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

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 logging all of the devices as well as extra analysis on the devices alongside me. You set a really good base for us to build upon!
- Cal Poly and Cisco for funding this project. We would not have been able to get all of the IoT devices without your help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF CODE LISTINGS

Chapter 1

INTRODUCTION

There are 23.14 billion IoT devices in use worldwide; by 2025 that number is broadcast to increase to 75.44 billion[1]. With so many manufacturers creating IoT devices, each with differing update policies, some devices will inadvertently have better support. For example, some devices may drop out of a manufacturers update cycle, introducing privacy and security concerns. Several large-scale cybersecurity attacks such as Mirai [2] have already highlighted flaws in IoT security and privacy.

To address these concerns, I have built 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 rows into a database. To help researchers sort and view this data, I have written a Python program that graphs network traffic and power data over time from the database. I then used graphs created by this tool to analyze IoT devices in the testbed while idle, during startup, and while in use. From this analysis, I characterized common patterns for each devices' network and power usage. From these graphs, I have also found that it is possible to identify the smart speaker in use when viewing just one minute of the shared power usage.

This paper focuses on addressing security and privacy flaws, that if fixed, do not affect the core features of an IoT device. For example, it is required for a smart speaker to listen to its user and store the audio in increments so that it can parse the audio 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. This is not what this paper covers. However, Google Homes had an issue with their wake button caused the Home to listen 24/7 [3]. In another case, 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]. In this case, a manufacturer cannot expect a user to change their login info and should create a more secure sign-in process. These flaws are what this paper searches for through network and power usage.

1.1 Previous Work

This section presents and analyzes related works on the topic of analyzing and characterizing IoT devices. This section presents the previous works individually. Because these papers are similar to each other, commentary on how their work is different from ours and how it is useful to us as a group in section 1.2.

1.1.1 An Analysis of Home IoT Network Traffic and Behaviour

The scope and work of *An Analysis of Home IoT Network Traffic and Behaviour* [5] are the most similar to the work and goals of this paper. 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, conecting them to a router, sniffing their network packets while idle, and storing these packets on a Linux box's disk. 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 individually and as a whole. For example, they noticed that they can identify most devices from the first three MAC address bits. The Hue bridge broadcasts credentials over HTTP, which is unencrypted. The authors state that these seemingly small security flaws create a privacy risk. A users 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 for comparison. These devices connect to a WiFi access point that recorded 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 formatted as source IP, destination IP, source port Number, destination port Number. When intercepting network traffic, they extract the information they need from each TCP packet to form the four-tuple data type. 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. Then out of their nine IoT devices, they can classify brand and model of the IoT device with 99.281 percent accuracy when run 7376 times.

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 the devices.

1.2 Scope

The first 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.

Then, like the second paper from subsection 1.1.2, this paper also focuses on classification of devices from data. However, instead of using machine learning techniques on network data, this paper focuses on manual analysis, looking for spikes in power usage, the height of the spike, the length of the power spike, and other graphical heuristics.

In comparison to the first paper in 1.1.1 and second 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 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 developer environment to run the analysis tool 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. Chapter 3 also shows the graphs used to fingerprint the smart speakers while handling different commands and under noise. Chapter 4 discusses the data presented in Chapter 3, why a particular device might have higher throughput traffic and the feasibility of classifying devices within a household from a shared power line. Finally, Chapter 5 finishes with concluding thoughts and future work.

Chapter 2

METHOD

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 will query 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 net provided through the WAP intercepts network packets of devices connected to it (IoT devices in tesbed), parses them into power and network entries, and pushes them to the database. Finally, for analysis, a python script called realTimeGrapher.py pulls from the database and graphs this data in real time or from set intervals in the past.

Section 2.1 discuss what devices we used, the set up of each device, and the physical layout of the IoT test bed. Section 2.2 will go over the software part of the IoT test bed. It covers the usage flow of this test bed and what was done to obtain results for analysis.

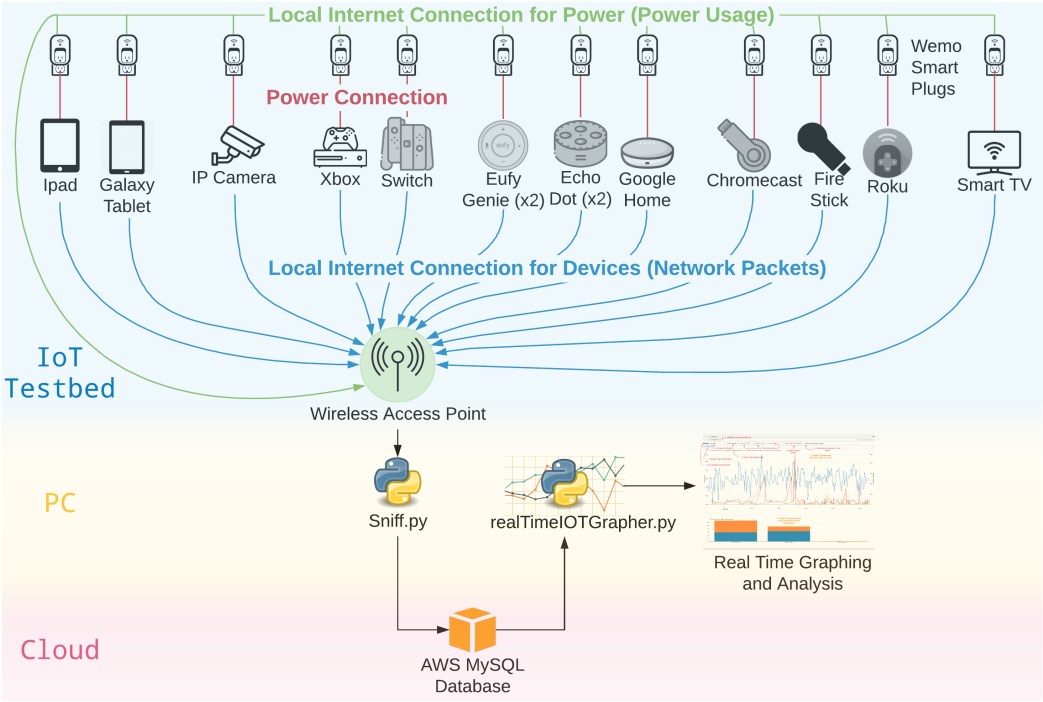


Figure 2.1: Network Diagram

Figure 2.2: Smart Speakers Connected to Wemo Insight Switches

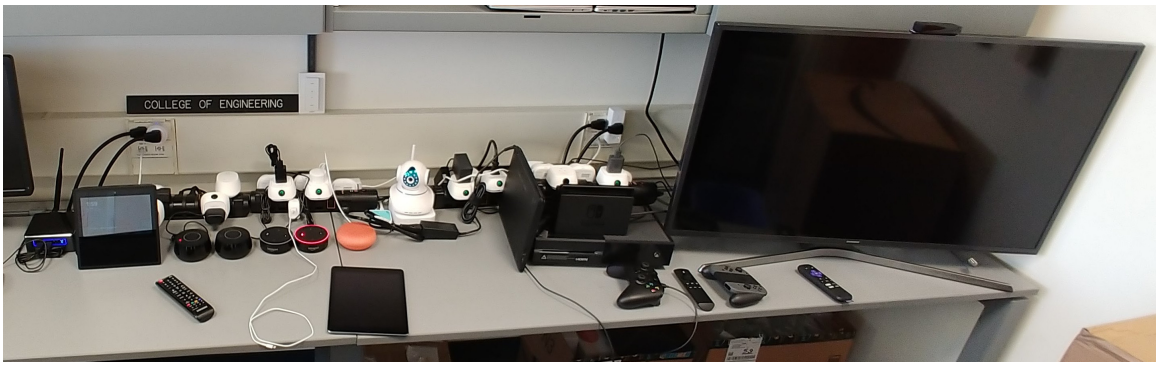


Figure 2.3: IoT Devices Under Examination

2.1 Physical Layout and Setup

This section covers the physical setup and layout of the hardware used when setting up the IoT test bed, the IoT devices we tested, and how each device is set up. Figures 2.2 and 2.3 show the IoT testbed set up.

2.1.1 Wireless Access Point

The wireless access point is an Intel NUC [12]. We loaded Ubuntu[13] on it because its the most common Linux OS [14] and most flexible for developing scripts and programs to log network packets, query power information, and push those values to a database.

One challenging thing about the NUC is that its internal wireless chip 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] (25 Mb/s is the minimum possible for broadband). This rate limitation caused many of the Belkin Wemo smart switches [17] 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, solving the issue where devices would lose network connectivity.

2.1.2 Devices

This section goes over the smart speakers, streaming devices, video game consoles, tablets, security cameras, or other devices as listed in figure 2.1. These devices were included based on their popularity, as the goal of the paper 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 WiFi 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, such as 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[18]. It is one of the top-selling IoT devices and it can act a central voice control for other IoT devices. 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[19]. 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 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 streams videos or music from online services. In 2017 it was recorded that 70 million US households (58.7 percent of homes) had a television connected to streaming devices [20].

The specific streaming devices we look at include 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 [21].

To view content from these streaming devices, we use a Samsung smart TV. This TV also as streaming capabilities.

Video Game Consoles

Video game consoles are systems that were initially built specifically to play video games. We use 2 out of the three most popular gaming devices including the Nintendo Switch which sold 17.8 million units [22] and the Xbox One, which sold 30 million units [23] as of January 2018.

Tablets

Tablets are devices that have phone operating systems running on them such as the Apple IPad, Samsung Galaxy tablet, and the Amazon Fire tablet. Ultimately these devices were mainly used for interacting with the other IoT devices since most of them require a smartphone or tablet for set up or control. Some of the IoT devices are also limited to either Android, which runs on the Galaxy, or IOS, which runs on the IPad, so we got both.

Security Camera

Security cameras are cameras meant to run constantly as surveillance, streaming the video footage a user can view from any connected device at all times. Two out of the five largest recorded cybersecurity attacks targeted security cameras [24]. The specific camera we use is the Eray camera. It is a generic Alibaba device with a weak username:password of admin:1234. A weak username and password makes it very susceptible to cyber attacks. There have also been reports that smart cameras have been found to send unencrypted data [25]. Because this device is a third party, unknown manufacturer, this device is an outlier for which we hope to get more interesting data.

Other Devices

We also connected several other devices that are harder to categorize.

We have a smart hub connected to a smart lock, an indoor room temperature sensor, and smart LED light bulbs. These devices are difficult to analyze because they

communicate with the smart hub through Zigbee or Bluetooth at which point the smart hub aggregates the data and communicates with the network. In order to look at the network traffic caused by smart lights individually, we prevented all devices besides the smart LED bulbs from communicating with the smart hub. This isolation minimized other packet noise to the smart hub so that we can monitor the smart hub traffic as a replacement for the smart LED bulbs. However, even then, the smart hub might have extra overhead for other tasks, making isolated analysis difficult.

We also have a Chromebook which we used to control the Chromecast. However, because this is more a laptop than an IoT device, we decided to leave this out during our research

2.1.3 Smart Power Plugs

Smart power plugs are WiFi capable, pass through outlets that a device plugs into. The outlets collect and transmit power states over the network. For this reason, they are critical to the power consumption logging in this research.

We used the Belkin Wemo. We chose this smart plug because there are many open source Python libraries for pulling power information from them and because they were the easiest to set up.

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 reference data to a smart plug. This informal naming leads to issues when the wrong device is connected to the wrong smart plug, as discussed in section 2.2.5

2.1.4 Wiring and Configuration

When setting up the smart plugs, we manually named each smart plug in reference to the device connected to it. We also tried to set 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.

The next step was to set up each device and corresponding smart plug through

their set up application, filling in the devices, name, email, network configuration, and any other setup details.

When plugging in all of the devices for the first time for power, we worked to plug each device into the corresponding Wemo smart plug and connected to our WAP during device set up so that we could instantly log power and traffic information. It was essential to log power as soon as possible to catch data while a device is starting up. Some devices had already been opened and used, so they do not contain startup information(Xbox and the Echo Dot 1).

Set up and configuration took up a significant portion of the time. We worked to set up each device with a separate email account. To do this, we made around 20 AOL accounts. We wanted 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 do perform domain-specific optimizations. When setting up AOL accounts, we also faced a limitation of the number of email accounts tied to a single phone number. We had to use different phone numbers in order to circumvent this.

We set up each device in waves, taking around a week to set everything up. Incremental device setup with no plan resulted in a messy work environment, which made it difficult to match a device to it is corresponding smart plug. The Wemos also take up too much space to be plugged right next to each other. After a month of data logging, it was difficult to keep track of each device because we had not kept it organized. We unplugged everything, renamed the Wemos, organized the wiring, and organized the location of each smart device resulting in the set up in figure 2.3 and 2.2. With limited power strips, the organization helped maximize power ports. We also made sure to face the button for each power plug towards our view so that we can quickly notice whether the power plugs were on or off. Organization streamlined the logging process in the long run. Debugging network and connection issues was much simpler with an ordered set up. Now we could quickly identify each device and go down the line when running procedures on the devices.

2.2 Software

This section covers the software components used to control the IoT test bed. It includes the scripts for logging power and network traffic, the database information, and the python script for visually viewing the database.

2.2.1 Wireless AP Code

In order to turn the NUC into a wireless access point, we used a create_ap script[24]CITEoblique_2017. This app takes in the WiFi name, WiFi password, and the wireless interface. This script turned the NUC into a wireless access point (WAP). 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[24]CITEm_2016. We later used this feature. Partially through our research, as we set up more IoT devices and demand for bandwidth through the WAP increased, we had to switch to an external antenna and rerun the script in order to set the USB antenna as the WAP wireless interface causing a void of traffic due to this switching.

2.2.2 sniff.py

Sniff.py is a python script that runs on the Intel NUC. It 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 mysqlclient library. If the database connection is lost, the script automatically reconnects.

Network Logging

Network logging is the largest part of the script with 180 lines of code out of 290. It 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 raw metadata in the form of a hex dump. We preserve the hex dump block because it is a raw, untouched packet. The fields we write to the database are the source IP, source host, destination IP, destination host, time, size, type, protocol, source port, destination port, source host, destination host, and hex dump. We will explain what each field in the database section.

Power Logging

The second addition to our IoT device analysis is the power information. To obtain this information, we rely heavily on the Belkin Wemos. Each Belkin Wemo Insight connects to an outlet and each IoT device plugs into the Wemos.

We used the PyWemo python script to read power usage form the Wemos once per second. However, this is the highest we were able to obtain. 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 as well when pushing this data to our database.

One issue with PyWemo and the Belkin Wemo's was that they would sometimes disconnect and the script would miss out on power data. To solve this issue, the script

rescans for Wemos until it finds all the Wemos. If it does not find the of Wemo's in our testbed (hardcoded value). The code for this is shown below in figure 2.4

```
219 def scan_until_all_found():
220     print("Discovering Wemos")
221     switches = pywemo.discover_devices()
222     print("Discovered {} switches".format(len(switches)))
223     print(switches)
224     try_num = 1
225 
226     while len(switches) < 16:
227         print("Did not discover enough switches, trying again{}...".format(try_num))
228         switches = pywemo.discover_devices()
229         try_num += 1
230         print("Discovered {} switches".format(len(switches)))
231 
232     return switches
```

Figure 2.4: Rescan if all wemos not found.

2.2.3 Database

To store all the result of the sniffed packets and power data, we push the data to a MySQL database hosted on an Amazon AWS server. As of writing this paper, the database currently holds 172,445,929 entries that take up 184.94 GB of space.

Network Table

The largest part of our database is the IP network table. This table contains all the network packets that have gone through the NUC WAP. It is currently 179.1 GB and contains 116830077 entries. It is so large because it consists of constant network packets through at least 12 months of research. In those 12 months, we did many high bandwidth tasks such as playing music or video. These entries also 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.

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

With these columns, some common SQL commands we used for our analysis includes common queries shown in figures 2.6 and 2.7. The most useful commands generally required examining total throughput or device throughput. The SQL query shown in 2.5.

```
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 2.5: 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. The hex dump contains

the information of all the columns in the table and more. This flexibility is useful for other researchers who may have slightly different needs. As one of the original goals is to provide a database that other network researchers can use, the addition of the hex dump strongly hits that point

In the tables, each device can be tracked down by looking for the IP address of that device in the source or destination columns. However, note that during set up, we did not set up static IPs for these devices. Because of this, 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.

Power Table

The power table holds the most important data to this research, and we 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 61240189. The large difference in size between the IP table and power table is because each device only produces one entry power entry every second. The Power table is shown below in figure2.3.

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 database, we generally query for all the most recent power packets or a range of power packets in a given time range. We do this for either all

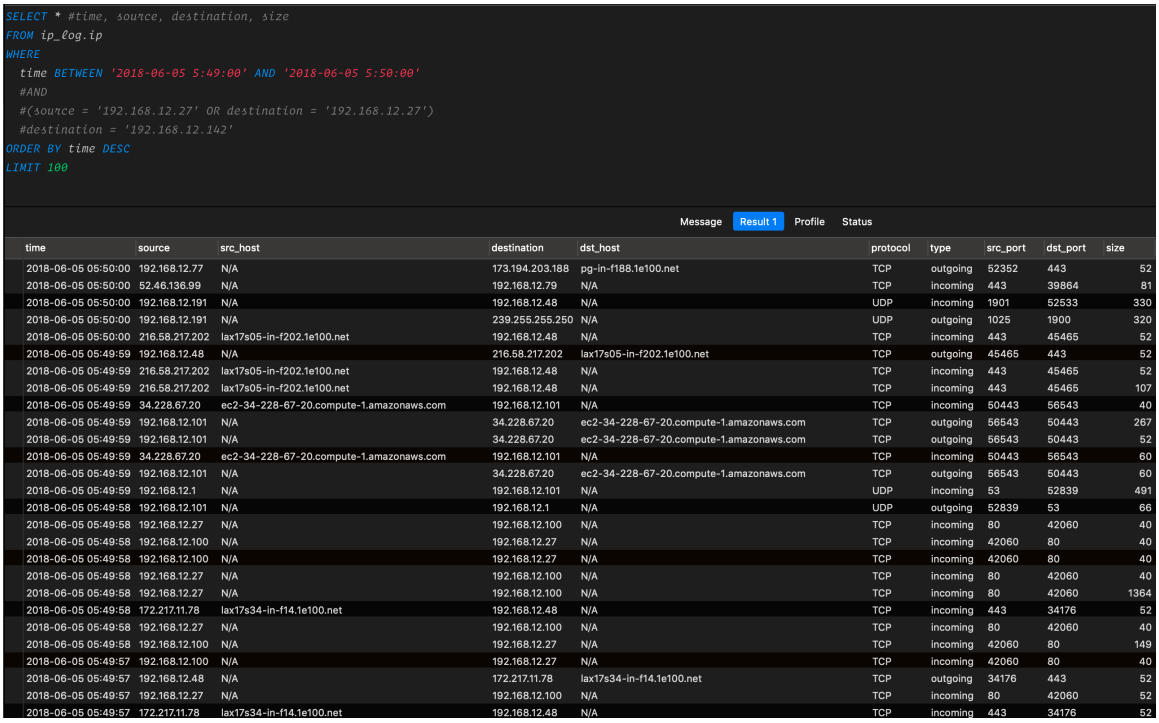
devices, a subset of devices, or a single device. Some of the commands and results are shown below in figures 2.6 and 2.7.

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

Message | Result 1 | Profile | Status

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

Figure 2.6: Power query form database with navicat.



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

Message | Result 1 | Profile | Status

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

Figure 2.7: Network query form database with Navicat.

Within the power table, the name field is extracted from the name of the Wemo, which we manually name through the app. This can cause issues if the wrong device is connected to a Wemo. For example, the Echo Dot could've accidentally have been connected to the Wemo named Google home. 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 in TIME RANGE. 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. A possible solution to this could've been to name the Wemo's by the IPs of the devices they connect to, but still, they are not static.

The sampling frequency for the power table is also low and missing some points. To handle missing points, interpolate points as shown in the 2.2.5 section. There is nothing that can be done about the low sampling rate. This low sampling rate can cause some imprecise data through interpolation, some missing points may not have been a smooth change, drastic changes may have occurred that we may have missed.

2.2.4 Usage Flow

As previously stated, one of the goals of this paper 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 20 weeks of this research. To build a proper dataset that other people could use for research, we set up a list of things that we should do for each device at a minimum when interfacing with it. While doing this, we would log what the activities into a Google Sheets file so that these events could be correlated to entries in the database, giving context when looking back on this data. 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 the Google sheets, we include the start, end time, name of the device(s), action performed, and any individual notes. When naming the devices, we used the same naming as we did for the Wemos in order to maintain consistency and correlations from the log to the database. Because we manually enter entries through a Google sheet. An example of entries into the table is shown below in table2.4.

Table 2.4: 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

In the following subsections, we will explain the script created to automate the event logging portion of the research. Then we will follow up by explaining the specific procedure we ran each group of IoT devices through whenever we interfaced with 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. This can lead to poorly formatted entries. To minimize this as much as possible, we wrote a script to automatically populate the start time, end time, and date when entering entries. The code for the script can be shown below in listing 2.2.

```
1  function onEdit() {
2    var s = SpreadsheetApp.getActiveSpreadsheet().
        getSheetByName("Event Log");
3    var r = s.getActiveCell();
4    var c = r.getColumn();
5
6    if( c == 4 || c == 5) { // checks the column
7      var dateCell = r.offset(0, -3);
8
9      if( dateCell.getValue() === '' ) {// is empty?
10       var startTimeCell = r.offset(0, -2);
11
12       // fill in the start time and date
13       var date = new Date();
14       dateCell.setValue(date);
15       startTimeCell.setValue(date.toLocaleTimeString());
16     }
17   }
18
```

```
19    if( c == 6 ) { // checks that the description is being
          entered
20      // if so fill in the end time
21      var endTimeCell = r.offset(0, -3);
22
23      if ( endTimeCell.getValue() === '' ) {
24        endTimeCell.setValue(new Date().toLocaleTimeString());
25      }
26    }
27  }
```

Listing 2.2: Open and Read from a Socket

Smart Speakers

Within the smart speakers' section, we include the Google Home, Echo Dot, Eufy Genie, and Echo Show. Whenever interfacing with these devices, we made sure to at least query for the weather and the news. The specific phrases we would use would be "¡wakeword¿ what's the weather" and "¡wakeword¿ what's the news". Then we would also make sure to set a reminder for a random task in a random timeframe. Then we would mute the devices for a few minutes. We muted these devices to see if they would still be listening if we said the wake word.

Video Game Consoles

Within the video game consoles section, we include the Xbox One and Nintendo Switch. When interfacing with these devices, we made sure to play a game on the device for at least 15 minutes. Every week, we would also browse for games on the game store and download free demos.

Afterward, we would turn off the Xbox and put the Nintendo Switch into sleep

mode. The Nintendo Switch could not be put to sleep when docked, only when disconnected from the dock.

Media Players

Within the Media Players section, we include the Roku Express, Chromecast, and Amazon Firestick. When interfacing with these devices, we made sure to watch youtube for at least one video or 3 minutes.

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

Tablets

Within the tablets, we include the Samsung Galaxy Tablet and the IPad.

In this experimental setup, we used the tablets less so for investigative purposes, but more so that we can set up and interface with other devices. These devices are less IoT devices and more full-fledged computers. Thus they would have widely varied network patterns that would be difficult to track.

We mainly used these IPads to set up the WeMos and any device that had to be connected to a SmartHub. It was also used for controlling or changing the names of the WeMos through the app if necessary and for setting up the security camera.

However, regardless, 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 under the app (NVAS2) in the IPad for at least 2 minutes. Usage includes streaming video from the camera to

tablet and viewing it. We also controlled the camera through the app through 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 on IPad.

2.2.5 realtimeIoTGrapher.py

In this section, we will talk about further automation when researching Network and Power patterns. When analyzing the power and network for a particular device, we can only see so much when looking straight at the database entries. Even when filtering by device or time, we can only catch text but cannot visualize any patterns. To resolve this, we graphed the data into Google Sheets and graphed it. Seeing a graph made analysis and detection of patterns much easier. The tables and formulas are shown in figure2.8. This process was tedious and time-consuming. We had to copy the data over after making a sequel query, extract unique time columns, then sum the size up at each unique time frame. So we looked to optimize it.

The result was the realtimeIoTgrapher. This python script would automatically form the graphs shown in figure 2.9 when given the time range and the devices. This automation sped up the analysis process and provided extra features which we will explain in section 2.2.5.

This python script leveraged the Plotly library, which, given data, will graph it onto a local web page for viewing. It comes with many useful tools for further analysis and can be extended to run on a public web page for public viewing.

When we release this database to the public, we hope to complement it with this software so that researchers view the power and network information of any/all of the devices at a glance. At which point, we can set up a public website to host this software so that anyone can view the data at any time.

fx =UNIQUE(A:A)

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

Figure 2.8: 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

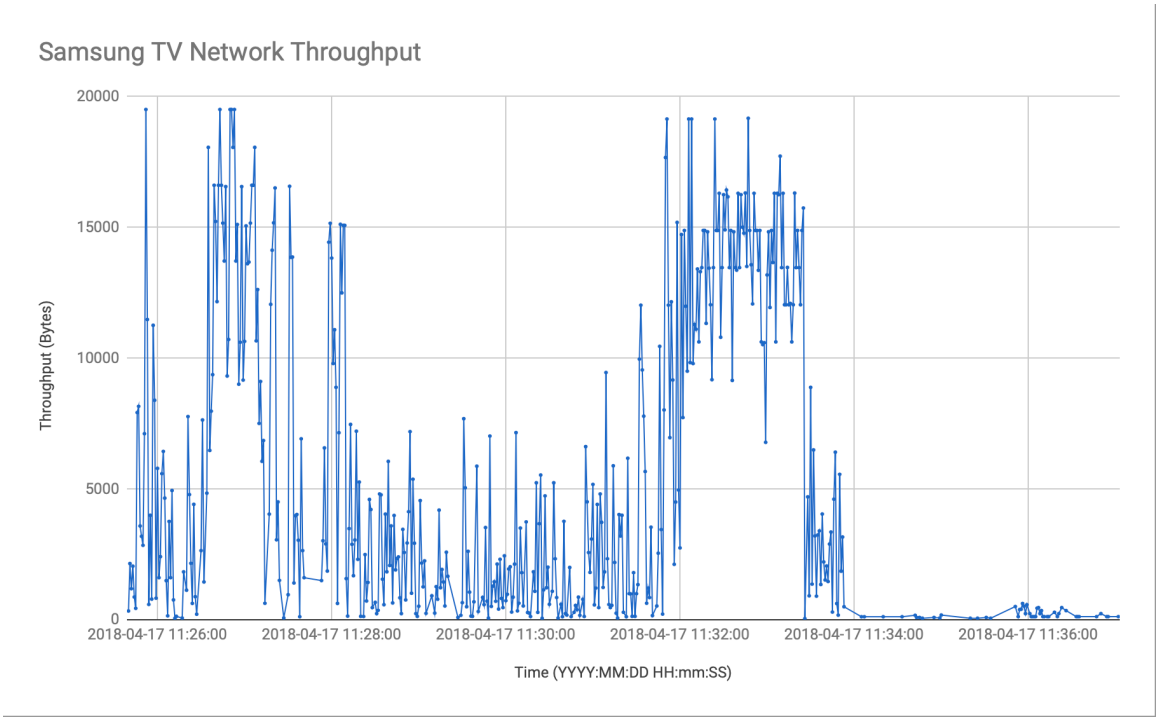


Figure 2.9: Resulting graph of dataset shown in Figure 2.8

Features

In this section, we will discuss the features that this RealTimeIoTGrapher (grapher) provides.

Firstly, at its core, this program automates the creation of the graph shown in figure 2.9. However, now, instead, we can specify a time range that the graph should be in, then we specify what devices we want to graph, and then the program will create the graph. It will show the power, input traffic, output network traffic, and the total network traffic over the time range specified as a line graph. The grapher will do this for all the devices specified.

On top of this core information, the grapher will also annotate the data by including a line denoting the average for each of the traces above over the time range that is currently displayed. It will also annotate the maximum and minimum values in the time range for each trace. Also, instead of a set time range, the grapher can also display this information in real time for an interval amount of 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, Plotly also provides a save feature that allows further editing of the graph so that it is formatted the way the user wants. There are many other features that the Plotly libraries provide, but these are the main ones used in our research.

```
64  def throughput_query_in_range(db_connection, device, start_time, end_time):
65      sql_query = """
66          SELECT
67              time,
68              CASE WHEN type IS NULL THEN 'total' ELSE type END type,
69              SUM(size) AS total_throughput
70          FROM ip_log.ip
71          WHERE
72              time BETWEEN '%s' AND '%s' AND
73              (source = '%s' OR destination = '%s')
74          GROUP BY time, type WITH ROLLUP;
75      """ % (start_time, end_time, ip[device], ip[device])
76
77      dataframe = pd.read_sql_query(sql_query, db_connection)
78
79      return dataframe
```

Figure 2.10: Efficient SQL query to obtain total Network throughput at each second

To reduce local computation, the grapher offloads work to the server through the SQL query. When querying for network data, it will perform a rollup in order to sum up the incoming and outcoming network throughput to obtain the total throughput. This roll-up query can is formatted as shown in 2.10.

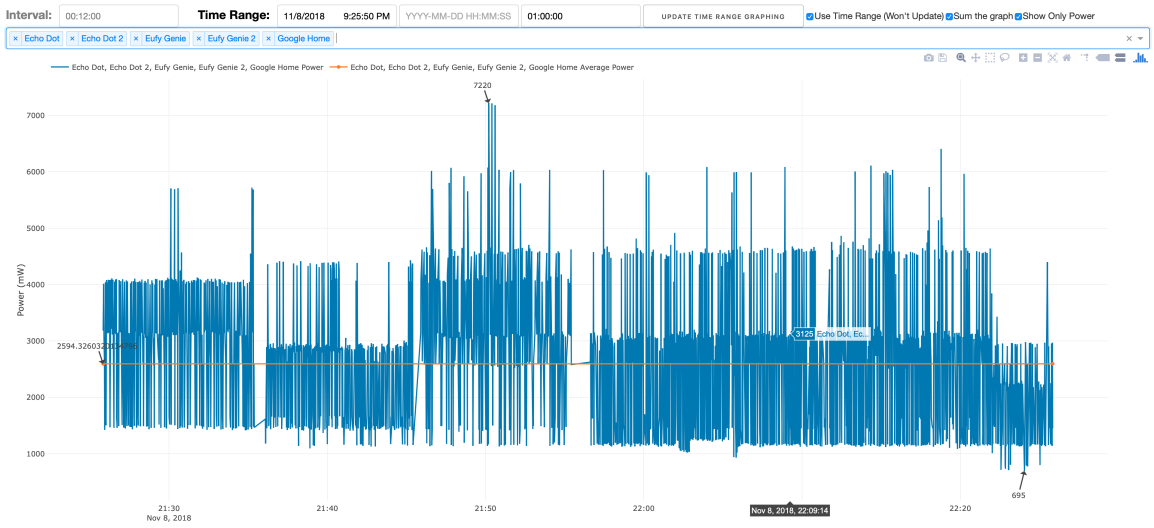


Figure 2.11: Summed up power trace without interpolation.

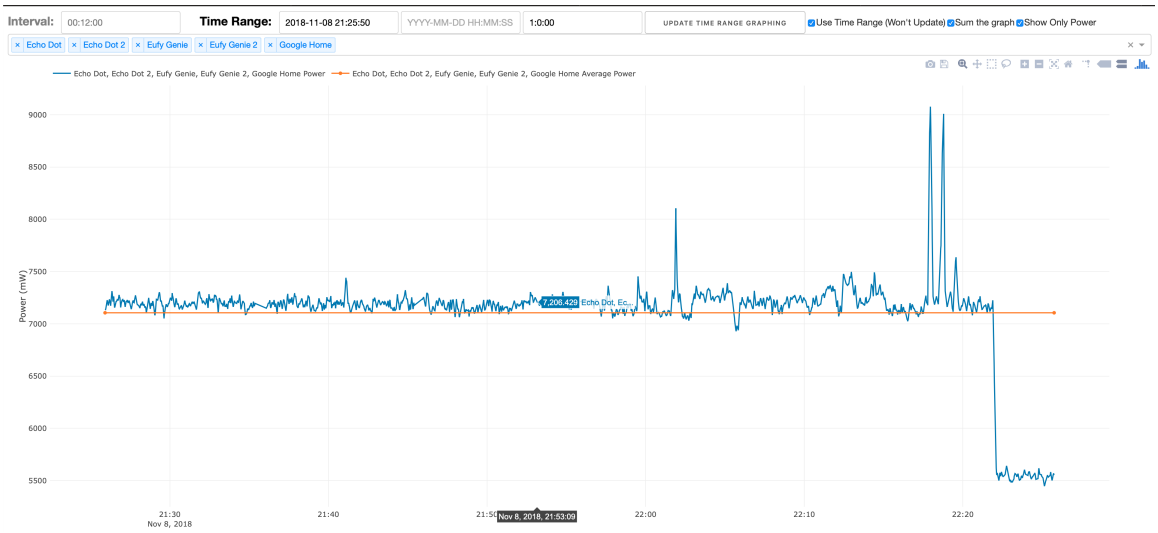


Figure 2.12: Summed up power trace graph with interpolation.

The grapher can also sum up all power traces into one single trace. Originally, when summing the graphs, it would add up the two Pandas data frames that are representing each trace and then graph that resultant trace, shown in figure 2.11. However, the Wemo doesn't always sample at every second. This is an issue if, at some second in time T, some device X has a power point, but the other device Y does not. The summation will count the empty data point in device Y at T as 0 and take device X's data point as the sum. Device Y does not use 0 power at time T but doesn't have a data point for that time. So to solve this, we interpolated data for every second for each device before summing their data frames up, with a builtin Pandas feature. After using interpolation, the same time summed graph looks like figure 2.12. It is now much more accurate.

Why It is Useful

As stated before, this tool is meant to be released as a complementary tool to our database. It has already been used by multiple researchers when interfacing with our database.

The realTimeIoTGrapher will helps analysis of the database at a glance and gives insight into each device in relation to the event log. This tool created most of the

images shown in section 3. Many other researchers have used this tool as well to gain a surface understanding of the database. The grapher provides real-time or historical graphs on any device for individual analysis or comparison with little effort or time. Additional features from Plotly also provides an easy way to format the graph for presentations or papers.

The summation feature can simplify the graph and provide a more holistic view of power usage as a whole, making it easier to spot patterns. In this paper, the feature is also 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 in a constrained time frame, there are some edge cases that it fails to handle.

Every once in a while, since so many people are reading from the database, the database will have too many connections, at this point, the software will refuse to connect, and the grapher will not work.

The graphing tool also gets much slower when the time frame requested is much larger. We have noticed a slow down for queries in time frames longer than 7 hours. In the static graphing mode, it will slow down before graphing the information. However, in real time graphing mode, the tool will make many slow queries, eventually causing too many connections to the database and undefined behavior.

Because the IP addresses of the devices connected to the database are not static, they will change every once in a while. The Wemo names have also been changed a few times throughout this research for clarity purposes. 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 these fields change, the graphing tool will not be able to find the device and will not graph the devices information. In some cases, this situation is handled, and the graphing tool will ignore them. However, in some cases, the graphing tool will crash and require restarting. To solve this, manually change the IP addresses defined in the lookup table in line 23-39 shown in figure2.13 below and increment/decrement the value of the IP

a few times. In our experience, we at most had to change the IP up 3, but never more than that. This is just a guess and check method.

However, to figure out the new IP address for the table, we have to perform a SQL query in the time frame that a particular device is doing something, and manually examine to see which IP has the highest network throughput. With this method, we had to ensure only one device was in use so that an IP address for another device would not be confused for the current device. To change the corresponding name, check the device inventory excel sheet for a possibly updated naming scheme.

```
#region IP Addresses of all devices we are tracking
ip = {
    'Chromecast':       '192.168.12.77',
    'Echo Dot 2':       '10.42.0.132',
    'Echo Dot':         '10.42.0.150',
    'Eufy Genie':       '10.42.0.223',
    'Eufy Genie 2':     '10.42.0.172',
    'Fire Stick':       '192.168.12.113',
    'Google Home':      '10.42.0.236',
    'IP Camera':        '192.168.12.58',
    'Nintendo Switch':  '192.168.12.160',
    'Roku':             '192.168.12.69',
    'Samsung Hub':      '192.168.12.100',
    'Samsung TV':       '192.168.12.191',
    'Smart Light':      '192.168.12.27',
    'Xbox':             '192.168.12.251',
    'Echo Show':        '192.168.12.122',
    'Appliance':        '192.168.12.122',
    'Appliance1':        '192.168.12.122',
}
```

Figure 2.13: IP lookup table in real time iot grapher

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' will start the backend and display its current operation onto the terminal.

The link to the local IP address will contain the graphing tool. Clicking on that will open it onto the default browser for the computer.

Usage Directions

Once the software is running, the grapher can be viewed on a web browser as shown in figure 2.14. This section will describe each numbered element in the image.

Figure 2.14: 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 will be 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 will display 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 will be dark instead of grey when it is in static graph mode. In this state, the graphing tool will specify the network traffic and power information in the time window specified.

4. In these three text boxes, the time window for static graphing will be specified. The first box will be the start time, the second box will be the end time, and the last box will be the interval. To specify the time range, fill in any 2 of the three text boxes, and the graphing tool will infer the other number. If all three text boxes are filled,

then the graphing tool will use the start and end time text box. To further explain, if the start time and interval are specified, then the window will be start time → start time + interval. If the end time and interval are specified, then the time window will be 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 will update the graph with the new time range.

6. The "Use Time Range" checkbox will toggle the software between live graph mode and static mode. The "Sum Graph" checkbox will sum up all power graphs into one. The "Show Only Power" checkbox will allow 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 will create 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 will display the number of bytes sent through the network by a device going in, out, and in total. It will also show the power usage over time. Also, it will show 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 will display an individual bar graph for each device and protocol. Similarly, the traces can be clicked on to hide its visibility.

Chapter 3

RESULTS

In this section, we will go over the data set we have built from our IoT test bed and present graphs from that data set. This section will start with the effects of changing the smart plug that the device connects to. Then the paper will cover the protocol spread over the whole database. The next two subsections will cover smart speakers and streaming devices specifically. Then the paper will sum power graphs and conclude with a comparison between smart speakers.

Sections 3.2, 3.3, and 3.4 have already been covered in Ryan Frawley's paper[8] so those results will be paraphrased. His paper will have a more in-depth analysis in those sections and include GeoIP information that will highlight the spread of locations the database and each device communicates with the most.

When obtaining data, most of the packets are encrypted, so the data we present will focus on metadata such as size, protocol, source IP, dest IP. For power analysis, we only use the power used at each second.

3.1 Swapping the Power Switch

We wanted to check if the smart switches were consistent between devices. To do so, we swapped various devices with various power switches and checked to see if the idle power was still the same. After doing so multiple times with various devices, we found no change when swapping the switches. One example is shown in figure 3.1 where we plugged the Echo Dot 1 into the Eufy Genie 1's corresponding Wemo smart switch and vice versa. After starting restarting up from power on, the Eufy Genie's power reads the same as the Echo Dot had previously read and vice versa.

Figure 3.1: Switching the Echo Dot 1 and Eufy Genie 1's smart switch

3.2 Whole Database

When examining the whole database, most of the traffic goes through TCP so that communication can be more reliable. Then UDP comes next as it is beneficial for streaming and communication services, then finally IGMP and everything else. The spread of protocols can be shown in figure 3.2.

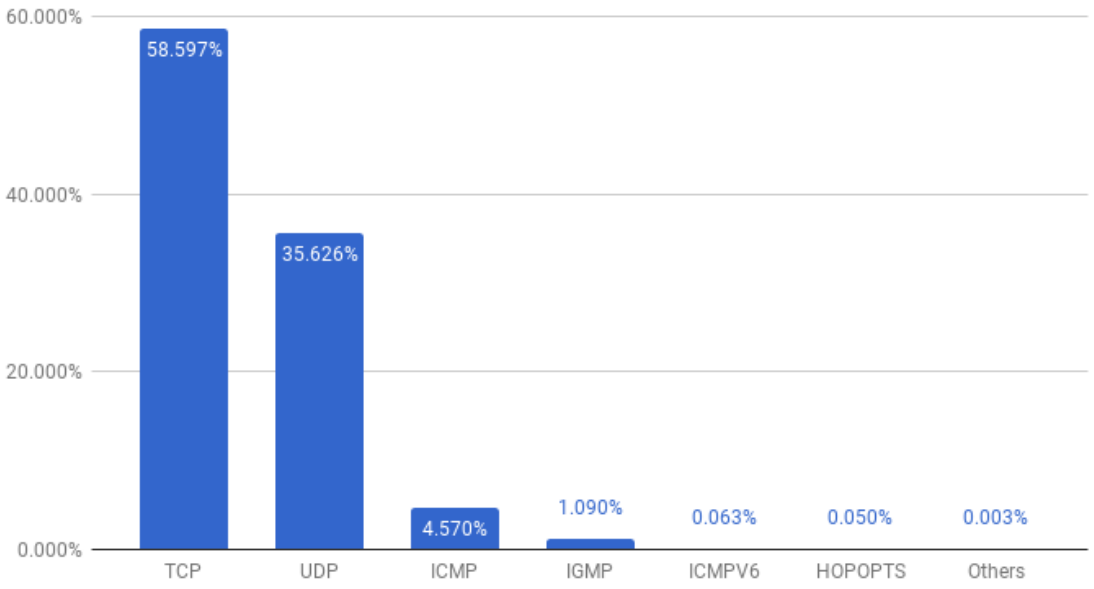


Figure 3.2: Internet and Transport Layer Protocols in Database

3.3 Smart Speakers

3.3.1 Google Home Mini

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, streaming devices, and the Chromebook would 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. The Google Home Mini's idle traffic is shown in figure 3.3. The high peaks are the encrypted TCP packets, and the smaller spikes are the SSDP/UPnP packets.

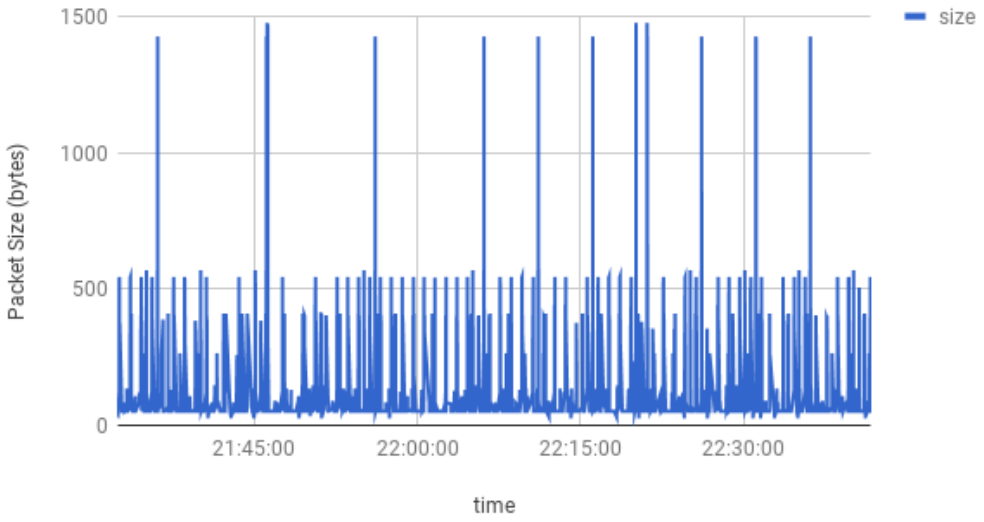


Figure 3.3: Idle Traffic of Google Home Over 1 Hour Period

A graph of the Google Home Mini is shown in the figure 3.4. 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.

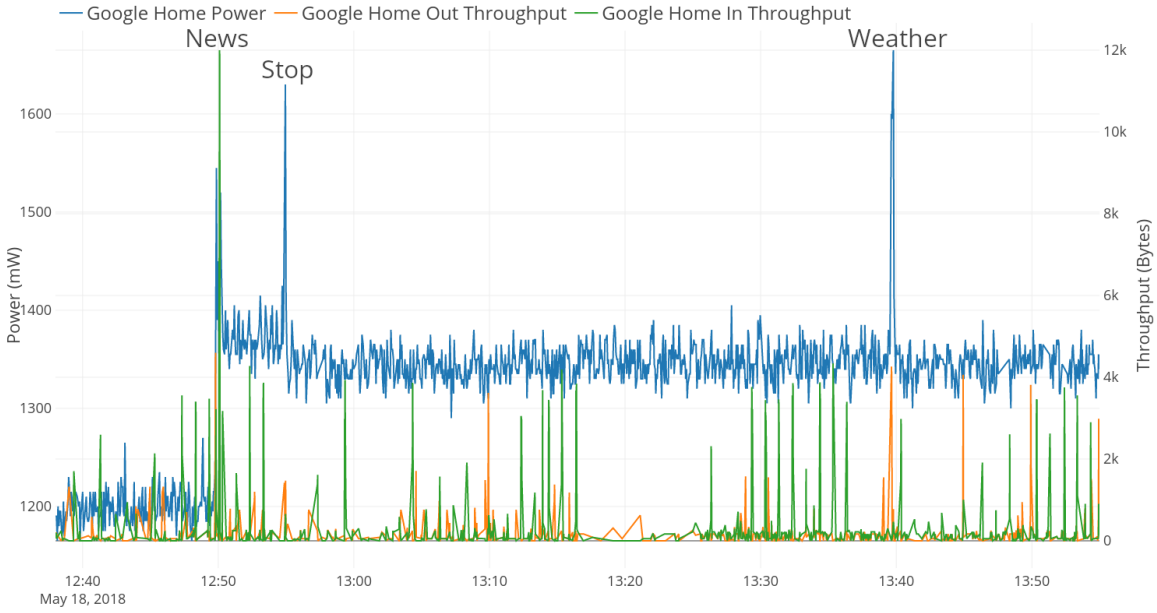


Figure 3.4: Home Mini Response to News and Weather

3.3.2 Echo Dot

One interesting finding from the Echo Dot is that it has a light monitor. When we turn on the lights, the LEDs on the Echo Dot brightens to adapt. The brightness change causes the Echo Dot to use more idle power as shown in figure 3.5.

Figure 3.5: Echo Dot Response to Lights

3.4 Streaming Devices

Each subsection will show the streaming devices usage during startup and when streaming.

3.4.1 Google Chrome Cast

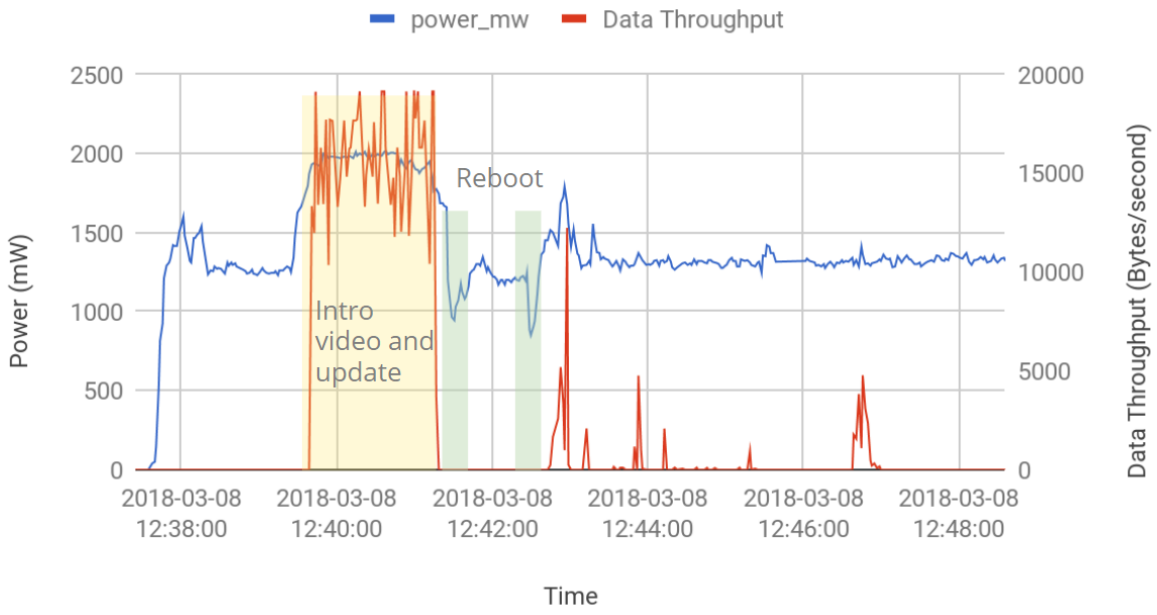


Figure 3.6: Chromecast First Time Boot Network Traffic and Power Consumption

The Chrome Cast startup graph is shown in figure 3.6. On startup, the Chrome Cast shows an intro tutorial video and downloads a firmware update. After rebooting twice, it is ready to go.

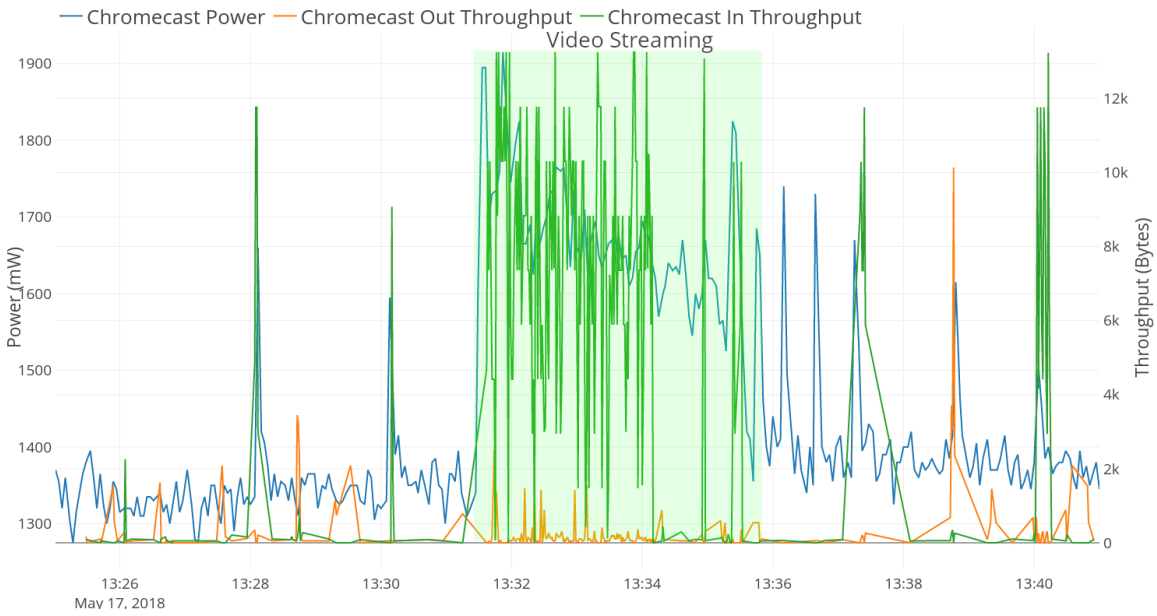


Figure 3.7: Chromecast Video Streaming

The Chrome Cast streaming graph is shown in figure 3.7. In this time frame, the chrome cast streams video in the marked box.

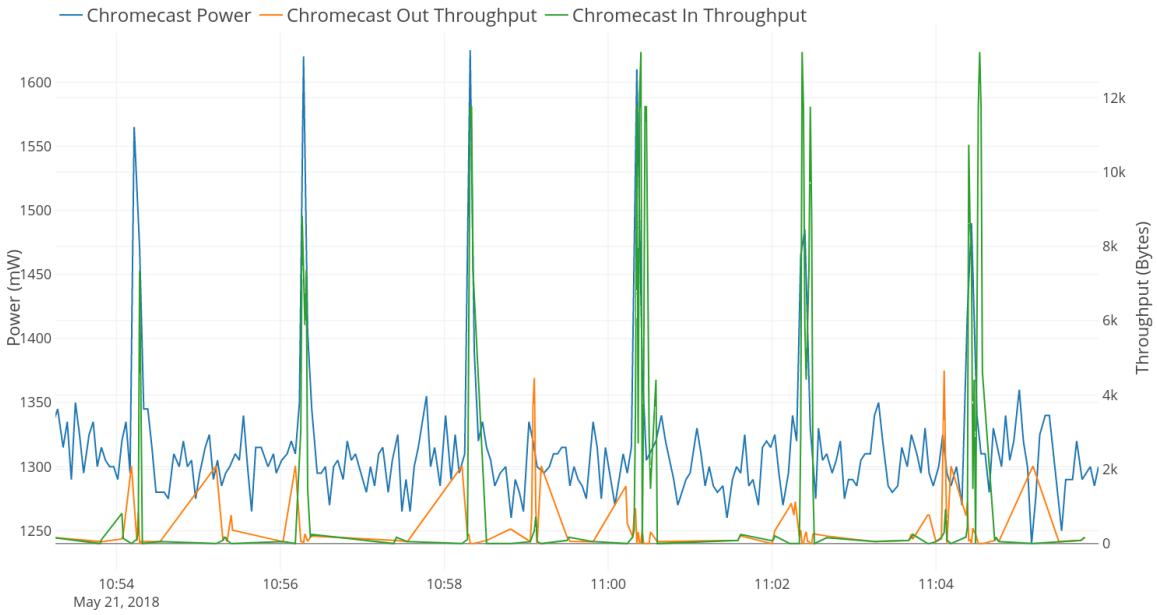


Figure 3.8: Chromecast Idle Traffic

Additionally, the Chrome Cast idle graph is shown in figure 3.8. 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.

3.4.2 Amazon Fire Stick

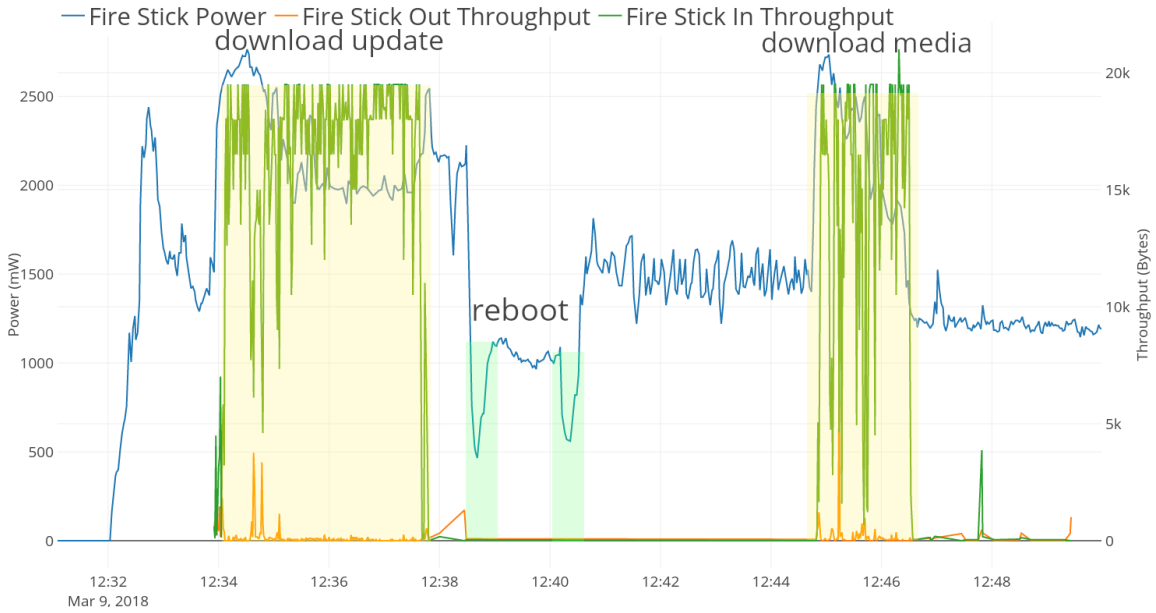


Figure 3.9: Fire TV Stick First Time Boot Network Traffic and Power Consumption

The Amazon Fire Stick startup graph is shown in figure 3.9. 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 3.10.

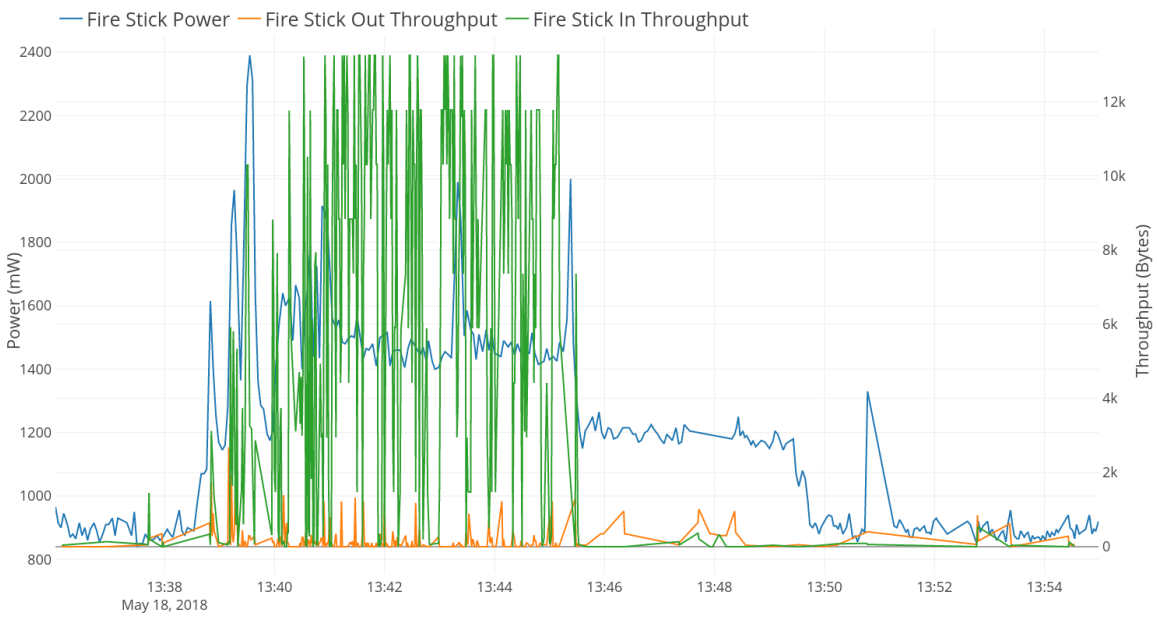


Figure 3.10: Fire TV Stick Video Streaming

3.4.3 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.

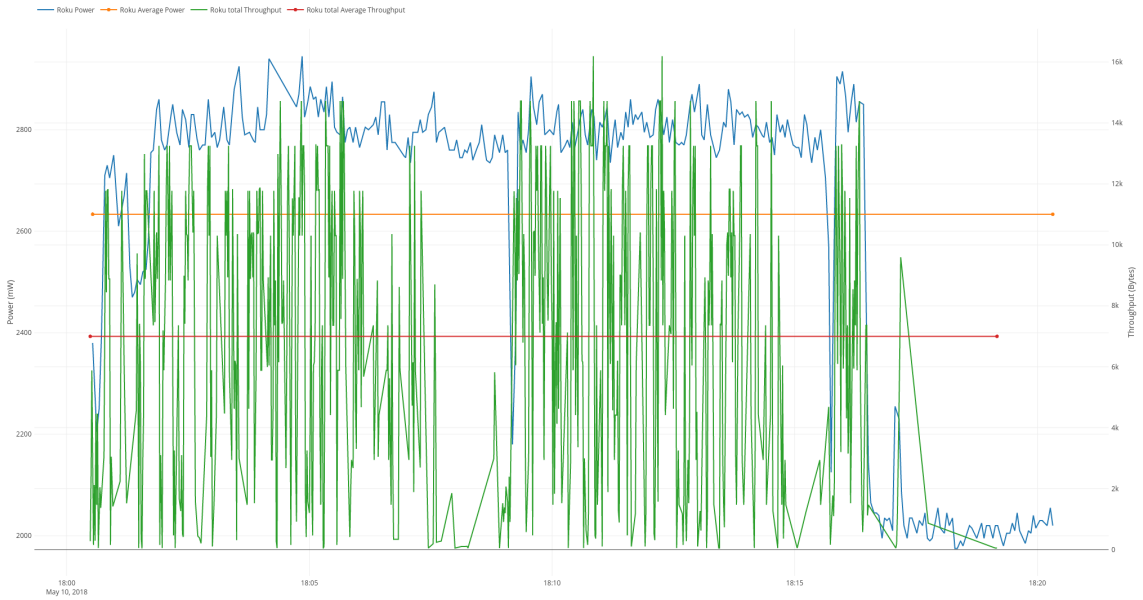


Figure 3.11: Roku Express Video Streaming

3.5 Summed Power Graph

In this section, we will show graphs of the five smart speakers summed up as one graph of power usage. The devices used includes two echo dots, 2 Eufy genies, and 1 Google chrome cast.

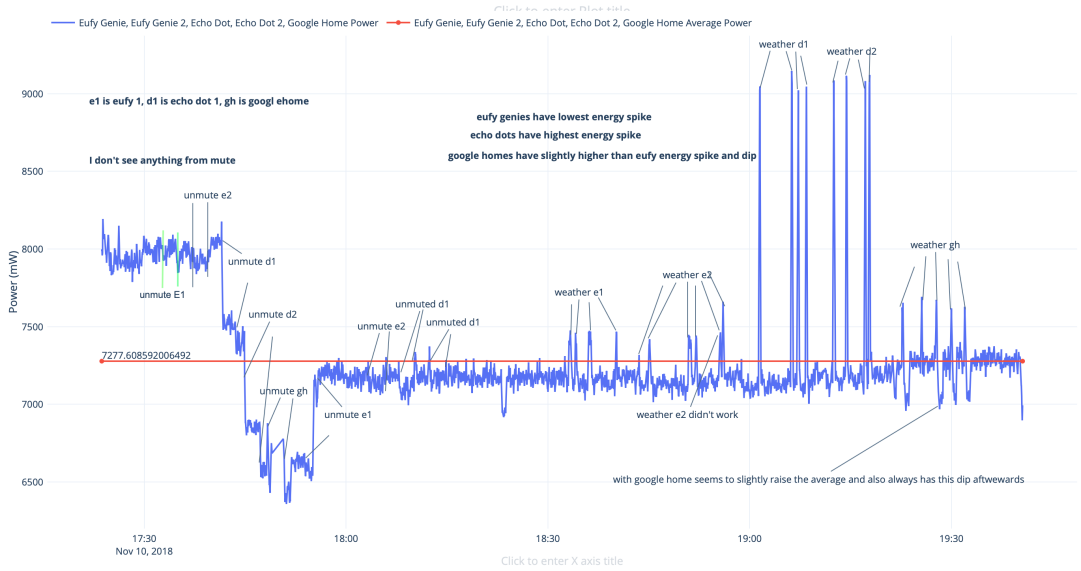


Figure 3.12: 5 Smart Speakers Power Summed Up. Toggled the mute button and queried each device for the weather.

The first graph 3.12 is shown in a time frame where each device is unmuted and muted for a period. We wanted to see if muting and unmuting the devices changed anything. In this graph, there are changes, but they do not correlate to the muting/unmuting. We did this a few more times and found that most of the time, the power usage stayed the same. Then we queried each smart speaker for the weather while all other smart speakers were muted at the 18:30 mark. We queried each device in order 3-4 times in consecutive order so that it is easy to see and to check for consistent results. The Eufy had the smallest spike for this command at 400 mW, then the google home at 600 mW, and finally the echo dot at 2000 mW.

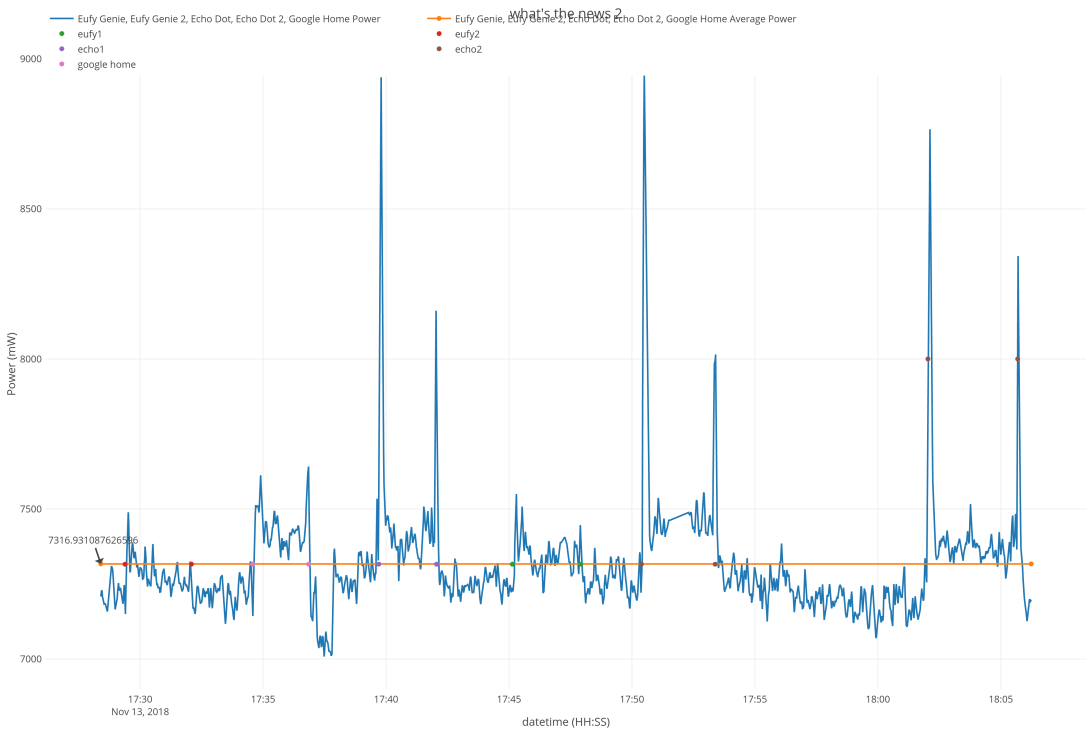


Figure 3.13: 5 Smart Speakers Power Summed Up. Queried each device for the news.

The next graph 3.13 shows the smart speakers when asked for the news. We asked each device for the news at random as to prevent any optimizations. This graph and the rest of the summed graphs in this section will use the same notation for signifying commands. 2 corresponding dots of the same color will signify the start and end of the command for a specific device. If the dots are on another level, then this command has been repeated another time.

In this graph, 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.

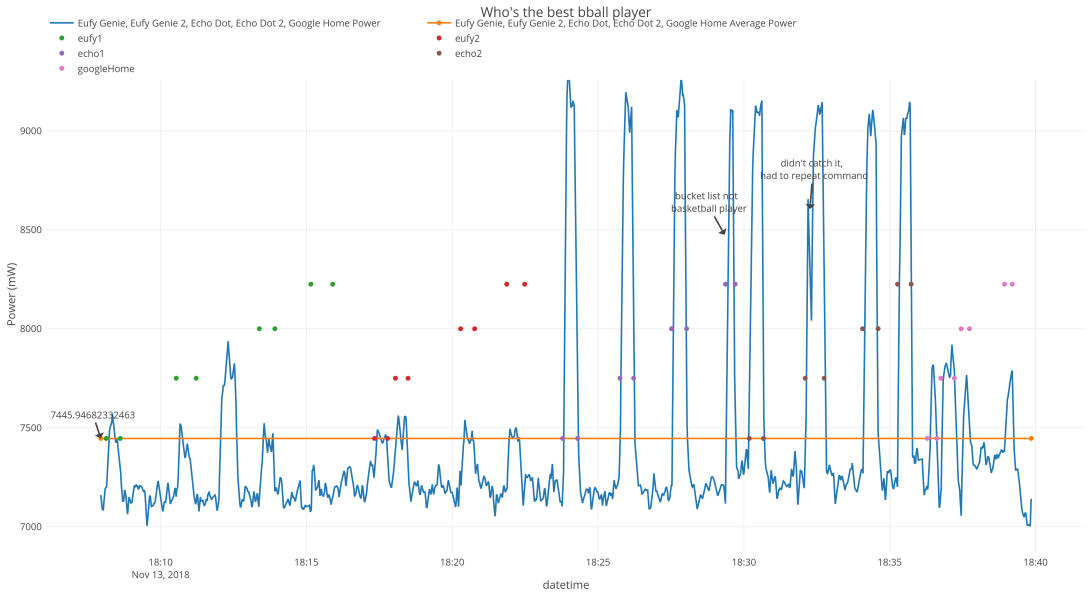


Figure 3.14: 5 Smart Speakers Power Summed Up. Queried each device for the best basketball player.

The next graph 3.14 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. Similar to 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.

However, when looking at graph 3.14, there is a power spike that is unaccounted for in correspondence to the event log. 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.

To figure what the power spike at 18:12 is, we separated the graphs into individual power traces as shown in figure 3.15. From the graph, the power spike at 18:12 is attributed to the Echo Dot 2. We then looked at the individual network usage for each of these devices in this time frame as shown in figure 3.16. At 18:12, there is no significant network usage. We had no conclusive evidence to decide what the power spike is accounted to, but we will speculate what we think it was in section 4.

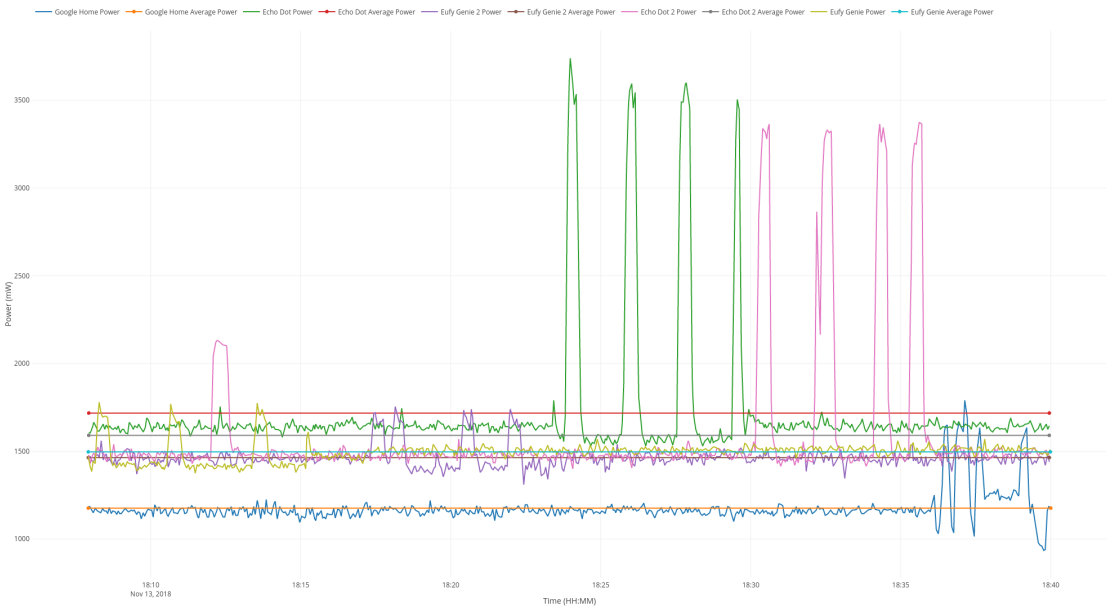


Figure 3.15: 5 Smart Speakers Power Usage over time. Queried each device for the best basketball player.

Figure 3.16: 5 Smart Speakers Network throughput over time. Queried each device for the best basketball player.

The power spike information for each query type ("what's the weather" "what's the news", "who's the best basketball player") are shown below in figure 3.17. This bar graph also includes the average spike height for each device over these three 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, the Echo Dot with a 2026.7 mW spike with 149.89 mW standard deviation.

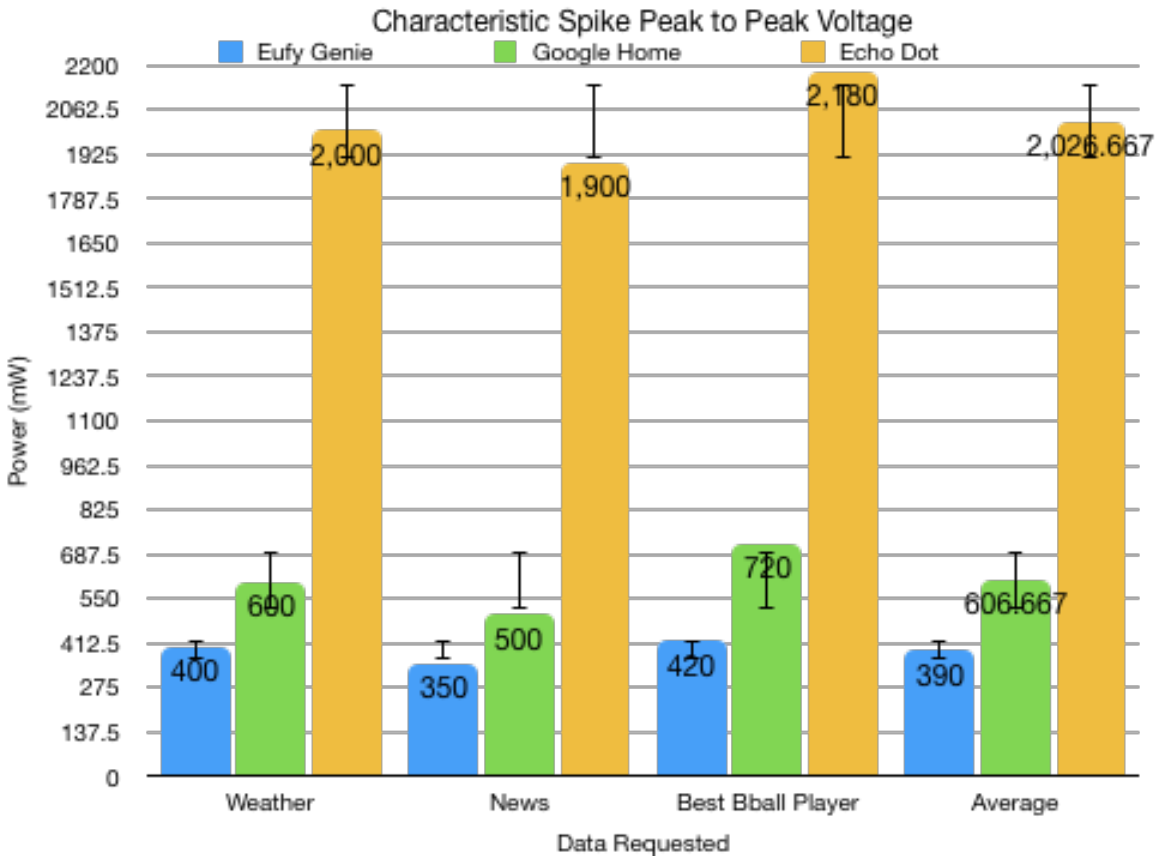


Figure 3.17: Spike summary of for each query.

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 will be three traces. The first trace (blue trace) will be the power usage over time for a high power device. We will swap this trace out with various devices. The second trace (green trace) is the summed power usage of our five

smart speakers (2 Echo Dots, 2 Eufy Genies, 1 Google Home). For this second trace, we used the same power usage graph as figure 3.14 for every graph in the subsections below. Using the same graph provides our control for each smart speakers power spike. The third trace (yellow trace) is the sum of both trace 1 and 2. It is the power usage of the smart speakers in the green trace with the noise from the device in the blue trace.

In all these graphs there will be two y-axes which will represent the power (mW) used by a device at that time. The graphs have two y-axes so that the traces can be more zoomed in. Usually, one trace is significantly larger than the other, so if they are on the same scale, the other gets compressed. The trace for the high power device (blue) will always in reference to the first y-axis on the left side. The trace for the smart speakers (green) will always in reference to the second y-axis on the right side. The sum of the two traces (yellow trace) will about the axis that it is closer into magnitude. The graph will state which traces are using the axis in the axis title. Also note that to further zoom in on the graphs, the y-axes will not always start at 0.

The x-axes in these graphs are also not actual time stamps. The x-axis is the time elapsed (s) from when the green smart speaker trace (green) 3.14 began. Because we had already recorded the smart speaker trace (green), we recorded each high power appliance trace (blue) individually and included it on the same time range.

Below, we will display the smart speaker trace with a PC (Intel NUC), fan, refrigerator, or microwave. We do this while these devices are idle and in use.

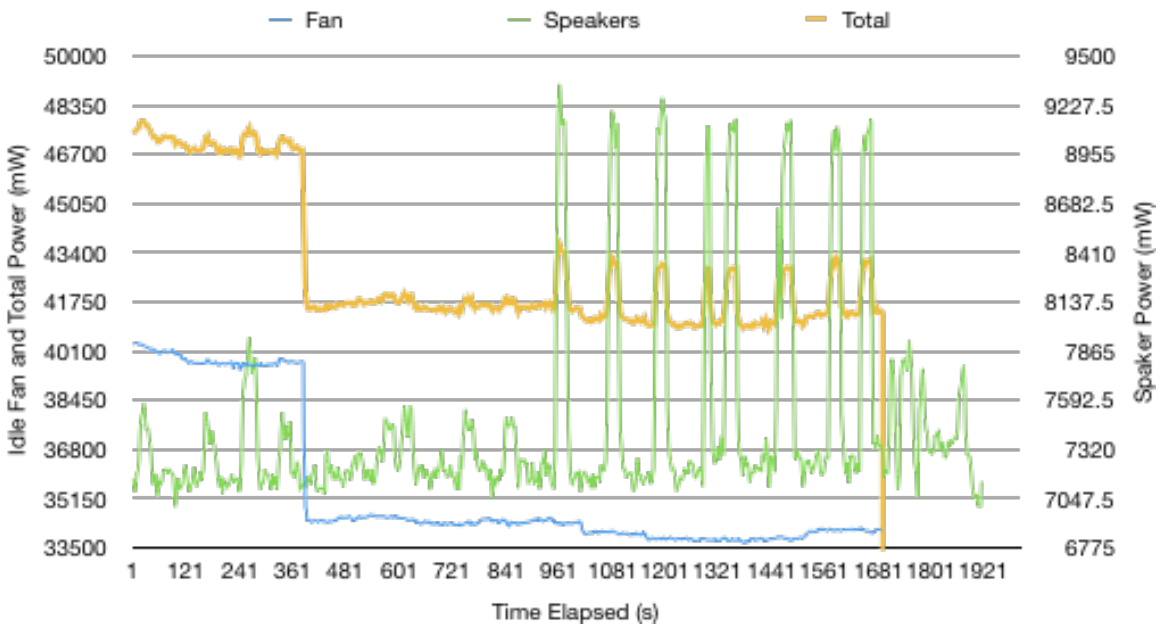


Figure 3.18: Idle Fan with figure 3.14 trace.

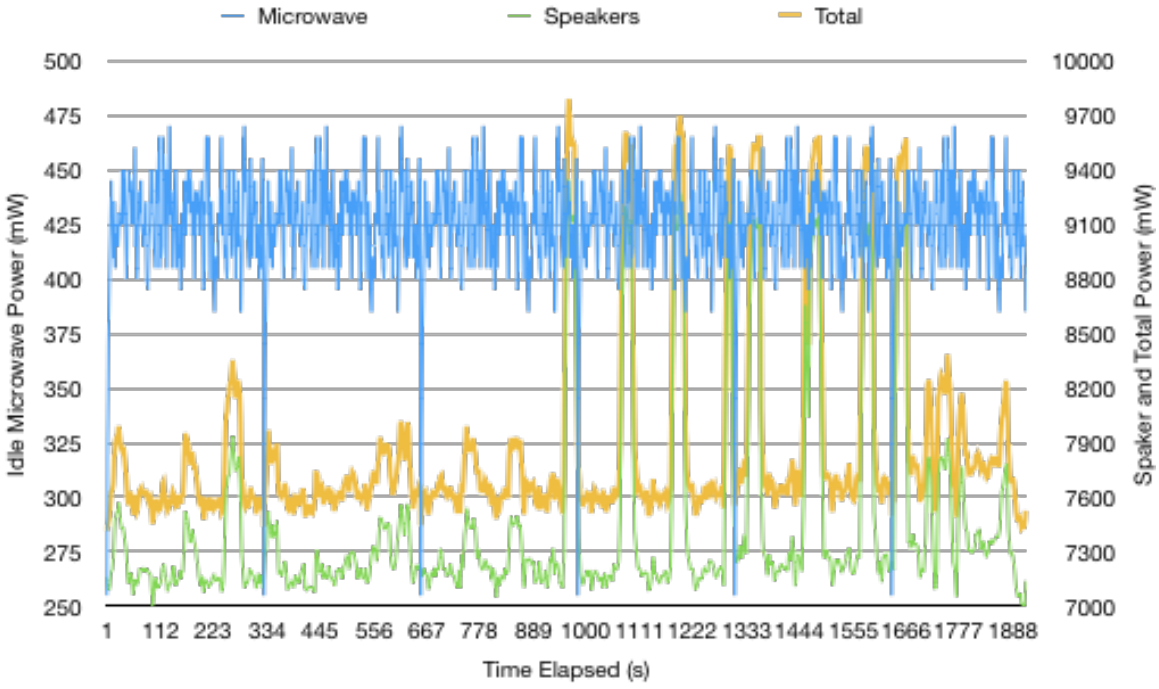


Figure 3.19: Idle microwave with figure 3.14 trace.

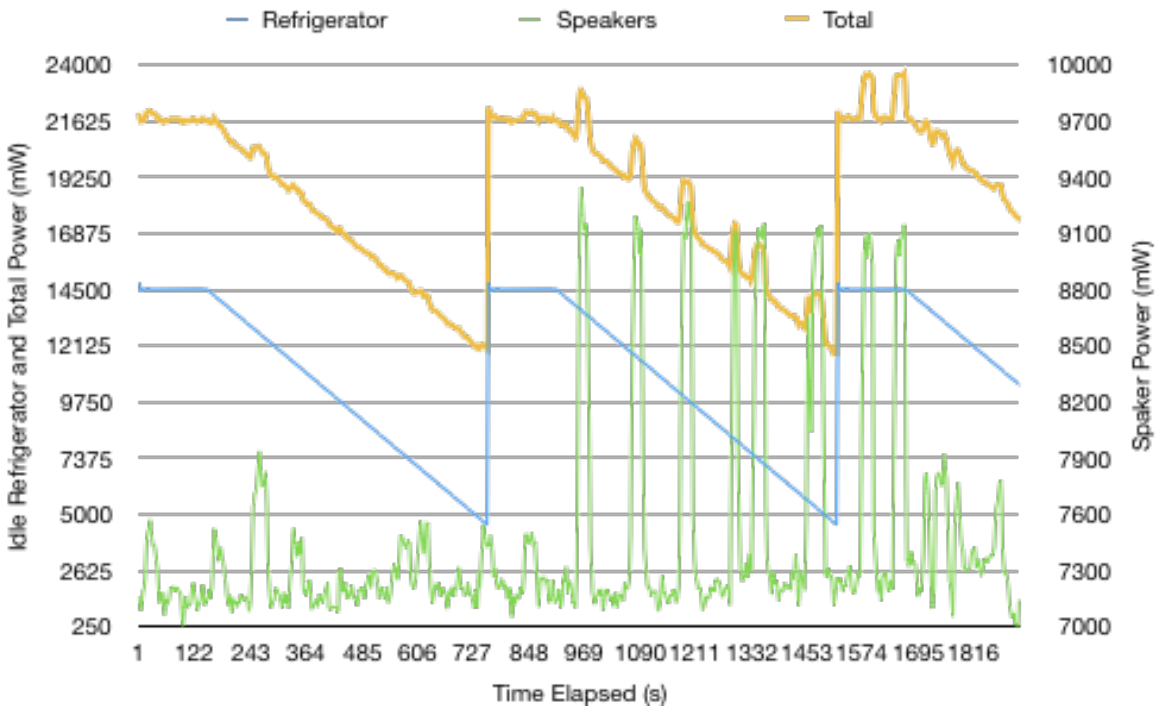


Figure 3.20: Idle refrigerator with figure 3.14 trace.

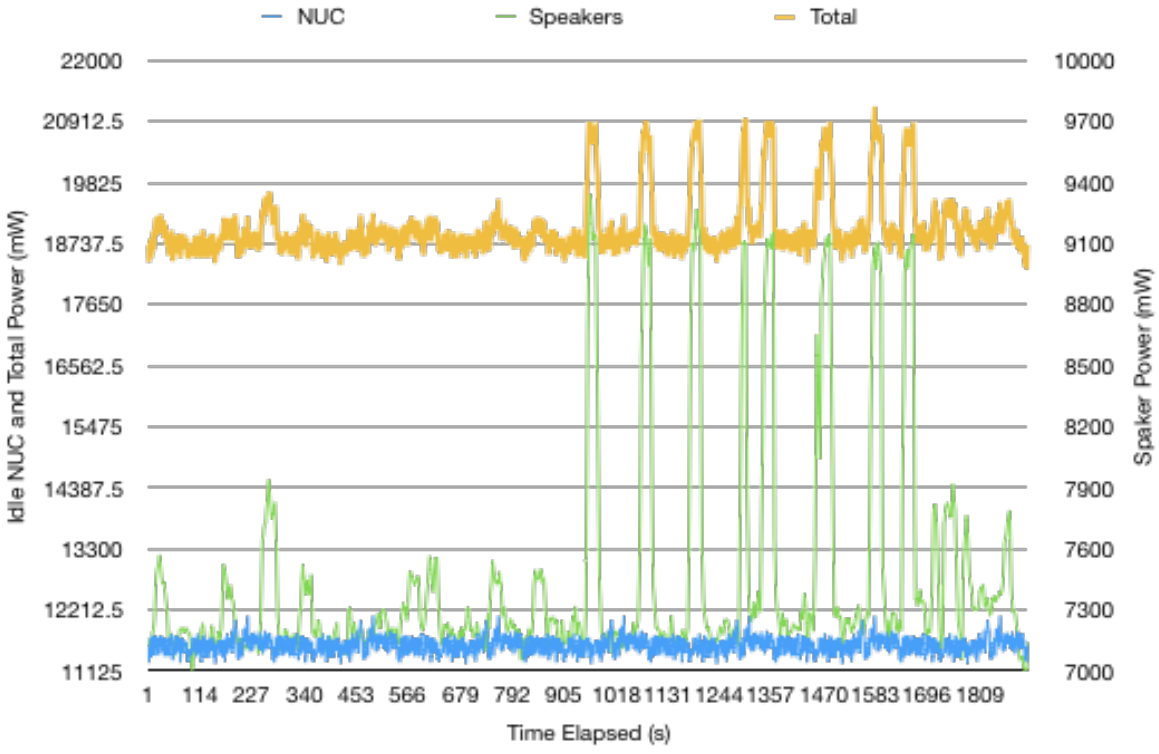


Figure 3.21: Idle PC (Intel NUC) with figure 3.14 trace.

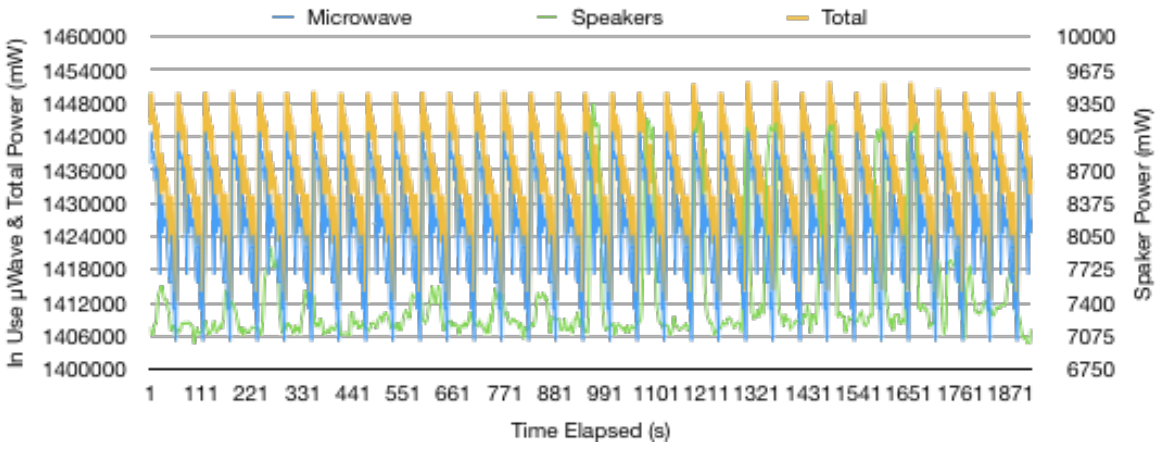


Figure 3.22: In use Microwave with figure 3.14 trace.

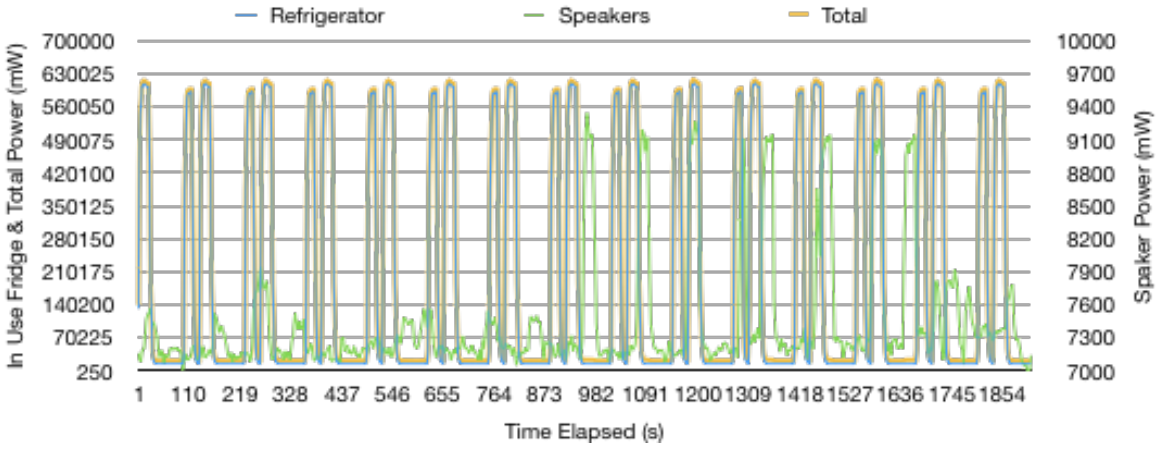


Figure 3.23: Fridge in the middle of cooling with figure 3.14 trace.

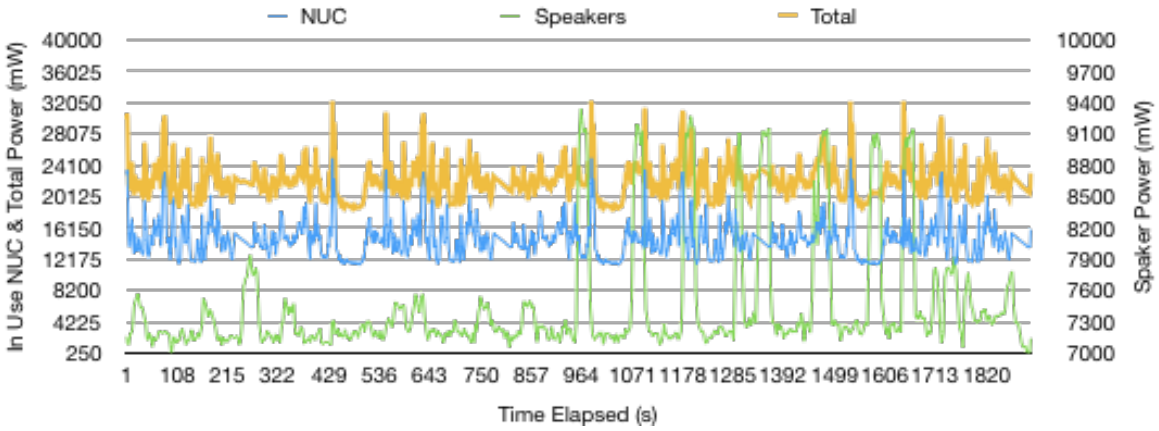


Figure 3.24: PC in use with figure 3.14 trace.

3.7 Smart Speaker Comparison

From the figures in section 3.5, the Echo Dot uses a lot more energy than the other smart speakers. Because of that, 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 graphs below, we display 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 3.15 for the two graphs below. We removed the second Echo Dot and Eufy Genie for simplicity.

In the same time frame as the two graphs below, we also recorded the average power usage and throughput for the smart speakers. For average power, from greatest to least, we have the Echo Dot (1650 mW), Eufy Genie (1325 mW), Google Home (1175 mW). For average throughput, from greatest to least, we have the Eufy Genie (1743 bytes), Google Home (800 bytes), Echo Dot (325 bytes).

Figure 3.25: 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 3.15 with Echo DOt 2 and Eufy Genie removed.

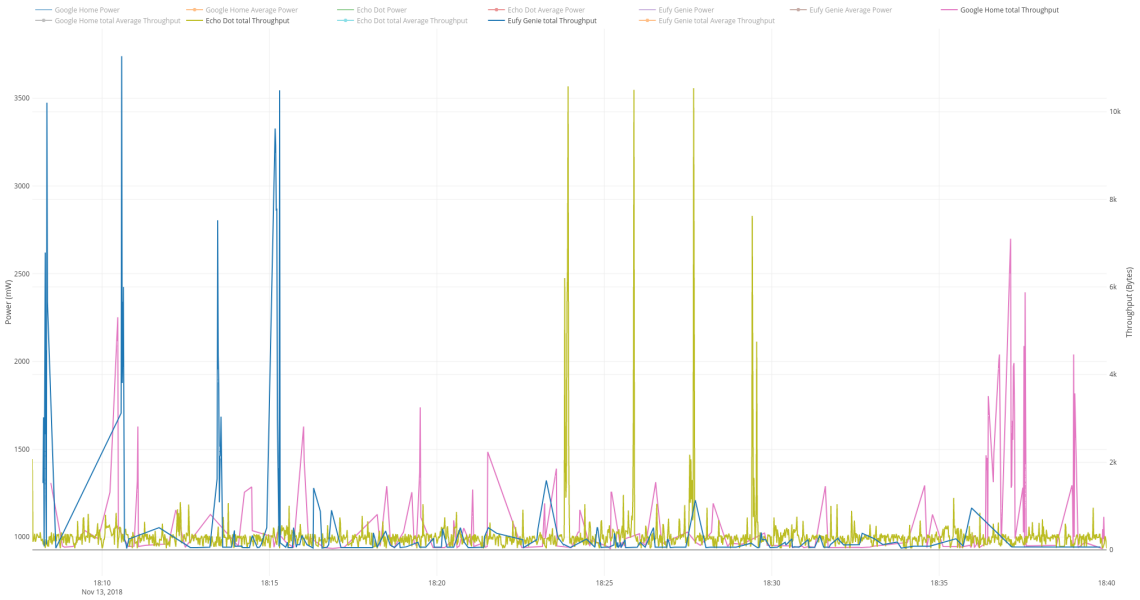


Figure 3.26: Power usage of Echo Dot 1, Eufy Genie 1, and Google Home over time when asked "who is the best basketball player". Same time frame as figure 3.25

Chapter 4

DISCUSSION

In this chapter, we will discuss the implications of the data presented in chapter 3.

4.1 Switching Smart Plug

In section 3.1, we displayed figure 3.1. We do this to ensure consistency between measurements. The network traffic is assumed to be consistent because all network traffic will go through our NUC AP, following a standard WiFi connection. However, each device has their power plug in different locations on different power strips. We wanted to check to see that this variation causes minimal recording differences. From the figure 3.1, we can see that after restarting, the power level goes back to where it was before. We did this multiple times with other switches to different locations and obtained nominal changes in the idle power usage. Thus we can conclude that this power logging scheme is relatively consistent.

4.2 Metadata Analysis

In sections 3.2, 3.3 and, 3.4 we go over the protocol spread, and general power/network usage of varying devices. However, because Frawley's paper [8] already analyzes these sections, refer to his paper for analysis.

4.3 Power Spike at 18:12

In figure 3.14, we saw an unaccounted power spike at 18:12. When looking at figures 3.15 and 3.16, we believe that the power surge is from the LEDs. Because there is no network usage in this time, we do not think 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.

4.4 Device Fingerprinting

We want to see if it is possible to determine what devices are in use if given access to home's power line. Classifying devices from a single power usage graph over time consisting of multiple devices is a good simulation for a total powerline.

In sections 3.6 and 3.5. We show how the smart speakers perform under various operations such as "what's the weather", "who is the best basketball player", or "what's the news". We did this various times to confirm consistency, and we always found that there is a start spike (voltage spike peak to peak value) of around 390 mW with 36.1 mW standard deviation for the Eufy Genie, 606.7 mW spike with 110 mW standard deviation for the Google Home, and 2026.7 mW spike with 149.89 mW standard deviation for the Echo Dot as shown in figure 3.17. Whether summing the power usage graphs for the devices up or keeping separate, from the graph, we can tell what device model is being used if it is within those thresholds. The models with multiple units are consistent with each other. Ordering the queries results in the same power spike. Because of all of this, we conclude that we can classify a device model from just a visual examination of its power usage. This is our first step to testing our hypotheses that it is possible to see what devices are in use from analysis of someone's power line.

The next step would be to add noise into our graphs, to see if the smart speaker spikes from commands would still be legible 3.6. From the graphs shown, if the power is stable enough, then it will not be a problem for classifying the devices. The device spikes will just be shifted up. This statement is supported by our graphs with the fan, the idle microwave, idle fridge, and idle NUC.

However, if the power usage has noise on the same magnitude or larger than the smart speaker spikes, then it becomes difficult to make out which smart speaker model is currently in use. This is evident when the NUC, fridge, and microwave are in use. However, the microwave and fridge are idle for most the time, so it would still be

possible to monitor someone's power line on a time frame that these devices are not being used. However, PCs, smartphones, and streaming devices are constantly in use. So it would be difficult to find a time one the NUC is not in use and make it difficult to determine what smart speaker model is in use form just visual analysis of its power usage. We think that it still might be possible with machine learning, or if network data is added in as well, or both.

From the results of section 3.5 and 3.6, it is possible to classify the model of a smart speaker from visual examination of a single power usage graph during a time when most other devices are idle. However, with multi-use devices such as a PC, smartphone, or IoT device that is in constant use for long periods of times, it would be challenging to classify a smart speaker model.

4.5 Smart Speaker Comparison

Finally, we noticed that the Echo Dot is characterized by a power spike that is a lot larger than the other smart speakers and was interested to see what trade-offs these devices made.

From the results in section 3.7, 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, than then Google Home, then the Echo Dot. We believe the Eufy is constantly communicating through the network so that it can cache results, minimizing onboard operations. The Echo Dot, on the other hand, uses the least network throughput and opts to do as much on board work as possible. The Google Home sits in the middle, querying the network, for more relevant things that it can cache for when the user asks the questions. It seems they pre-request results better. It is interesting that the Google Home still sits in the middle for network throughput when it is the only one that sends UPnP at such a high frequency. However, this is mostly speculation and would be interesting to see more research on this.

Chapter 5

CONCLUSION

To conclude, we have shown that it is easy to tell which smart speaker is in use based on just the initial spike. Devices are each making their unique trade-off between power and network throughout when idle and in use. This tradeoff leads to unique signatures that allow us to classify devices. For the smart speakers, it is as obvious as the first power spike during usage.

Some patterns can be difficult to make out with so much noise coming from within a house for power data. However, if enough devices are idle, it is still possible. It shows an indirect security flaw in our power-line and how that defense should be fortified. This paper showed that we could guess 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.

This proves that more research should be done to analyze and classify IoT devices. We have created an expansive database of network and power data for 16 IoT devices that spread over ten months, accumulating 184.94 GB of data and 172,445,929 entries in the database. Along with that, we have created an analysis tool to examine the database visually. We hope this will catalyze future research in IoT devices.

5.1 Future Work

This paper creates many opportunities for future work to continue research surrounding IoT devices.

The first thing we want to do is to maintain an ever-growing database. We want to clean up the database for quick queries, available to the public with a relevant event log. There are also some inconsistencies in the database, such as cases where we plug a device into an incorrectly labeled smart switch. Changing all those names would be apart of maintenance of this database.

Next, we want to make the realtimeIoTgrapher public and improve its features.

Hosting it on a publically hosted website would analyze the database much more accessible for new researchers.

We want to also utilize both power and network data in tandem for analysis to see if that yields better classification results.

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 any of the ideas above or this paper's project would be open to future work as well. We quickly tried a 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. However, we did not have time for that, and that would be interesting for future work as well.

Regardless, there are many opened opportunities for future work because of the database. They can quickly utilize the realtimeIoTGrapher to analyze the database or manually query data. There have already been students who have used the database for various research projects.

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] iotforall, "The 5 Worst Examples of IoT Hacking and Vulnerabilities in Recorded History," 2017. [Online]. Available: https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/

[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] Belkin, "Wemo." [Online]. Available: https://www.belkin.com/us/Products/smarthome-iot/c/wemo/

[18] 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/

[19] ——, "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/

[20] 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/

[21] "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

[22] https://www.nintendo.co.jp/ir/en/finance/hard_soft/index.html, "Dedicated Video Game Sales Units."

[23] A. Souppouris, "Sony Has Sold 50 Million PlayStation 4s," 2016. [Online]. Available: https://www.engadget.com/2016/12/07/sony-playstation4-50-million-sales/

[24] "The 5 Worst Examples of IoT Hacking and Vulnerabilities in Recorded History," 2018. [Online]. Available: https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/

[25] N. Feamster, "Who Will Secure the Internet of Things," 2016. [Online]. Available: https://freedom-to-tinker.com/2016/01/19/who-will-secure-the-internet-of-things/