LOGGING, VISUALIZATION, AND ANALYSIS OF NETWORK/POWER DATA OF IOT DEVICES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Neal Nguyen

December 2018

COMMITTEE MEMBERSHIP

TITLE:	Logging, Visualization, and Analysis of Network/Power Data of IoT Devices
AUTHOR:	Neal Nguyen
DATE SUBMITTED:	December 2018
COMMITTEE CHAIR:	Andrew Danowitz, Ph.D. Assistant Professor of Electrical Engineering
COMMITTEE MEMBER:	Bruce Debruhl, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Lynne Slivovsky, Ph.D. Professor of Electrical Engineering

ABSTRACT

Logging, Visualization, and Analysis of Network/Power Data of IoT Devices

Neal Nguyen

There are approximately 23.14 billion IoT(Internet of Things) devices currently in use worldwide. This number is projected to grow to over 75 billion by 2025. Despite their ubiquity little is known about the security and privacy implications of IoT devices. Several large-scale attacks against IoT devices have already been recorded.

To help address this knowledge gap, we have collected a year's worth of network traffic and power data from 16 common IoT devices. From this data, we show that we can identify different smart speakers, like the Echo Dot, from analyzing one minute of power data on a shared power line.

ACKNOWLEDGMENTS

Thanks to:

- My parents and sister for supporting me throughout life. I wouldn't have made it here without your time and effort.
- Professor Andrew Danowitz for the guidance and knowledge provided. I really enjoyed working with you.
- Yuhsin Chang for supporting me throughout my studies.
- Ryan Frawley for setting up the scripts for device logging and analysis alongside me. You really set a strong foundation for us to build upon!
- Cal Poly and Cisco for funding this project. We would not have had the resources needed to accomplish our task.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF CODE LISTINGS xiii

CHAPTER

1 Introduction 1

1.1 Previous Work 2

1.1.1 An Analysis of Home IoT Network Traffic and Behaviour . . . 2

1.1.2 ProfilIoT 3

1.1.3 Logging and Analysis of Internet of Things (IoT) Device Network Traffic and Power Consumption 4

1.2 Scope 4

1.3 Thesis Organization 5

2 Method 7

2.1 IoT Testbed Overview 7

2.2 Physical Layout and Setup 9

2.2.1 Smart Power Plugs 9

2.2.2 Wireless Access Point 10

2.2.3 Devices 10

2.2.3.1 Smart Speakers 11

2.2.3.2 Streaming Devices 12

2.2.3.3 Video Game Consoles 12

2.2.3.4 Tablets 13

2.2.3.5 Security Camera 13
2.2.3.6 Other Devices 13
2.2.4 Wiring and Configuration 14
2.3 Software 16
2.3.1 Wireless WAP Code 16
2.3.2 sniff.py 16
2.3.2.1 Network Logging 17
2.3.2.2 Power Logging 17
2.3.3 Database 19
2.3.3.1 Network Table 19
2.3.3.2 Power Table 22
2.3.4 Device Inventory 26
2.3.5 IP History 26
2.3.6 Usage Flow 27
2.3.6.1 Google Sheets Script 28
2.3.6.2 Smart Speakers 29
2.3.6.3 Video Game Consoles 30
2.3.6.4 Streaming Devices 30
2.3.6.5 Tablets 30
2.3.6.6 Security Camera 31
2.4 realtimeIoTGrapher.py 31
2.4.0.1 Features 33
2.4.0.2 Visual Takeaway 38
2.4.0.3 Limitations 38
3 Results and Discussion 41

3.1 Swapping the Power Switch 41
3.2 General Analysis 43
3.2.1 Google Home Mini General Analysis 44
3.2.1.1 Google Chromecast 45
3.2.1.2 Amazon Fire Stick 47
3.2.1.3 Roku Express 49
3.2.2 General Analysis Discussion 50
3.3 Power Analysis on Smart Speakers 50
3.3.1 Smart Speaker Comparison 51
3.3.2 Echo Dot Brightness Sensor 53
3.3.3 Baseline Speaker Power 54
3.3.4 Smart Speaker Power Spikes When Asked for Weather 55
3.4 Summed Power Graph 56
3.5 Summed Power Graph with Noise 62
4 Conclusion 70
4.1 Future Work 71
BIBLIOGRAPHY 73
APPENDICES
A How to install and run RealTimeIoTGraph.py 74
A.1 How To Install and Run 74
A.2 Usage Directions 74

LIST OF TABLES

Table		Page
2.1	IoT Devices Being Monitored	11
2.2	Columns in Network Traffic Table	20
2.3	Columns in Power Table	23
2.4	Device inventory excerpt. Password column not shown.	26
2.5	History of IP addresses for each device	27
2.6	Event Log Excerpt	28

LIST OF FIGURES

Figure		Page
2.1	IoT testbed.	8
2.2	Smart Speakers Connected to WeMo Insight Switches	8
2.3	IoT Devices Under Examination	9
2.4	Navicat IP query with rollup.	21
2.5	Power query from database with Navicat.	24
2.6	Network query from database with Navicat.	25
2.7	Manual Analysis of IoT devices in Google Sheets: Raw data shown in columns A and B are then formatted into columns C and D for Graphing	32
2.8	Resulting graph of dataset shown in Figure 2.7	33
2.9	Screen of saved plotly graphs.	34
2.10	Screen to edit saved Plotly graphs containing an edit tool bar, the graph data, and the graph.	35
2.11	Summed power traces without interpolation.	37
2.12	Summed power traces with interpolation.	37
3.1	Connected the Echo Dot 1 to WeMo for 'Eufy Genie 1' and vice versa. The traces swap before and after switching.	42
3.2	Connected the Eufy Genie 1 to WeMo for 'Echo Dot 1' and vice versa. The traces swap before and after switching.	42
3.3	Connected the Google Home to WeMo for 'Eufy Genie 1' and vice versa. The traces swap before and after switching.	43
3.4	Idle Traffic of Google Home Over 1 Hour Period	44
3.5	Home Mini Response to News and Weather	45
3.6	Chromecast First Time Boot Network Traffic and Power Consumption	46

3.7 Chromecast Video Streaming 46

3.8 Chromecast Idle Traffic 47

3.9 Fire TV Stick First Time Boot Network Traffic and Power Consumption 48

3.10 Fire TV Stick Video Streaming 49

3.11 Roku Express Video Streaming 50

3.12 Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked "who is the best basketball player". Same graph as Figure 3.21 with Echo Dot 2 and Eufy Genie removed. 52

3.13 Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked "who is the best basketball player". Same time frame as Figure 3.12 52

3.14 Average power and network usage at each second for the Echo Dot, Eufy Genie, and Google Home in the time frame of Figures 3.12 and 3.13 53

3.15 Echo Dot Response to Lights 54

3.16 Baseline smart speaker power usage. 55

3.17 Smart speakers' power usage when asked 'whatś the weather' four times. 56

3.18 5 Smart Speakers Power Summed Together. 57

3.19 5 Smart Speakers Power Summed Up. Queried each device for the news. 58

3.20 5 Smart Speakers Power Summed Up. Queried each device for the best basketball player. 59

3.21 5 Smart Speakers Power Usage over time. Queried each device for the best basketball player. 60

3.22 5 Smart Speakers Network throughput over time. Queried each device for the best basketball player. 60

3.23 Spike summary of for each query. 61

3.24 Figure 3.20 summed smart speaker power, in use fan noise, and total power shown separately. 63

3.25 Figure 3.20 summed smart speaker power, in use fan noise, and total power. 64

3.26 Figure 3.20 summed smart speaker power, idle microwave noise, and total power. 64

3.27 Figure 3.20 summed smart speaker power, idle refrigerator noise, and total power. 65

3.28 Figure 3.20 summed smart speaker power, idle idle Intel NUC noise, and total power. 65

3.29 3.25, 3.26, 3.27, and 3.28 figures zoomed in 66

3.30 In use Microwave with Figure 3.20 summed smart speaker power separately. 66

3.31 In use Microwave with Figure 3.20 summed smart speaker power. . 67

3.32 Fridge in the middle of cooling with Figure 3.20 summed smart speaker power. 67

3.33 PC in use with Figure 3.20 summed smart speaker power. 68

A.1 RealTimeIoTGrapher usage image 75

Listings

2.1 Open and Read from a Socket 17
2.2 Rescan if all WeMos not found. 18
2.3 Open and Read from a Socket 29
2.4 Efficient SQL query to obtain total Network throughput at each second. 36
2.5 IP lookup table in real time IoT grapher. 40

Chapter 1

INTRODUCTION

There are 23.14 billion IoT devices in use worldwide that number is expected to grow to 75.44 billion by 2025 [**?**]. To support the demand for these devices, 2,888 companies have jumped into the field, contributing various types of IoT devices. Some devices have no support, some devices get updates but have bad support, some devices are outright malicious. Some devices have privacy and security concerns even when brand new and still technically "supported".

To address these concerns, this thesis contributes an IoT testbed that logs network and power data from 16 IoT devices over one year, accumulating 184.94 GB of data and 172,445,929 data points into a database. To help researchers sort and view this data, this thesis adds a Python [**?**] program that graphs network traffic and power data from the database. The graphs created by this tool were used to analyze IoT devices network and power usage in the testbed while idle, during startup, and while in use. From these graphs, it appears to be possible to identify the smart speaker in use when viewing just one minute of the shared power usage, for multiple commands.

This paper focuses on security and privacy flaws, that if fixed, do not affect the core features of an IoT device. For example, a smart speaker must store audio snippets to parse for the wake word. If the smart speaker occasionally hears a false positive wake word and sends the audio to its server, that is reasonable. However, Google Homes had an issue with their wake button, which caused the Home to listen 24/7 [**?**]. This type of unexpected behavior is a privacy concern.

In one of the largest IoT cybersecurity attacks, the Mirai Botnet, an attacker was

able to use weak login credentials to take control of 2.5 million IoT devices to perform a denial of service attack [**?**]. This is one of many events that introduces security concerns. We design the IoT database to better understand and detect these types of vulnerabilities in the future.

1.1 Previous Work

This section presents and analyzes related works on the topic of characterizing IoT devices. It presents the previous works individually. Because these papers are similar to each other, commentary on how their work is different and useful to this paper is covered in Section 1.2.

1.1.1 An Analysis of Home IoT Network Traffic and Behaviour

In *An Analysis of Home IoT Network Traffic and Behaviour* [**?**], the authors analyze IoT traffic in the home. The authors created an IoT testbed by setting up multiple IoT devices, connecting them to a router, sniffing their network packets while idle, and storing these packets. The IoT testbed consists of a smart air quality monitor, Amazon Echo, a few Apple devices, a smart hub, and a smart vacuum cleaner.

After 22 days of network logging, the authors analyzed each IoT device's network traffic individually and as a whole. They noticed that they can identify most devices' vendor from the first three MAC address bytes and make an assumption to what device it is. This information does not give away the exact device but knowing a specific vendor for devices within a house creates a privacy risk.

The Hue bridge communicates over unencrypted HTTP with other devices and the outside world. The authors state that the security flaw creates a privacy risk. A

user's presence in a room or house can be determined from these unencrypted HTTP packets. The authors also show the percentage of network packets by protocol and various other device network patterns. This general analysis fingerprints each device.

An Analysis of Home IoT Network Traffic and Behaviour most closely matches this paper. The authors have the same overall idea to collect network data and then use it to analyze metadata surrounding the networks.

1.1.2 ProfilIoT

The paper, *ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis* [**?**] uses machine learning algorithms to classify IoT devices. The researchers of this paper collect traffic from 13 different IoT and non-IoT devices. The IOT devices include a baby monitor, motion sensor, printer, refrigerator, security camera, socket, thermostat, smartwatch, and television. The non-IoT devices include two PCs and two smartphones. These devices connect to a Wi-Fi access point that records their network traffic with Wire Shark[**?**].

The researchers use machine learning on single-sessions to classify a device as an IoT device or non-IoT device. Then, they can classify the IoT devices by brand and model(e.g. Samsung Refrigerator, LG TV, WeMo Motion Sensor) with multi-sessions. A single-session is a 4-tuple consisting of the source IP, destination IP, source port number, and destination port number of a single TCP packet from a device. A multi-session is a list of single-sessions. Another machine learning model determines the minimum number of single-sessions needed to classify each device, determining the size of a multi-session. With single-sessions, they could determine if the device is an IoT device or not with 100 percent accuracy. Out of their nine IoT devices, they can classify brand and model of the IoT device with 99.281 percent accuracy.

Like *ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis*, this paper also focuses on classification of devices from data.

1.1.3 Logging and Analysis of Internet of Things (IoT) Device Network Traffic and Power Consumption

Logging and Analysis of Internet of Things (IoT) Device Network Traffic and Power Consumption[**?**], written by Ryan Frawley, was formed in conjunction with this paper. Frawley's paper and this paper were both directed by advisor Andrew Danowitz at Cal Poly.

Frawley's paper documents the steps necessary to construct a reliable IoT testbed capable of capturing network traffic and power data for connected devices and analyzing these devices further. He performed GeoIP[**?**] lookups on each device, showing the percentage of packets originating from each country and company. He also analyzed the packets of any unencrypted data in these devices.

1.2 Scope

Our work expands on the paper from Subsection 1.1.1 by contributing a portable database consisting of 10 months of data rather than 22 days of data. This paper adds more devices in our study and focuses more on device power/network usage over time rather than specific network packet information.

Instead of using machine learning techniques on network data, like the paper form Subsection 1.1.2, this paper focuses on manual analysis, looking for spikes, the height of the spike, the length of the power spike, and other graphical heuristics.

Compared to the paper in 1.1.1 and paper in 1.1.2, this paper also adds power usage

over time to the data set. The two papers mentioned only focus on network traffic. This paper also puts a significant emphasis on creating an extensive database rather than a smaller set of data to create graphs on network and power usage over time.

This paper is a continuation of the third paper in Subsection 1.1.3. Due to overlap between these two works, certain aspects of the IoT testbed setup and usage is only covered in cursory detail here. The reader is advised to access Frawley's work for full information. We both assembled the IOT test bed and interacted with the devices on a daily basis to simulate regular usage. We both also performed a preliminary analysis of the device network and power usage together.

The contributions of this work are:

- Logging of IoT device Power and Network Usage
- A custom data visualization tool that attaches to the database
- Analysis smart speaker and streaming device startup, idle, and in use network and power usage over time
- Analysis of smart speaker network and power usage
- Figures that show it is likely possible to identify a smart speaker through analysis of the powerline over time.

1.3 Thesis Organization

Chapter 2 discusses steps in setting up the IOT test bed, the analysis tool, and the logging system for interaction with devices. Chapter 2 also highlights the steps to set up a development environment to run the analysis tool (realTimeIoTGrapher.py) and how to use it. Chapter 3 presents power and network traffic for smart speakers and

streaming devices while idle, in use, and during startup in the form of line graphs and discuss the implications of these graphs. Chapter 3 also shows the graphs used to fingerprint the smart speakers while handling different commands and under noise. Finally, Chapter 4 finishes with concluding thoughts and future work.

Chapter 2

METHOD

To log and analyze network and power usage for our IoT devices, we set up an IoT testbed. This chapter documents the setup process for each IoT device, the logging process, how those two things resulted in the IoT testbed, and the software created to analyze the logged data.

2.1 IoT Testbed Overview

The flow diagram in Figure 2.1 outlines the IoT test bed constructed for this project. IoT devices pull power from the outlets through the smart switches, which log the power usage, and the WAP (Wireless Access Point) queries the smart switches for that data and pushes it to an AWS (Amazon Web Services)[**?**] MySQL[**?**] database as an entry in the 'power' table.

The wireless network provided through the WAP intercepts network packets of the IoT devices connected to it, parses them into power and network entries, and pushes them to the database. Then, for analysis, a user can run a python script called realTimeGrapher.py, which pulls from the database and graphs this data in real time or from set intervals in the past.

Section 2.2 discusses the physical components of the IoT testbed: devices used, setup of each device, and the physical layout. Section 2.3 covers the software part of the IoT testbed: how it was used and the tools to analyze it. Section 2.4 covers a Python program that automated visual analysis of the devices in the testbed.

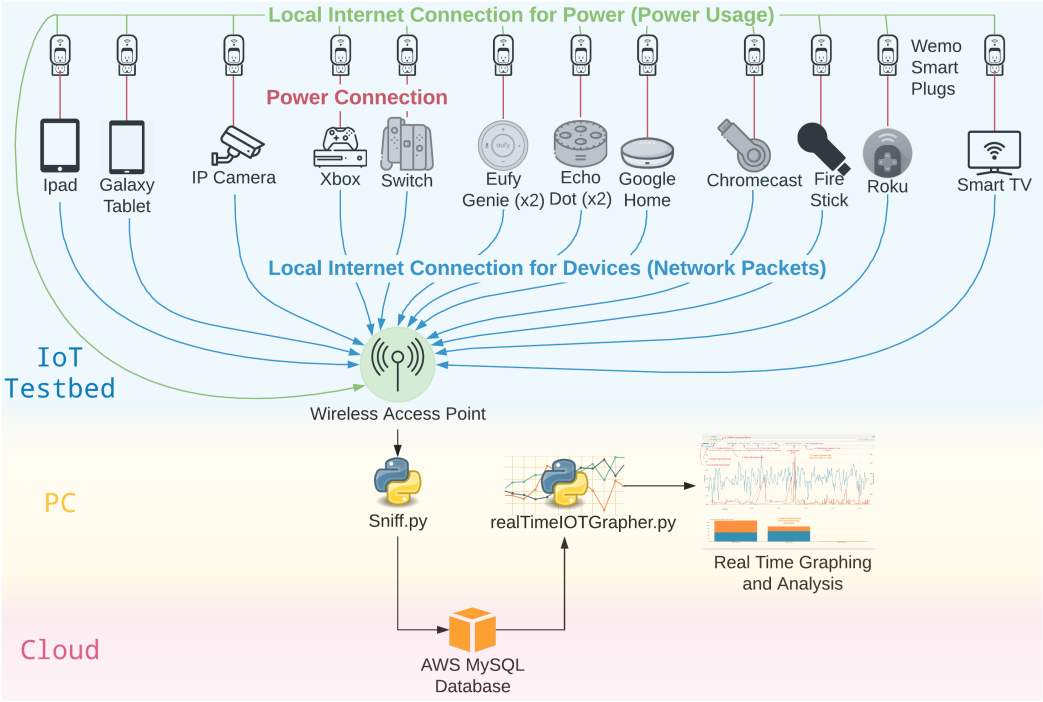


Figure 2.1: IoT testbed.

Figure 2.2: Smart Speakers Connected to WeMo Insight Switches

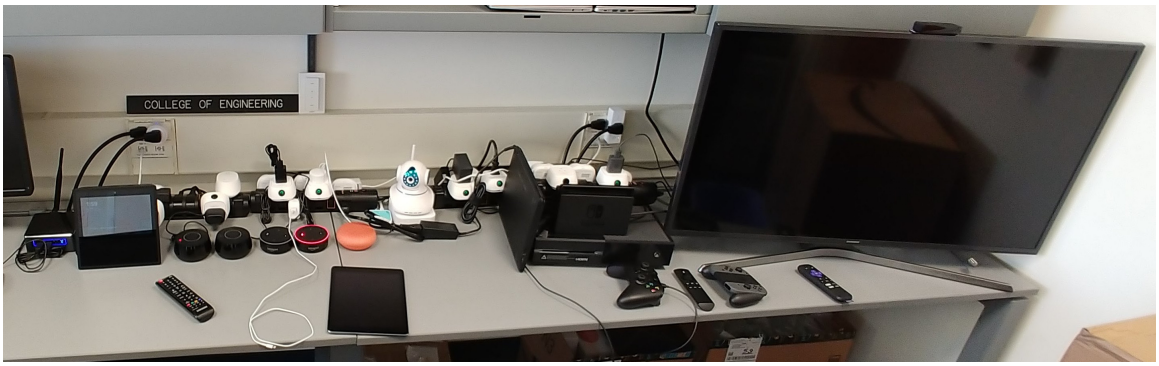


Figure 2.3: IoT Devices Under Examination

2.2 Physical Layout and Setup

Figures 2.2 and 2.3 show the IoT testbed set up. Setup was performed with Ryan Frawley and explained in his master's thesis [**?**]; this paper provides an updated description.

2.2.1 Smart Power Plugs

Smart power plugs are pass-through outlets that can be controlled/communicate wirelessly. The outlets collect and transmit power usage over the network. They are critical to the power consumption logging in this research.

We used the Belkin WeMo Insight. We used smart outlets in general because they're affordable. We chose the WeMo insights specifically because of the Python bindings.

When setting these smart plugs up, we named each smart plug based on the device connected to it so that when we pull power information with our scripts, we can easily relate plug data to a specific IoT device. This informal naming led to issues when the wrong device is connected to the wrong smart plug, as discussed in Section 2.4

2.2.2 Wireless Access Point

The wireless access point is an Intel NUC [?]. We loaded Ubuntu[?] on it because it is the most common Linux OS [?] and thus has the largest development support. The NUC was setup up as an access point that also sniffed network packets, query for power usage, and log data to a database through a Python script.

The NUC's built-in wireless interface is very slow. The throughput when using the wireless chip was measured to be 12 Mb/s down on Speedtest.net by Ookla [?] when connected to with a Chromebook [?] (for reference, the FCC defines 25 Mb/s as the minimum rate to be considered broadband [?]). This rate limitation caused many of the Belkin WeMo smart switches [?] to drop their connection intermittently, causing gaps in recorded power data. To solve this, we added a USB wireless antenna to the NUC. This antenna improved our internet speed to 30 Mb/s down when tested on Speedtest.net by Ookla with the same Chromebook, and solved the issue of devices losing connectivity.

2.2.3 Devices

This section goes over the smart speakers, streaming devices, video game consoles, tablets, security cameras, or other devices in the IoT testbed as listed in Figure 2.1. These devices were included based on their popularity, as the goal of master's thesis is to build a data set from common household IoT items.

Table 2.1: IoT Devices Being Monitored

Category	Manufacturer	Device	Quantity
Game Console	Nintendo	Switch	1
Game Console	Microsoft	Xbox One	1
Laptop	HP	Chromebook	1
Media Player	Samsung	Smart TV	1
Media Player	Google	Chromecast	1
Media Player	Amazon	Fire TV Stick	1
Media Player	Roku	Express	1
Security Camera	Eray	Hi3518 Wi-Fi Camera	1
Smart Speaker	Amazon	Echo Dot	2
Smart Speaker	Eufy/Anker	Genie	2
Smart Speaker	Amazon	Echo Show	1
Smart Speaker	Google	Home Mini	1
Tablet	Amazon	Fire 7 Tablet	2
Tablet	Samsung	Galaxy Tablet	1
Tablet	Apple	iPad	1

2.2.3.1 Smart Speakers

Smart speakers are IoT devices that combine speakers with built-in voice assistants. Some voice assistants include the Amazon Alexa, Google Assistant, or Apple's Siri. These devices are mainly controlled by voice commands, preceded by a wake word such as "hey Google".

Currently, around 39 million people (16 percent of the US population) use smart speakers[**?**]. In 2022 it is projected that 70 million US households will have at least one

smart speaker (55 percent of US households) and around 175 million smart speakers total[**?**]. For these reasons, we included this group of devices in our research.

Within the smart speaker category, we included the Amazon Echo Dot, Google Home, and Eufy Genie. The Amazon Echo Dot and the Google Home are the two leading smart speaker products. We added the Eufy Genie because it is a third party Amazon Alexa device that we can compare with the Echo Dot. We also included the Amazon Show, which is a voice assistant with a screen.

2.2.3.2 Streaming Devices

Streaming devices are IoT products that connect to a television or are built into a television (smart TVs) and stream videos or music from online services. In 2017 it was recorded that 70 million US households (58.7 percent of homes) had an internet enabled device capable of streaming to a TV [**?**].

The specific streaming devices we look at are the Roku Express, Amazon Fire TV, and the Google Chromecast. These devices represent three popular streaming platforms (Roku, Google Chromecast, Amazon Fire TV) that have a combined user base of over 110 million users who watch content at least once a month [**?**].

To view content from these streaming devices, we use a Samsung smart TV. This TV also as streaming capabilities which we log but do not analyze in this paper.

2.2.3.3 Video Game Consoles

Video game consoles are systems specifically built to play video games. We use two out of the three most popular gaming devices including the Nintendo Switch which sold 17.8 million units [**?**] and the Xbox One, which sold 30 million units [**?**] as of

January 2018.

2.2.3.4 Tablets

Tablets are mobile devices that have a touch screen with computer-like capabilities, but with limited IO such as an Apple iPad, Samsung Galaxy tablet, and the Amazon Fire tablet. These devices were used for interacting with the other IoT devices, which require a smartphone or tablet app for set up or control. Some of the IoT devices are limited to either Android, which runs on the Galaxy, or IOS, which runs on the iPad. Although there is no analysis on these devices in the paper, the network packets are logged. We did not track the charging or device power usage for these devices.

2.2.3.5 Security Camera

Security cameras are cameras meant to run constantly as surveillance, streaming the video footage so a user can view it from any connected device at any time. Two out of the five largest recorded cybersecurity attacks targeted security cameras [**?**]. The camera we use is the Eray camera. It has a weak username:password of admin:1234. A weak username and password make it very susceptible to cyber-attacks. There have also been reports that smart cameras have been found to send unencrypted data [**?**].

2.2.3.6 Other Devices

We also connected several other devices that are harder to categorize, analyze, or both.

We have a smart hub connected to a smart lock, an indoor room temperature sensor, and smart LED light bulbs. These devices communicate with the smart hub through

Zigbee or Bluetooth rather than through network packets.

To look at the network traffic caused by smart lights individually, we disconnected all devices besides the smart LED bulb from the smart hub. This isolation minimized other packet noise to the smart hub. Now the smart hub's network packets should mostly be due to actions by the LED bulb because it is the only one connected to it. We performed this process a few times as one of the experiments with it did no further analysis on this.

We also have a Chromebook which we used to control the Chromecast. However, because the Chromebook is more a laptop than an IoT device, we only log the network packets from this device.

2.2.4 Wiring and Configuration

After obtaining all the hardware necessary, the next thing was to set up the IoT testbed. This section briefly covers WeMo Insight setup, IoT device setup, and challenges/considerations faced when setting up the IoT testbed. Frawley's paper [**?**] previously covered this section, refer to his paper for more details.

It was essential to log power as soon as possible to capture a first-boot power network usage for each device. So first, before setting up the IoT devices, Ryan set up the NUC for logging. We considered setting static IPs for each device, but we decided that the average user would not set up a static IP for their device and left dynamic IPs. Some devices had already been used, so they do not contain startup information(Xbox and the Echo Dot 1).

Then, we set up each smart plug through the WeMo iOS app [**?**], filling in the WeMo's, username, email, network configuration, and device name. A WeMo Insight is named

after the device it provides power to, e.g. the Echo Dot was given the username "Echo Dot".

After the logging system was set up (WeMo for power usage, NUC for network traffic and logging data to database) we began setting up each device. We set up each device with a separate email account. To do this, we made 20 AOL accounts. We used email addresses that were not under control of the manufacturer of any of the devices. For example, we did not want to use any Gmail accounts because we did not want the Google Home or Google Chromecast to perform domain-specific optimizations. When setting up AOL accounts, we also faced a limitation of the number of email accounts that could be tied to a single phone number. We used two different phone numbers to circumvent this as each phone number can is allowed 15 accounts.

We plugged each device into their corresponding WeMo. Then, each device was set up through their corresponding app. Within that process, we connected the IoT device to the NUC for WiFi and created an account for the IoT device with the emails covered in the previous paragraph.

We set up each device in waves, taking around a week to set up most IoT devices. Incremental device setup with no planned organization resulted in a messy work environment, which made it difficult to match a device to it is corresponding smart plug. After a month of data logging in a disorganized scheme, it became difficult to keep track of each device. The WeMos were also placed in an inefficient way that covered up some ports, causing us to run out of ports for new WeMos to connect to. We unplugged everything, renamed the WeMos, organized the wiring, and organized the location of each smart device resulting in the setup in Figures 2.2 and 2.3. With limited power strips, the organization helped maximize power ports. We also faced the button for each power plug towards the user so that we could quickly notice whether the power plugs were on or off.

Organization streamlined the logging process overall; debugging network and connection issues were much simpler with an ordered setup. We could quickly identify redundant devices (there are two Echo Dots and Eufy Genies), run procedures on groups of devices, and determine the corresponding smart plug for each device.

2.3 Software

This section covers the software components used to keep track of the IoT test bed, the scripts for logging power and network traffic, and information about the database.

2.3.1 Wireless WAP Code

To turn the NUC into a wireless access point, we used a create_ap script [**?**]. This script takes in the desired network name, Wi-Fi password, wireless interface, and turns the NUC into a wireless access point. We later found that, with/ newer versions of Ubuntu, there is a built-in feature for creating a hotspot through a wireless interface on the device [**?**]. We used this feature to simplify the number of programs running and instead have the OS handle the hotspot. As discussed in Section 2.2.2 we eventually had to switch to an external antenna and rerun the script to set the USB antenna as the WAP wireless interface.

2.3.2 sniff.py

Sniff.py runs a thread for power logging and a thread for network logging. As each logger runs, it writes the network packets and power information to a MySQL database hosted on AWS through the python MySQL client [**?**] library. If the database connection is lost, the script automatically reconnects.

2.3.2.1 Network Logging

Network Logging begins by opening a socket on the wireless interfaces connected. All IP packets are sniffed through this socket connection as shown in Listing 2.1.

```
1 self.socket = socket.socket(socket.AF_PACKET, socket.
    SOCK_RAW, socket.htons(ETH_P_ALL))
2 self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF,
    2**30)
3 self.socket.bind((self.interface_name, ETH_P_ALL))
4 while True:
5     packet, address = self.socket.recvfrom(MTU)
```

Listing 2.1: Open and Read from a Socket

Once we have sniffed a packet, we can obtain the metadata from the hex dump. We preserve this hex dump in case we want to extract more information from it later. From each hex dump, we extract and store the source IP, source host, destination IP, destination host, time, size, type, protocol, source port, destination port, source host, and destination host. We explain what each fields are in Section 2.3.3.

2.3.2.2 Power Logging

In addition to network traffic, we also log the power usage. To obtain this information, we rely heavily on the Belkin WeMo Insights.

We used the PyWemo python script [**?**] to read power usage from the WeMos once per second. This is the highest frequency we could poll for power. The WeMo is capable of reading both power and energy in mW and kW hours.

Once the script queries the WeMo for its power information, we relate the WeMo to

the device connected to it by extracting the name of the WeMo. This information is used when pushing power data to our database.

One issue with PyWemo and the Belkin WeMos is that they sometimes disconnect, causing the script to miss power data. To handle this issue, the script rescans for WeMos until it finds all 16 WeMos in our testbed. The rescan code for this is shown in Listing 2.2

```
1     def scan_until_all_found():
2         print("Discovering WeMos")
3         switches = pywemo.discover_devices()
4         print("Discovered {} switches".format(len(switches))
              )
5         print(switches)
6         try_num = 1
7
8         while len(switches) < 16:
9             print("Did not discover enough switches, trying
                  again{}...".format(try_num))
10            switches = pywemo.discover_devices()
11            try_num += 1
12            print("Discovered {} switches".format(len(
                  switches)))
13
14        return switches
```

Listing 2.2: Rescan if all WeMos not found.

2.3.3 Database

To store all the results of the sniffed packets and power data, we push the data to a MySQL database hosted on an Amazon AWS server. As of writing, the database holds 172,445,929 entries that take up 184.94 GB of space. The database contains two tables, one for the network traffic and one for the power usage data. To interface with our database, we used a combination of the RealTimeIoTGrapher from Section 2.4, Navicat [**?**], and MySQL's command line tool [**?**].

2.3.3.1 Network Table

The largest part of our database is the IP network table. It is currently 179.1 GB and contains 116,830,077 entries. Its size is attributed to 12 months of network traffic. In those 12 months, we did many high bandwidth tasks such as playing music or video. These entries contain the raw hex dump of the whole packet, which contains all the metadata plus data load, further contributing to the size of this table. The data table's rows represent a single packet with the columns shown in Table 2.2. The research presented in Chapter 3 primarily relies on the time, source, destination, and size fields to get network throughput over time.

Table 2.2: Columns in Network Traffic Table

Column Number	Column Name	Data Type
1	time	datetime
2	source	varchar
3	src_host	varchar
4	destination	varchar
5	dst_host	varchar
6	protocol	varchar
7	type (in/out)	varchar
8	src_port	varchar
9	dst_port	varchar
10	size	int
11	hexdump	longtext

Common SQL commands we used for our analysis include those shown in listings 2.5 and 2.6. The most useful commands generally required examining total throughput or device throughput using ROLLUP, shown in Figure 2.4.

```
SELECT time, source, destination,
       CASE WHEN type IS NULL THEN 'total' ELSE type END type,
       SUM(size) AS total_throughput
FROM ip_log.ip
WHERE
  time BETWEEN '2018-05-17 16:30:00' AND '2018-05-17 16:40:00'
    AND
  (source = '192.168.12.48' OR destination = '192.168.12.48')
GROUP BY time, type WITH ROLLUP
LIMIT 20
```

time	source	destination	type	total_throughput
2018-05-17 16:30:27	192.168.12.48	184.31.3.33	outgoing	540
2018-05-17 16:30:27	192.168.12.48	184.31.3.33	total	540
2018-05-17 16:30:29	184.31.3.33	192.168.12.48	incoming	52
2018-05-17 16:30:29	184.31.3.33	192.168.12.48	total	52
2018-05-17 16:30:33	192.168.12.147	192.168.12.48	incoming	392
2018-05-17 16:30:33	192.168.12.147	192.168.12.48	total	392
2018-05-17 16:30:39	184.31.3.33	192.168.12.48	incoming	1500
2018-05-17 16:30:39	184.31.3.33	192.168.12.48	total	1500
2018-05-17 16:30:55	192.168.12.48	192.168.12.147	incoming	171
2018-05-17 16:30:55	192.168.12.48	192.168.12.147	total	171
2018-05-17 16:30:57	172.217.4.142	192.168.12.48	incoming	52
2018-05-17 16:30:57	172.217.4.142	192.168.12.48	total	52
2018-05-17 16:31:02	192.168.12.48	35.186.224.44	outgoing	94
2018-05-17 16:31:02	192.168.12.48	35.186.224.44	total	94
2018-05-17 16:31:12	172.217.4.142	192.168.12.48	incoming	855
2018-05-17 16:31:12	192.168.12.48	172.217.4.142	outgoing	569
2018-05-17 16:31:12	192.168.12.48	172.217.4.142	total	1424
2018-05-17 16:31:18	192.168.12.48	184.31.3.33	outgoing	52
2018-05-17 16:31:18	192.168.12.48	184.31.3.33	total	52
2018-05-17 16:31:35	35.186.224.44	192.168.12.48	incoming	91

Figure 2.4: Navicat IP query with rollup.

The hex dump column drastically increases the size of each entry in the network table. However, as a raw network packet, it is very flexible and can be manipulated for many

more use cases than the targeted columns we have. This flexibility is useful for future research that may require data we did not explicitly pull out for this project.

In the database, each device can be tracked down by looking for the IP address of that device in the source or destination columns. However, due to the use of dynamic IPs in our network setup a single device can be under multiple IPs within the table. In our research environment, we have no formal solution to this. When a query for a particular IP stopped working, we invoked a high bandwidth command on the device whose IP changed. This floods the database with packets from that device. The device's new IP is then correlated to the IP with the most packets and similar characteristics to the device in the network table in that time frame. We logged all devices and their IP addresses as they change in a Google Sheets [**?**] file covered in Subsection 2.3.5.

2.3.3.2 Power Table

The power table holds the most critical data for the research discussed in Chapter 3, and I refer to it for the majority of the paper's findings. The size of this table is much smaller than the network table. It contains 5.84 GB of data and 61,240,189 entries in the database. The significant difference in size between the IP table and power table is because each device only produces one power entry every second. The columns of the power table are shown in Figure 2.3.

Table 2.3: Columns in Power Table

Column Number	Column Name	Data Type
1	name	varchar
2	power_mw	int
3	time	datetime
4	today_kwh	varchar
5	on_for	varchar
6	today_on_time	varchar

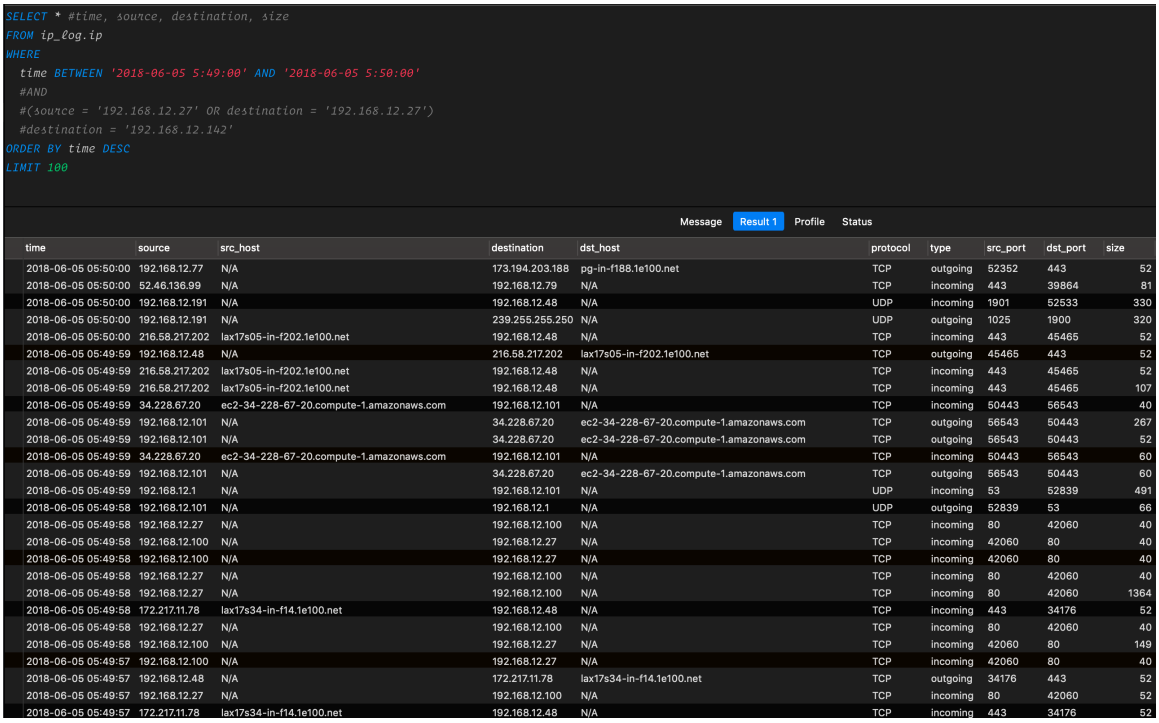
When working with this table, we generally query for a range of power packets in a given time frame. We do this for all devices, a subset of devices, or a single device. Some of the commands and results are shown in Figures 2.5 and 2.6.

```
SELECT time, name, power_mw
FROM ip_log.power
WHERE
  time BETWEEN '2018-5-26 1:30:00' AND '2018-7-26 1:40:00' AND
  (name = 'Echo Dot'
    OR name = 'Echo DOt 2'
    OR name = 'Chromecast'
    OR name = 'Roku'
    OR name = 'Samsung TV'
    OR name = 'Samsung Hub'
    OR name = 'Nintendo Switch'
    OR name = 'Xbox'
    OR name = 'Eufy Genie'
    OR name = 'Eufy Genie 2'
    OR name = 'Echo Show'
  )
ORDER BY time DESC
LIMIT 20
```

Message | Result 1 | Profile | Status

time	name	power_mw
2018-07-26 01:39:59	Eufy Genie 2	1470
2018-07-26 01:39:59	Echo Dot	1340
2018-07-26 01:39:58	Chromecast	1275
2018-07-26 01:39:58	Echo Dot 2	1385
2018-07-26 01:39:58	Samsung TV	50
2018-07-26 01:39:58	Eufy Genie	1405
2018-07-26 01:39:58	Roku	2335
2018-07-26 01:39:58	Nintendo Switch	170
2018-07-26 01:39:58	Samsung Hub	2190
2018-07-26 01:39:57	Echo Show	2840
2018-07-26 01:39:57	Xbox	15785
2018-07-26 01:39:56	Eufy Genie 2	1465
2018-07-26 01:39:55	Echo Dot	1345
2018-07-26 01:39:55	Chromecast	1265
2018-07-26 01:39:55	Echo Dot 2	1375
2018-07-26 01:39:55	Samsung TV	25
2018-07-26 01:39:55	Eufy Genie	1405
2018-07-26 01:39:55	Roku	2330

Figure 2.5: Power query from database with Navicat.



```
SELECT * #time, source, destination, size
FROM ip_log.ip
WHERE
  time BETWEEN '2018-06-05 5:49:00' AND '2018-06-05 5:50:00'
  #AND
  #(source = '192.168.12.27' OR destination = '192.168.12.27')
  #destination = '192.168.12.142'
ORDER BY time DESC
LIMIT 100
```

Message | Result 1 | Profile | Status

time	source	src_host	destination	dst_host	protocol	type	src_port	dst_port	size
2018-06-05 05:50:00	192.168.12.77	N/A	173.194.203.188	pg-in-f188.1e100.net	TCP	outgoing	52352	443	52
2018-06-05 05:50:00	52.46.136.99	N/A	192.168.12.79	N/A	TCP	incoming	443	39864	81
2018-06-05 05:50:00	192.168.12.191	N/A	192.168.12.48	N/A	UDP	incoming	1901	52533	330
2018-06-05 05:50:00	192.168.12.191	N/A	239.255.255.250	N/A	UDP	outgoing	1025	1900	320
2018-06-05 05:50:00	216.58.217.202	lax17s05-in-f202.1e100.net	192.168.12.48	N/A	TCP	incoming	443	45465	52
2018-06-05 05:49:59	192.168.12.48	N/A	216.58.217.202	lax17s05-in-f202.1e100.net	TCP	outgoing	45465	443	52
2018-06-05 05:49:59	216.58.217.202	lax17s05-in-f202.1e100.net	192.168.12.48	N/A	TCP	incoming	443	45465	52
2018-06-05 05:49:59	216.58.217.202	lax17s05-in-f202.1e100.net	192.168.12.48	N/A	TCP	incoming	443	45465	107
2018-06-05 05:49:59	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	192.168.12.101	N/A	TCP	incoming	50443	56543	40
2018-06-05 05:49:59	192.168.12.101	N/A	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	TCP	outgoing	56543	50443	267
2018-06-05 05:49:59	192.168.12.101	N/A	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	TCP	outgoing	56543	50443	52
2018-06-05 05:49:59	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	192.168.12.101	N/A	TCP	incoming	50443	56543	60
2018-06-05 05:49:59	192.168.12.101	N/A	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	TCP	outgoing	56543	50443	60
2018-06-05 05:49:59	192.168.12.1	N/A	192.168.12.101	N/A	UDP	incoming	53	52839	491
2018-06-05 05:49:58	192.168.12.101	N/A	192.168.12.1	N/A	UDP	outgoing	52839	53	66
2018-06-05 05:49:58	192.168.12.27	N/A	192.168.12.100	N/A	TCP	incoming	80	42060	40
2018-06-05 05:49:58	192.168.12.100	N/A	192.168.12.27	N/A	TCP	incoming	42060	80	40
2018-06-05 05:49:58	192.168.12.100	N/A	192.168.12.27	N/A	TCP	incoming	42060	80	40
2018-06-05 05:49:58	192.168.12.27	N/A	192.168.12.100	N/A	TCP	incoming	80	42060	40
2018-06-05 05:49:58	192.168.12.27	N/A	192.168.12.100	N/A	TCP	incoming	80	42060	1364
2018-06-05 05:49:58	172.217.11.78	lax17s34-in-f14.1e100.net	192.168.12.48	N/A	TCP	incoming	443	34176	52
2018-06-05 05:49:58	192.168.12.27	N/A	192.168.12.100	N/A	TCP	incoming	80	42060	40
2018-06-05 05:49:58	192.168.12.100	N/A	192.168.12.27	N/A	TCP	incoming	42060	80	149
2018-06-05 05:49:57	192.168.12.100	N/A	192.168.12.27	N/A	TCP	incoming	42060	80	40
2018-06-05 05:49:57	192.168.12.48	N/A	172.217.11.78	lax17s34-in-f14.1e100.net	TCP	outgoing	34176	443	52
2018-06-05 05:49:57	192.168.12.27	N/A	192.168.12.100	N/A	TCP	incoming	80	42060	52
2018-06-05 05:49:57	172.217.11.78	lax17s34-in-f14.1e100.net	192.168.12.48	N/A	TCP	incoming	443	34176	52

Figure 2.6: Network query from database with Navicat.

Within the power table, the name field is extracted from the WeMo. This can cause issues if the wrong device is connected to a WeMo. There is no way to check for this besides manual examination. After a few weeks, we reset all devices, causing a temporary void in power data for a few hours. Manual examination required turning the WeMo off from the iPad (where the WeMo name is displayed) and looking to see what device turned off. When we reset all devices, we also renamed some of the WeMos. For example, we changed "echoDot" to "Echo Dot". This way, all devices follow the naming convention of separating words with spaces and capitalizing each word.

The sampling frequency for the power table is also low and missing some points. Because the WeMo network connection is unstable, the script to query power data is unable to get power data every second. To handle missing points, I interpolate them

as shown in Section 2.4.

2.3.4 Device Inventory

To keep track of all devices and important details about them, we log a device's service, device, related email, IP address, dependencies, and password into a Google Sheets file called 'Device Inventory'. An example of entries into the Device Inventory are shown in Table 2.4. The service field denotes the service a device provides, for example, the Google Home is a smart speaker. The device field denotes the specific manufacturer and device name. The email and password fields denotes the email and password used when setting up the device. Some devices such as the Nintendo Switch were set up without an account. The IP address field denotes the IP address of the device within our local network. The dependencies field denotes anything the device is connected to or uses. For example, the Roku express is connected to a WeMo for power logging and comes with a controller for use.

Table 2.4: Device inventory excerpt. Password column not shown.

Service	Device	Email	IP Address	Dependencies
Speaker	Amazon Echo Dot 1	amazonEchoDotSS0@aol.com	192.168.12.79	WeMo
Speaker	Google Home	googleHomeMiniSS0@aol.com	192.168.12.48	WeMo
Streaming	Google Chromecast	googleChromecastSD0@aol.com	192.168.12.78	WeMo
Streaming	Roku Express	rokuExpressSD0@aol.com	192.168.12.68	WeMo, controller
Game Console	Nintendo Switch	n/a	192.168.12.160	WeMo, controller

2.3.5 IP History

As mentioned in Subsection 2.2.4, the IP addresses of each device are dynamic. They change over time as DHCP assigns new IP addresses to a device. A Google Sheets file is used to keep track of each device's IP address as it changes as shown in Table

2.5. The 'Date Noticed' column is the date that the IP address was noted to have changed. The 'Device' is the name of the device that changed IP address. The 'Old IP' and 'New IP' are the IP addresses for the device before and at that current time respectively. Newly added devices will have an 'Old IP' entry as 'None'.

Table 2.5: History of IP addresses for each device

Date Noticed	Device	Old IP	New IP
April 10, 2:15 PM	Samsung TV	192.168.12.148	192.168.12.149
April 11, 2:27 PM	Chromecast	192.168.12.77	192.168.12.78
April 12, 5:16 PM	Fire Stick	192.168.12.113	192.168.12.114
April 20, 10:31 PM	Galaxy Tablet	192.168.12.222	192.168.12.223
April 20, 5:37 PM	Camera	None	192.168.12.57
April 24, 5:36 PM	Home Montioring Kit	None	192.168.12.191
May 8, 5:59 PM	Home Montioring Kit	192.168.12.191	192.168.12.100
May 8, 6:21 PM	Camera	192.168.12.57	192.168.12.84/83
May 18, 12:04	Samsung TV	192.168.12.149	192.168.12.191

2.3.6 Usage Flow

One of the goals of this work is to create a data set that represents the baseline of normal network traffic and power usage. To do this, we used each IoT device at least twice a week for the year length of this research. There is a void in usage for Summer 2018 because no students were at Cal Poly to log data. To build a proper dataset that other people could use for research, we set up a list of things that we should do with each device. We logged the activities into Google Sheets so that these events could be correlated to entries in the database. For example, if someone were to look at the database and notice that the power and network traffic was high for 3 minutes,

they could look at our logs and see that the device was streaming music for those 3 minutes and assume that it was normal usage.

When logging events into Google Sheets, we include the start, end time, name of the device(s), action performed, and any individual notes. When naming the devices, we use the same naming as we do for the WeMos to maintain consistency from the log to the database. An example of entries into the table is shown in Table 2.6.

Table 2.6: Event Log Excerpt

Date	Start Time	End Time	Device	Event
5/25/2018	12:44:42	12:47:04	Home Mini	Ask for news
5/25/2018	12:55:07	12:56:37	Echo Dot	Ask for news
5/25/2018	12:55:57	12:56:17	Echo Show	Ask for weather
5/25/2018	12:56:44	13:03:07	Home Mini	Play music

The following subsections explains the script created to automate the event logging portion of the research. Then it explains the specific procedure we ran each group of IoT devices through whenever we used them.

2.3.6.1 Google Sheets Script

Logging information into the spreadsheet is a tedious task, especially when trying to analyze each device while running tasks on them. To minimize human error, I created a script in Google Sheets to automatically populate the date and time an entry is added. The code for the script is shown in Listing 2.3.

```
function onEdit() {
  var s = SpreadsheetApp.getActiveSpreadsheet().
     getSheetByName("Event Log");
  var r = s.getActiveCell();
  var c = r.getColumn();

  if( c == 4 || c == 5) { // checks the column
    var dateCell = r.offset(0, -3);

    if( dateCell.getValue() === '' ) {// is empty?
      var startTimeCell = r.offset(0, -2);

      // fill in the start time and date
      var date = new Date();
      dateCell.setValue(date);
      startTimeCell.setValue(date.toLocaleTimeString());
    }
  }

  if( c == 6 ) { // checks that the description is being
     entered
    // if so fill in the end time
    var endTimeCell = r.offset(0, -3);

    if ( endTimeCell.getValue() === '' ) {
      endTimeCell.setValue(new Date().toLocaleTimeString());
    }
  }
}
```

Listing 2.3: Open and Read from a Socket

2.3.6.2 Smart Speakers

Whenever interfacing with these devices, we queried for the weather and the news. The specific phrases we used were "<wake word> what's the weather" and "<wake word> what's the news", we set a reminder for a random task in a random timeframe, and we muted the devices for a few minutes to see if they were still listening if we said the wake word.

2.3.6.3 Video Game Consoles

When interfacing with these devices, we played a game on the device for at least 15 minutes. Every week, we also browsed for games on the game store and downloaded free demos.

Afterward, we turn off the Xbox and put the Nintendo Switch into sleep mode. The Nintendo Switch was only set to sleep because it can only shut down when disconnected from the dock.

2.3.6.4 Streaming Devices

When interfacing with these devices, we watched YouTube for at least three minutes.

For the Chromecast, we cycled through different devices to cast videos from, including the iPad and the Chromebook. We wanted to get varied data in case the Chromecast prioritized streaming from different devices.

2.3.6.5 Tablets

In this experimental setup, we used the tablets less for investigative purposes, but more to set up and interface with other devices.

However, we still tracked the data and had a small set list of things to do on these devices. We used each tablet to browse the web, watch YouTube, and use various apps.

2.3.6.6 Security Camera

The only security camera we worked with was the Eray Security Camera.

Whenever interfacing with this device, we used the device through the app NVAS2 [**?**] on the iPad for at least 2 minutes. Usage included streaming video from the camera to iPad to view it. We also controlled the camera through the app with pan and tilt commands. When done experimenting with the security camera, we disconnected the tablet from the camera by using the "end stream" button in the NVAS2 app.

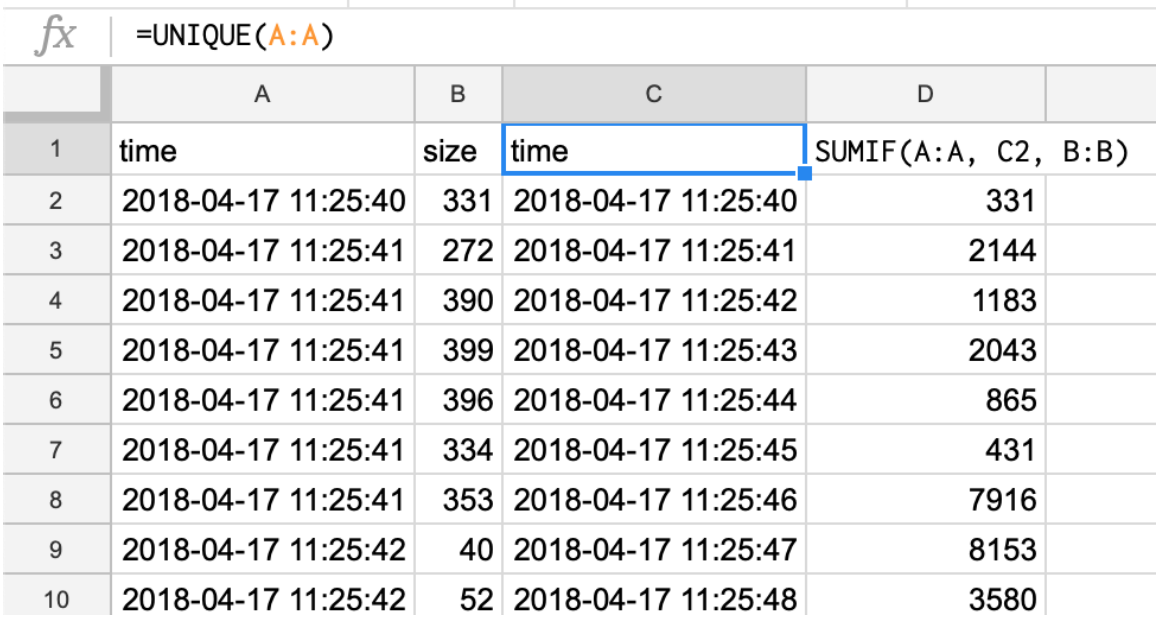
2.4 realtimeIoTGrapher.py

This section covers a visual tool I created to look for trends. Originally, for trend analysis, we queried the database for the necessary data, copied the data over to Google Sheets, created the necessary formulas, graphed the network and power usage over time, and finally analyzed this graph. The tables and formulas are shown in Figure 2.7. This process was tedious and time-consuming because we had to manually format an SQL query, entering unique time values and the IP address of the device. Copying large datasets from SQL into Google Sheets also causes browser lag.

To address these problems, I created a Python script that automatically forms graphs, including the one shown in figure, 2.12 when given a time range and the desired devices. This automation sped up the analysis process and provided extra features covered in Section 2.4.0.1.

This Python script leverages the Plotly [**?**] library, which, given data, graphs it onto a local web page for viewing. Plotly comes with many useful tools for further analysis and can be extended to run on a public web page for public viewing.

I plan to distribute this tool with our database so that researchers can view the power and network information of any device at a glance.



fx =UNIQUE(A:A)

	A	B	C	D
1	time	size	time	SUMIF(A:A, C2, B:B)
2	2018-04-17 11:25:40	331	2018-04-17 11:25:40	331
3	2018-04-17 11:25:41	272	2018-04-17 11:25:41	2144
4	2018-04-17 11:25:41	390	2018-04-17 11:25:42	1183
5	2018-04-17 11:25:41	399	2018-04-17 11:25:43	2043
6	2018-04-17 11:25:41	396	2018-04-17 11:25:44	865
7	2018-04-17 11:25:41	334	2018-04-17 11:25:45	431
8	2018-04-17 11:25:41	353	2018-04-17 11:25:46	7916
9	2018-04-17 11:25:42	40	2018-04-17 11:25:47	8153
10	2018-04-17 11:25:42	52	2018-04-17 11:25:48	3580

Figure 2.7: Manual Analysis of IoT devices in Google Sheets: Raw data shown in columns A and B are then formatted into columns C and D for Graphing

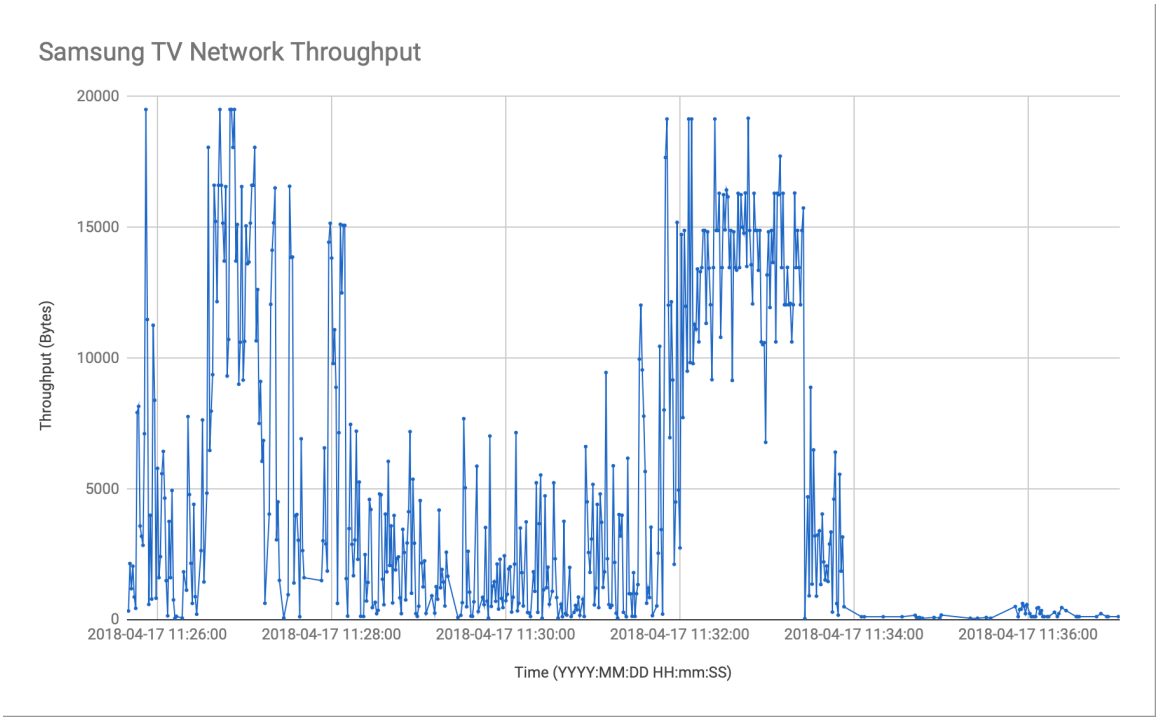


Figure 2.8: Resulting graph of dataset shown in Figure 2.7

2.4.0.1 Features

This section discusses the features that RealTimeIoTGrapher (grapher) provides.

At its core, this program automates the creation of the graph shown in Figure 2.8. The user can specify a time range that the graph should cover and what devices to graph. By default, the graph shows power, input traffic, output traffic, and total network traffic over the time range specified.

The grapher also annotates the data by including a line denoting the average for each of the traces over the time range currently displayed. It also annotates the maximum and minimum values in the time range for each trace. The grapher can also display information in real time for any specified window size (update live showing ten minutes at a time), updating every second.

The Plotly libraries also provide useful tools when displaying these IoT graphs. It is possible to zoom in and out the displayed graph, select specific traces for viewing, save the graph, and hover over data points to display the specific value. Finally, once a graph is saved as shown in Figure 2.9 Plotly allows further editing of the graph so that it is formatted the way the user wants as shown in Figure 2.10.

Figure 2.9: Screen of saved plotly graphs.

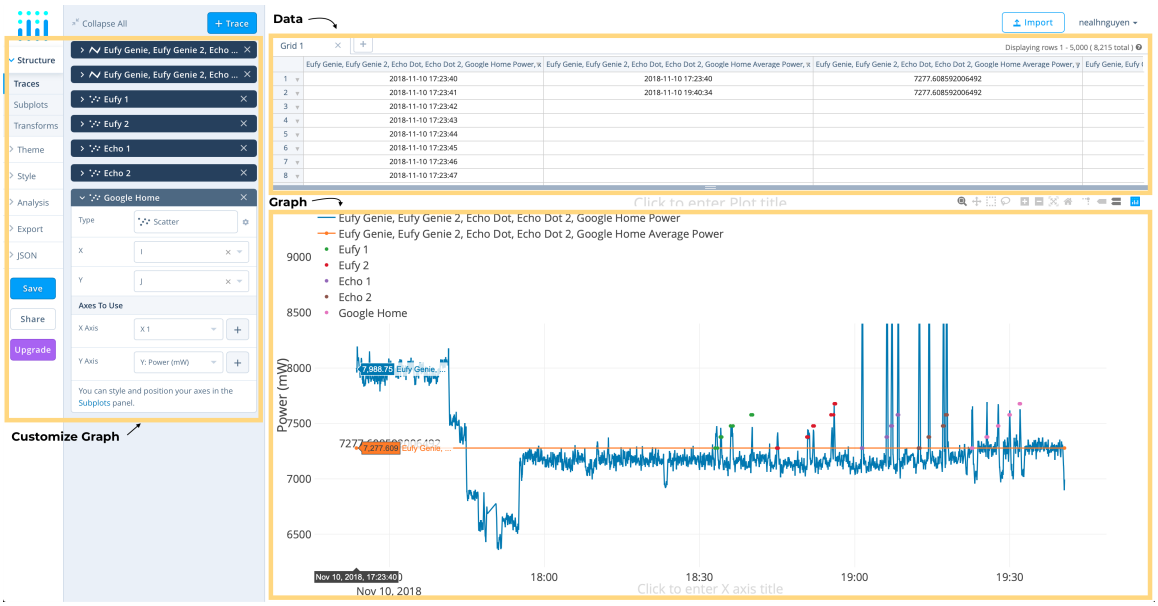


Figure 2.10: Screen to edit saved Plotly graphs containing an edit tool bar, the graph data, and the graph.

```
1    def throughput_query_in_range(db_connection, device,
        start_time, end_time):
2    sql_query = """
3        SELECT
4            time,
5            CASE WHEN type IS NULL THEN 'total' ELSE type
                END type,
6            SUM(size) AS total_throughput
7        FROM ip_log.ip
8        WHERE
9            time BETWEEN '%s' AND '%s' AND
10           (source = '%s' OR destination = '%s')
11       GROUP BY time, type WITH ROLLUP;
12   """ % (start_time, end_time, ip[device], ip[device])
13
14   dataframe = pd.read_sql_query(sql_query, db_connection)
15
16   return dataframe
```

Listing 2.4: Efficient SQL query to obtain total Network throughput at each second.

To reduce local computation, the grapher offloads work to the server through an SQL query. ROLLUP causes the query to group data by different combinations specified in the ROLLUP. When querying for network data, the grapher performs an SQL ROLLUP to sum the incoming and outgoing network throughput to obtain the total throughput. This query is formatted as shown in Listing 2.4.

To simplify data handling in the grapher, the grapher stores queried data into Pandas [**?**]. Pandas is a common library used for dealing with tabular data. It integrates with

SQL and has builtin functions to query directly from a database into a Pandas object. Pandas provides easy key, value scheme for indexing by column or row. It also comes with many useful built-in mathematical and scientific functions to perform on the table.

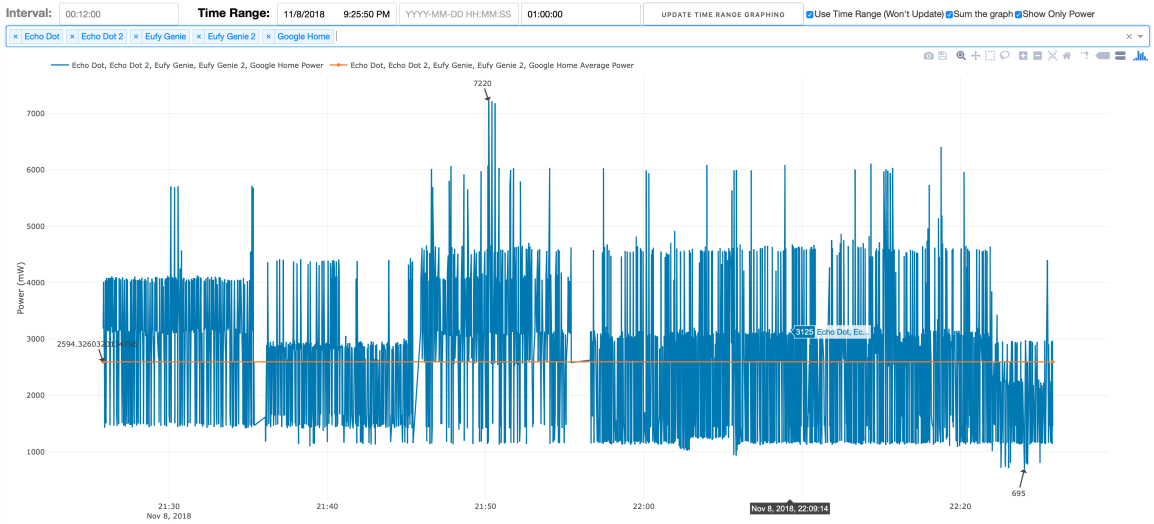


Figure 2.11: Summed power traces without interpolation.

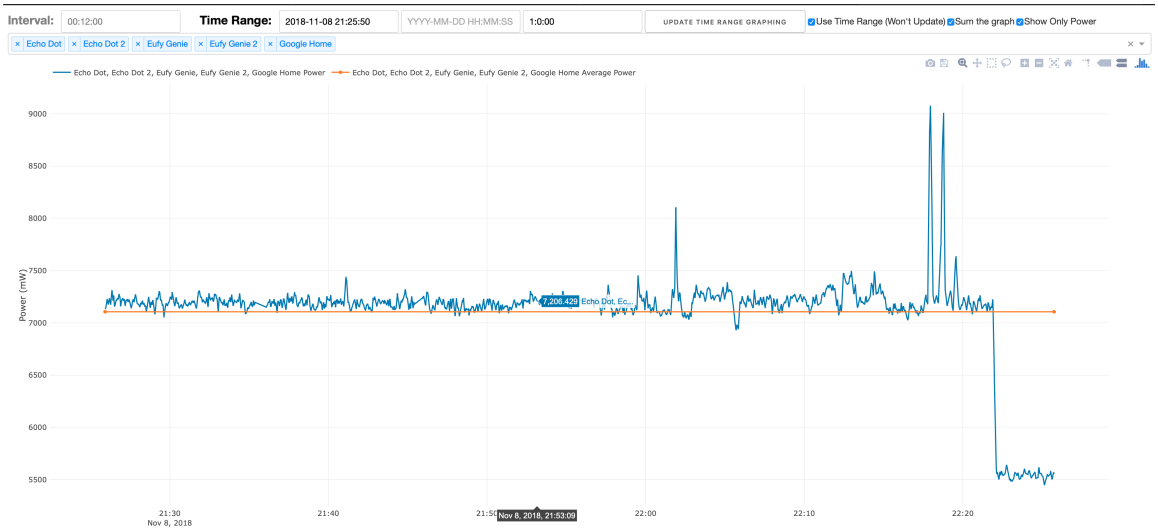


Figure 2.12: Summed power traces with interpolation.

The grapher can also sum all power traces into one single trace. Originally, when

summing the graphs, it would add the two Pandas data frames that represent each trace, and graph the result, shown in Figure 2.11. However, the WeMo does not always sample every second. This is an issue if, at some time, one device X has a power point, but another device does not. The summation counts the empty data point as 0, resulting in an erroneous sum for certain frames. This was solved by interpolating data for every second for each device before summing data frames with a built-in Pandas feature. Interpolation is shown in 2.12 as a graph.

2.4.0.2 Visual Takeaway

This tool created most of the images shown in Chapter 3. A few other researchers, including Frawley, have used this tool for data visualization. The grapher provides real-time or historical graphs on any device in the database with little effort. Additional features from Plotly also provides an easy way to format graphs for presentations or papers.

The summation feature can simplify the graph and provide a more holistic view of power usage, making it easier to spot patterns. In this paper, the summation feature is used to simulate the concept of a household's "powerline" where all energy usage is be summed up into one power meter.

2.4.0.3 Limitations

As this tool was created entirely for research purposes, there are some edge cases that it fails to handle.

When the database has too many connections, the grapher refuses to connect, and the grapher will not work. When graphing in real time, the grapher makes a new connection

for each query. If a query hangs in the graphing tool in live mode, connections can quickly build up as the grapher continually makes a new connection every second without the previous queries closing.

The graphing tool also slows as the time frame increases. We have noticed a slow down for queries with time frames longer than seven hours. In the static graphing mode, the tool slows down before graphing the information. However, in real time graphing mode, the tool makes many slow queries, eventually causing too many connections to the database and undefined behavior.

The graphing tool uses a lookup table that maps the IP address of the device to the corresponding name given to the WeMos. Because we manually create the lookup table, when any of the IoT device's IP changes, the graphing tool is not able to find the device. In some cases, this causes the graphing tool to crash. To solve this, the user must manually change the IP addresses defined in the lookup table in lines 23-39 shown in Listing 2.5.

```
    #region IP Addresses of all devices we are tracking
    ip = {
        'Chromecast':       '192.168.12.77',
        'Echo Dot 2':       '10.42.0.132',
        'Echo Dot':         '10.42.0.150',
        'Eufy Genie':       '10.42.0.223',
        'Eufy Genie 2':     '10.42.0.172',
        'Fire Stick':       '192.168.12.113',
        'Google Home':      '10.42.0.236',
        'IP Camera':        '192.168.12.58',
        'Nintendo Switch':  '192.168.12.160',
        'Roku':             '192.168.12.68',
        'Samsung Hub':      '192.168.12.100',
        'Samsung TV':       '192.168.12.191',
        'Smart Light':      '192.168.12.27',
        'Xbox':             '192.168.12.251',
        'Echo Show':        '192.168.12.122',
        'Appliance':        '192.168.12.122',
        'Appliance1':        '192.168.12.122',
    }
```

Listing 2.5: IP lookup table in real time IoT grapher.

Chapter 3

RESULTS AND DISCUSSION

This chapter covers different network and power usage patterns for the IoT devices in the testbed. First, Section 3.1 informally compares the smart switches between devices to confirm that they are accurate enough for this research's purposes. Section 3.2 covers smart speaker and streaming device network and power use while idle, in use, and during first boot. Section 3.2 serves as precursory analysis to fingerprint devices and give example uses of the database and visualizer tool.

After general analysis, Section 3.3 shows how a power trace may highlight the presence of a user in their home. Sections 3.4 and 3.5 cover summed power graphs of the smart speakers with and without noise from high power household devices.

When obtaining data, most of the packets are encrypted, so the data presented focuses on metadata such as size, protocol, source IP, and destination IP. For power analysis, we focus on the power used at each second.

3.1 Swapping the Power Switch

To check if the smart switch voltage readings are accurate, we swapped various devices with various power switches. Some examples are shown in Figures 3.1, 3.2, and 3.3. After swapping the switches, each device takes some time to start up, the colored boxes in Figures 3.1, 3.2, and 3.3 highlight this time.

In every case, the traces swapped as we swapped the device connected to each smart switch. This informal sanity check confirmed that the power switches were precise

enough to enable basic device comparisons.

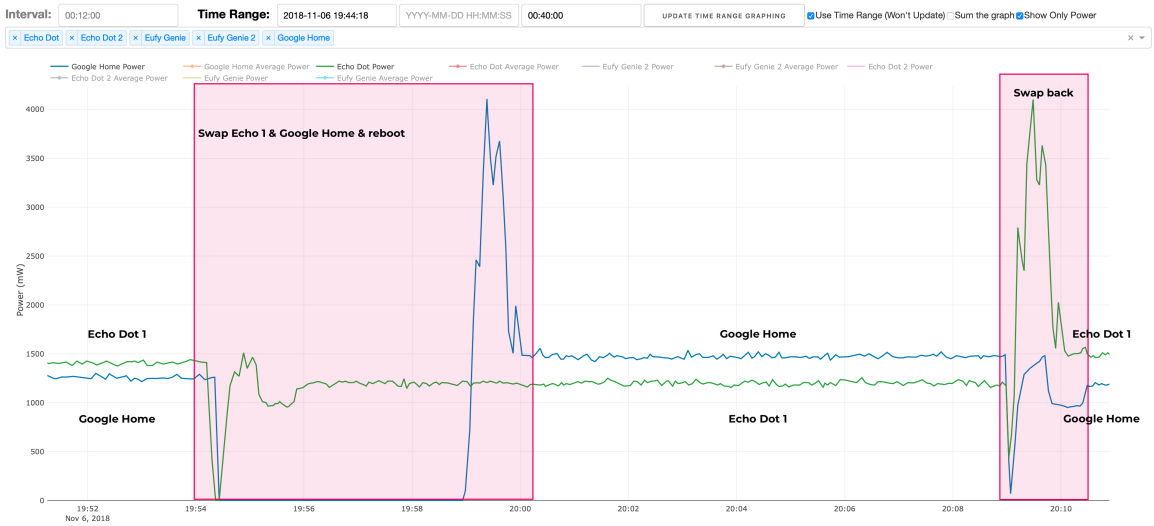


Figure 3.1: Connected the Echo Dot 1 to WeMo for 'Eufy Genie 1' and vice versa. The traces swap before and after switching.

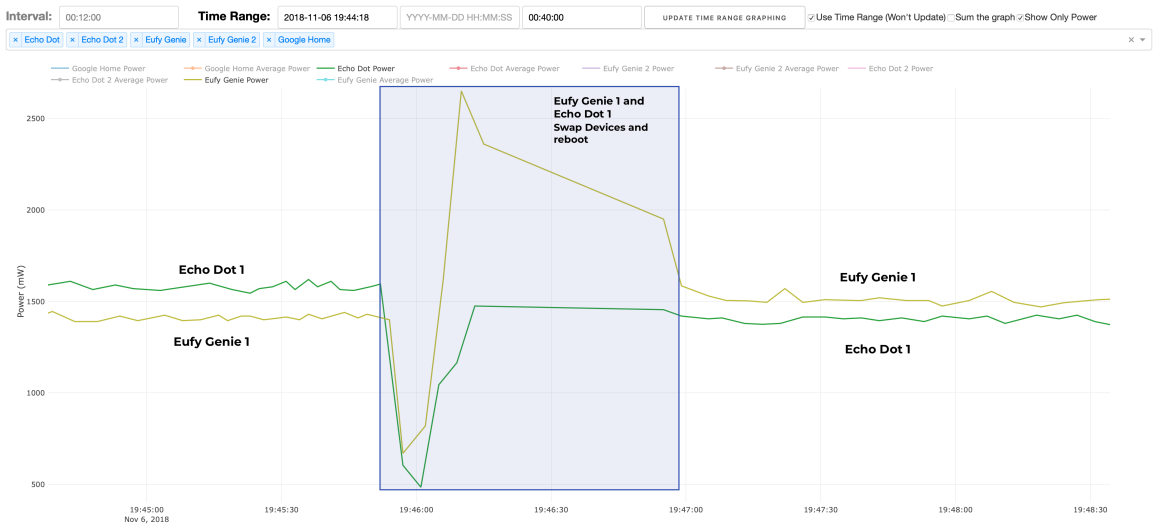


Figure 3.2: Connected the Eufy Genie 1 to WeMo for 'Echo Dot 1' and vice versa. The traces swap before and after switching.

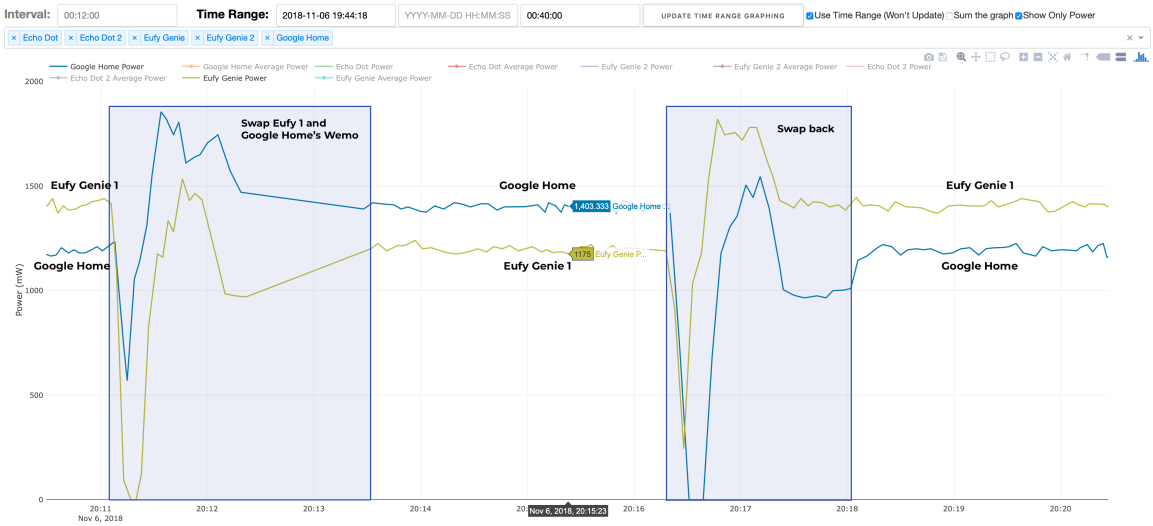


Figure 3.3: Connected the Google Home to WeMo for 'Eufy Genie 1' and vice versa. The traces swap before and after switching.

3.2 General Analysis

The figures shown in this section give examples of how the database and grapher tool is used. The figures also demonstrate a strong correlation between a device's network/power usage and what it is doing, serving as data to fingerprint each smart speaker.

Frawley's paper contains a more in-depth analysis of this topic. His paper includes GeoIP [**?**] information from Cal Poly's ITS servers [**?**] that highlights the locations and companies the network packets are sent to and received from. Frawley's paper also includes data of unencrypted packets being sent and received along with the text within them.

3.2.1 Google Home Mini General Analysis

The Google Home Mini's idle traffic broadcasts SSDP (Simple Service Discovery Protocol) packets once every minute. SSDP packets are a discovery request packet for every IoT device on the network that supports UPnP (Universal Plug and Play). After devices such as the Echo Dot, Samsung TV, and the Chromebook receive an SSDP packet, they respond with a .xml file containing details regarding the device, operating system, and more. Google also sends encrypted TCP packets to Google every 10 minutes. Figure 3.4 shows the Google Home Mini's idle traffic. The large spikes are the encrypted TCP packets, and the smaller spikes are the SSDP/UPnP packets.

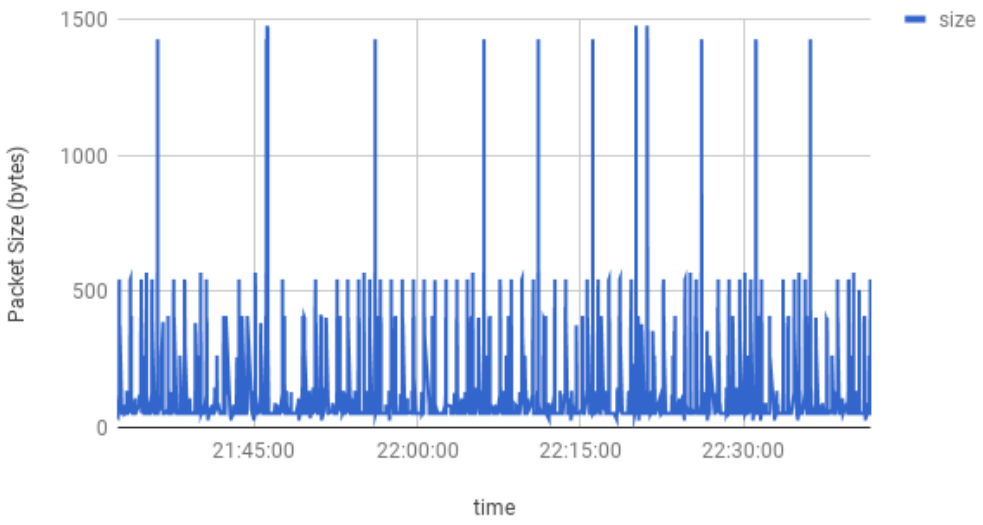


Figure 3.4: Idle Traffic of Google Home Over 1 Hour Period

Figure 3.5 shows a graph of the Google Home Mini's network and power usage while performing various commands. The Google Home Mini is first asked for the news and then told to stop. After waiting for 40 minutes, we ask it for the weather forecast.

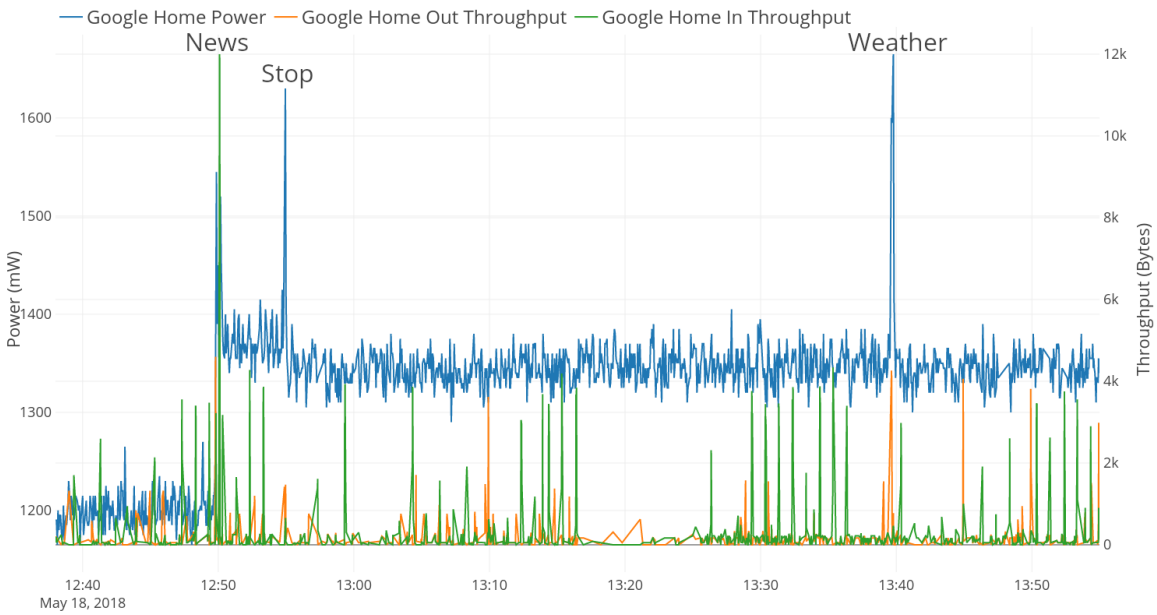


Figure 3.5: Home Mini Response to News and Weather

3.2.1.1 Google Chromecast

Figure 3.6 shows the Chromecast's power and network usage during startup. On startup, the Chromecast shows an intro tutorial video and downloads a firmware update. After rebooting twice, it is ready to go.

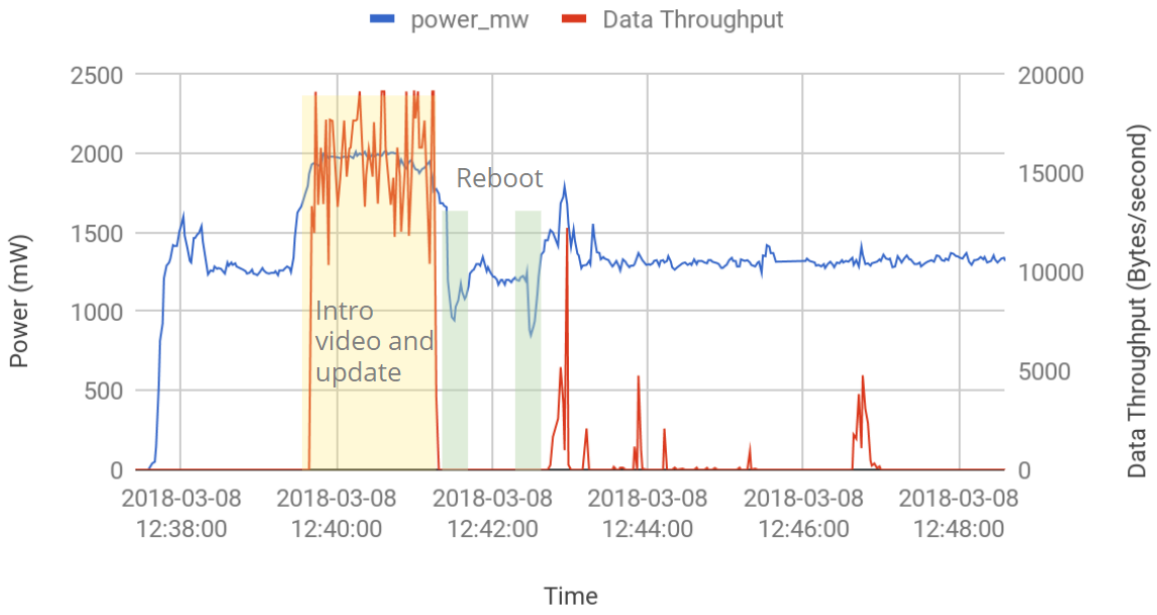


Figure 3.6: Chromecast First Time Boot Network Traffic and Power Consumption

Figure 3.7 shows the Chromecast's power and network usage while streaming highlighted in green.

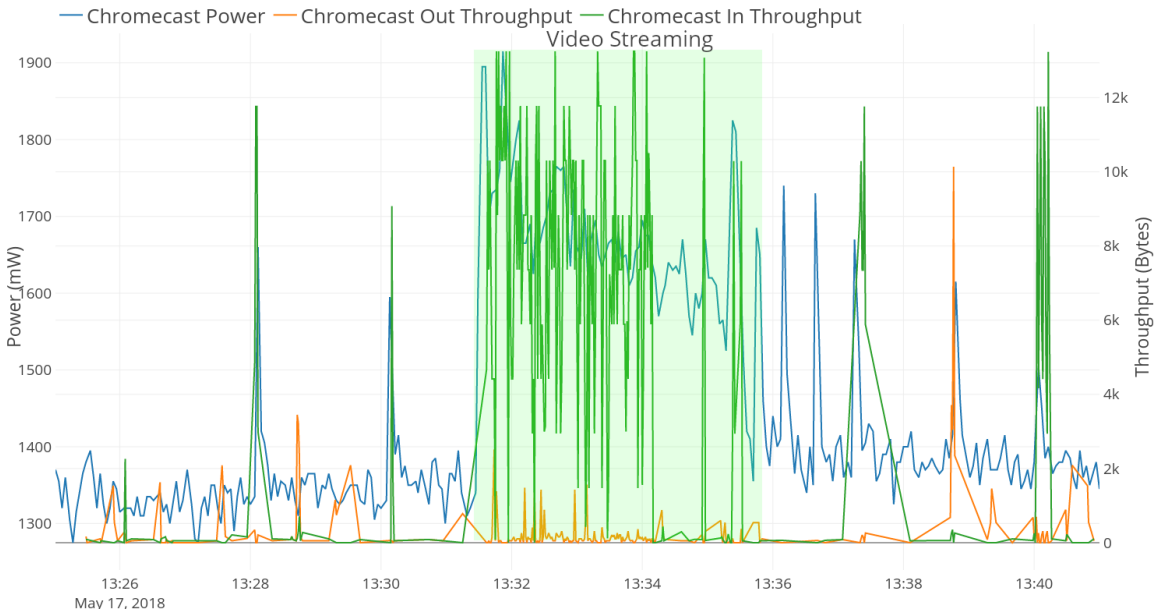


Figure 3.7: Chromecast Video Streaming

Additionally, Figure 3.8 shows the Chromecast's network and power usage while idle. In this time frame, there are consistent power and network spikes every two minutes. During these spikes, the Chromecast is downloading an image to update the background on the TV it is connected to.

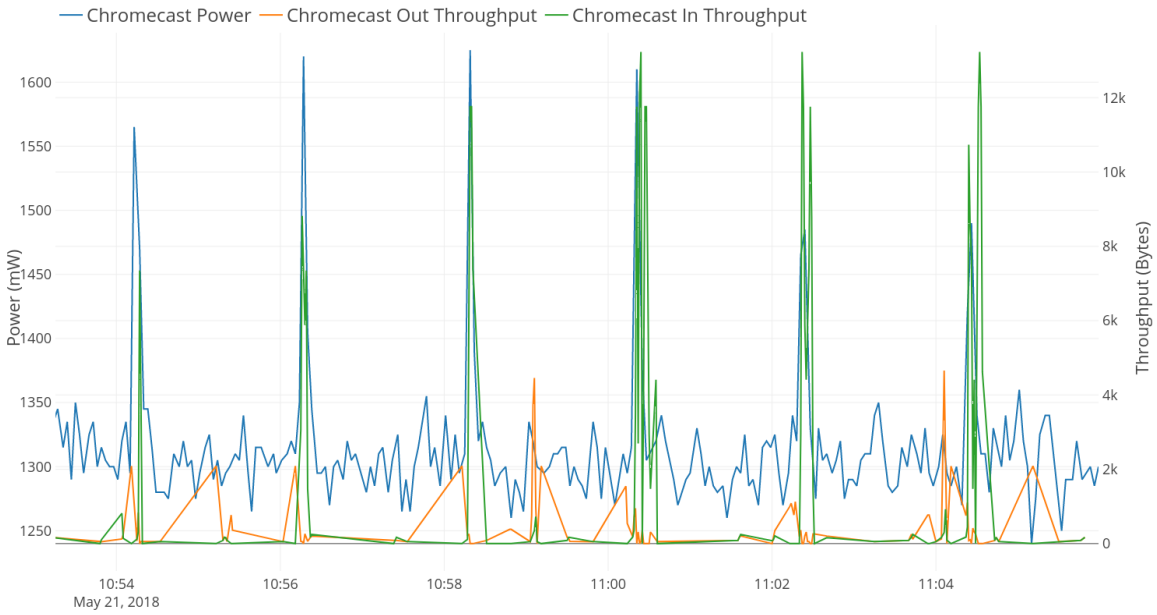


Figure 3.8: Chromecast Idle Traffic

3.2.1.2 Amazon Fire Stick

Figure 3.9 shows the Amazon Fire Stick's power and network usage during startup. On first boot, the fire stick downloads an update, reboots twice, then downloads certificates from Symantec and Verisign.

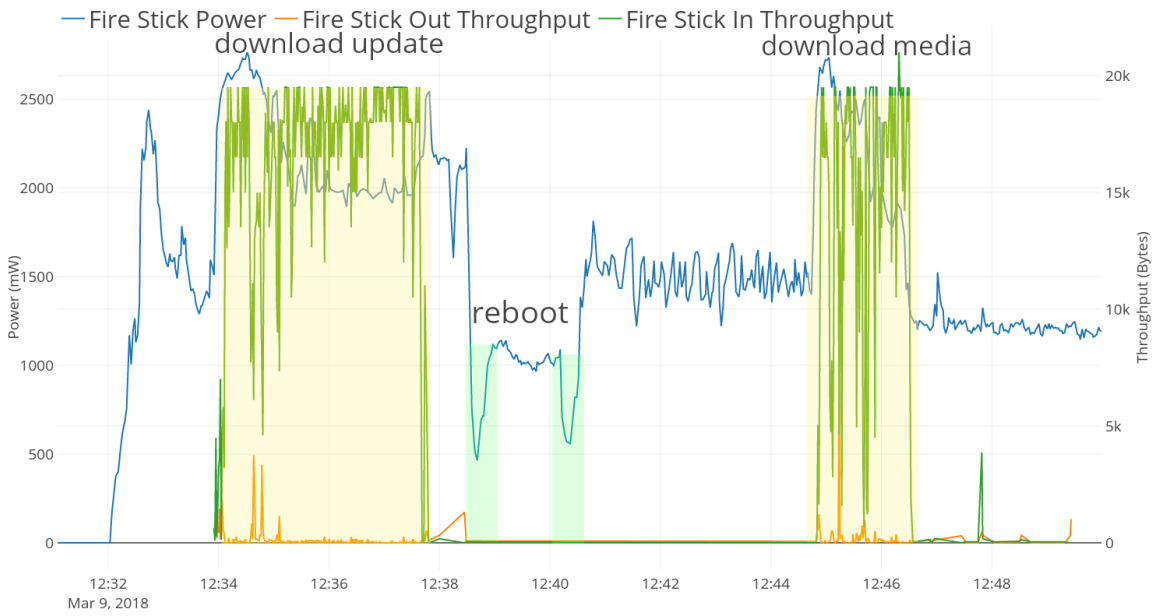


Figure 3.9: Fire TV Stick First Time Boot Network Traffic and Power Consumption

Figure 3.10 shows the Amazon Fire Stick's power and network usage while streaming. When streaming, the Fire Stick increases throughput and power usage. There are a few power spikes at the beginning of the stream time and one power spike at the end with raised power usage in between.

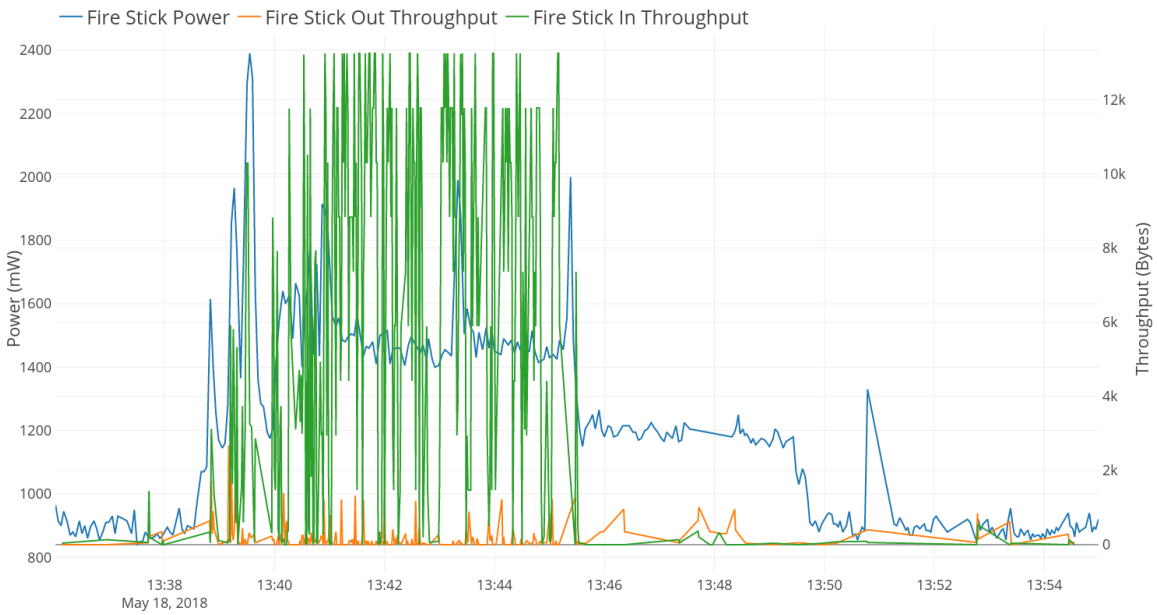


Figure 3.10: Fire TV Stick Video Streaming

3.2.1.3 Roku Express

Figure **??** shows the Roku Express's power and network usage while streaming. When streaming, the Roku's power usage and network throughput rise and stay at a constant level until the video streaming is complete. At which point it drops back down when done.

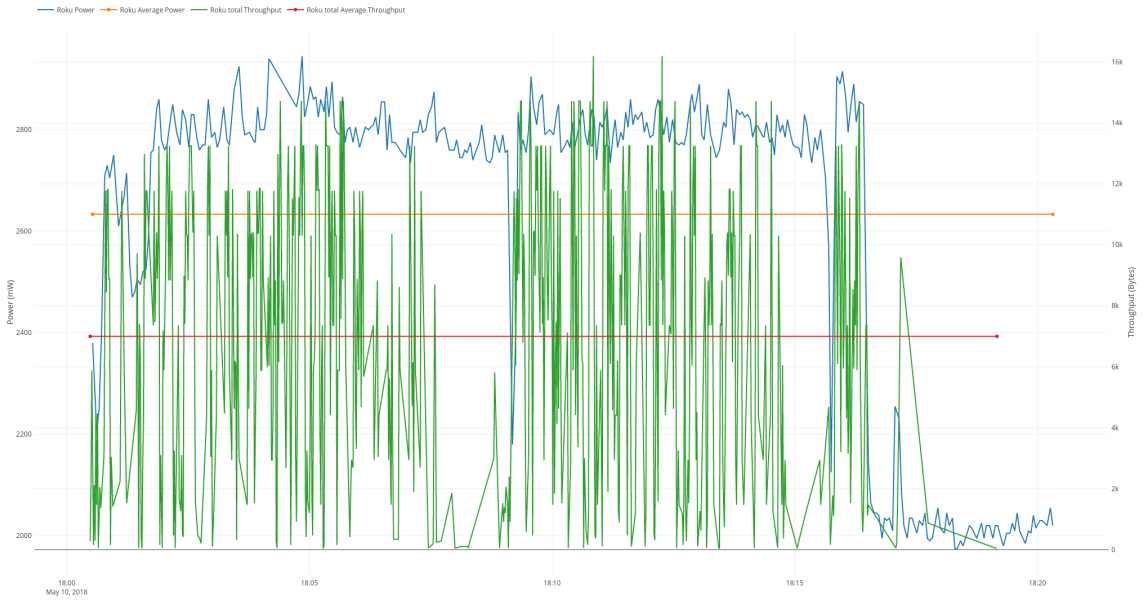


Figure 3.11: Roku Express Video Streaming

3.2.2 General Analysis Discussion

From all the figures in this section, there is a visual correlation between network/power usage and what a device is doing. During periodic updates, the smart speakers and streaming devices have periodic network/power usage spikes as shown in Figure 3.8. While streaming, the power/network usage increases for the period of the stream as shown in 3.7, 3.10, and 3.11. This strong correlation shows the value of the database and visualizer tool for IoT device analysis. It also indicates that it may be possible to use network throughput/power usage over time to understand what a device is doing.

3.3 Power Analysis on Smart Speakers

There is already significant analysis on network usage of IoT devices as shown in the previous works Section 1.1. This paper focuses on the smart speakers' power usage to

focus the scope of research.

This section first examines the power usage of the smart speakers separately before examining total power usage in Section 3.4.

3.3.1 Smart Speaker Comparison

This Subsection compares the energy and network usages of the three smart speakers individually and speculates trade-offs that they make.

In the Figures 3.12 and 3.13, we display the power and network traces for the Echo Dot 1, Eufy Genie 1, and Google Home. We used the same time frame as the graph in Figure 3.21 for the two graphs for Figures 3.12 and 3.13. We removed the second Echo Dot 2 and Eufy Genie 2 so that there is one of each device.

In the time frame of Figures 3.12 and 3.13, the average power, from greatest to least, starts with the Echo Dot (1650 mW), then the Eufy Genie (1325 mW), and the Google Home (1175 mW). For average throughput, from greatest to least, there is the Eufy Genie (1743 bytes), Google Home (800 bytes), and Echo Dot (325 bytes). These results are shown in Figure 3.14

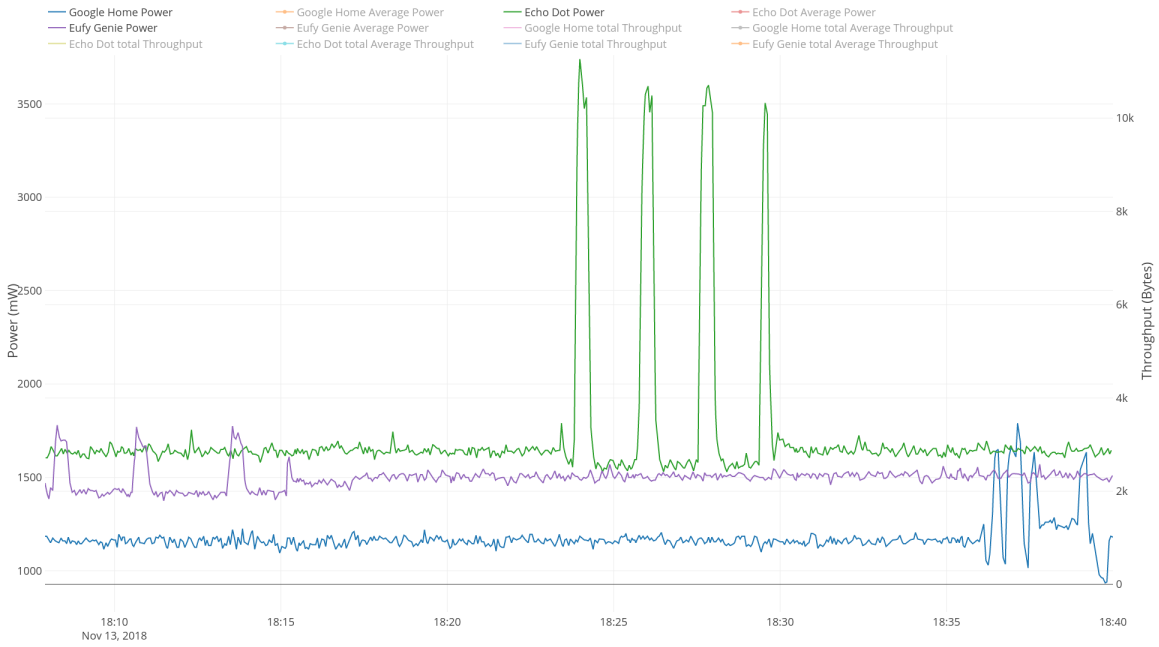


Figure 3.12: Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked "who is the best basketball player". Same graph as Figure 3.21 with Echo Dot 2 and Eufy Genie removed.

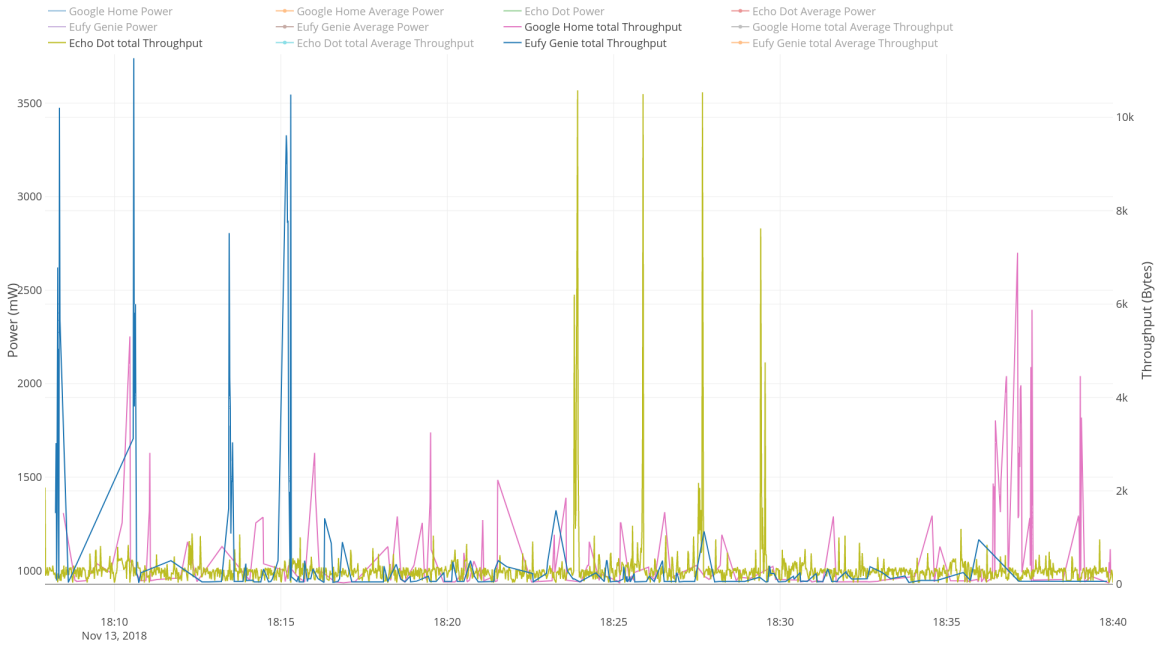


Figure 3.13: Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked "who is the best basketball player". Same time frame as Figure 3.12

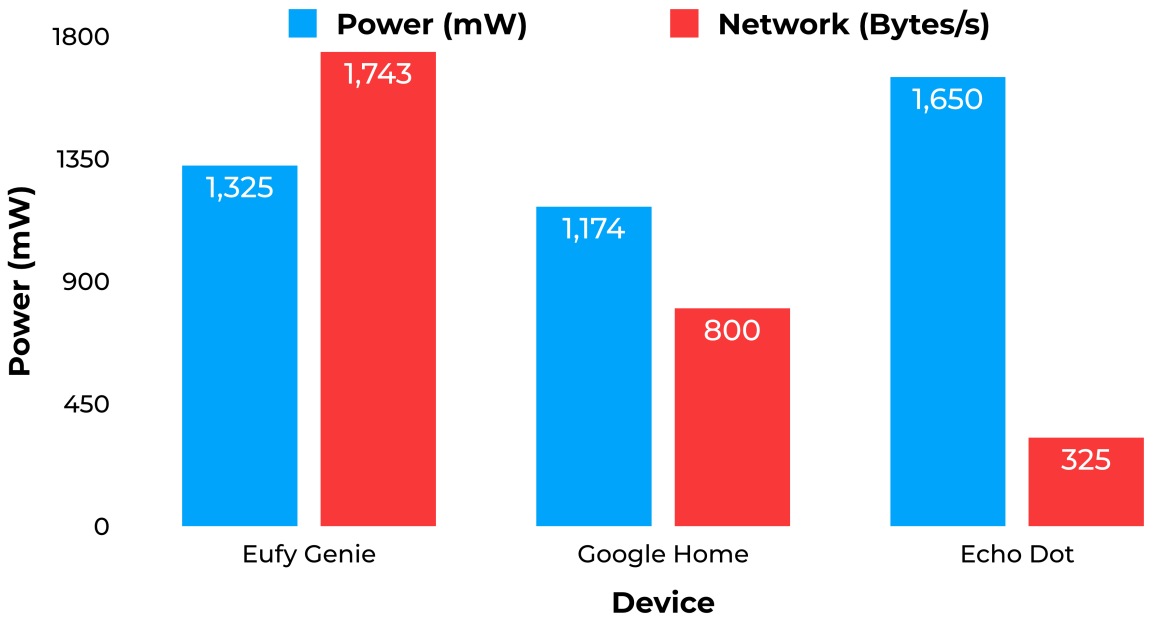


Figure 3.14: Average power and network usage at each second for the Echo Dot, Eufy Genie, and Google Home in the time frame of Figures 3.12 and 3.13

From Figure 3.14, we can see that the Echo uses the most power, then the Eufy, then the Google Home. However, the Eufy uses the most network throughput, then the Google Home, then the Echo Dot. We believe each device is making some tradeoff between power and network usage. Possibly, the Echo dot is trying to do as much on board operations as possible, increasing the power usage. Possibly, the Eufy Genie is constantly querying for information, thus reducing power usage, but increasing network usage. But it is also possible the Echo Dot has poor power optimizations. Further analysis to understand this could be interesting.

3.3.2 Echo Dot Brightness Sensor

One interesting finding from the Echo Dot is that it has a light sensor When turning on the room lights, the LEDs on the Echo Dot brighten to adapt. The brightness change causes the Echo Dot to use more idle power as shown in Figure 3.15.

Figure 3.15 shows that the power usage of an Echo Dot can show if the lights are on in a house. This can help someone determine if someone is in their house or not or what room they are in, potentially introducing some privacy concerns.

Figure 3.15: Echo Dot Response to Lights

3.3.3 Baseline Speaker Power

To determine what else can be learned from a smart speaker's power usage, this section examines the idle power of each smart speaker. Figure 3.16 shows this from 1:00 AM to 2:30 AM when none of the devices are in use.

The Echo Dot 1 has the highest idle power as shown in the pink trace, the Echo Dot 2 has second highest idle power as shown in the green trace, the Eufy 1 and Eufy 2 have roughly the same idle power usage as shown in the yellow and purple traces, and the Google Home has the lowest idle power usage as shown in the blue trace. With visual analysis, the smart speaker can be matched to idle power usage. But the power usages can overlap between the Echo Dot 2 and Eufys. This makes it difficult to

differentiate them and shows this method to identify a device is difficult. Also, on a shared powerline where the all power use is combined, these idle traces disappear, with no way to differentiate what idle devices are contributing to the total power usage.

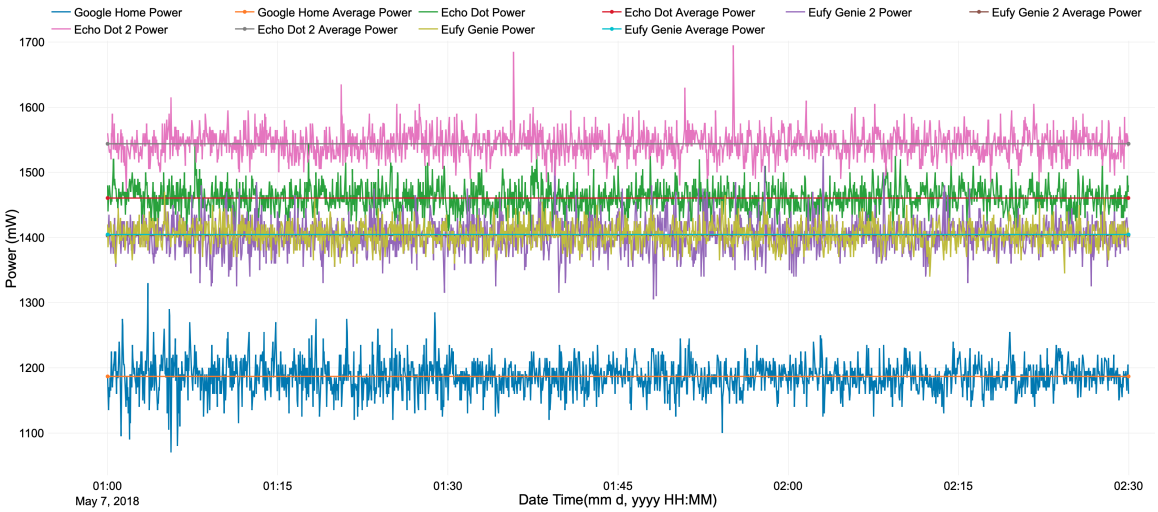


Figure 3.16: Baseline smart speaker power usage.

3.3.4 Smart Speaker Power Spikes When Asked for Weather

This section continues to see if a smart speaker can be determined from power by focusing on power characteristics while the devices are in use. Figure 3.17 shows the power data where each device is asked for the weather four times.

In Figure 3.17 there is a visual spike for each device while it is giving the weather forecast. Each device is highlighted while in use. The Google Home was queried for the weather five times and thus contains an extra spike. From this data, we can note a Eufy trace has 400 mW peak to peak spikes, the Google Home has 600 mW peak to peak spikes, and the Echo Dot has 2,100 mW peak to peak spikes. This difference in magnitude implies that a device can be determined from a power trace if the device is asked for the weather. The next section analyzes more use cases, and the traces are summed together to simulate a shared home powerline more closely.

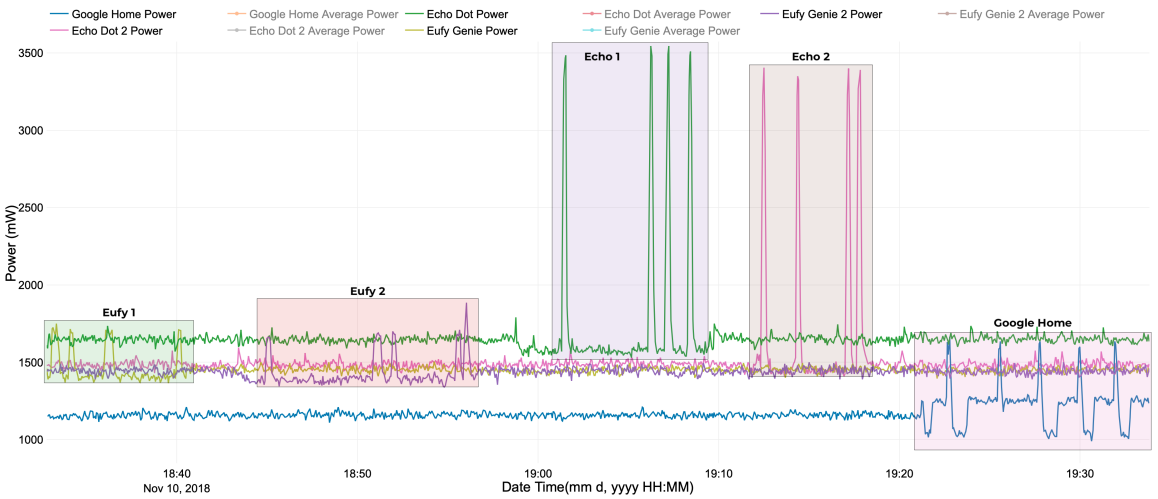


Figure 3.17: Smart speakers' power usage when asked 'whatś the weather' four times.

3.4 Summed Power Graph

This section attempts to analyze the power usage of all 5 of the smart speakers for different commands.

In Figure 3.18 we queried each smart speaker for the weather while all other smart speakers were muted at the 18:30 mark. We query each device 3-4 times for the weather. The Eufy had the smallest spike for the "what is the weather" command at 400 mW, then the Google Home at 600 mW, and finally the Echo Dot at 2000 mW, consistent with results in Section 3.3.4.

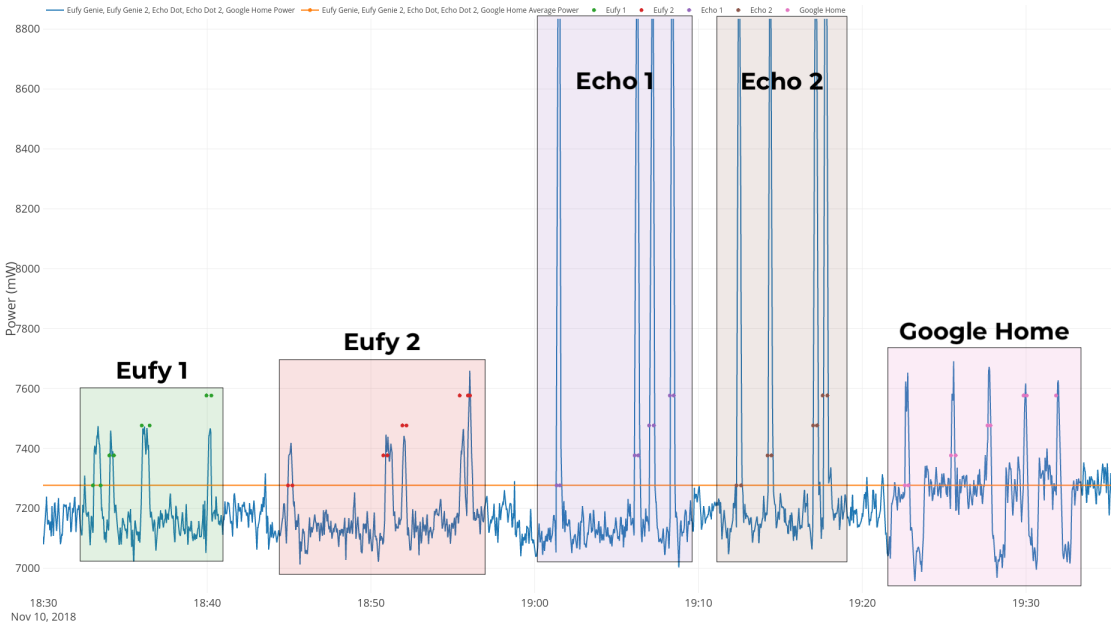


Figure 3.18: 5 Smart Speakers Power Summed Together.

Figure 3.19, shows the smart speakers power usage when asked for the news. This graph and the rest of the summed graphs in this section use the same notation for signifying commands: Two dots of the same color on the same y-axis level signify the start and end of the command for a specific device.

In Figure 3.19, all devices have a spike at the beginning and end of the command and maintain a steady energy usage in between that is slightly higher than the idle energy used. The peak to peak spike of the Eufy Genie is the smallest at 350 mW, then the Google Home at 500 mW, and finally the Echo Dot at 1900 mW.

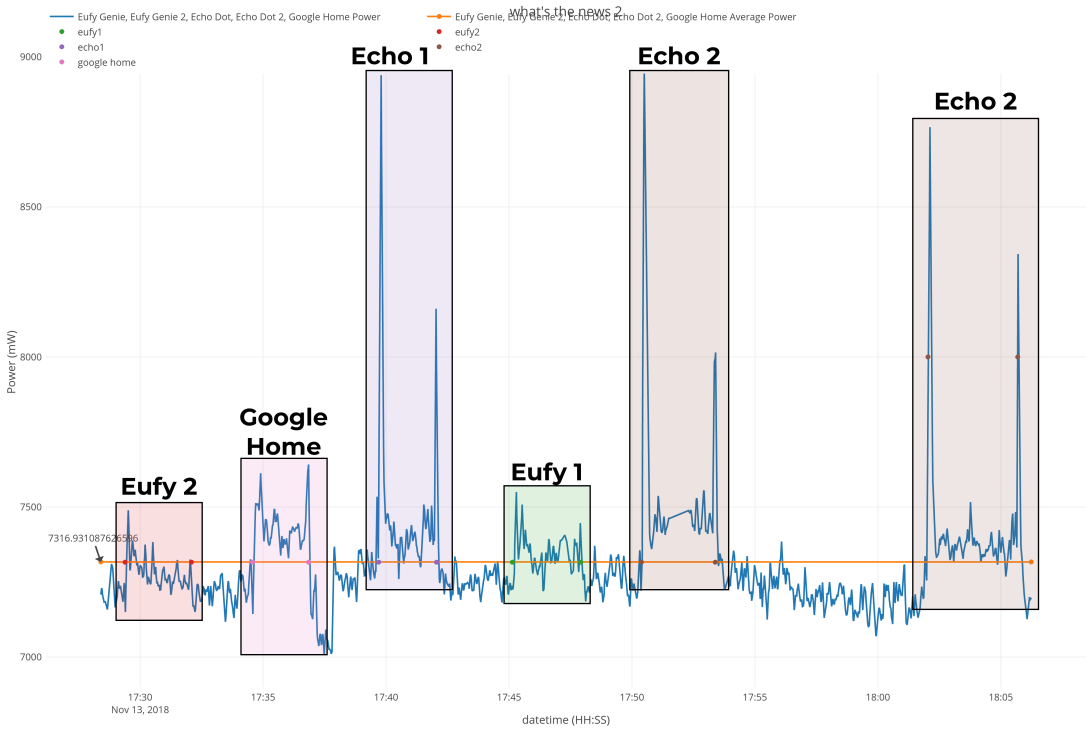


Figure 3.19: 5 Smart Speakers Power Summed Up. Queried each device for the news.

Figure 3.20 shows the smart speakers when asked for the best basketball player. The annotation scheme is the same as before. Each smart speaker is queried for the best basketball player in consecutive order four times. Like before, the Eufy has the smallest power spike at 420 mW peak to peak, the Google Home has a power Spike of 720 mW, and the Echo Dot has a power spike of 2180 mW.

When looking at graph 3.20, there is a power spike that is unaccounted for in correspondence to the event log at 18:12.

To figure what the power spike at 18:12 is, we separated the graphs into individual power traces as shown in Figure 3.21. From this graph, the power spike at 18:12 is attributed to the Echo Dot 2 because it is the only trace with a spike occurring. We then looked at the individual network usage for each of these devices in this time frame as shown in Figure 3.22. At 18:12, there is no significant network usage.

From Figures 3.21 and 3.22, we speculate that because there is no network usage during this time but a power spike, the Echo Dot was briefly exposed to more light, causing the LEDs to brighten. The power usage before the spike and during the spike exactly match that of Figure 3.15, starting at around 1500 mW, rising to about 2300 mW.

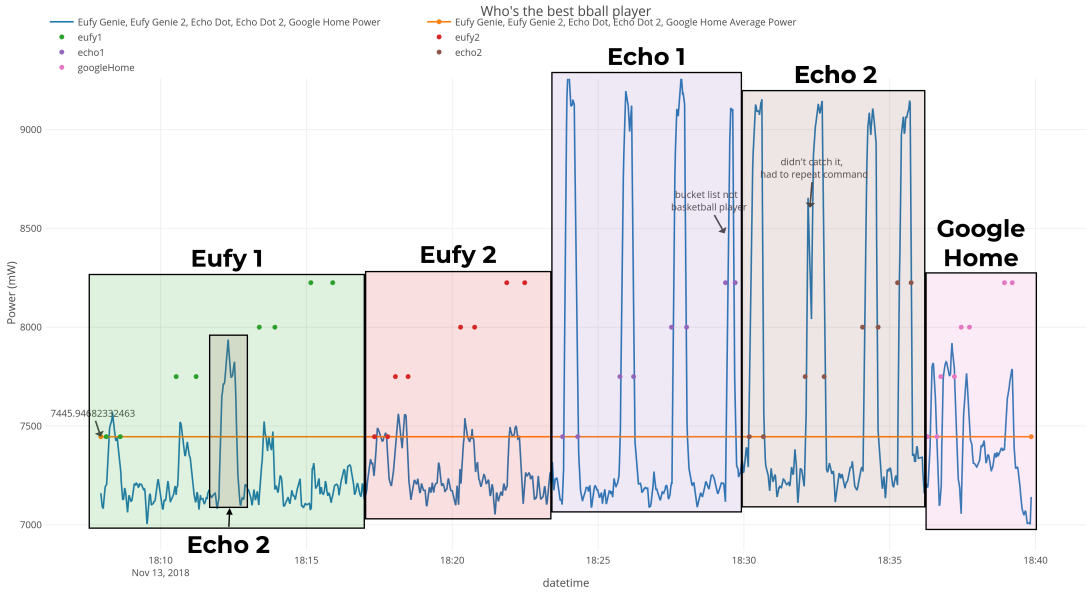


Figure 3.20: 5 Smart Speakers Power Summed Up. Queried each device for the best basketball player.

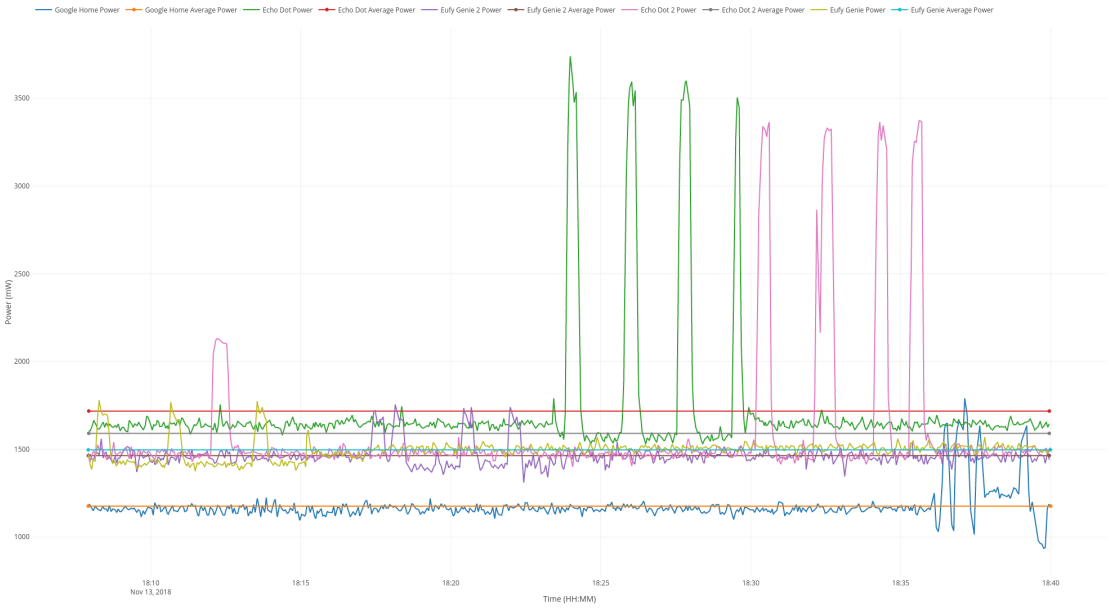


Figure 3.21: 5 Smart Speakers Power Usage over time. Queried each device for the best basketball player.

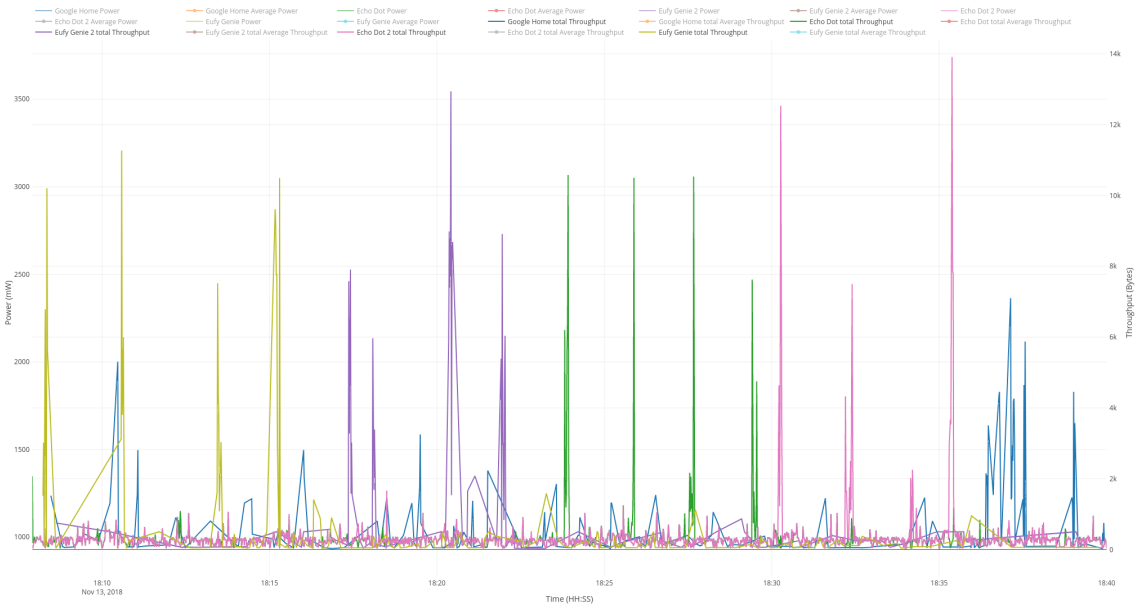


Figure 3.22: 5 Smart Speakers Network throughput over time. Queried each device for the best basketball player.

Figure 3.23 shows the power spike averages for each query type (“what’s the weather” “what’s the news”, “who’s the best basketball player”). The black line at the top of

each bar shows the standard deviation. Averaging peak to peak voltage spikes for each device across all commands shows the Eufy Genie with a 390 mW spike with 36.1 mW standard deviation, the Google Home with a 606.7 mW spike with 110 mW standard deviation, and the Echo Dot with a 2026.7 mW spike with 149.89 mW standard deviation.

Figure 3.23, indicates that it is likely possible to determine a smart speaker from a shared power trace if a power spike occurs within the thresholds shown. From this, we conclude it is possible to determine a device model from visual examination of its power usage.

This is the first step to determine if it may be possible to see what devices are in use from analysis of someone's power line. But in a real house, there are more than just five smart speakers on a power line. The next step is to add noise from high power devices to see if it is still possible to visually determine the device in use from power spikes 3.5.

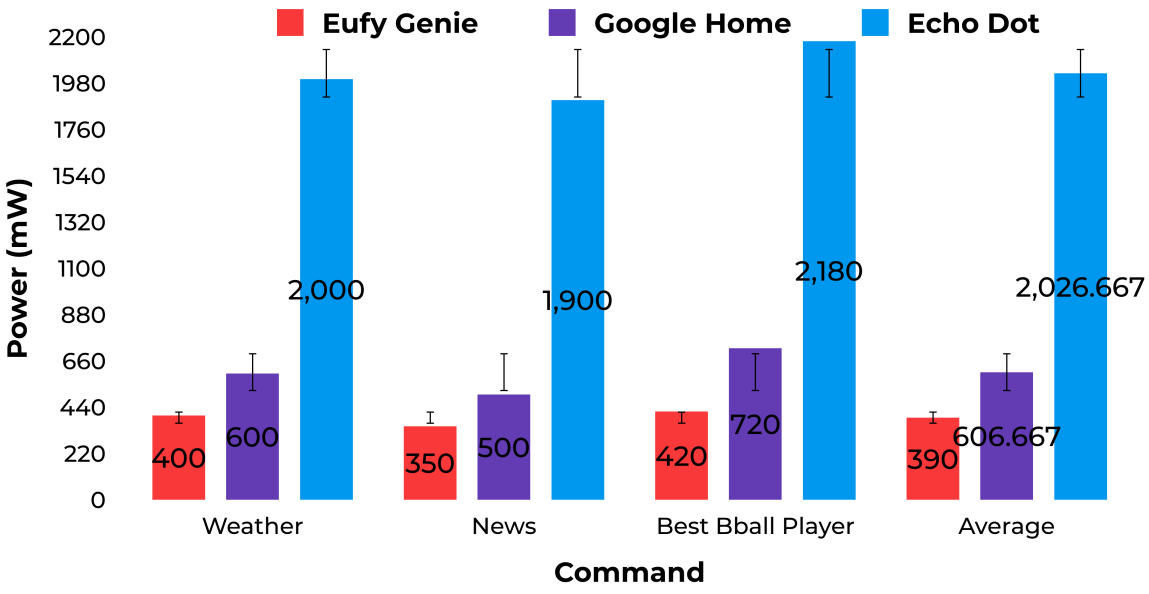


Figure 3.23: Spike summary of for each query.

3.5 Summed Power Graph with Noise

In this section, we introduce noise into the summed smart speaker setup so that we can determine if the power spike from each smart speaker is still discernible within a summed power graph when we add high power devices.

In each subsection, there are three traces. The first trace (blue trace) is the summed power usage of our five smart speakers (2 Echo Dots, 2 Eufy Genies, 1 Google Home) when asked for the best basketball player four times. This is the same trace as shown in Figure 3.20. The second trace (purple trace) is the power usage of a high power device. It maps to various devices as we switch them out. The third trace (red trace) is the sum of both trace 1 and 2.

The X-axis is the time elapsed in seconds rather than a specific time stamp because the smart speaker trace and 'noise' trace are in different time frames. The Y-axis is the power used by each device at that time.

Figure 3.24 shows an example of the traces used separately. This is shown separately at first, but are summed together for the rest of the paper so that the scale of the noise can be understood.

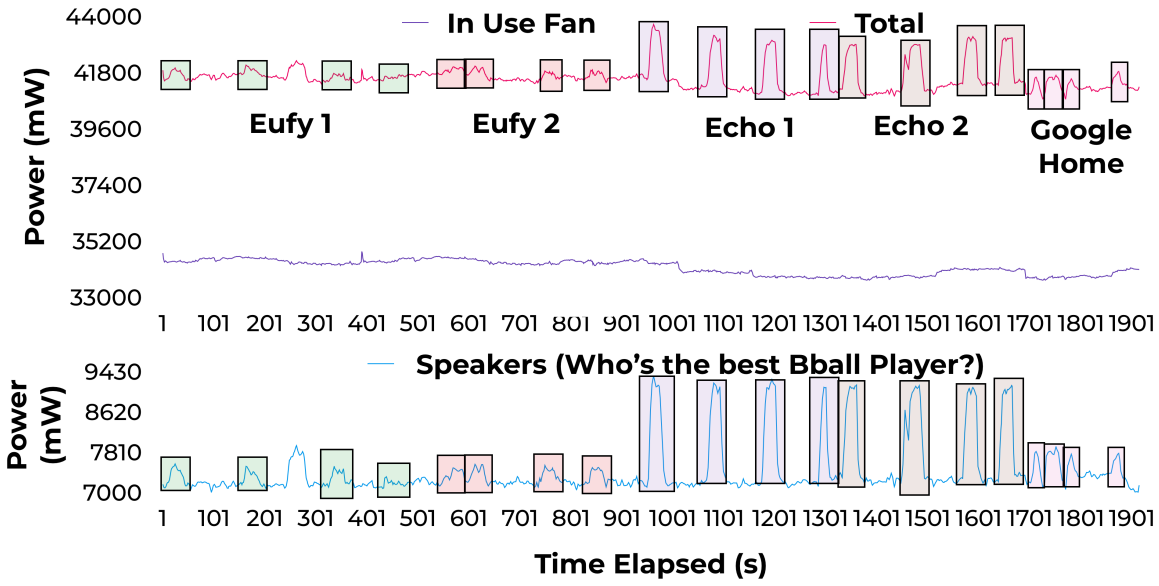


Figure 3.24: Figure 3.20 summed smart speaker power, in use fan noise, and total power shown separately.

Figures 3.24, 3.25, 3.26, 3.27, 3.28, and 3.29show the smart speaker power trace when asked for the best basketball player with noise from a PC (Intel NUC), fan, refrigerator, or microwave. In these graphs, high power devices are in idle or steady power state. Then Figures 3.31, 3.32, and 3.33 show noise from the high power devices while in use, introducing noise that make visual detection of smart speakers more difficult.

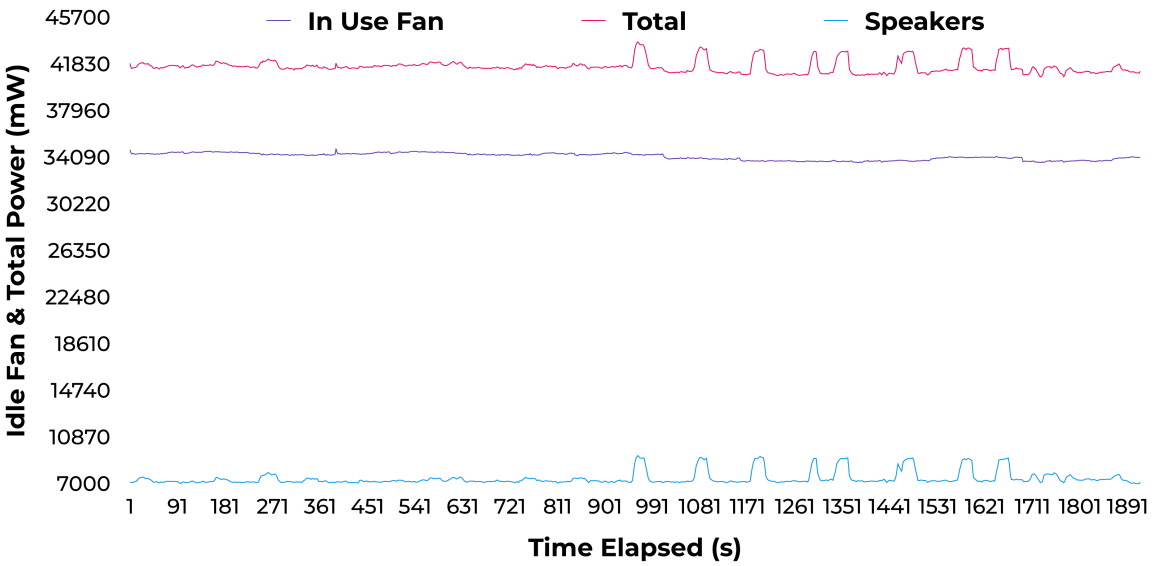


Figure 3.25: Figure 3.20 summed smart speaker power, in use fan noise, and total power.

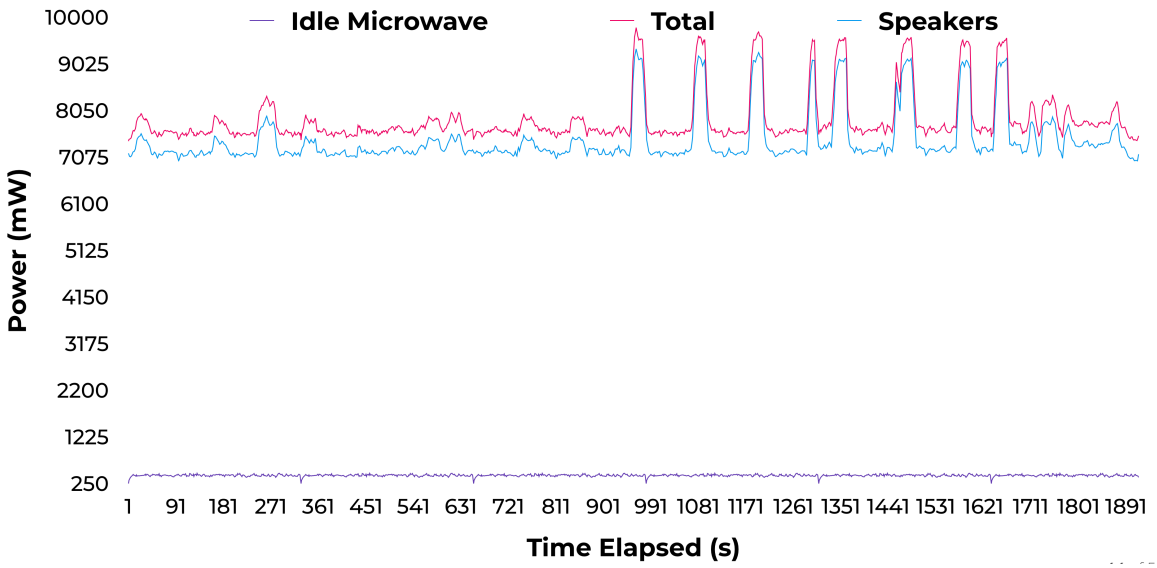


Figure 3.26: Figure 3.20 summed smart speaker power, idle microwave noise, and total power.

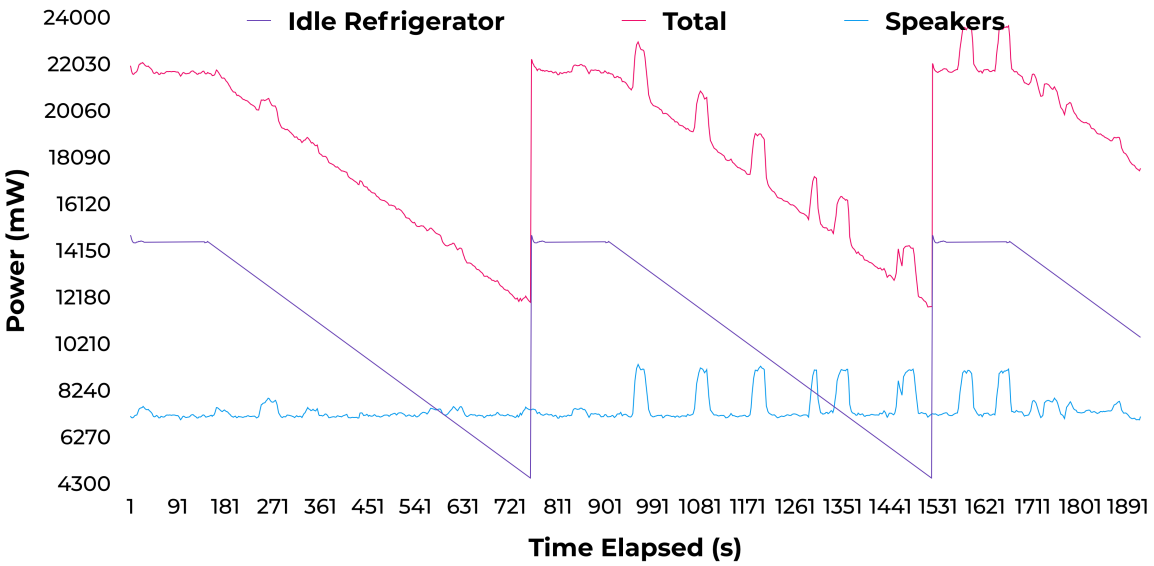


Figure 3.27: Figure 3.20 summed smart speaker power, idle refrigerator noise, and total power.

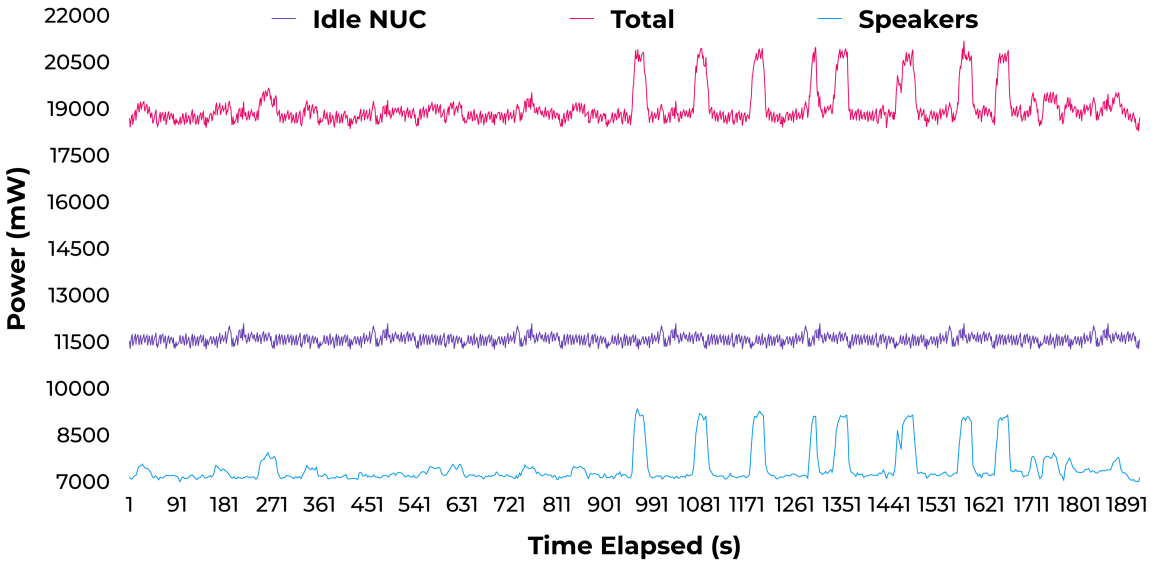


Figure 3.28: Figure 3.20 summed smart speaker power, idle idle Intel NUC noise, and total power.

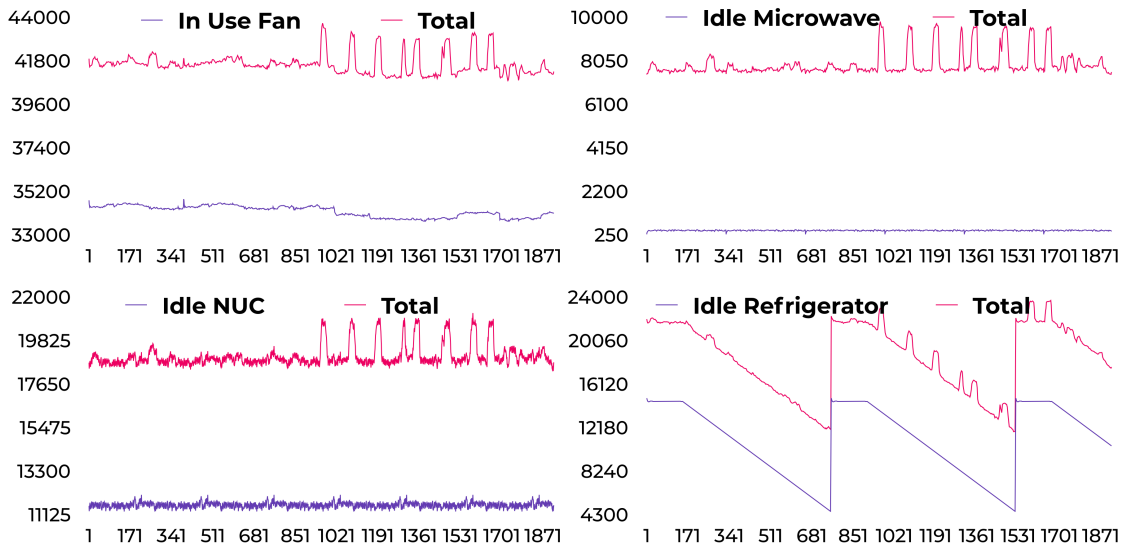


Figure 3.29: 3.25, 3.26, 3.27, and 3.28 figures zoomed in

In Figures 3.25, 3.26, 3.27, and 3.28, the power spikes from the smart speakers can still be seen. The Echo Dot power spikes are still visible in the red trace for each graph. The power spikes for the Google Home and Eufy Genie are less visible but can still be seen. Especially when zoomed in as shown in Figure 3.29.

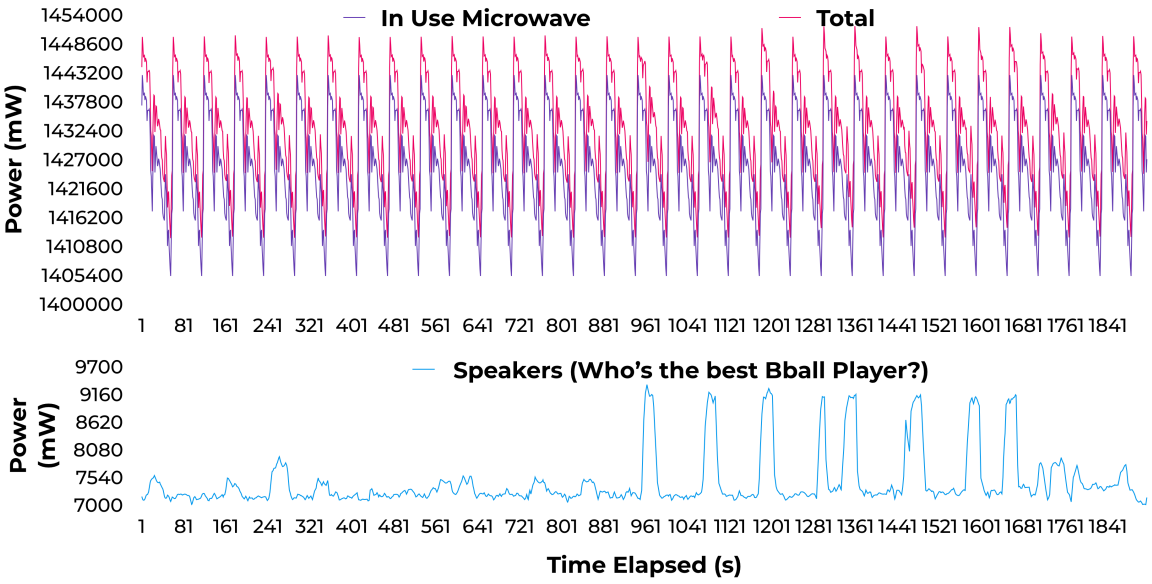


Figure 3.30: In use Microwave with Figure 3.20 summed smart speaker power separately.

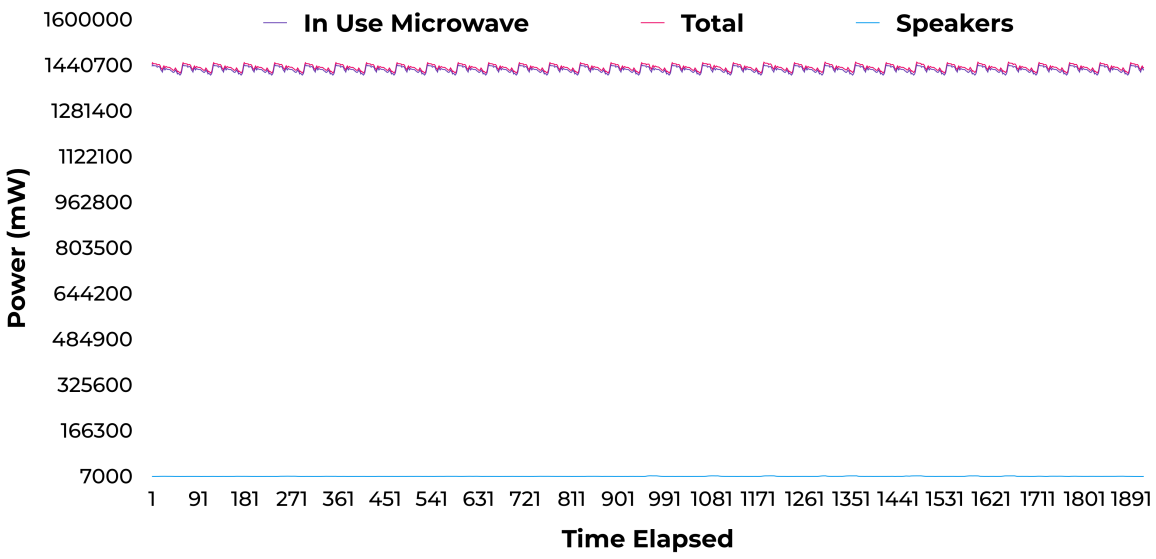


Figure 3.31: In use Microwave with Figure 3.20 summed smart speaker power.

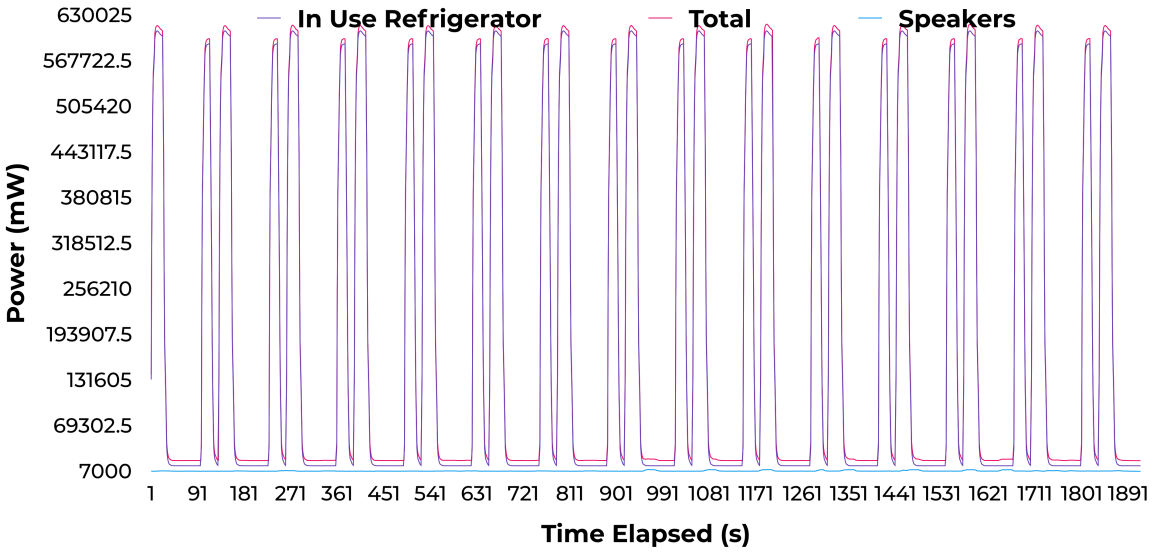


Figure 3.32: Fridge in the middle of cooling with Figure 3.20 summed smart speaker power.

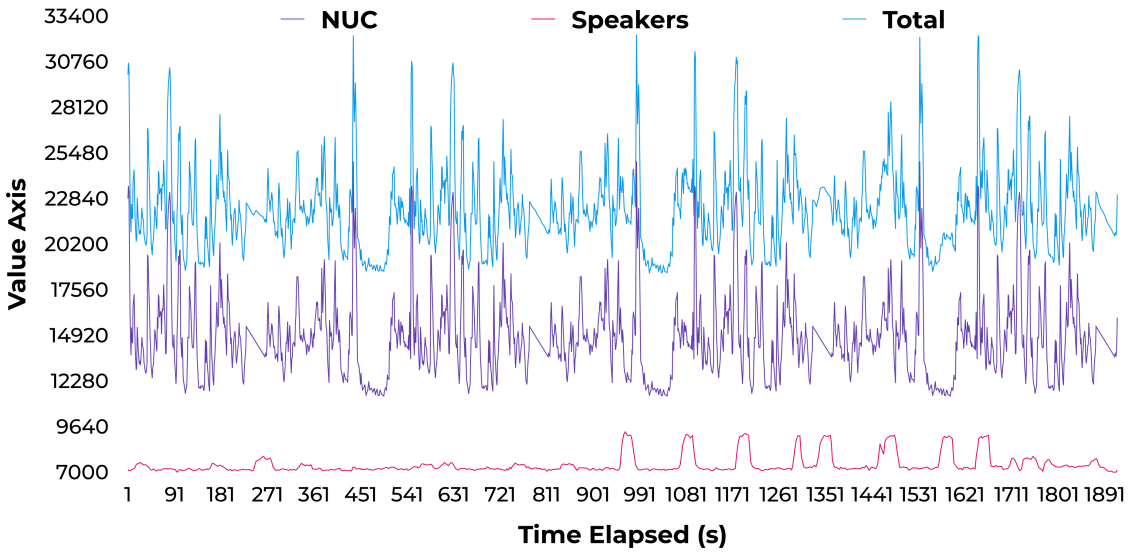


Figure 3.33: PC in use with Figure 3.20 summed smart speaker power.

Figures 3.25 - 3.33 show that it is possible to determine what smart speaker is in use from visual examination of power spikes even in the presence of noise. If the noise within the house is stable, the power spikes are shifted up as shown in Figure 3.25. If the noise within the house has a low magnitude, the smart speaker power spikes visually overpower the noise, as shown in Figure 3.28.

But if the noise within the house is considerable in magnitude such as the microwave or fridge while in use 3.31 3.32, this noise will squash the power spikes from the smart speakers. The spikes can no longer be seen, even when zoomed in, as shown in Figure 3.30. The smart speaker spikes also disappear if the noise of a device is on the same or a larger magnitude than the power spikes, as shown in Figure 3.33. The power spikes can still be seen in the sum graph for this figure, but it is impossible to visually differentiate between a power spike from a speaker and the introduced noise without knowing the original smart speaker graph.

These two cases vary depending on the household. If many battery power devices

such as laptops (replacing the NUC) or phones are used, then it may still be possible to determine the power spikes visually. If high power devices such as the microwave or fridge are not in use often, then the smart speaker spikes may be visually correlated. But if a household has many devices plugged into an outlet for power, then the smart speaker power spikes are likely to disappear from the additive noise of all the devices. Regardless, there are privacy implications in a household's power line. With easy, direct, access to a household's power line [**?**], someone could determine the device in use within a household.

Chapter 4

CONCLUSION

To conclude, this paper documents the creation of a large database of IoT devices' network and power usage, introduces a graphing tool for this database and shows that it is possible to tell what smart speaker is in use based on just the initial spike made during a command.

After one year of logging common household IoT devices' network and power usage, our AWS database contains 172,445,929 entries that take up 184.94 GB of data. This database provides a large dataset for others to use without having to repeat the steps to form a testbed. This database was incredibly useful in Frawley and my work. Other students have already used it for their research projects.

To further streamline IoT research, a visualizer tool was created that graphs power and network usage for any device in the database, making visual analysis quick and easy. The visualizer provides many useful features that Frawley and I used for our research.

Some patterns can be difficult to make out with so much noise coming from within a house for power data. However, in some cases, it is still likely possible to determine what smart speaker is in use. With more research, it is likely that other private information can be inferred from the power line data as the research here has yet to leverage advanced statistical methods.

4.1 Future Work

With an ever-growing database, maintenance is necessary. We want to clean up the database by indexing all columns for quick lookup; as of now, only some columns are indexed. There are also some inconsistencies in the database because the IP addresses of the devices and WeMo names change over time. Unifying a device under one IP address and a WeMo under one name in the database are clear first steps.

The next plan is to improve the realtimeIoTgrapher and add features. The biggest interest is to add automatic annotations such as boxes denoting peak to peak spikes, or trend line plots. Plotly also has a feature to display the grapher on a public domain. Altering the grapher program to do so would make this tool more easily accessible as compared to the locally hosted setup shown in this paper.

We also want to utilize both power and network data in tandem for analysis to see if even more information can be extracted.

With many botnet attacks in the recent past, we would like to hack into these devices with the Mirai Botnet and analyze power usage and network throughput of these devices. This paper shows trends between power use, network traffic, and what an IoT device is doing. It is possible that power use and network traffic could correlate with Mirai's operations. Getting Mirai working in the restricted time turned out to be infeasible, but would be interesting to continue for future work.

Finally, applying advanced statistical methods to the power data would be interesting to pursue. We quickly tried machine learning with a feed-forward neural net with 0 percent accuracy and an LSTM and achieved 30 percent accuracy given five devices (slightly better than random). We had also planned to alter a human activity recognition machine learning model to classify an IoT device from power data over a set time

period. It is likely that machine learning can find patterns that visual examination could not.

Regardless, there are many opened opportunities for future work because of the database. There is already a student who is using the database for their research project. Researchers can quickly use the realtimeIoTGrapher to analyze the database or manually query data.

The world of IoT needs careful analysis and research to make IoT devices more private and secure. Hopefully, my contributions will help enable these studies.

BIBLIOGRAPHY

[1] Cal Poly Github. http://www.github.com/CalPoly.

APPENDICES

Appendix A

HOW TO INSTALL AND RUN REALTIMEIOTGRAPH.PY

This chapter discusses how to install, run, and use the database visualization tool.

A.1 How To Install and Run

To install and run this analysis tool, follow the instructions at the GitHub site: github.com/nealhnguyen/realtimeiotgrapher.

First, pull the code for this tool from git hub and install the dependencies in the requirements file through pip. At this point, the dependencies are set to run this code on python 2.7.

Next, a 'loginCredentials' file must be created containing the host, user, password, and database information. At which point, running 'python realTimeIoTGraph.py' starts the backend and display its current operation onto the terminal.

The link to the local IP address contains the graphing tool. Clicking on that opens it onto the default browser for the computer.

A.2 Usage Directions

Once the software is running, the grapher can be viewed on a web browser as shown in figure A.1. This section describes each numbered element in the image.

Figure A.1: RealTimeIoTGrapher usage image

0. The local IP address presented in the terminal where the grapher is hosted.

1. The interval field and the text box next to it is dark when the software is in live update mode.

2. The text box here relates to the live update feature of the graph. This is a time field, denoting the window of time the graph displays at a time while it updates. The format is HH:MM:SS where HH is the hour, MM is minutes, and SS is seconds.

3. The time range field and the four elements consecutively after it are dark instead of grey when it is in static graph mode. In this state, the graphing tool specifies the network traffic and power information in the time window specified.

4. In these three text boxes, the time window for static graphing is specified. The first box is the start time, the second box is the end time, and the last box is the interval. To specify the time range, fill in any 2 of the three text boxes, and the graphing tool infers the other number. If all three text boxes are filled, then the graphing tool uses the start and end time text box. To further explain, if the start time and interval are specified, then the window is start time $\rightarrow$ start time + interval. If the end time and interval are specified, then the time window is the end time - interval $\rightarrow$ end time.

5. This button causes the graph to update. After updating the time range in step 4, clicking this button updates the graph with the new time range.

6. The "Use Time Range" checkbox toggles the software between live graph mode and static mode. The "Sum Graph" checkbox sums up all power graphs into one. The "Show Only Power" checkbox allows the grapher to avoid a SQL query for the network throughput, speeding up the process in case of pure power analysis.

7. This selection field specifies all the devices the software creates graphs. One, or many devices can be specified at a time. If too many devices are specified at a time, the software may run slow.

8. The traces shown can be selected or deflected to toggle its visibility.

9. This line graph displays the number of bytes sent through the network by a device going in, out, and in total. It also shows the power usage over time. Also, it shows the average value for a graph in the interval of the graph

10. This bar graph displays the spread of network traffic by protocol for each device. It displays the outgoing packets as orange and incoming packets as blue. It displays an individual bar graph for each device and protocol. Similarly, the traces can be clicked on to hide its visibility.