthurs et al.,"A comparative study of ultra-low voltage digital circuit design," *International Journal of VLSI design & Communication Systems (VLSICS)*, vol.3, no.3, June 2012. [Online]. Available: http://airccoline.com/vlsics/V3N3/3312vlsics01.pdf

[19]   "New design concepts in ultra low voltage and Nanopower(TM) circuits with EPAD MOSFET arrays," Advanced Linear Devices Inc, 2005. [Online]. Available: http://www.aldinc.com/pdf/NewDesignConcepts.pdf

[20]   R. Min, T. Furrer and A. Chandrakasan, "Dynamic voltage scaling techniques for distributed microsensor networks," *Proceedings IEEE Computer Society Workshop on VLSI 2000. System Design for a System-on-Chip Era*, Orlando, FL, 2000, pp. 43–46, Aug. 2002. [Online]. doi: 10.1109/IWV.2000.844528

[21]   O. Hahm, E. Baccelli, H. Petersen and N. Tsiftes, "Operating systems for low-end devices in the Internet of Things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 720–734, Oct. 2016. [Online]. doi: 10.1109/JIOT.2015.2505901

[22]   H. Rahman, N. Ahmed and I. Hussain, "Comparison of data aggregation techniques in Internet of Things (IoT)," *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, Chennai, 2016, pp. 1296–1300, Mar. 2016. [Online]. doi: 10.1109/WiSPNET.2016.7566346

[23]   K. Karamitsios and T. Orphanoudakis, "Efficient IoT data aggregation for connected health applications," *2017 IEEE Symposium on Computers and Communications (ISCC)*, Heraklion, 2017, pp. 1182–1185, Jul. 2017. [Online]. doi: 10.1109/ISCC.2017.8024685

[24]   Ian Poolel, "IEEE 802.11 Wi-Fi standards," Radio-electronics.com. Accessed February 20, 2018 [Online]. Available: http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11-standards-tutorial.php

[25]   "Bluetooth Smart and Bluetooth Smart Ready," Laird. Accessed February 20, 2018 [Online]. Available: https://www.lairdtech.com/blog/cs/bluetooth-smart-and-bluetooth-smart-ready

[26]   Jon Mundy, "What is Bluetooth 5? The iPhone 8 and iPhone X's new wireless tech explained," September 23, 2017. [Online]. Available: http://www.trustedreviews.com/news/what-is-bluetooth-5-2947121

[27]   Greg Hackmann, "802.15 Personal Area Networks," Washington University, March 21, 2006. [Online]. Available: https://www.cse.wustl.edu/~jain/cse574-06/ftp/wpans/index.html

[28]   G. Montenegro et al.,"Transmission of IPv6 packets over IEEE 802.15.4 networks," September 2007. [Online]. Available: https://tools.ietf.org/html/rfc4944

[29]   JPVasseur et al.,"RPL: The IP routing protocol designed for low power and lossy
        networks," Internet Protocol for Smart Objects Alliance, April 2011. [Online].
        Available: http://www.ipso-alliance.org/wp-content/media/rpl.pdf

[30]   "BU-601: How does a smart battery work?" Battery University, October 10, 2016.
        [Online]. Available:
        http://batteryuniversity.com/learn/article/inner_workings_of_a_smart_battery

[31]   "Roost smart home telematics," Roost Inc. Accessed February 20, 2018 [Online].
        Available: https://www.getroost.com

[32]   Aboudou, "Picheckvoltage," June 3, 2013. [Online]. Available:
        https://github.com/aboudou/picheckvoltage

[33]   Felipe Morgan, "Openwsn-Berkeley/ Amp-o-meter." Accessed March 1, 2018.
        [Online]. Available: https://github.com/openwsn-berkeley/amp-o-meter

[34]   "BU-903: How to measure state-of-charge," Battery University, October 25,
        2017. [Online]. Available:
        http://batteryuniversity.com/learn/article/how_to_measure_state_of_charge

[35]   "BU-901: Fundamentals in battery testing," Battery University, May 27, 2016.
        [Online]. Available:
        http://batteryuniversity.com/learn/article/difficulties_with_testing_batteries

[36]   "Energizer E92 datasheet," Energizer. Accessed February 20, 2018 [Online].
        Available: http://data.energizer.com/pdfs/e92.pdf

[37]   Ashish, "What are the different methods to estimate the state of charge of
        batteries?" May 2017. [Online]. Available:
        https://www.scienceabc.com/innovation/what-are-the-different-methods-to-
        estimate-the-state-of-charge-of-batteries.html

[38]   Sunny, "The secret marketing trick behind powerbank capacity," Banggood, June
        2, 2015. [Online]. Available: https://blog.banggood.com/the-secret-marketing-
        trick-behind-powerbank-capacity-29982.html

[39]   "The ultimate guide to capacity (mAh,Wh) and efficiency of portable chargers
        (a.k.a. Power Banks or external batteries)," Zendure, December 1, 2013. [Online].
        Available: https://zendure.com/blogs/zendure-team-blog/the-ultimate-guide-to-
        capacity-mah-wh-and-efficiency-of-portable-chargers-a-k-a-power-banks-or-
        external-batteries

[40]   "Power Bank charging," Flashbay, Accessed March 4, 2018. [Online]. Available:
        http://www.flashbay.com/support/faq/power-bank-charging

[41]   "Adafruit USB power gauge mini-Kit," Adafruit. Accessed February 20, 2018.
        [Online]. Available: https://www.adafruit.com/product/1549

[42]   "DIY USB LINE POWER METER STICK," Solo-labs, August 28, 2015.
        [Online]. Available: http://www.solo-labs.com/diy-usb-line-power-meter-stick

[43]   "USB tester backpack 2.0: TLS-0047," Fried Circuits. Accessed February 20,
        2018. [Online]. Available: https://friedcircuits.us/47

[44]   Mike Grusin, "LTC4150 Coulomb Counter hookup guide," Accessed February
        20, 2018. [Online]. Available: https://learn.sparkfun.com/tutorials/ltc4150-
        coulomb-counter-hookup-guide

[45]   "Raspberry Pi - teach, learn, and make with Raspberry Pi," Raspberry Pi
        Foundation, Accessed February 20, 2018. [Online]. Available:
        https://www.raspberrypi.org

[46]   "Welcome to Raspbian," Raspbian. Accessed February 20, 2018 [Online].
        Available: https://www.raspbian.org

[47]   Croston, "Raspberry-gpio-python," SourceForge, Accessed March 10, 2018.
        [Online]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs

[48]   Andr.oid Eric, "Python to get MAC address using uuid, run on Raspberry
        Pi/Raspbian Jessie," February 7, 2016. [Online]. Available:
        http://helloraspberrypi.blogspot.com/2016/02/python-to-get-mac-address-using-
        uuid.html

[49]   "Chart.js," Chartjs, Accessed March 10, 2018. [Online]. Available:
        http://www.chartjs.org/docs/latest

[50]   "Chartjs/Chart.js," MIT License. Accessed March 10, 2018. [Online]. Available:
        https://github.com/chartjs/Chart.js

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1.   Defense Technical Information Center
       Ft. Belvoir, Virginia

2.   Dudley Knox Library
       Naval Postgraduate School
       Monterey, California