

LOGGING, VISUALIZATION, AND ANALYSIS OF NETWORK AND 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

© 2018
Neal Nguyen
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Logging, Visualization, and Analysis of Network and 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 and Power Data of IoT Devices

Neal Nguyen

It is estimated that there are 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 for supporting me throughout life. I wouldn't have made it here without your time and effort.
- 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	viii
LIST OF FIGURES	ix
LIST OF CODE LISTINGS	x
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	2
1.1.3 Logging and Analysis of Internet of Things (IoT) Device Network Traffic and Power Consumption	3
1.2 Scope	3
1.3 Thesis Organization	4
2 Method	6
2.1 IoT Testbed Overview	6
2.2 Physical Layout and Setup	8
2.2.1 Smart Power Plugs	8
2.2.2 Wireless Access Point	9
2.2.3 Devices	9
2.2.4 Wiring and Configuration	13
2.3 Software	14
2.3.1 Wireless WAP Code	14
2.3.2 sniff.py	15
2.3.3 Database	17
2.3.4 Device Inventory	23
2.3.5 IP History	23
2.3.6 Usage Flow	24
2.3.7 realtimeIoTGrapher.py	28
3 Results and Discussion	36
3.1 Swapping the Power Switch	36

3.2	Whole Database	38
3.3	General Analysis	39
3.3.1	Google Home Mini General Analysis	39
3.3.2	General Analysis Discussion	46
3.4	Power Analysis on Smart Speakers	46
3.4.1	Echo Dot Brightness Sensor	47
3.4.2	Baseline Speaker Power	48
3.4.3	Smart Speaker Power Spikes When Asked for Weather	48
3.5	Summed Power Graph	49
3.6	Summed Power Graph with Noise	55
3.7	Smart Speaker Comparison	61
4	Conclusion	65
4.1	Future Work	66
	BIBLIOGRAPHY	68
A	How to install and run RealTimeIoTGraph.py	72
A.1	How To Install and Run	72
A.2	Usage Directions	72

LIST OF TABLES

Table	Page
2.1 IoT Devices Being Monitored	10
2.2 Columns in Network Traffic Table	18
2.3 Columns in Power Table	20
2.4 Device inventory excerpt. Password column not shown.	23
2.5 History of IP addresses for each device	24
2.6 Event Log Excerpt	25

LIST OF FIGURES

Figure		Page
2.1	IoT testbed.	7
2.2	Smart Speakers Connected to WeMo Insight Switches	7
2.3	IoT Devices Under Examination	8
2.4	Navicat IP query with rollup.	19
2.5	Power query from database with Navicat.	21
2.6	Network query from database with Navicat.	22
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	29
2.8	Resulting graph of dataset shown in Figure 2.7	29
2.9	Screen of saved plotly graphs.	30
2.10	Screen to edit saved Plotly graphs containing an edit tool bar, the graph data, and the graph.	31
2.11	Summed power traces without interpolation.	33
2.12	Summed power traces with interpolation.	33
3.1	Connected the Echo Dot 1 to WeMo for ‘Eufy Genie 1’ and vice versa. The traces swap before and after switching.	37
3.2	Connected the Eufy Genie 1 to WeMo for ‘Echo Dot 1’ and vice versa. The traces swap before and after switching.	37
3.3	Connected the Google Home to WeMo for ‘Eufy Genie 1’ and vice versa. The traces swap before and after switching.	38
3.4	Internet and Transport Layer Protocols in Database	39
3.5	Idle Traffic of Google Home Over 1 Hour Period	40
3.6	Home Mini Response to News and Weather	41
3.7	Chromecast First Time Boot Network Traffic and Power Consumption	42
3.8	Chromecast Video Streaming	42
3.9	Chromecast Idle Traffic	43
3.10	Fire TV Stick First Time Boot Network Traffic and Power Consumption	44
3.11	Fire TV Stick Video Streaming	45

3.12	Roku Express Video Streaming	46
3.13	Echo Dot Response to Lights	47
3.14	Baseline smart speaker power usage.	48
3.15	Smart speakers' power usage when asked 'what's the weather' four times.	49
3.16	5 Smart Speakers Power Summed Up. Toggled the mute button and queried each device for the weather.	50
3.17	5 Smart Speakers Power Summed Up. Queried each device for the news.	51
3.18	5 Smart Speakers Power Summed Up. Queried each device for the best basketball player.	52
3.19	5 Smart Speakers Power Usage over time. Queried each device for the best basketball player.	53
3.20	5 Smart Speakers Network throughput over time. Queried each device for the best basketball player.	53
3.21	Spike summary of for each query.	54
3.22	Idle Fan with figure 3.18 trace shown separately.	56
3.23	Idle Fan with figure 3.18 trace.	56
3.24	Idle microwave with figure 3.18 trace.	57
3.25	Idle refrigerator with figure 3.18 trace.	57
3.26	Idle PC (Intel NUC) with figure 3.18 trace.	58
3.27	3.23, 3.24, 3.25, and 3.26 figures zoomed in	58
3.28	In use Microwave with figure 3.18 trace separately.	59
3.29	In use Microwave with figure 3.18 trace.	59
3.30	Fridge in the middle of cooling with figure 3.18 trace.	60
3.31	PC in use with figure 3.18 trace.	60
3.32	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 3.19 with Echo Dot 2 and Eufy Genie removed.	62
3.33	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 3.32	63
3.34	Average power and network usage at each second for the Echo Dot, Eufy Genie, and Google Home in the time frame of figures 3.32 and 3.33	63

A.1	RealTimeIoTGrapher usage image	73
-----	--------------------------------	----

LIST OF CODE LISTINGS

2.1	Open and Read from a Socket	15
2.2	Rescan if all WeMos not found.	16
2.3	Open and Read from a Socket	26
2.4	Efficient SQL query to obtain total Network throughput at each second.	32
2.5	IP lookup table in real time IoT grapher.	35

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 [1]. 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 [2] 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.

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 [3]. This is not expected behavior and highlights privacy concerns.

In another security flaw, one of the largest IoT cybersecurity attacks, Mirai, was able to use weak login credentials to take control of 2.5 million IoT devices to perform a denial of service attack [4]. This is one many events that introduces security concerns. This paper uses these flaws as a focus when collecting and analyzing network and power usage.

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* [5], 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.

1.1.2 ProfilIoT

The paper, *ProfilIoT: A Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis* [6] 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[7].

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.

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[8], 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[9] 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

The paper from subsection 1.1.1 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. Our work expands on this concept 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.

Like the paper from subsection 1.1.2, this paper also focuses on classification of devices from data. Instead of using machine learning techniques on network data, 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 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 unique contribution of this work is its analysis of IoT device power usage and the introduction of a custom data visualization tool that attaches to the database. This paper focuses on a select few devices, analyzing their startup, idle, and in use network and power usage over time. I compare the smart speakers' network and power usage and show that it is likely possible to identify a smart speaker through analysis of the powerline over time.

1.3 Thesis Organization

Chapter 2 discusses my 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)[10] MySQL[11] 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. Finally, 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 devices we used, the setup of each device, and the physical layout of the IoT test bed. Section 2.3 covers the software part of the IoT test bed, including the usage of this test bed and what was done to obtain results for analysis.

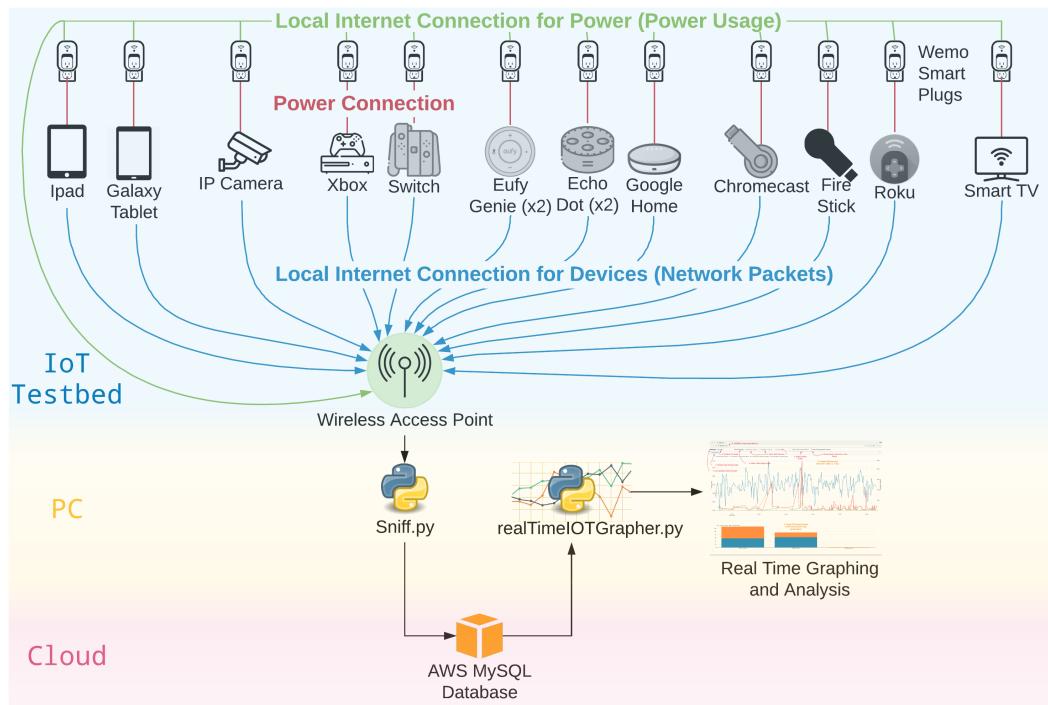


Figure 2.1: IoT testbed.



Figure 2.2: Smart Speakers Connected to WeMo Insight Switches

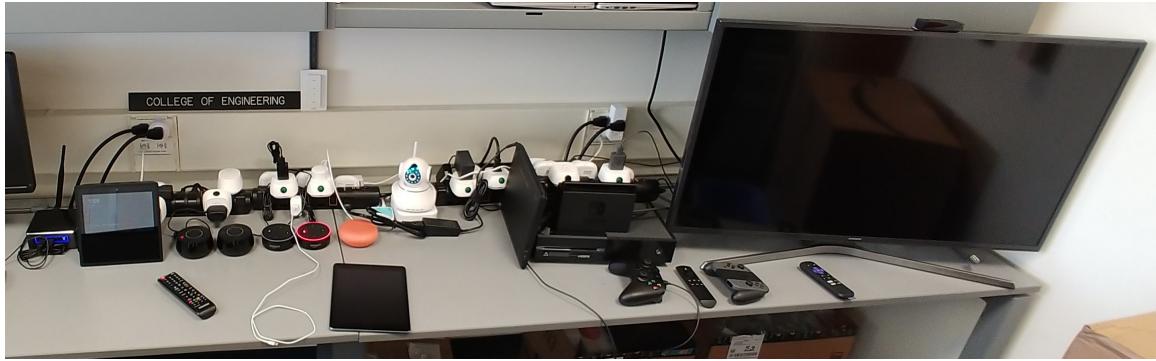


Figure 2.3: IoT Devices Under Examination

2.2 Physical Layout and Setup

This section covers the physical setup and layout of the IoT devices, and how each device is set up. 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 [8]; 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 2.3.7

2.2.2 Wireless Access Point

The wireless access point is an Intel NUC [12]. We loaded Ubuntu[13] on it because it is the most common Linux OS [14] 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 [15] when connected to with a Chromebook [16] (for reference, the FCC defines 25 Mb/s as the minimum rate to be considered broadband [17]). This rate limitation caused many of the Belkin WeMo smart switches [18] 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

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[19]. 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[20]. 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.

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 [21].

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 [22].

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

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 [23] and the Xbox One, which sold 30 million units [24] as of January 2018.

Tablets

Tablets are mobile devices that have a touch screen, performing computer tasks, 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 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 power usage of these devices because they ran on battery power. Tracking a device's charging method as opposed to its actual power usage was out of the scope of this project.

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 [25]. 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 [26].

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. But we 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 decided to leave this out of our research.

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 [8] previously covered this section, refer to his paper for more details. Also, the NUC was previously acquired and setup by Ryan Frawley and other students and is not covered in this section.

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 [27], 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 be 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 its 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 control the IoT test bed. It discusses the scripts for logging power and network traffic, the database information, and the python script for viewing the database.

2.3.1 Wireless WAP Code

To turn the NUC into a wireless access point, we used a `create_ap` script [28]. 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 [29]. 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 [30] library. If the database connection is lost, the script automatically reconnects.

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.

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 [31] 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 would sometimes disconnect, and the script would 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 below 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
10             again{}...".format(try_num))
11          switches = pywemo.discover_devices()
12          try_num += 1
13          print("Discovered {} switches".format(len(
14              switches)))
15
16      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 2.3.7, Navicat [32], and MySQL's command line tool [33].

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. It is so large because it consists of all network traffic generated over 12 months. 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 massive size of this table. The data table's rows represent a single packet with the columns shown in the table below 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. We ignored this issue until a query for a particular IP stopped working. At which point we would invoke a command on the device whose IP changed to flood the database with packets from that device, query the database in the time frame of the command, and obtain the new IP. We logged all devices and their current IP in a Google Sheets [34] file covered in subsection 2.3.5.

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

```

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.

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 05:50:22	216.5.2.17.202	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	216.5.2.17.202	lax17s05-in-f202.1e100.net	TCP	outgoing	45465	443	52
2018-06-05 05:49:59	216.5.2.17.202	lax17s05-in-f202.1e100.net	192.168.12.48	N/A	TCP	incoming	443	45465	52
2018-06-05 05:49:59	216.5.2.17.202	lax17s05-in-f202.1e100.net	192.168.12.48	N/A	TCP	incoming	443	45465	107
2018-06-05 05:49:59	ec2-34-228-67-20.compute-1.amazonaws.com	192.168.12.101	34.228.67.20	ec2-34-228-67-20.compute-1.amazonaws.com	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	192.168.12.101	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	17.2.217.11.78	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	17.2.217.11.78	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, which we did after a few weeks, resetting all devices and causing a temporary void in power data. 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 would 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 subsection 2.3.7.

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 below 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.

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 below in Listing 2.3.

```

1 function onEdit() {
2     var s = SpreadsheetApp.getActiveSpreadsheet().
3         getSheetByName("Event Log");
4     var r = s.getActiveCell();
5     var c = r.getColumn();
6
7     if( c == 4 || c == 5) { // checks the column
8         var dateCell = r.offset(0, -3);
9
10    if( dateCell.getValue() === '' ) { // is empty?
11        var startTimeCell = r.offset(0, -2);
12
13        // fill in the start time and date
14        var date = new Date();
15        dateCell.setValue(date);
16        startTimeCell.setValue(date.toLocaleTimeString());
17    }
18
19    if( c == 6 ) { // checks that the description is being
20        entered
21        // if so fill in the end time
22        var endTimeCell = r.offset(0, -3);
23
24        if( endTimeCell.getValue() === '' ) {
25            endTimeCell.setValue(new Date().toLocaleTimeString());
26        }
27    }
}

```

Listing 2.3: Open and Read from a Socket

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.

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 would 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.

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.

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.

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 [35] 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.

2.3.7 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 would also lag the browser.

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 2.3.7.

This Python script leverages the Plotly [36] 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.

	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

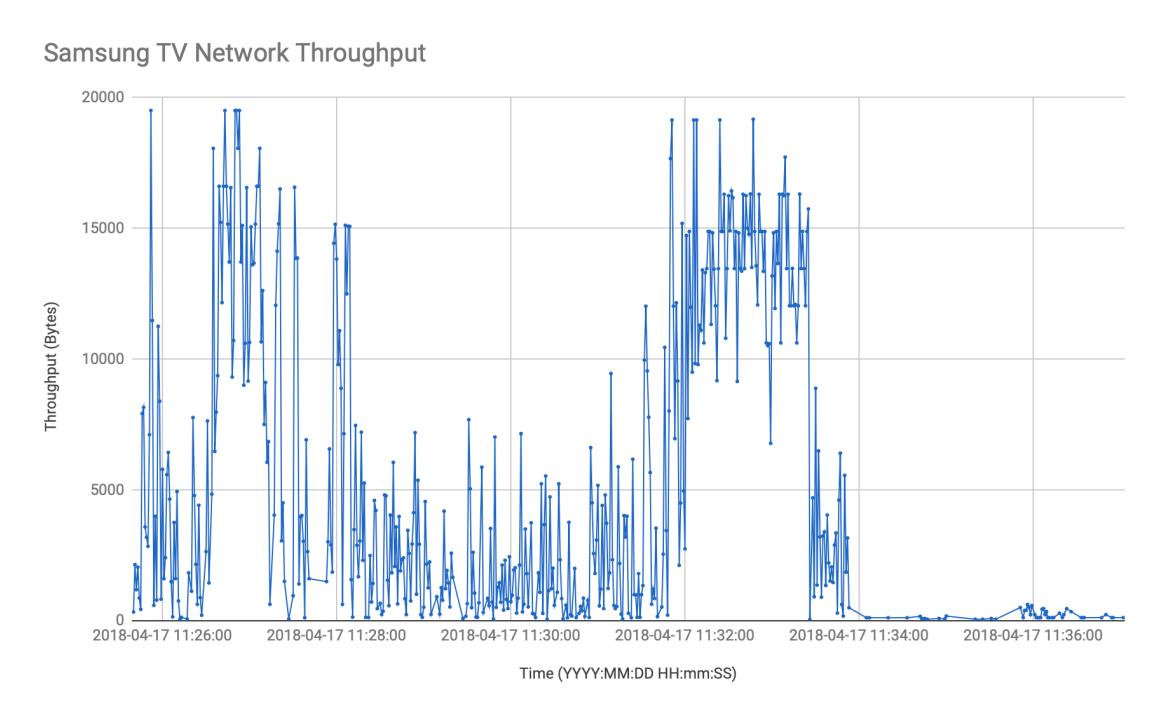


Figure 2.8: Resulting graph of dataset shown in Figure 2.7

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.

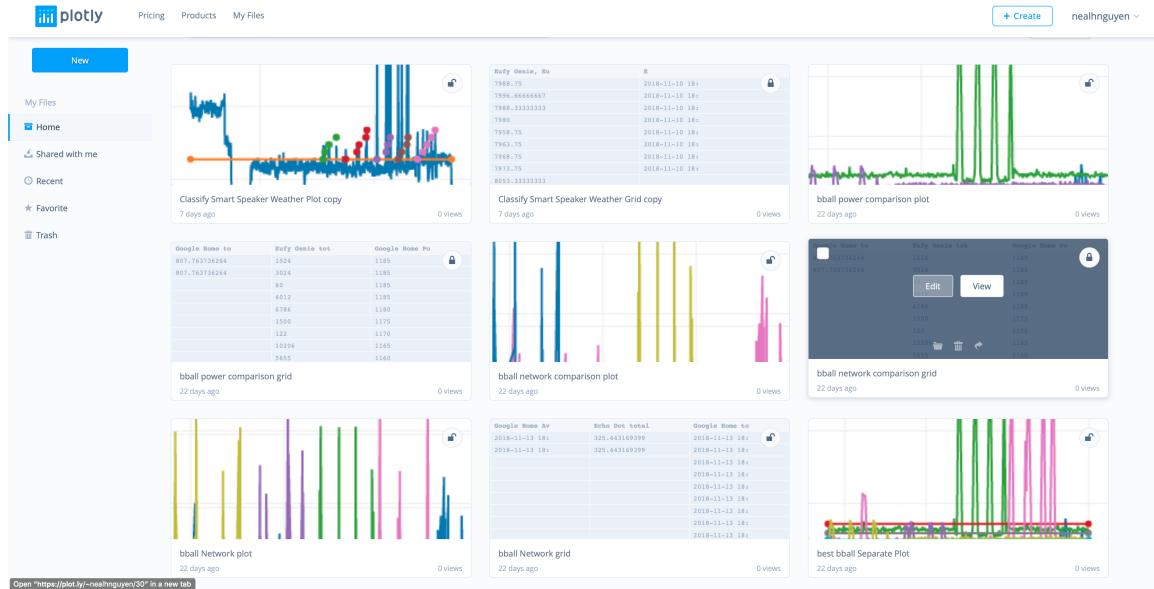


Figure 2.9: Screen of saved plotly graphs.

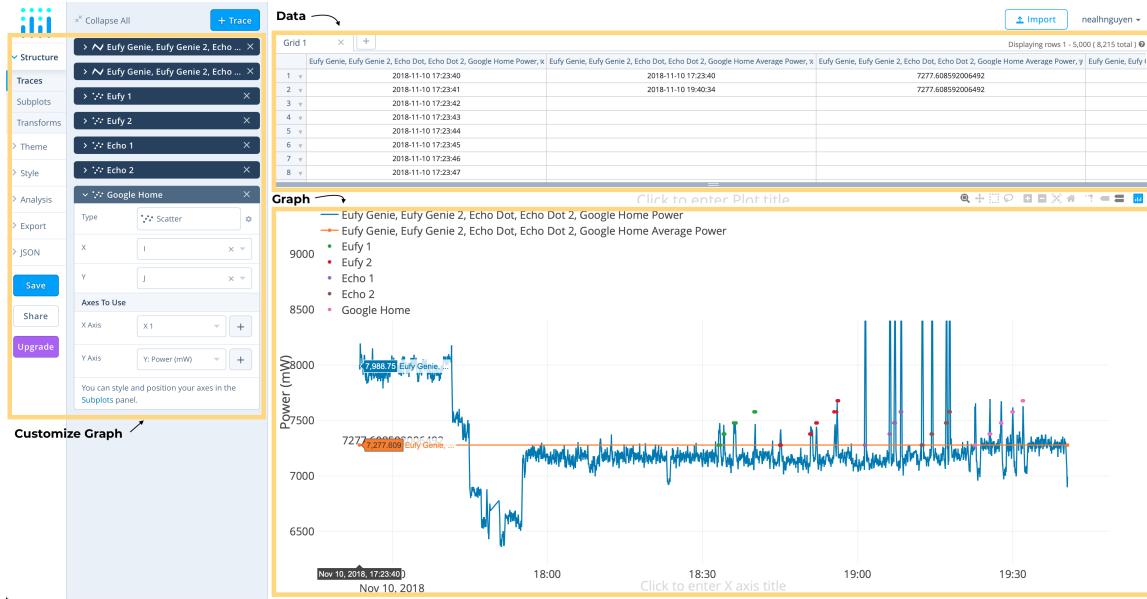


Figure 2.10: Screen to edit saved Plotly graphs containing an edit tool bar, the graph data, and the graph.

```

1  def throughput_query_in_range(db_connection, device,
2      start_time, end_time):
3
4      sql_query = """
5          SELECT
6              time,
7                  CASE WHEN type IS NULL THEN 'total' ELSE type
8                      END type,
9                  SUM(size) AS total_throughput
10             FROM ip_log.ip
11             WHERE
12                 time BETWEEN '%s' AND '%s' AND
13                     (source = '%s' OR destination = '%s')
14             GROUP BY time, type WITH ROLLUP;
15
16 """ % (start_time, end_time, ip[device], ip[device])
17
18     dataframe = pd.read_sql_query(sql_query, db_connection)
19
20     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 provides nested queries and causes the query to sum up the data by time for each type of packet. 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 [37]. Pandas is the industry standard library 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 and for parsing the data.

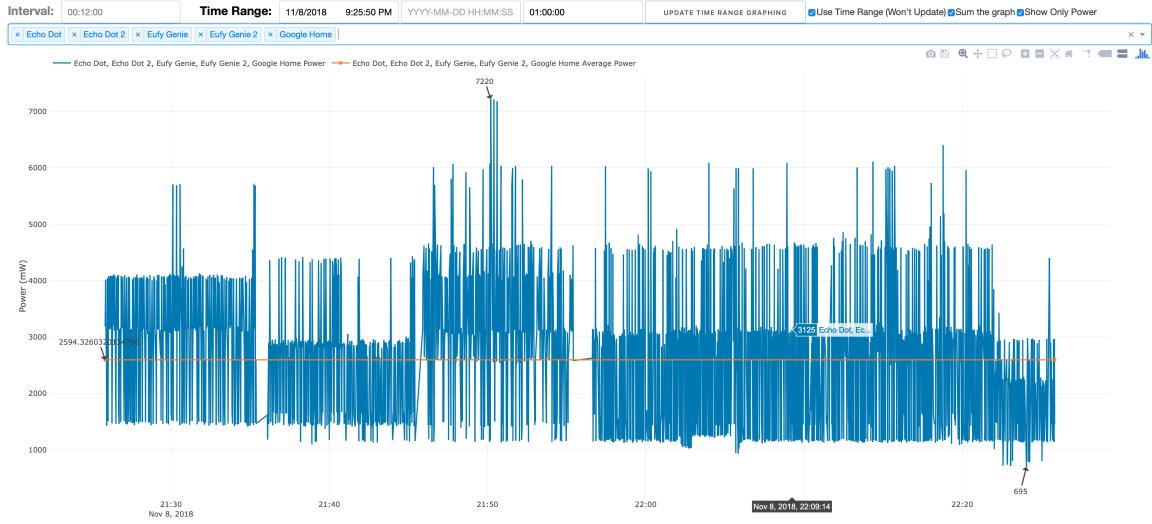


Figure 2.11: Summed power traces without interpolation.

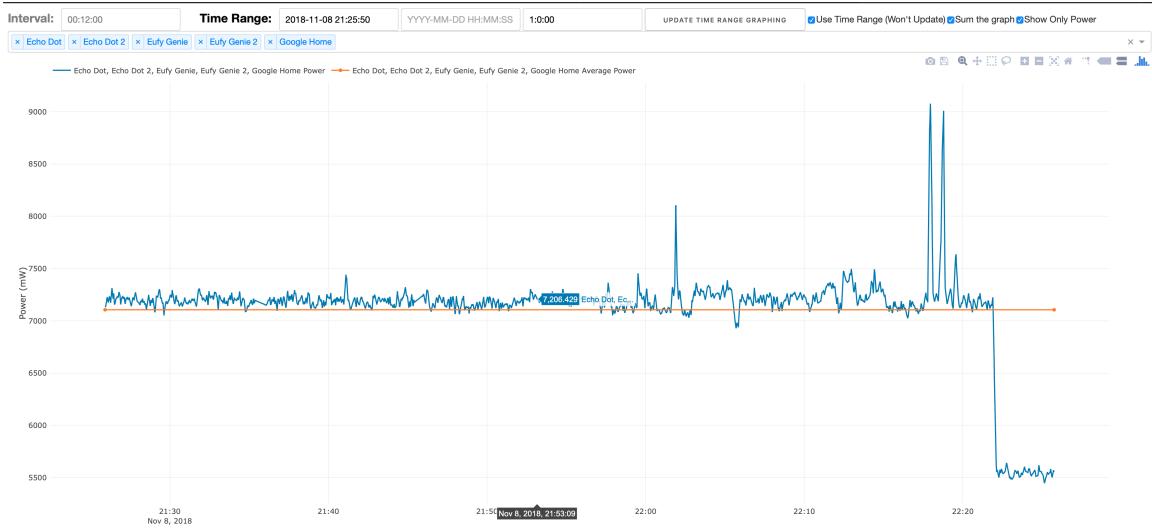


Figure 2.12: Summed power traces with interpolation.

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.

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 would be summed up into one power meter.

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 below.

```

1      #region IP Addresses of all devices we are tracking
2
3      ip = {
4          'Chromecast':           '192.168.12.77',
5          'Echo Dot 2':          '10.42.0.132',
6          'Echo Dot':            '10.42.0.150',
7          'Eufy Genie':          '10.42.0.223',
8          'Eufy Genie 2':        '10.42.0.172',
9          'Fire Stick':          '192.168.12.113',
10         'Google Home':        '10.42.0.236',
11         'IP Camera':          '192.168.12.58',
12         'Nintendo Switch':    '192.168.12.160',
13         'Roku':                '192.168.12.68',
14         'Samsung Hub':         '192.168.12.100',
15         'Samsung TV':          '192.168.12.191',
16         'Smart Light':         '192.168.12.27',
17         'Xbox':                 '192.168.12.251',
18         'Echo Show':           '192.168.12.122',
19         'Appliance':            '192.168.12.122',
20         '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 show that they are accurate for this research’s purposes. Section 3.2 covers the percentage of each protocol in the database for an overall view of home IoT devices. Section 3.3 covers smart speakers’ and streaming devices’ network and power use while idle, in use, and during first boot. Sections 3.2 and 3.3 serve as precursory analysis to fingerprint devices and give example usages of the database and visualizer tool.

After general analysis, section 3.4 shows how the presence of a user in their house and a smart device can be determined from power traces. This starts the focus of the research to use a shared power line (simulation for a home power line) to determine what smart speaker is in use visually. Sections 3.5 and 3.6 cover summed power graphs of the smart speakers with and without noise from high power household devices. Finally, section 3.7 ends with a comparison between smart speakers for further general analysis.

Within each section, after presenting the results, the paper discusses the implications of the data. The results and discussion presented are visual and informal, but strongly support the claims they make.

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 3.1, 3.2, and 3.3. After swapping the switches, each device would take some time to start up, the colored box 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 accurate enough power switches for visual analysis.

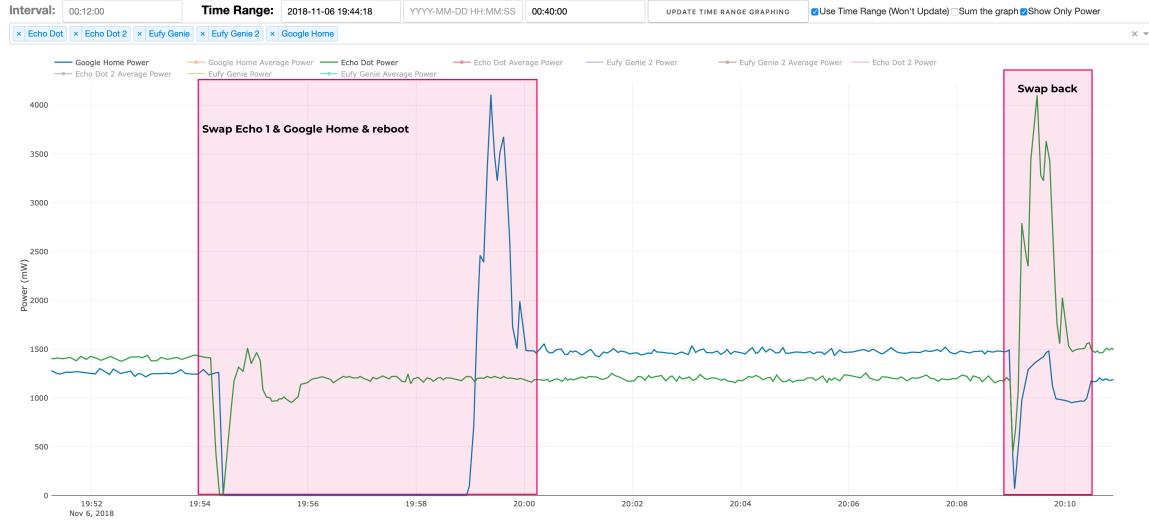


Figure 3.1: Connected the Echo Dot 1 to WeMo for ‘Eufy Genie 1’ and vice versa. The traces swap before and after switching.

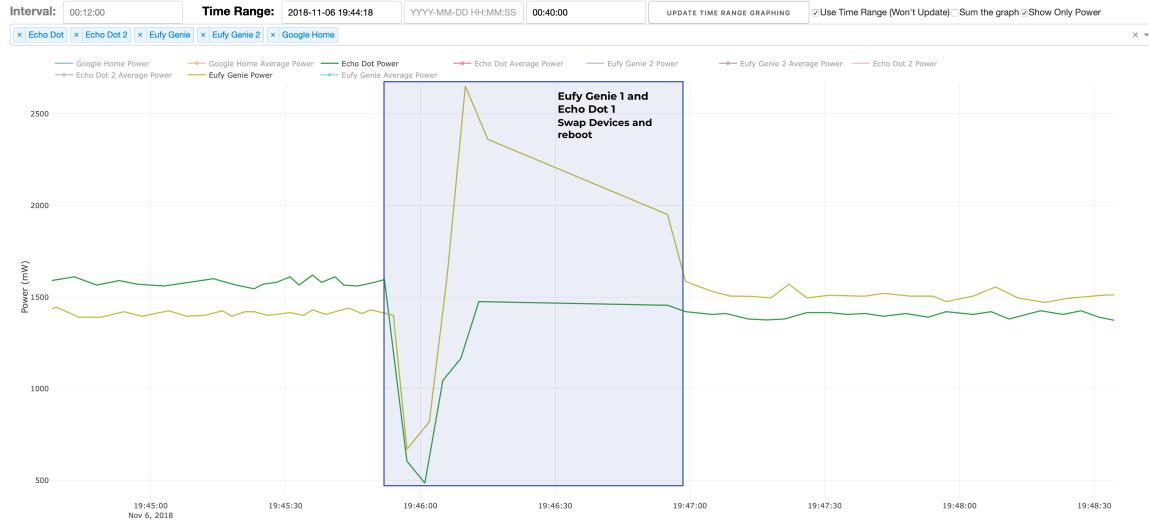


Figure 3.2: Connected the Eufy Genie 1 to WeMo for ‘Echo Dot 1’ and vice versa. The traces swap before and after switching.

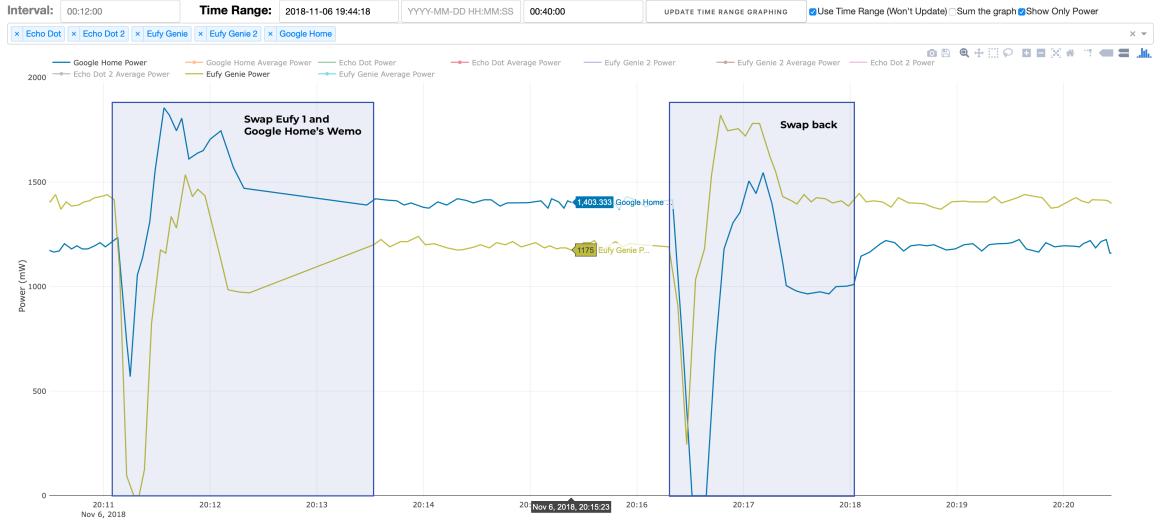


Figure 3.3: Connected the Google Home to WeMo for ‘Eufy Genie 1’ and vice versa. The traces swap before and after switching.

3.2 Whole Database

When examining the whole database, most of the traffic is TCP packets. The second most significant protocol is UDP. This figure is an example of some information the database provides.

It shows the percentage of certain protocols that would pass through a typical home, adding to the fingerprint of IoT devices. Refer to Frawley’s paper for more information on trends in the whole database1.1.3.

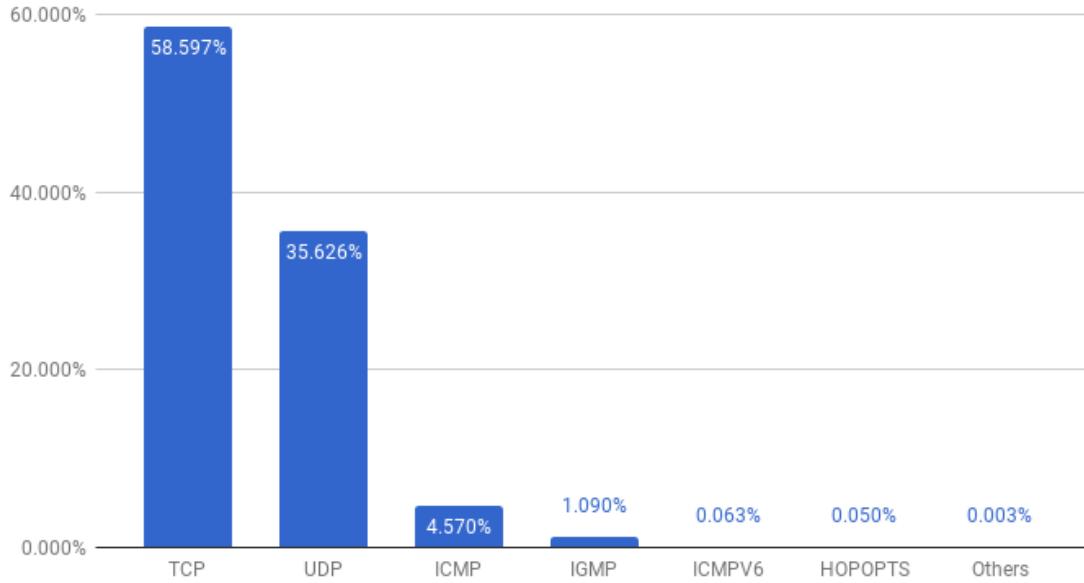


Figure 3.4: Internet and Transport Layer Protocols in Database

3.3 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 what the device is doing and its network/power usage, serving as data to fingerprint each smart speaker.

Frawley's paper contains a more in-depth analysis of this section. His paper includes GeoIP [9] information from Cal Poly's ITS servers [38] that highlight 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.

3.3.1 Google Home Mini General Analysis

The Google Home Mini's idle traffic broadcasts SSDP (Simple Service Discovery Protocol) packets once every minute. The SSDP packets are a discovery request packet

for every IoT device on the network that supports UPnP (Universal Plug and Play). At which point, devices such as the Echo Dot, Samsung TV, and the Chromebook respond with information about themselves in a .xml file. This XML file contains details regarding the device operating system and more. Google also sends encrypted TCP packets to Google every 10 minutes. Figure 3.5 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.

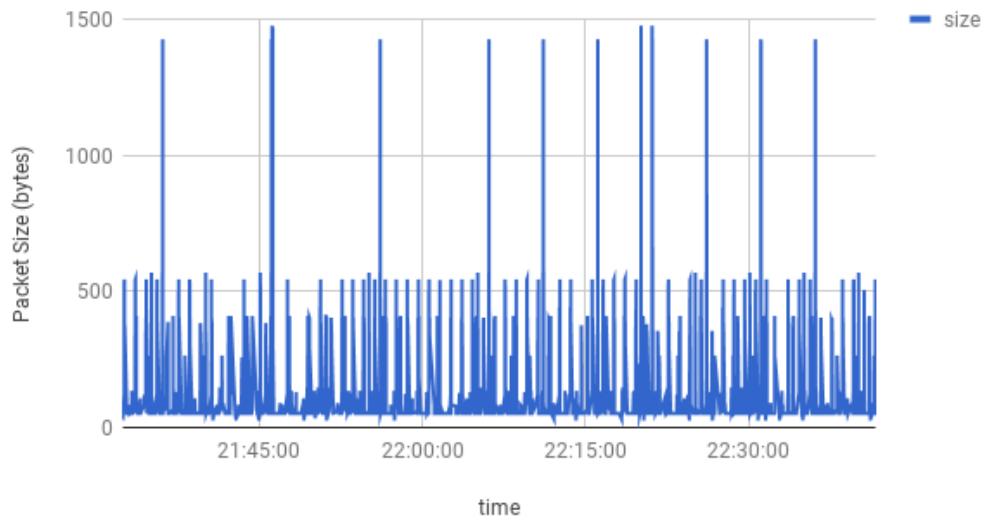


Figure 3.5: Idle Traffic of Google Home Over 1 Hour Period

Figure 3.6 shows a graph of the Google Home Mini's network and power usage under 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.

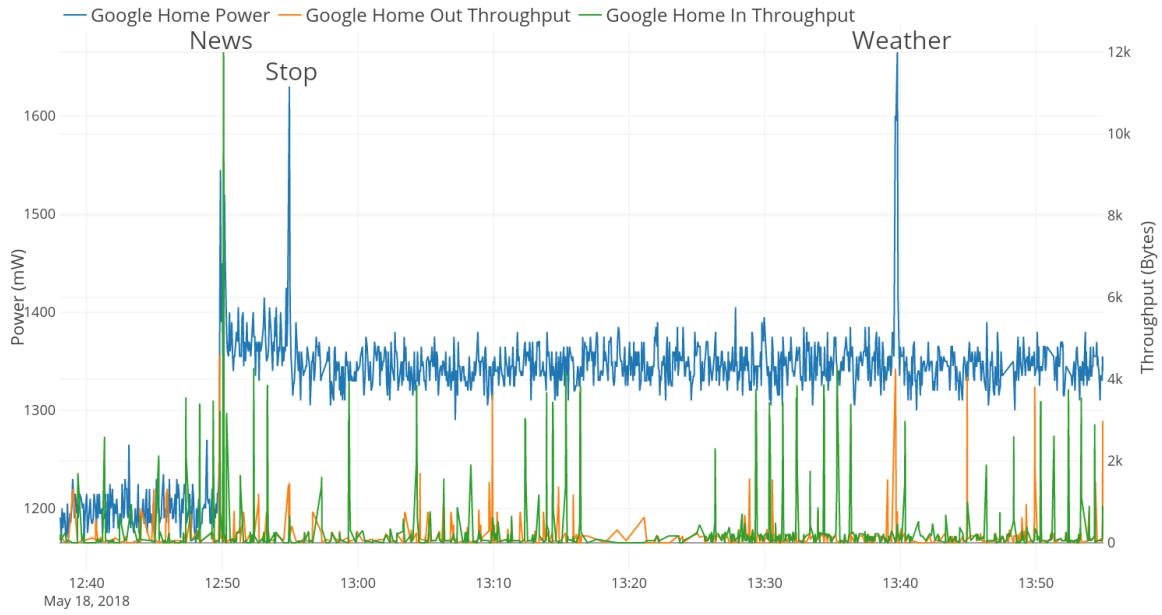


Figure 3.6: Home Mini Response to News and Weather

Google Chrome Cast

Figure 3.7 shows the Chrome Cast startup graph. On startup, the Chrome Cast shows an intro tutorial video and downloads a firmware update. After rebooting twice, it is ready to go.

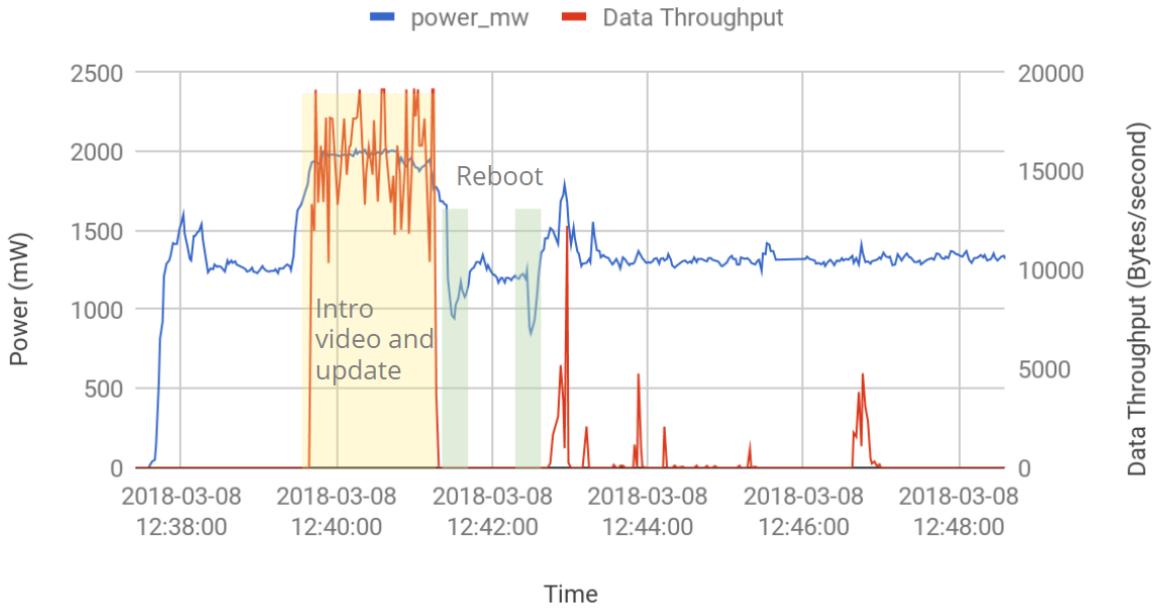


Figure 3.7: Chromecast First Time Boot Network Traffic and Power Consumption

Figure 3.8 shows the Chrome Cast streaming graph. In this time frame, the chromecast streams video in the marked box.

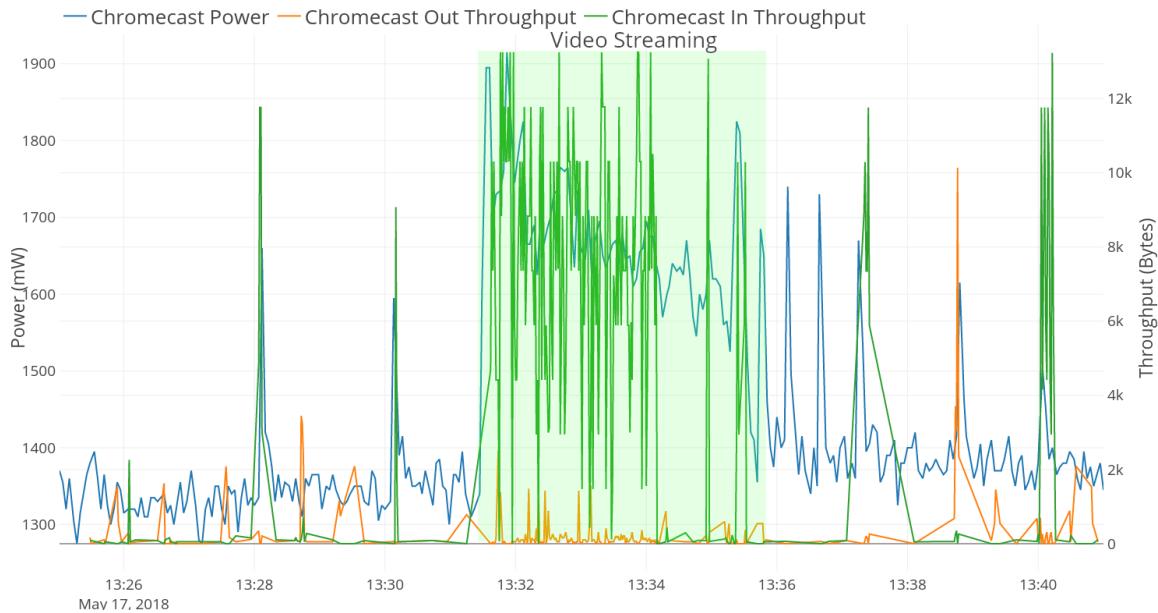


Figure 3.8: Chromecast Video Streaming

Additionally, the Chromecast idle graph is shown in figure 3.9. In this time frame, there are consistent spikes to the chrome cast every 2 minutes. During these spikes, the chrome cast is showing a new background that it downloads from Google Servers.

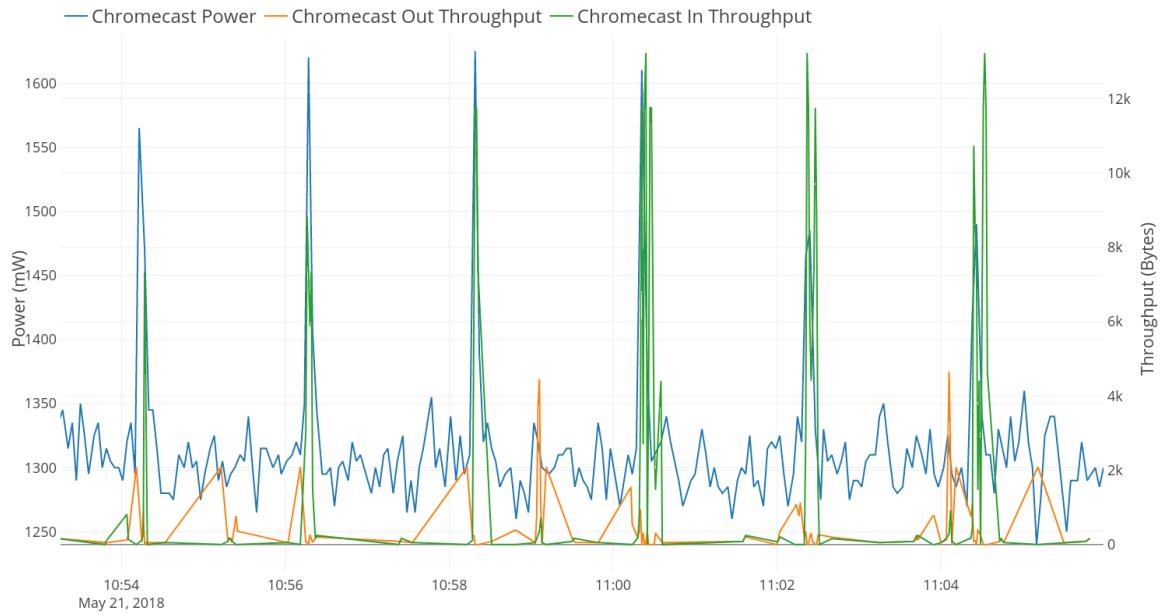


Figure 3.9: Chromecast Idle Traffic

Amazon Fire Stick

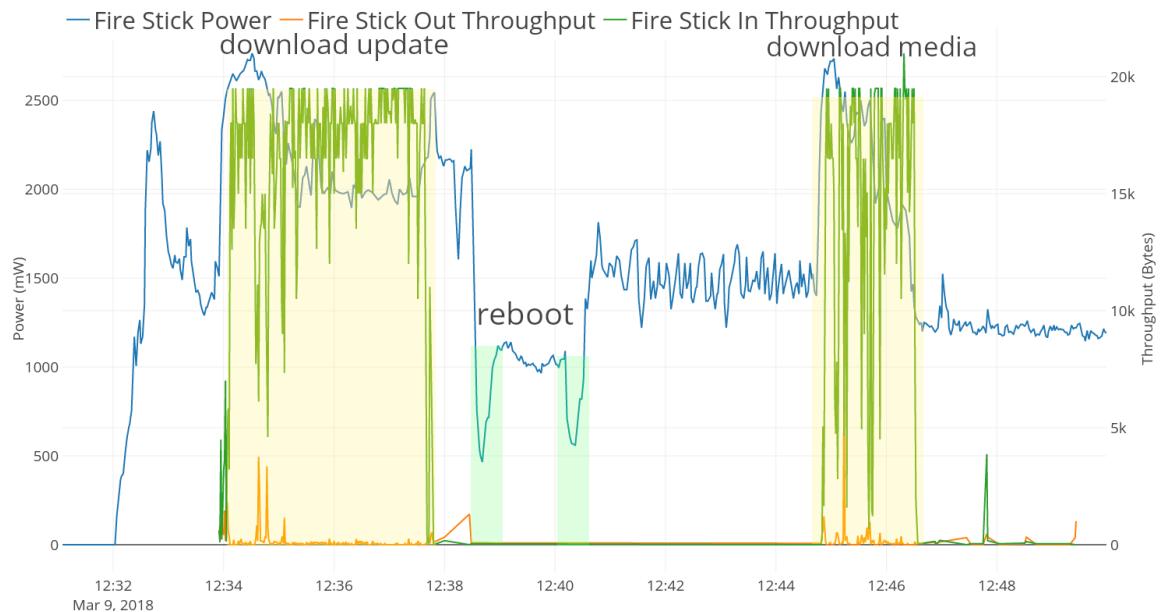


Figure 3.10: Fire TV Stick First Time Boot Network Traffic and Power Consumption

Figure 3.10 shows the Amazon Fire Stick startup graph. On first boot, the fire stick downloads an update, reboots twice, then downloads certificates from Symantec and Verisign.

When streaming, the Fire Stick increases throughput and power usage with a spike at the beginning and end of streaming in power usage as shown in figure 3.11.

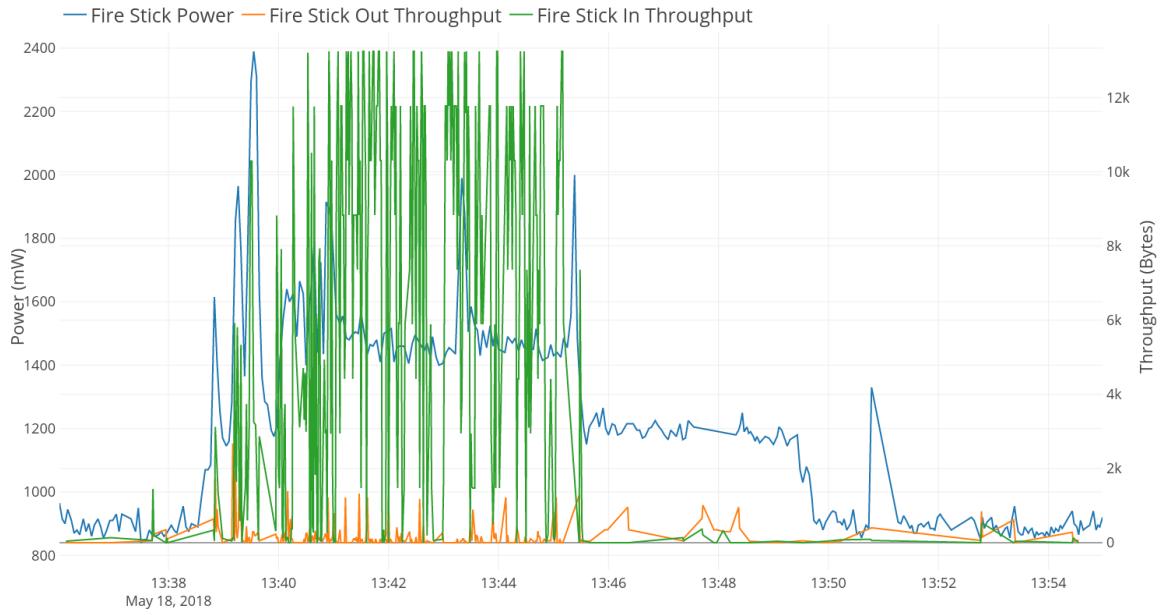


Figure 3.11: Fire TV Stick Video Streaming

Roku Express

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.

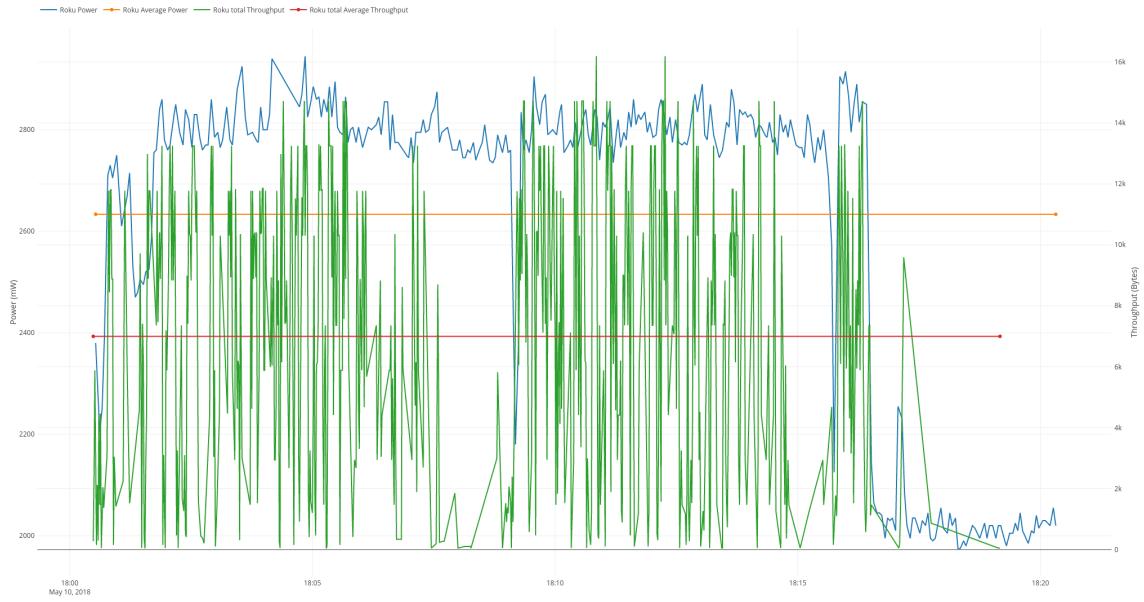


Figure 3.12: Roku Express Video Streaming

3.3.2 General Analysis Discussion

From all the figures in this section, there is a visual correlation between network/power usage and what each device is doing. During periodic updates, the smart speakers and streaming devices would have periodic network/power usage spikes as shown in figure 3.9. While streaming, the power/network usage would increase for the period of the stream as shown in 3.8, 3.11, and 3.12. This strong correlation shows the value of the database and visualizer tool for visual pattern recognition. It also shows the viability of using network throughput/power usage over time to understand what a device is doing.

3.4 Power Analysis on Smart Speakers

After determining a strong correlation between network/power usage and device operation, this section begins to focus on power usage over time. There is already significant analysis on network usage 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 3.5. Subsection 3.4.2 examines the idle power of smart speakers separately, and how correlates to a specific smart speaker. Then subsection 3.4.3 shows that through visual examination of a power trace of an individual smart speaker, we can tell what smart speaker is in use.

3.4.1 Echo Dot Brightness Sensor

One interesting finding from the Echo Dot is that it has a light monitor. When turning on the 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 3.13.

Figure 3.13 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 even what room they are in, introducing some privacy concerns.

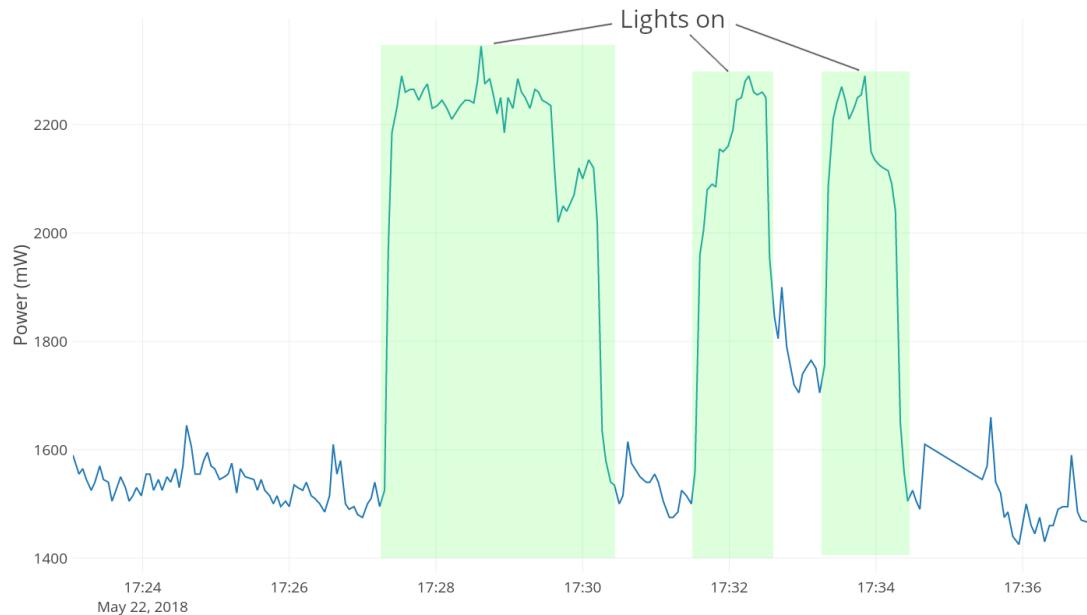


Figure 3.13: Echo Dot Response to Lights

3.4.2 Baseline Speaker Power

Once discovering how much a smart speaker's power usage can show, this section examines the power usage of each smart speaker while idle. Figure 3.14 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 traffic as shown in the pink trace, the Echo Dot 2 has second highest idle traffic as shown in the green trace, the Eufy 1 and Eufy 2 have the same idle power usage as shown in the yellow and purple trace, then 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 would make it difficult to differentiate them and shows this method to determine a device is difficult. Also, when summing the power usages together, these idle traces would disappear, with no way to differentiate what idle devices are contributing to the total power usage.

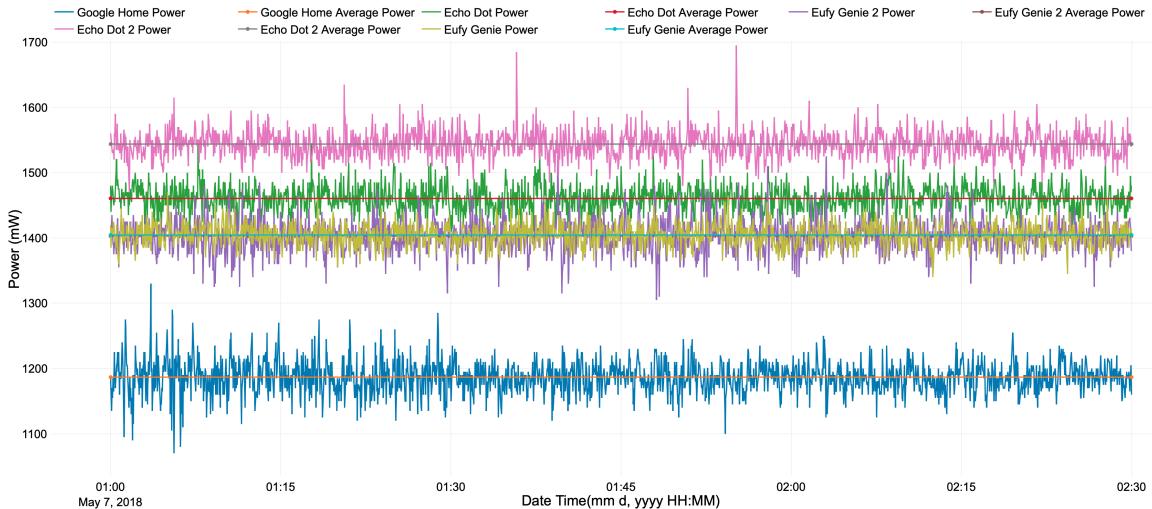


Figure 3.14: Baseline smart speaker power usage.

3.4.3 Smart Speaker Power Spikes When Asked for Weather

This section continues to see if a smart speaker can be determined from an individual power trace, focusing on traces while the device is in use. Figure 3.15 shows this, where each device is asked for the weather four times.

In figure 3.15 there is a visual spike for each device while it is giving the weather forecast. Each device is highlighted while in use, showing four spikes. The Google Home was queried for the weather five times, thus containing an extra spike. From this, we can determine a Eufy trace if it has small 400 mW peak to peak spikes, the Google Home if there is a medium 600 mW peak to peak spikes, and the Echo Dot if there is a large 2,100 mW peak to peak spikes. This 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.

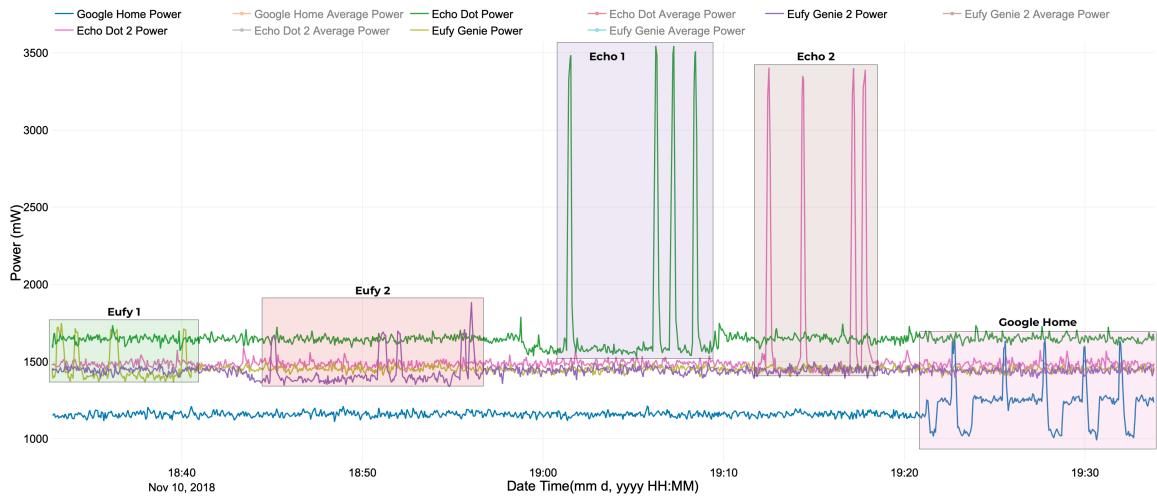


Figure 3.15: Smart speakers' power usage when asked 'what's the weather' four times.

This section highlights initial power findings, and how visual examination of a power graph over time gives a lot of information. Once discovering the insight, a power trace can provide, the research focused on the overarching goal to see if given a graph of a house's total power usage, is it possible to determine the device in use.

3.5 Summed Power Graph

This section visually analyzes the power usages of all 5 of the smart speakers summed together under different commands.

The first graph in figure 3.16 shows a time frame where each device was unmuted

and muted for a period. This shows if muting and unmuting the devices affected the power or network usage. There are changes on the first run-through, but when repeating the process nothing happens. Further run-throughs show that power usage stayed the same. In the last half of the graph, 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’s the weather” command at 400 mW, then the Google Home at 600 mW, and finally the Echo Dot at 2000 mW.

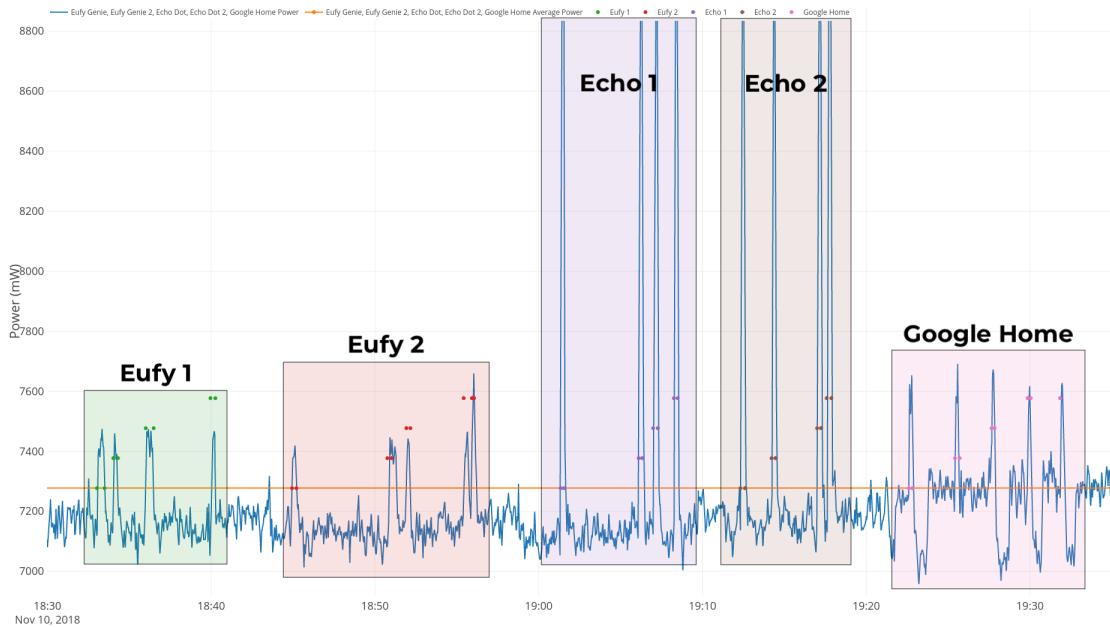


Figure 3.16: 5 Smart Speakers Power Summed Up. Toggled the mute button and queried each device for the weather.

The next graph, 3.17, shows the smart speakers power usage when asked for the news. Each speaker was 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 3.17, 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.

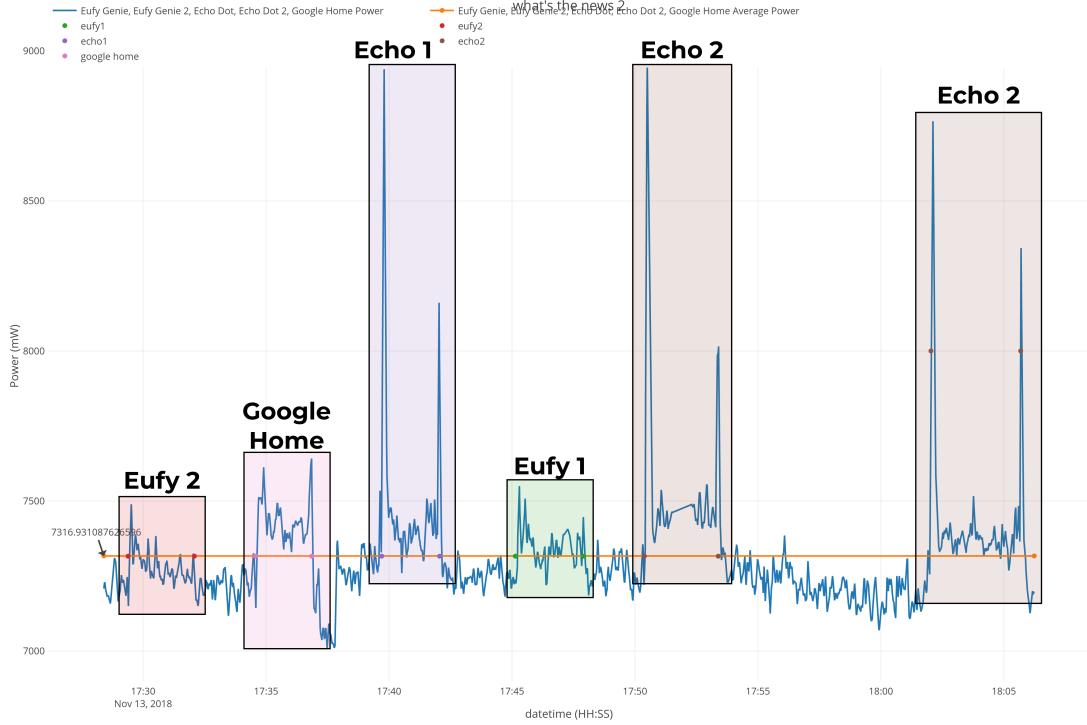


Figure 3.17: 5 Smart Speakers Power Summed Up. Queried each device for the news.

The next graph, 3.18, 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 a power spike of 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 3.18, there is a power spike that is unaccounted for in correspondence to the event log at 18:12. We made a query, invoking all other power spikes but we did not do anything for the power spike at 18:12 that is higher than the other Eufy power spikes shown in figure 3.18.

To figure what the power spike at 18:12 is, we separated the graphs into individual power traces as shown in figure 3.19. 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 3.20. At 18:12, there is no significant network usage.

From figures 3.19 and 3.20, we speculate that because there is no network usage

during this time, the Echo Dot was doing anything data recording, listening, or was activated. We believe that the computer was covering the Echo Dot, but is briefly moved, exposing the Echo Dot to more light, thus causing the LEDs to brighten. The power usage before the spike and during the spike exactly match that of figure 3.13, starting at around 1500 mW, rising to about 2300 mW.

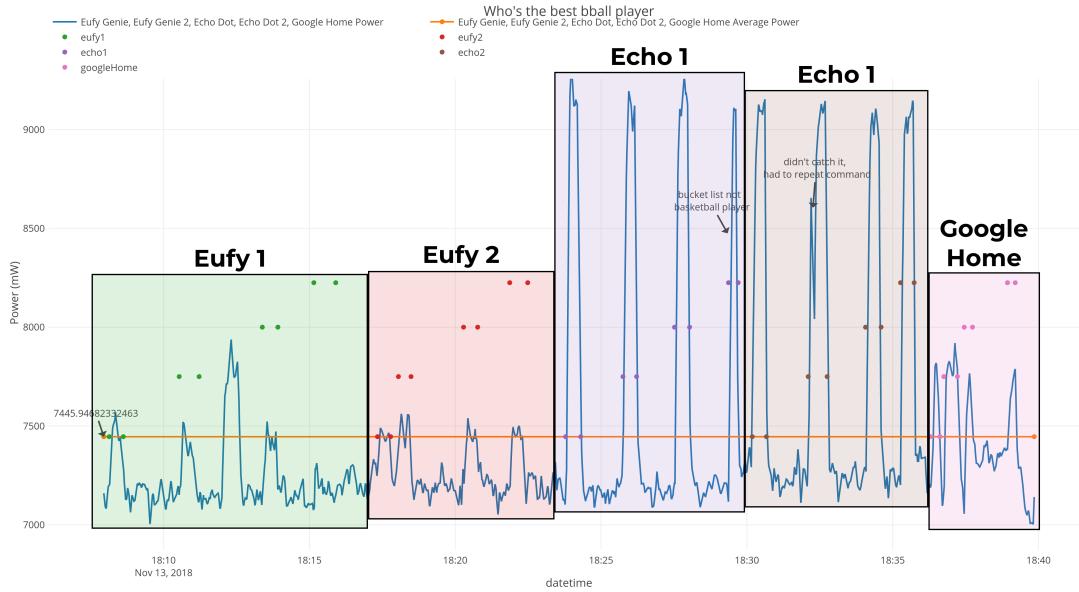


Figure 3.18: 5 Smart Speakers Power Summed Up. Queried each device for the best basketball player.

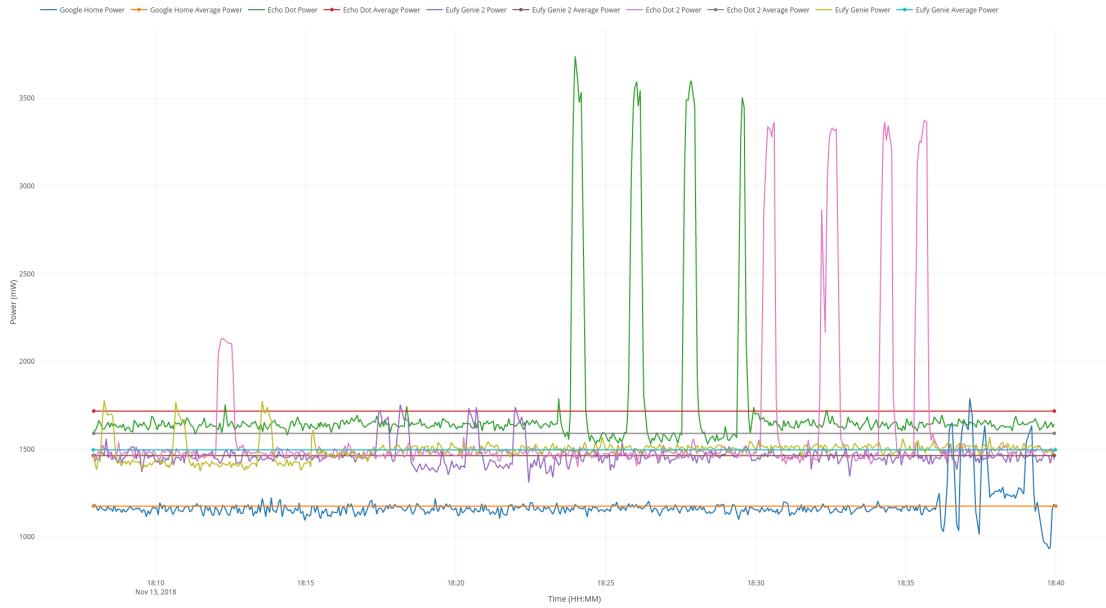


Figure 3.19: 5 Smart Speakers Power Usage over time. Queried each device for the best basketball player.

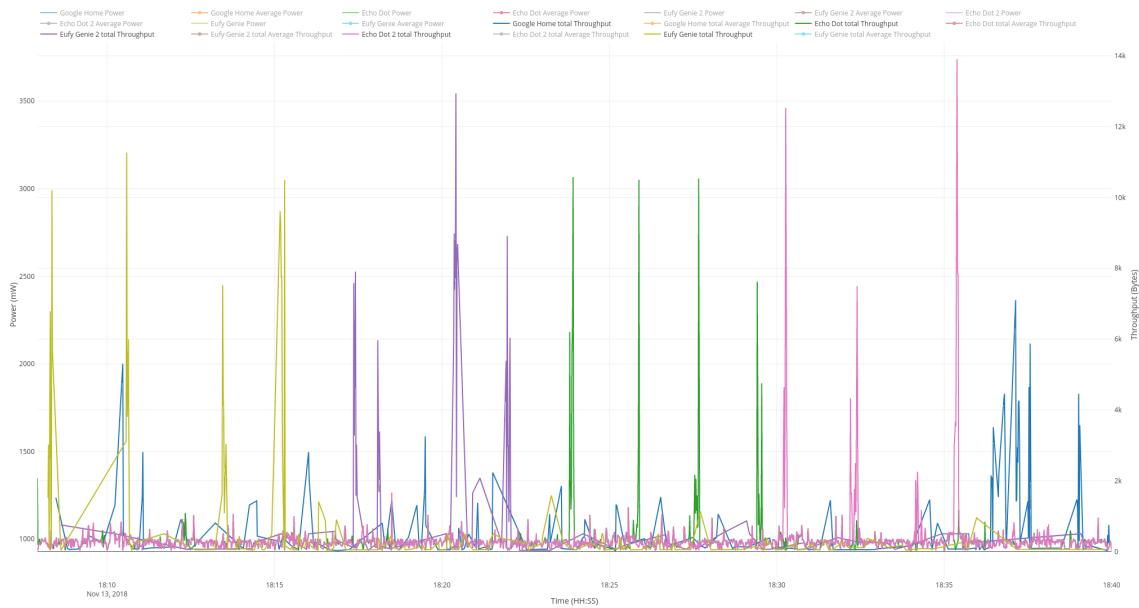


Figure 3.20: 5 Smart Speakers Network throughput over time. Queried each device for the best basketball player.

Figure 3.21 shows the power spike information for each query type (“what’s the weather” “what’s the news”, “who’s the best basketball player”). For each of these commands, there is always a characteristic power spike that occurs at the beginning

of the command. This bar graph also includes the total average spike height for each all three of these commands. The black line at the top of each bar shows the standard deviation. Averaging peak to peak voltage spikes for each device 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 3.21, shows that it is 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 testing our hypotheses that it is possible to see what devices are in use from analysis of someone's power line.

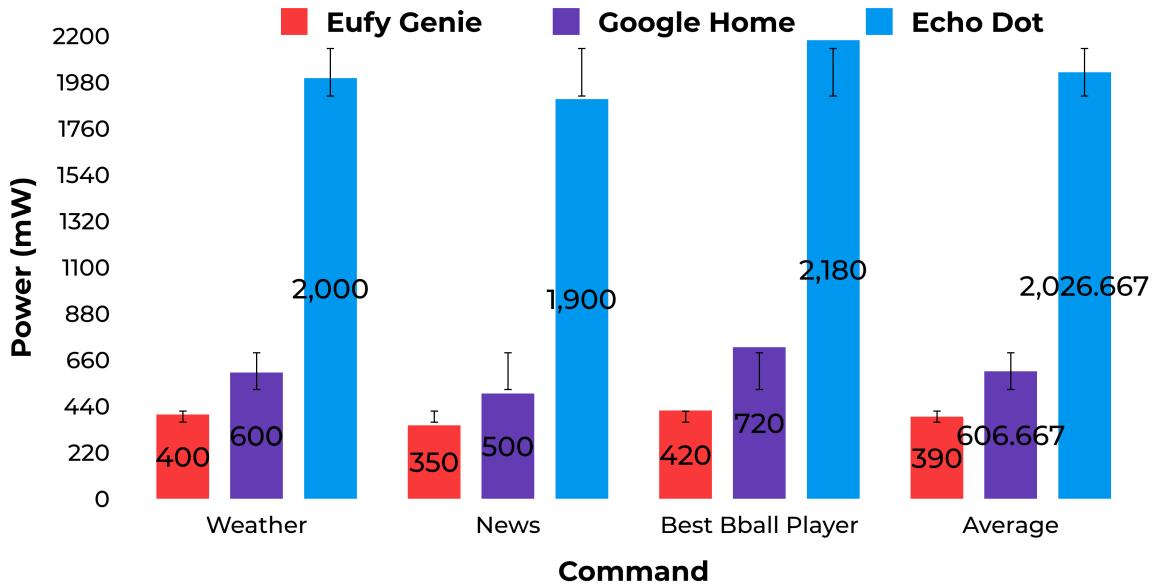


Figure 3.21: Spike summary of for each query.

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 3.6.

3.6 Summed Power Graph with Noise

In this section, we introduce noise into the system 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 3.18. The second trace (purple trace) is the power usage of a high power device. This trace is the added noise. It maps to various devices as we switch them out. The third trace (red trace) is the sum of both trace 1 and 2. It is the power usage of the smart speakers in the blue trace with the noise from the device in the purple trace.

The x-axis is the time elapsed in seconds. There is no time 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.

Below, in figure 3.22, is an example of the traces used separately. Taking the blue trace (smart speaker power usage) and the purple trace (noise-introducing device) the summed up graph results in the red trace (total power usage). 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.

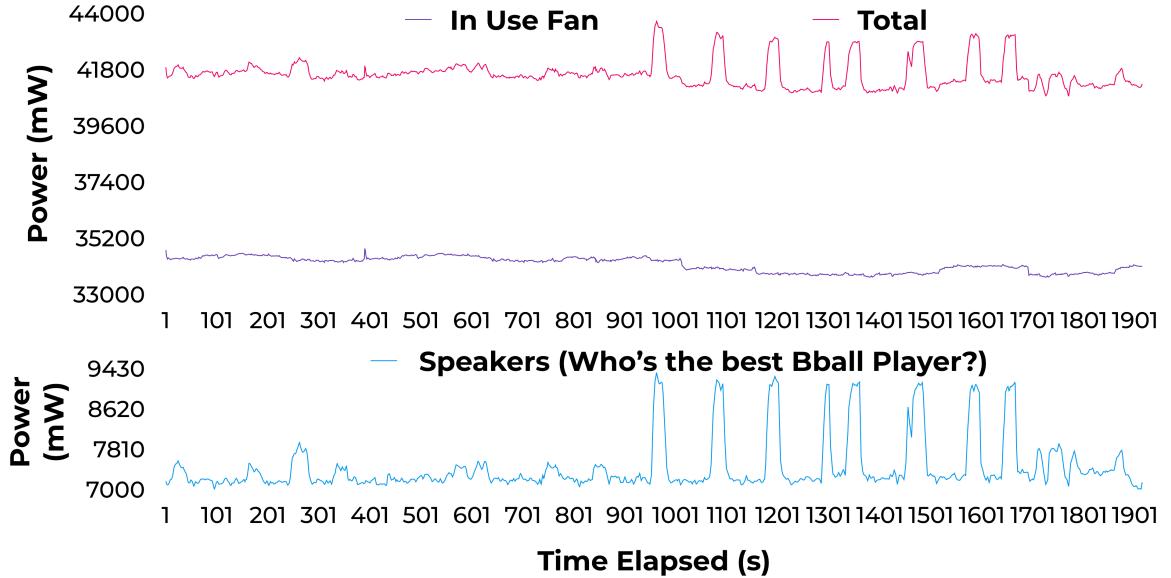


Figure 3.22: Idle Fan with figure 3.18 trace shown separately.

Below, we display the smart speaker trace with a PC (Intel NUC), fan, refrigerator, or microwave. Figures 3.22, 3.23, 3.24, 3.25, 3.26, and 3.27 begin with noise from devices while idle or during steady power usage to set a baseline. Then figures 3.29, 3.30, and 3.31 show noise from devices while in use, introducing noise that make visual detection of smart speakers most difficult.

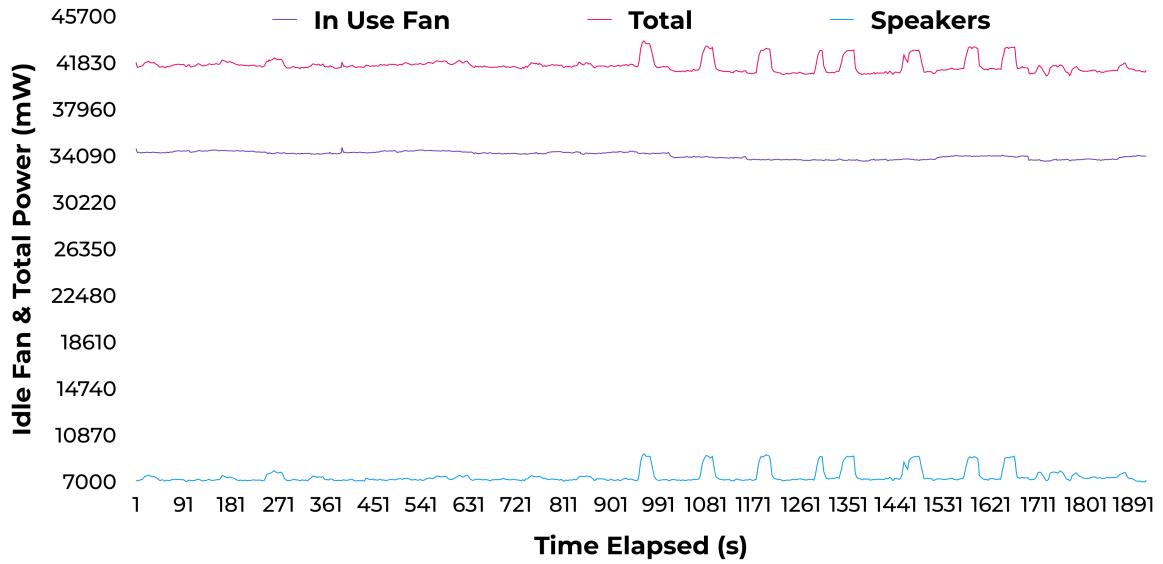


Figure 3.23: Idle Fan with figure 3.18 trace.

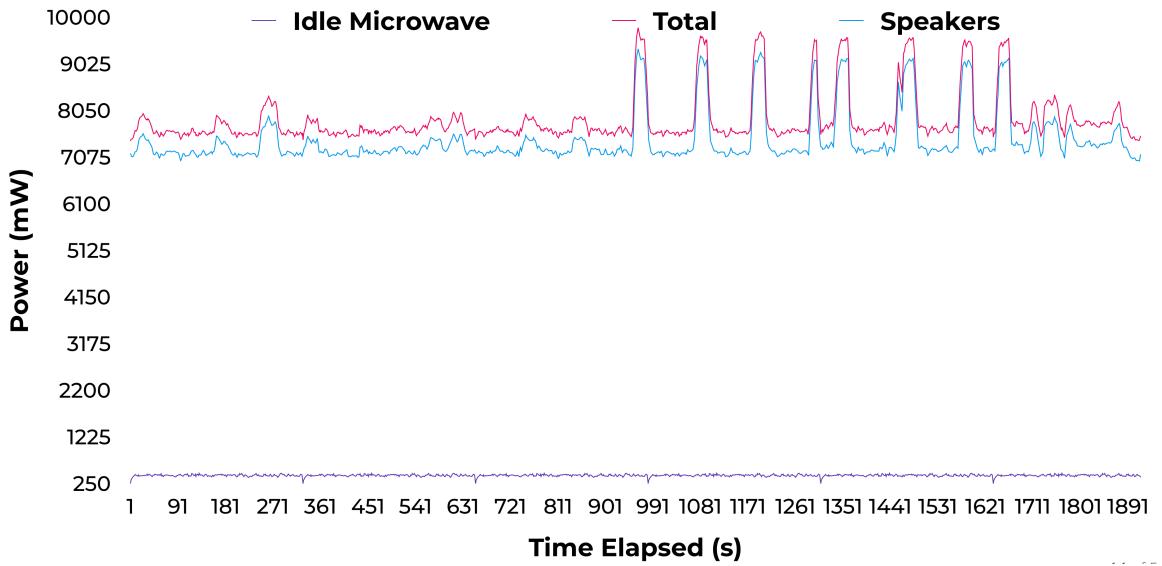


Figure 3.24: Idle microwave with figure 3.18 trace.

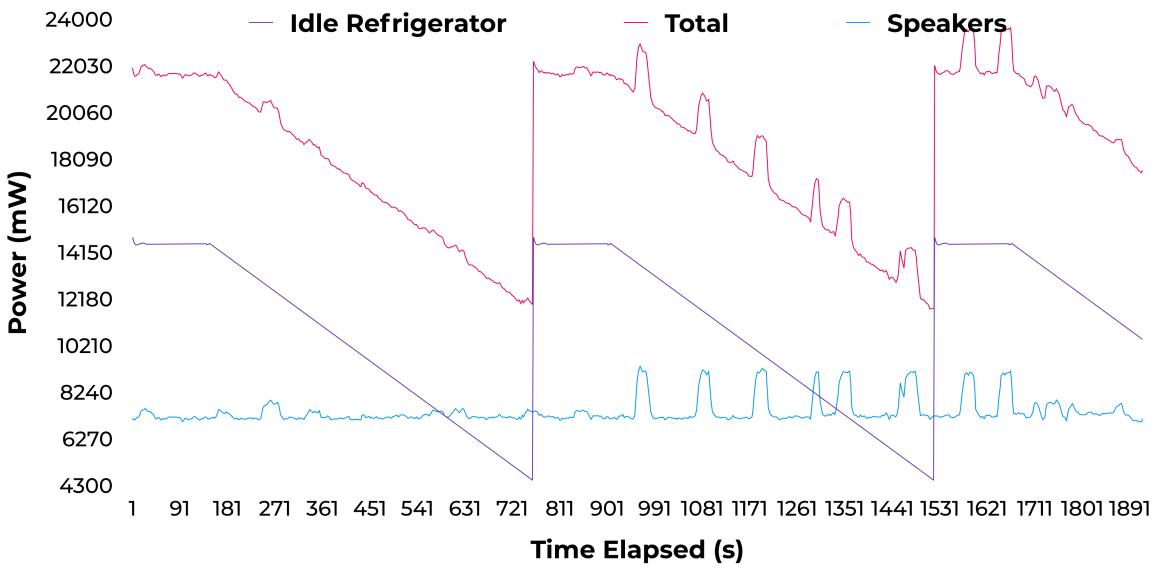


Figure 3.25: Idle refrigerator with figure 3.18 trace.

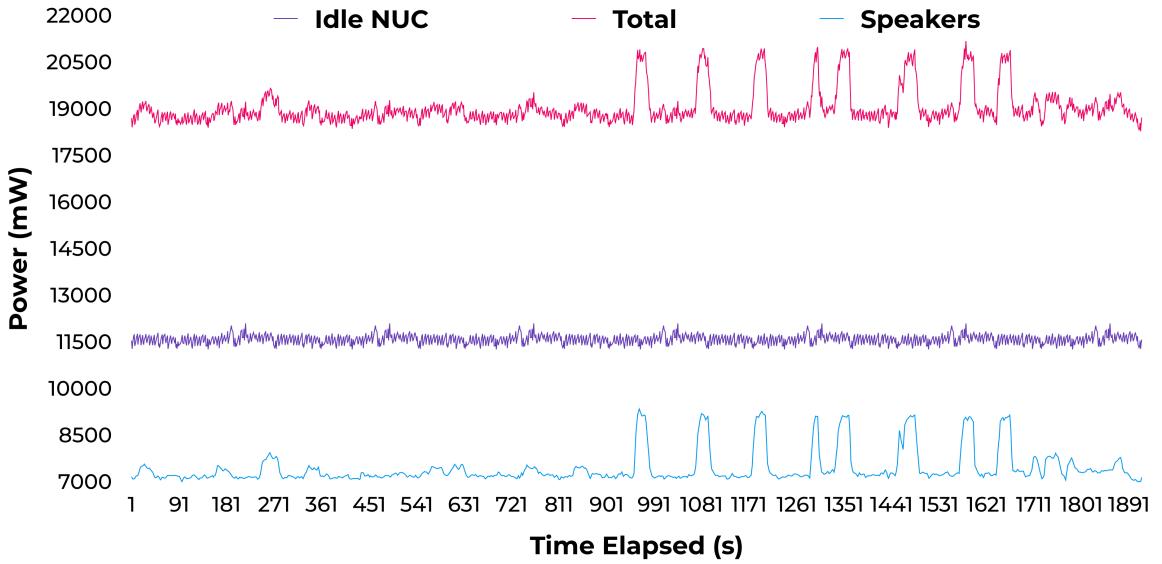


Figure 3.26: Idle PC (Intel NUC) with figure 3.18 trace.

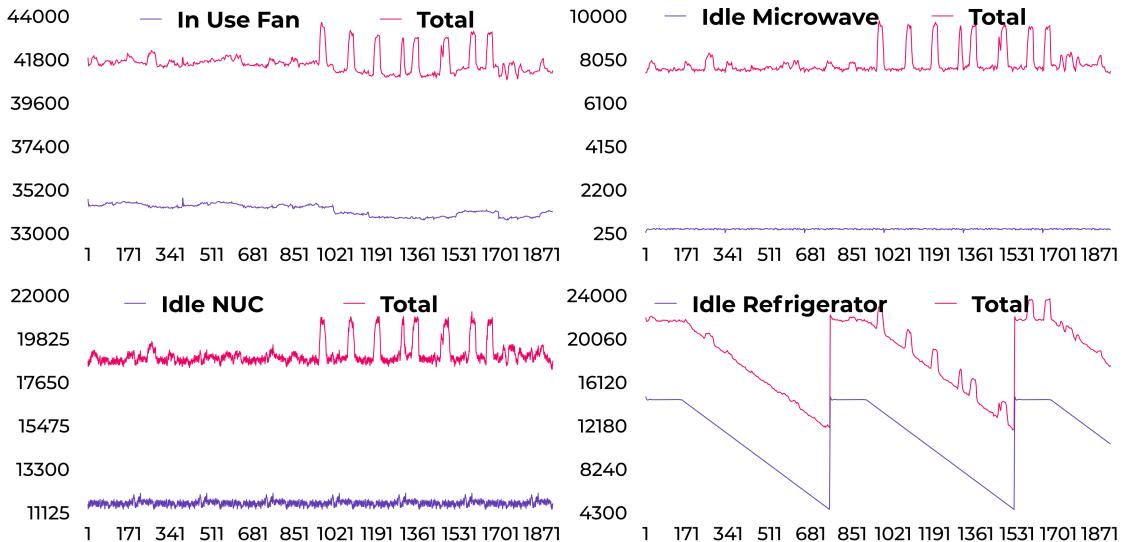


Figure 3.27: 3.23, 3.24, 3.25, and 3.26 figures zoomed in

In figures 3.23, 3.24, 3.25, and 3.26, 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 3.27 where the smart speaker trace is removed.

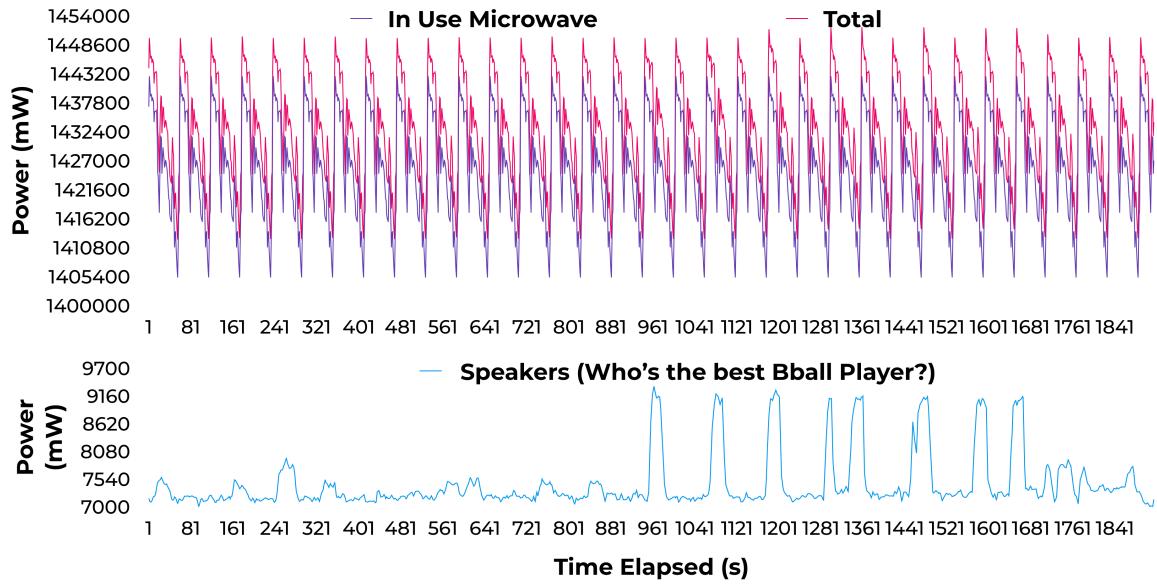


Figure 3.28: In use Microwave with figure 3.18 trace separately.

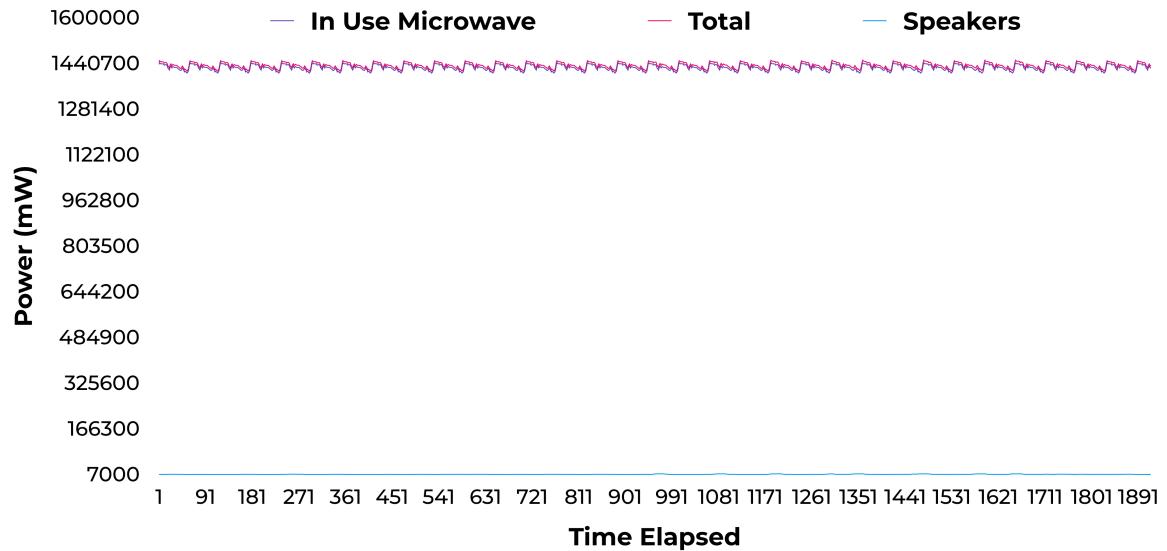


Figure 3.29: In use Microwave with figure 3.18 trace.

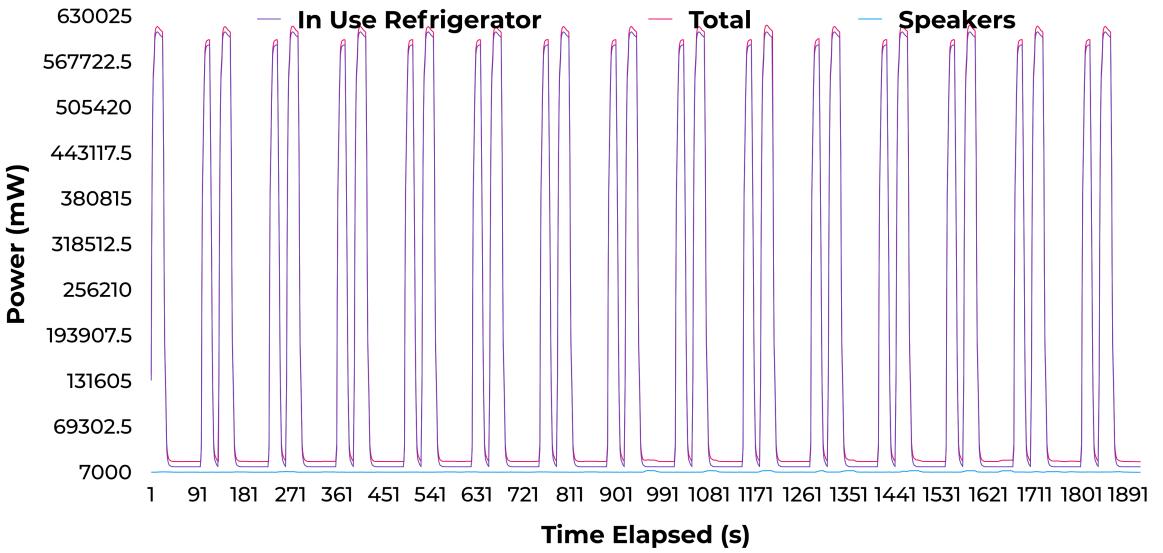


Figure 3.30: Fridge in the middle of cooling with figure 3.18 trace.

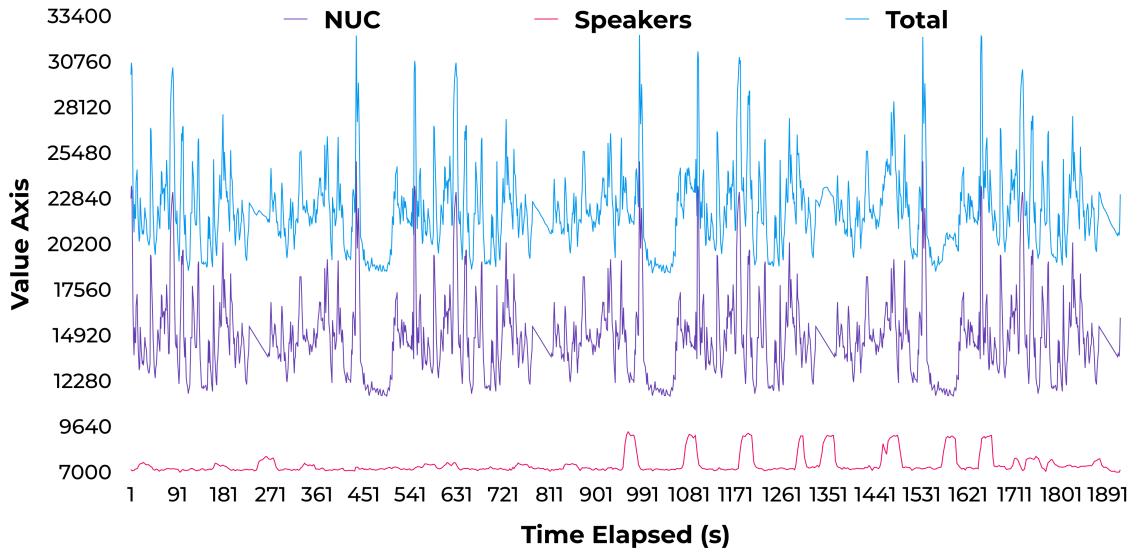


Figure 3.31: PC in use with figure 3.18 trace.

The figures above show that it is possible to determine the smart speaker 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 3.23. If the noise within the house is low power then, the spikes visually overpower the noise, as shown in figure 3.26.

But if the noise within the house is considerable in magnitude such as the microwave or fridge while in use 3.29 3.30, 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 3.28. 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 3.31. The power spikes can still be seen in the sum graph for this figure, but it would be impossible to visually differentiate between a power spike from a speaker and the 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 would still be possible to determine the power spikes visually. Alternatively, in another case, if high power devices such as the microwave or fridge are not in use often, then the smart speaker spikes can be visually correlated. But if a household has many devices plugged into an outlet for power, then the smart speaker power spikes will disappear from the additive noise of all the devices. Regardless, there are privacy implications in a household's power line. With easy, direct (even remote nowadays), access to a household's power line [39], someone could determine the device in use within a household or what room they are in.

3.7 Smart Speaker Comparison

The figures in section 3.5 show that the Echo Dot uses a lot more energy than the other smart speakers. Because of this, we wanted to compare the energy and network usages of the three smart speakers individually so that we can determine trade-offs that they make.

In the figures 3.32 and 3.33, 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 3.19 for the two graphs below. 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 reffig:smartSpeakerSeperate and 3.33, 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, we have the Eufy Genie (1743 bytes), Google Home (800 bytes), and Echo Dot (325 bytes). These results are shown in figure 3.34

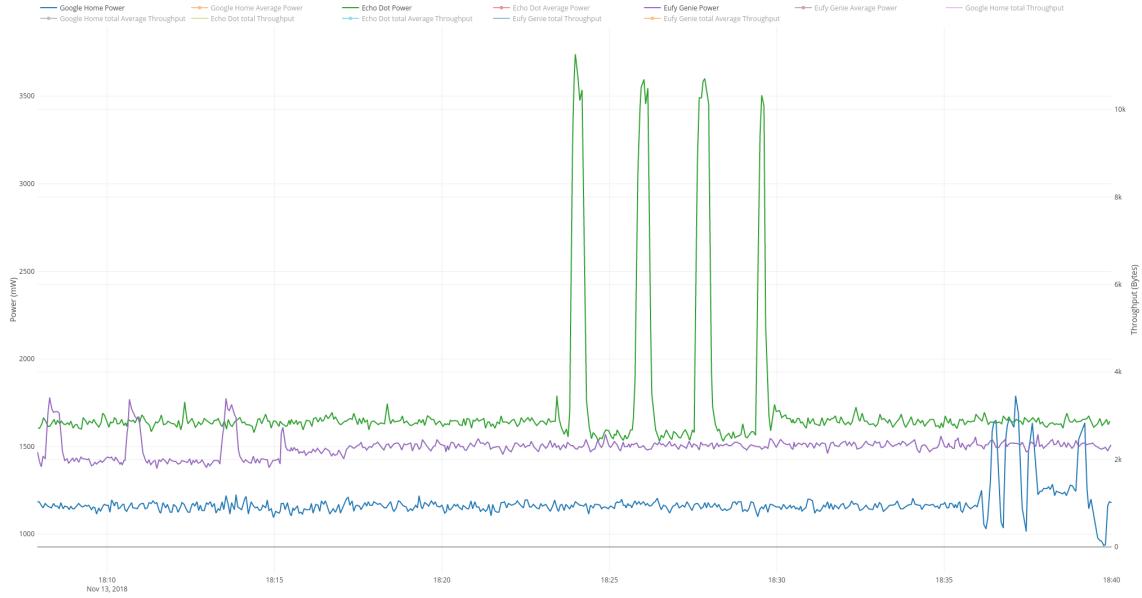


Figure 3.32: Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked “who is the best basketball player”. Same graph as figure 3.19 with Echo Dot 2 and Eufy Genie removed.

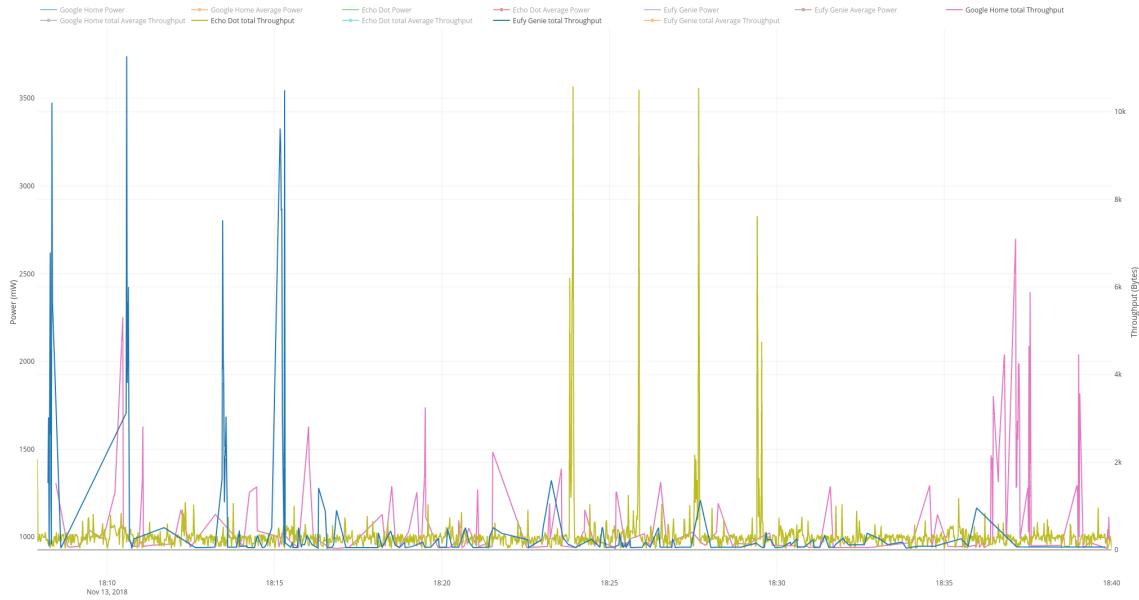


Figure 3.33: Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked “who is the best basketball player”. Same time frame as figure 3.32

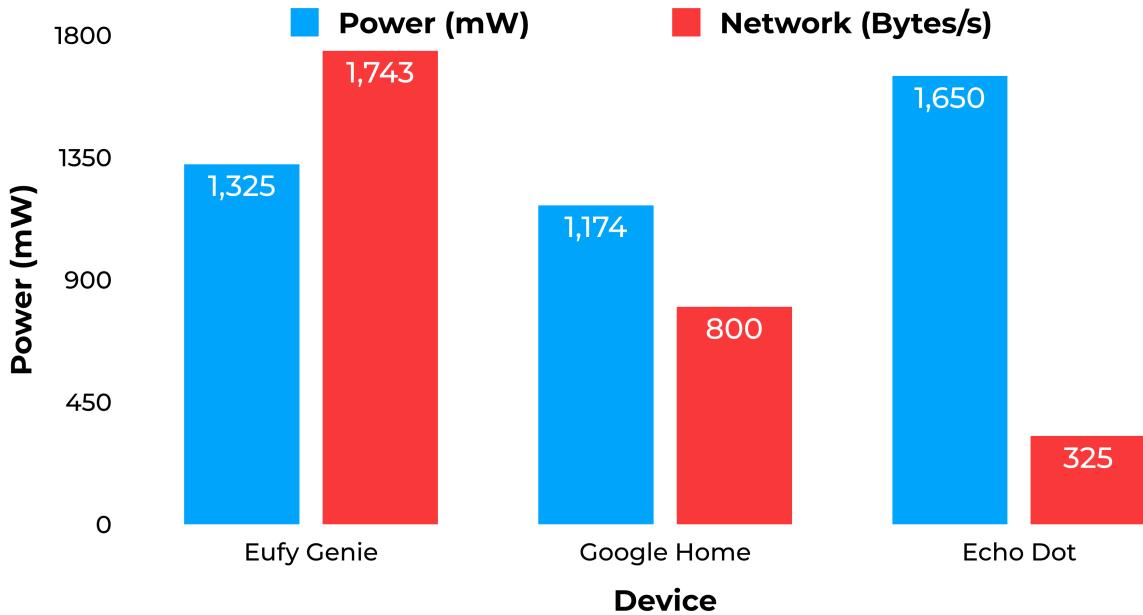


Figure 3.34: Average power and network usage at each second for the Echo Dot, Eufy Genie, and Google Home in the time frame of figures 3.32 and 3.33

From figure 3.34, 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. At its core, 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. Further analysis to prove this would be interesting.

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. Data is often overlooked as a crucial part of any project, such as research projects or AI projects. The process of obtaining it can be tedious. This database provides a large dataset for others to use without having to repeat the steps to form a testbed and log network traffic and power usage. This database was incredibly useful in Frawley and my work. Other students have already used it for their research projects.

Complementary to this database, a lot of data visualization was necessary to understand and extract patterns in IoT devices. To further streamline IoT research, a visualizer tool that graphs power and network usage for any device in the database makes 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 possible to determine what smart speaker is in use and whether the lights are on in the house from the power data. With more research, it is likely that other private information can be inferred from the power line data as the research here is highly informal and has yet to leverage machine learning to search for patterns. The results show a privacy flaw in household power lines.

Together, this proves that more research should be done to analyze and classify IoT devices. The latter analysis in smart speaker classification is informal, utilizing visual analysis and some numbers to strongly support our claims. A more formal

study into device classification from a shared powerline can be done. Also, analysis with more than just smart speakers and power spikes can be done.

4.1 Future Work

With an ever-growing database, its 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 an IP address of devices and naming of WeMos change over time. Changing the IP addresses and WeMo names that refer to the same device is necessary.

The next plan is to improve the realtimeIoTgrapher and add features. Hosting it on a public domain would make this tool more easily accessible, rather than running it locally and going to a local domain. Including more automated annotation on the graphs would also make anomaly detection easier.

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 also attempted to hack into these devices with Mirai and analyze power usage and network throughput of these devices. Getting Mirai working in the restricted time turned out to be unfeasible, but would be interesting to continue for future work.

Finally, applying machine learning to the power data would be interesting to pursue. We quickly tried 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 (HAR) model to fit the database for classification of device model given power data over a set 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 have already been students who have used the database for various research projects. Researchers can quickly utilize the realtimeIoTGrapher to analyze the database or manually query data. The world of IoT needs careful analysis and

research, and hopefully, my research reduces some friction for more projects about IoT devices.

BIBLIOGRAPHY

- [1] “IoT: Number of Connected Devices Worldwide 2012-2025,” 2016. [Online]. Available: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>
- [2] python, “Python,” 2018. [Online]. Available: <https://www.python.org>
- [3] S. Burke, “Google Admits Its New Smart Speaker Was Eavesdropping on Users,” 2017. [Online]. Available: <https://money.cnn.com/2017/10/11/technology/google-home-mini-security-flaw/index.html>
- [4] Z. Whittaker, “Fear the Reaper? Experts Reassess the Botnet’s Size and Firepower,” 2017. [Online]. Available: <https://www.zdnet.com/article/reaper-botnet-experts-reassess-size-and-firepower/>
- [5] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree, “An Analysis of Home IoT Network Traffic and Behaviour,” *ArXiv e-prints*, 2018.
- [6] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, “ProfilIoT: a Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis,” in *Proceedings of the Symposium on Applied Computing*, ser. SAC ’17. ACM, 2017, pp. 506–509. [Online]. Available: <http://doi.acm.org/10.1145/3019612.3019878>
- [7] Wireshark, “Wireshark.” [Online]. Available: <https://www.wireshark.org/>
- [8] R. J. Frawley, “Logging and Analysis of Internet of Things (IoT) Device Network Traffic and Power Consumption,” 2018. [Online]. Available: <https://digitalcommons.calpoly.edu/theses/1911/>
- [9] “Geoip databases and services: Industry leading ip intelligence.” [Online]. Available: <https://www.maxmind.com/en/geoip2-services-and-databases>
- [10] Amazon, “Amazon rds.” [Online]. Available: <https://aws.amazon.com/rds/>
- [11] MySQL, “Mysql.” [Online]. Available: <https://www.mysql.com/>

- [12] Intel, “Intel nuc.” [Online]. Available:
<https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html>
- [13] Ubuntu, “Ubuntu.” [Online]. Available: <https://www.ubuntu.com/>
- [14] linux, “linux.” [Online]. Available: <https://www.linux.org/>
- [15] Ookla, “Speedtest.” [Online]. Available: <https://www.speedtest.net/>
- [16] Google, “Chromebook.” [Online]. Available:
<https://www.google.com/chromebook/>
- [17] “2018 Broadband Deployment Report,” 2018. [Online]. Available:
<https://www.fcc.gov/reports-research/reports/broadband-progress-reports/2018-broadband-deployment-report>
- [18] Belkin, “Wemo.” [Online]. Available:
<https://www.belkin.com/us/Products/smarthome-iot/c/wemo/>
- [19] S. Perez, “Voice-enabled Smart Speakers to Reach 55 Percent of U.S. Households by 2022, Says Report,” 2017. [Online]. Available:
<https://techcrunch.com/2017/11/08/voice-enabled-smart-speakers-to-reach-55-of-u-s-households-by-2022-says-report/>
- [20] ——, “39 Million Americans Now Own a Smart Speaker, Report Claims,” 2018. [Online]. Available: <https://techcrunch.com/2018/01/12/39-million-americans-now-own-a-smart-speaker-report-claims/>
- [21] J. Lynch, “Nearly 70 Million U.S. Households Now Have a Connected-TV Streaming Device,” 2017. [Online]. Available:
<https://www.adweek.com/tv-video/nearly-70-million-u-s-households-now-have-a-connected-tv-streaming-device/>
- [22] “168 Million Will Watch Connected TV in the US This Year,” 2017. [Online]. Available: <https://www.emarketer.com/Article/168-Million-Will-Watch-Connected-TV-US-This-Year/1016233>
- [23] https://www.nintendo.co.jp/ir/en/finance/hard_soft/index.html, “Dedicated Video Game Sales Units.”

- [24] A. Souppouris, “Sony Has Sold 50 Million PlayStation 4s,” 2016. [Online]. Available:
<https://www.engadget.com/2016/12/07/sony-playstation4-50-million-sales/>
- [25] “The 5 Worst Examples of IoT Hacking and Vulnerabilities in Recorded History,” 2018. [Online]. Available:
<https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/>
- [26] N. Feamster, “Who Will Secure the Internet of Things,” 2016. [Online]. Available:
Available: <https://freedom-to-tinker.com/2016/01/19/who-will-secure-the-internet-of-things/>
- [27] Belkin, “Wemo,” 2018. [Online]. Available:
<https://itunes.apple.com/us/app/wemo/id511376996?mt=8>
- [28] Oblique, “Create ap,” 2017. [Online]. Available:
https://github.com/oblique/create_ap
- [29] J. M, “How to create wifi hotspot in ubuntu 16.04,” 2016. [Online]. Available:
<http://ubuntuhandbook.org/index.php/2016/04/create-wifi-hotspot-ubuntu-16-04-android-supported/>
- [30] mysqlclient, “mysqlclient.” [Online]. Available:
<https://pypi.org/project/mysqlclient/>
- [31] pavoni, “pywemo,” 2018. [Online]. Available: <https://github.com/pavoni/pywemo>
- [32] navicat, “Navicat,” 2018. [Online]. Available: <https://www.navicat.com/en/>
- [33] mysql, “Mysql commandline,” 2018. [Online]. Available:
<https://dev.mysql.com/doc/refman/5.5/en/mysql.html>
- [34] google, “Google sheets,” 2018. [Online]. Available:
<https://www.google.com/sheets/about/>
- [35] weimin chen, “nvas2,” 2018. [Online]. Available:
<https://itunes.apple.com/cn/app/nvas2/id1042160465?l=en&mt=8>
- [36] plotly, “plotly,” 2018. [Online]. Available: <https://plot.ly>

- [37] pandas, “Pandas,” 2018. [Online]. Available: <https://pandas.pydata.org>
- [38] C. P. I. T. Services, “Cal poly its.”
- [39] E. Griffith, “How to Measure Home Power Usage,” 2017. [Online]. Available: <https://www.pcmag.com/article/343177/how-to-measure-home-power-usage>

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 → start time + interval. If the end time and interval are specified, then the time window is the end time - interval → 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.