

SET YOUR TITLE USING \title

I. SHAA, CITIoT, and MIoTTL Design

1.1 Overview

This research introduces three novel contributions in analyzing and mitigating Internet of Things (IoT) data leakage in smart home environments: a Smart Home Automation Architecture (SHAA), the Classification, Identification, and Tracking of Internet of Things (CITIoT) tool, and the Mitigation of Internet of Things Leakage (MIoTL) tool. SHAA is a testbed that provides realistic smart home traffic by integrating Wi-Fi and Bluetooth Low Energy (BLE) commercial-off-the-shelf (COTS) devices with Apple’s home automation application, HomeKit. To analyze privacy leakage within SHAA, the CITIoT tool is used to classify IoT devices, identify IoT events, and track smart home users. The MIoTTL tool supplies mitigation techniques that neutralize aspects of CITIoT. This chapter provides a detailed description of SHAA, each component of the CITIoT and MIoTTL tools, and their respective roles within the experiment.

1.2 System Summary

Figure 1 displays a simplified system diagram of all components described in the design and their interactions: the SHAA, the CITIoT tool, and the MIoTTL tool (when active). The boundaries of the system are limited to these components; however, as discussed later in Chapter 4, the test environment attempts to accurately represent a real-world smart home by not taking any measures to filter out wireless traffic from

other devices outside of the network.

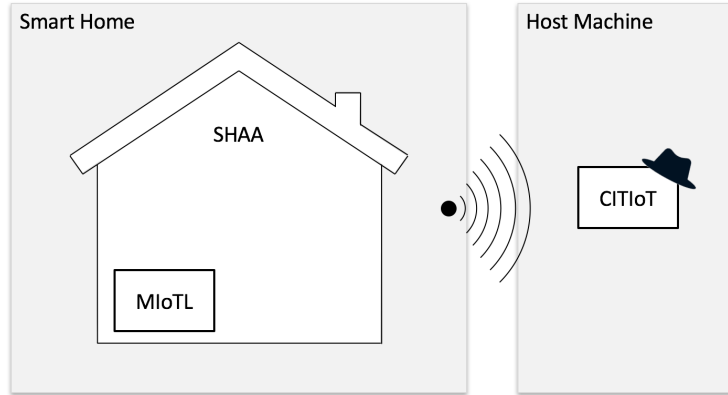


Figure 1. Overall system diagram

1.3 Smart Home Automation Architecture (SHAA)

The SHAA is used to provide real IoT traffic to be analyzed by the system under test. As shown in Figure 2, the SHAA includes three controller components and various connected devices. The controller components include (i) a Raspberry Pi running the Homebridge server that emulates the iOS HomeKit application programming interface (API) and exposes supported devices to Apple’s HomeKit, (ii) an iPhone 6+ running Apple’s HomeKit and device specific applications, and (iii) an Apple TV Generation 2 acting as a smart home hub to allow access to HomeKit supported devices while the user is away from the smart home. The communication between controllers and devices can be observed in Figure 2 and is described in the rest of this section.

1.3.1 Raspberry Pi.

The Raspberry Pi 3 Model B with Raspbian Jessie Lite version 4.9 operating system is connected to the smart home network via the on-board 802.11 b/g/n 2.4 GHz wireless chip [3]. The Raspberry Pi runs Homebridge version 0.4.14 as a systemd

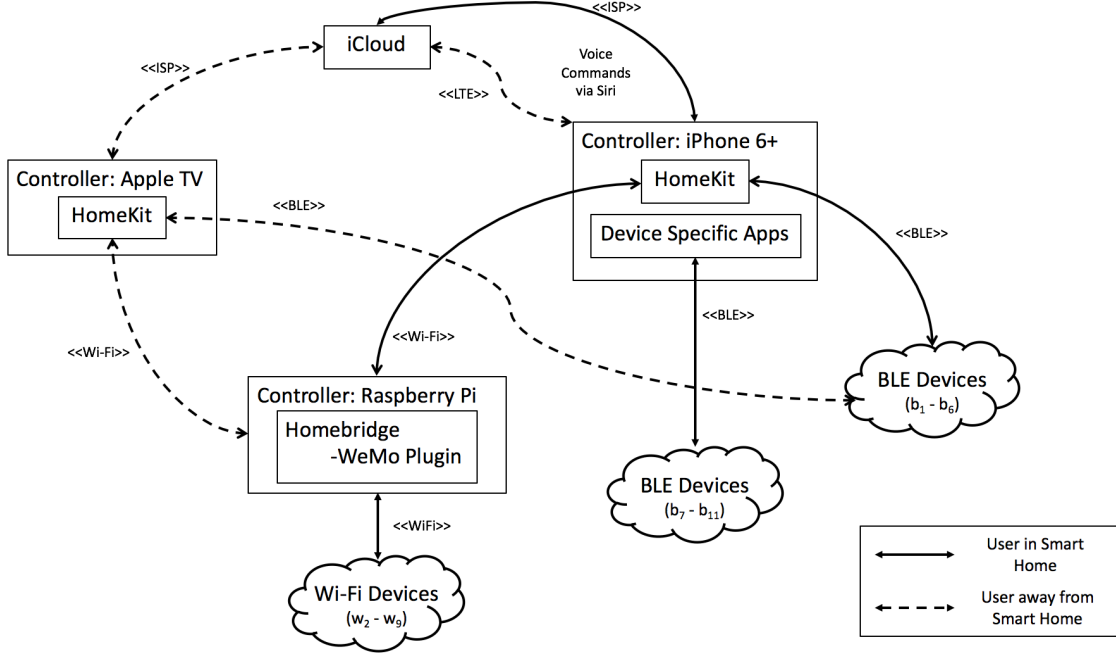


Figure 2. Diagram of SHAA components

service and each interaction between a controller and device is logged in the systemd journal [4]. A plugin is utilized to expose WeMo devices to the Apple Homekit and is loaded into Homebridge [2].

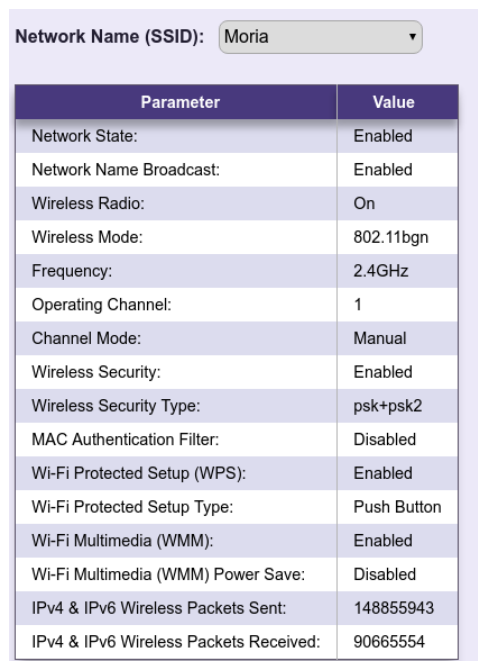
1.3.2 Apple Devices.

The iPhone 6+ and Apple TV act as controllers in the smart home architecture and connect to devices via Wi-Fi and BLE. When the user is home, the iPhone connects to Wi-Fi devices via the Homebridge and connects directly to the BLE devices. Some of the BLE devices are not supported by Apple's Homekit and can only be accessed through the manufacturer provided iOS application on the iPhone. When the user is away from the smart home, the iPhone can communicate with Homekit supported devices via the iCloud and Apple TV acting as a hub. For example, if the user is away from home and wants to access the temperature in a room, the iPhone communicates with the Apple TV via the iCloud and the Apple TV will communicate

with the device in the home via Wi-Fi or BLE. This will only work with Homekit supported devices, therefore, BLE devices b_7 - b_{12} (see Table ??) cannot be accessed while the user is away from the home.

1.3.3 Wi-Fi Devices.

To facilitate Wi-Fi communication in the smart home architecture, a 2.4 GHz Wi-Fi access point (AP), “Moria”, was setup with WPA2 security on channel 1 (see Figure 3 for a complete list of settings). A list of devices connected to the AP can be found in Table 1. The smart home devices include a camera, six outlets (four smart plugs, one mini plug, and one energy plug), and a motion sensor (w_2 - w_9). These devices use the Homebridge to communicate with Apple’s HomeKit on the iPhone.



Parameter	Value
Network State:	Enabled
Network Name Broadcast:	Enabled
Wireless Radio:	On
Wireless Mode:	802.11bgn
Frequency:	2.4GHz
Operating Channel:	1
Channel Mode:	Manual
Wireless Security:	Enabled
Wireless Security Type:	psk+psk2
MAC Authentication Filter:	Disabled
Wi-Fi Protected Setup (WPS):	Enabled
Wi-Fi Protected Setup Type:	Push Button
Wi-Fi Multimedia (WMM):	Enabled
Wi-Fi Multimedia (WMM) Power Save:	Disabled
IPv4 & IPv6 Wireless Packets Sent:	148855943
IPv4 & IPv6 Wireless Packets Received:	90665554

Figure 3. Prancing Pony Access Point Settings

Table 1. Wi-Fi Devices.

ID	Manuf	Device Type	Device Name	MAC	IP Address
w ₁	Calix	Wireless Router	Moria	EC:4F:82:73:D1:1A	-
w ₂	Belkin	Camera	NetCam	EC:1A:59:E4:FD:41	192.168.1.44
w ₃	Belkin	Outlet	Switch1	B4:75:0E:0D:33:D5	192.168.1.40
w ₄	Belkin	Outlet	Switch2	B4:75:0E:0D:94:65	192.168.1.41
w ₅	Belkin	Outlet	Switch3	94:10:3E:2B:7A:55	192.168.1.42
w ₆	Belkin	Outlet	Switch4	14:91:82:C8:6A:09	192.168.1.7
w ₇	Belkin	Motion Sensor	Motion	EC:1A:59:F1:FB:21	192.168.1.43
w ₈	Belkin	Outlet	Insight	14:91:82:24:DD:35	192.168.1.47
w ₉	WeMo	Outlet	Mini	60:38:E0:EE:7C:E5	192.168.1.51
w ₁₀	Raspberry Pi 3B	Computer	Pi	B8:27:EB:09:1A:81	12.168.1.50
w ₁₁	Apple	iPhone 6+	Steves-phone	A0:18:28:33:34:F8	192.168.1.4
w ₁₂	Apple	TV 2	Apple-TV	08:66:98:ED:1E:19	192.168.1.54

1.3.4 Bluetooth Low Energy Devices.

For Bluetooth communication to occur in the smart home architecture a Bluetooth master must be present. In the smart home architecture, the iPhone and Apple TV act as masters while each of the BLE devices are slaves. A list of devices operating in the BLE can be found in Table 2. The Media Access Control (MAC) addresses for each device are not included because they are randomized per the BLE protocol [1]. Devices b₁-b₆ are Homekit supported and can be accessed with voice commands via the iPhone 6+ or Apple TV. Devices b₇-b₁₂ are not Homekit supported and can only be accessed through their manufacture specific applications on the iPhone 6+.

Table 2. BLE Devices.

ID	Manuf	Device Type	Device Name
b ₁	Elgato	Indoor Temperature	Eve Room
b ₂	Elgato	Outdoor Temperature	Eve Weather
b ₃	Elgato	Motion Sensor	Eve Motion
b ₄	Elgato	Outlet	Eve Energy
b ₅	Elgato	Door Sensor	Eve Door
b ₆	Instant Pot	Smart Cooker	Instant Pot
b ₇	MPow	Lightbulb	Playbulb
b ₈	ZKTeco	Lock	BioLock
b ₉	BitLock	Lock	Bike lock
b ₁₀	SafeTech	Gunsafe	Gunsafe
b ₁₁	Apple	iPhone 6+	Steves-phone
b ₁₂	Apple	TV 2	Apple TV

1.4 Classification, Identification, and Tracking of Internet of Things (CITIoT)

The CITIoT tool contributes four capabilities enabled by data leakage in smart home Wi-Fi and BLE devices: device classification, event identification, user tracking, and network mapping. Figure 4 depicts how these capabilities are accomplished and can be summarized in six steps: (i) reconnaissance and scanning, (ii) passive sniffing, (iii) data preprocessing, (iv) tracking, (v) classification, and (vi) network mapping. Steps i and ii require user interaction, while steps iii-vi are executed via python scripts. The next sections provide a description of all components and their interactions.

1.4.1 Hardware.

The Host Machine is used to operate the software and sniffers. The prime component is a Hewlett-Packard 8570w with a 64-bit Intel Core i7 3270QM processor running at 2.60GHz, 16 GB DDR3 (4 x 4 GB) RAM, 120 GB SATA Hard Drive, and using Kali version 2017.1 as the operating system. These specifications are considered when observing processing time in Chapter 5. Three BLE sniffers (Ubertooth One with firmware 2017-03R2) and a wide range 802.11 ac dual band wireless adapter

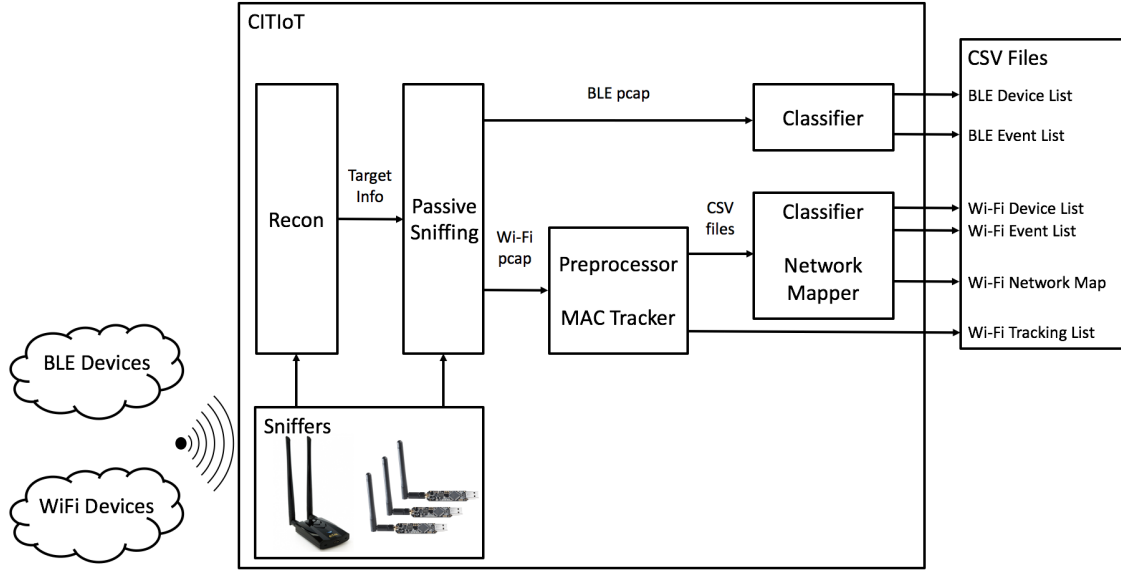


Figure 4. Diagram of CITIoT tool components and interactions

(Alfa Card AWUS036ACH) are connected to the host via Universal Serial Bus (USB). Each Ubertooth One device use a 2.4GHz 2.2dBi antenna (sparkfun) and connects to the host using USB 2.0, while the Alfa Card uses a 2.4GHz and 5GHz Dual-Band 5dBi dipole antenna (Amazon.com) and connects to the host using USB 3.0.

1.4.1.1 Ubertooth One Issues.

During testing it became clear that there were a few issues with the Ubertooth One firmware. First, an investigation of a 3-hour capture identified that the Ubertooth's clock would start drifting over time making it impossible to identify the real-time of a packet. This issue was resolved by updating the Ubertooth One firmware to utilize the clock of the host computer rather than the internal Ubertooth One clock for the packet timestamp. This fix was published to the Ubertooth Firmware GitHub issues page as issue #251 [1]. Second, the Ubertooth One sometimes froze when a master sent a connection request packet. Investigation proved that this was caused because newer devices such as the iPhone 6+ can elect to use a subset of available

BLE channels and the Ubertooth One did not support this feature. The Ubertooth One sniffers were observed during experimentation and if the sniffers froze, they were restarted. A bug report was submitted as issue #270 and has since been resolved by the Ubertooth One developers [1].

1.4.2 Reconnaissance and Scanning.

Reconnaissance is necessary to ascertain four elements about the smart home network which CITIoT requires for operation: AP MAC address, AP channel, Wi-Fi device MAC addresses and BLE device names, and controller MAC addresses. Figure 5 shows the command used to operate the Alfa Card and Airodump-ng to scan for devices and APs. This scan identifies the target and smart home information used during passive sniffing: the target's MAC address, associated AP MAC address, Service Set Identifiers (SSID) of the smart home, and AP channel.

```
root@gimli:gimli# airodump-ng wlan1
```

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
44:1C:A8:D5:E9:67	-63	2	0 0	6	54e	WPA2	CCMP	PSK	EIL
EC:4F:82:73:15:AF	-46	2	0 0	1	54e	WPA2	CCMP	PSK	Buckeyes2.4
EC:4F:82:73:D0:AE	-66	2	0 0	1	54e	WPA2	CCMP	PSK	EMF_411WS_106_2.4
EC:4F:82:73:D1:1A	-35	2	0 0	1	54e	WPA2	CCMP	PSK	Moria
EC:4F:82:73:15:B8	-65	2	0 0	1	54e	WPA2	CCMP	PSK	EMF_411WS_402_2.4
B8:EE:0E:E9:82:7E	-60	3	0 0	1	54e	WPA2	CCMP	PSK	MySpectrumWiFi78-2G
68:14:01:A8:E4:67	-49	3	0 0	1	54e	WPA2	CCMP	PSK	EWING-2.4
B8:A1:75:23:60:43	-48	2	0 0	1	54e	WPA2	CCMP	PSK	<length: 22>
EC:4F:82:73:17:0E	-67	2	0 0	1	54e	WPA2	CCMP	PSK	EMF_411WS_105_2.4
EC:4F:82:73:D3:87	-52	3	0 0	1	54e	WPA2	CCMP	PSK	EMF_411WS_302_2.4

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
EC:4F:82:73:D1:1A	A0:18:28:33:34:F8	-12	0 -24	3	5	

Figure 5. Command and results to accomplish a scan of Wi-Fi devices and associated APs

Next, Figure 6 shows the command used to scan for devices connected to the smart home network with the Alfa Card and Airodump-ng while filtering on the target AP MAC address. The list of device MAC addresses associated with the target AP is

collected and device manufacturers are discovered using an Organizational Unique Identifier (OUI) lookup tool found at wireshark.org. Figure 7 shows results from an OUI lookup using the Wi-Fi devices found during the scan. This information is used to infer which devices are IoT devices (e.g., Belkin devices) and which are controllers (e.g., Raspberry Pi or Apple devices).

```
root@gimli:~# airodump-ng wlan1 --bssid ec4f8273d11a
```

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
EC:4F:82:73:D1:1A	-23	242	47	0	1	54e	WPA2	CCMP	PSK Moria

BSSID	STATION	PWR	Rate	Lost	Frames	Probe
EC:4F:82:73:D1:1A	EC:1A:59:E4:FD:41	-23	54e-54e	0	64269	
EC:4F:82:73:D1:1A	EC:1A:59:F1:FB:21	-23	54e-46e	0	61572	
EC:4F:82:73:D1:1A	94:10:3E:2B:7A:55	-28	54e-54e	0	31353	
EC:4F:82:73:D1:1A	B4:75:0E:0D:33:D5	-36	46e-54e	0	32601	
EC:4F:82:73:D1:1A	60:38:E0:EE:7C:E5	-39	54e-1e	0	48631	
EC:4F:82:73:D1:1A	B8:27:EB:09:1A:81	-38	54e-54e	0	301214	
EC:4F:82:73:D1:1A	14:91:82:C8:6A:09	-41	54e-46e	0	28617	
EC:4F:82:73:D1:1A	A0:18:28:33:34:F8	-45	54e-24	476	205282	
EC:4F:82:73:D1:1A	14:91:82:24:DD:35	-48	54e-54e	0	42845	
EC:4F:82:73:D1:1A	08:66:98:ED:1E:19	-49	54e-54e	238	32549	
EC:4F:82:73:D1:1A	B4:75:0E:0D:94:65	-55	36e-1e	0	31496	

Figure 6. Command and results to scan for devices connected to the target AP

OUI search	
B4:75:0E:0D:33:D5	
B4:75:0E:0D:94:65	
94:10:3E:2B:7A:55	
14:91:82:C8:6A:09	
EC:1A:59:F1:FB:21	
14:91:82:24:DD:35	
60:38:E0:EE:7C:E5	
B8:27:EB:09:1A:81	
A0:18:28:33:34:F8	
08:66:98:ED:1E:19	
Results	
08:66:98	Apple, Inc.
14:91:82	Belkin International Inc.
60:38:E0	Belkin International Inc.
94:10:3E	Belkin International Inc.
A0:18:28	Apple, Inc.
B4:75:0E	Belkin International Inc.
B8:27:EB	Raspberry Pi Foundation
EC:1A:59	Belkin International Inc.

Figure 7. Wi-Fi MAC OUI search and results

Similarly, Figure 8 shows the command used to operate the Bluetooth wireless adapter (Plugable USB 2.0) and sniffing tool (BlueZ) to scan for BLE devices. The

Bluetooth service is started, the interface is activated, and scanning is initiated. The results show device names and MAC addresses found from Advertising Indication and Scan Response packets sent from within the smart home.

```
root@gimli:gimli# service bluetooth start
root@gimli:gimli# hciconfig hci0 up
root@gimli:gimli# hcitool lescan
LE Scan ...
EB:6E:E4:03:A0:67 Eve
EB:6E:E4:03:A0:67 Eve Energy 556E
FA:1B:EF:55:41:C8 Eve
FA:1B:EF:55:41:C8 Eve Motion 31A7
08:7C:BE:30:69:31 BLELock
08:7C:BE:30:69:31 BLELock
20:C3:8F:EC:29:DC Instant Pot Smart
F0:3A:A4:B1:3D:F0 Eve
F0:3A:A4:B1:3D:F0 Eve Weather 943D
AC:E6:4B:0A:74:81 PLAYBULB
FA:1B:EF:55:41:C8 Eve
FA:1B:EF:55:41:C8 Eve Motion 31A7
08:7C:BE:30:69:31 BLELock
08:7C:BE:30:69:31 BLELock
20:C3:8F:EC:29:DC Instant Pot Smart
F0:3A:A4:B1:3D:F0 Eve
F0:3A:A4:B1:3D:F0 Eve Weather 943D
AC:E6:4B:0A:74:81 PLAYBULB
FA:67:4F:5E:5C:CA Eve
FA:67:4F:5E:5C:CA Eve Door 91B3
DA:68:F2:6F:AC:72 Eve Room 4A04
```

Figure 8. Command and results to scan for BLE devices within the smart home

1.4.3 Passive Sniffing.

Passive sniffing is used to capture Wi-Fi and BLE traffic from the smart home. Sniffing occurs simultaneously for Wi-Fi and BLE traffic using the Alfa Card and three Ubertooth One sniffers respectively. Prior to capturing Wi-Fi traffic, the wireless interface that corresponds to the Alfa Card must be set to monitor mode to capture all Wi-Fi packets destined to all SSIDs on the selected channel. Figure 9 shows the five commands used to set the interface to monitor mode: (i) kill any processes that may interfere with the Aircrack-ng tool, (ii) bring down the interface, (iii) set the interface to monitor mode, (iv) bring the interface back up, and (iv) ensure the interface is in monitor mode.

```

root@gimli:gimli# airmon-ng check kill

Killing these processes:

PID Name
650 wpa_supplicant
651 dhclient

root@gimli:gimli# ifconfig wlan1 down
root@gimli:gimli# iwconfig wlan1 mode monitor
root@gimli:gimli# ifconfig wlan1 up
root@gimli:gimli# iwconfig
lo          no wireless extensions.

wlan1      IEEE 802.11  Mode:Monitor  Frequency:2.412 GHz  Tx-Power=20 dBm
Retry short limit:7  RTS thr:off  Fragment thr:off
Power Management:off

eth0       no wireless extensions.

wlan0      IEEE 802.11  ESSID:off/any
Mode:Managed  Access Point: Not-Associated  Tx-Power=15 dBm
Retry short limit:7  RTS thr:off  Fragment thr:off
Encryption key:off
Power Management:off

```

Figure 9. Commands used to set Wi-Fi interface to monitor mode

The Airodump-ng tool is then used to capture Wi-Fi traffic from the smart home. When operating the capture tool, the Alfa Card’s interface (“wlan1”), capture output file format (“.cap”), target AP MAC address (“ec4f8273d11a”), and target AP Wi-Fi channel (“1”) options are set. Figure 10 depicts the command and options used to capture Wi-Fi traffic.

```

root@gimli:gimli# airodump-ng --channel 1 wlan1 --output-format cap -w wifi --bssid ec4f8273d11a

```

Figure 10. Command and options used to capture Wi-Fi traffic

To capture BLE traffic, three Ubertooth One sniffers are set to operate with each device (“U0”-“U2”) tuned to one of three advertisement channels (“37”-“39”), to follow traffic (“f”), and output packets to a capture file (“.pcap”). A script was written using the Bash Unix shell and command language to simplify activating all

three of the Ubertooth One sniffers simultaneously. Appendix X includes the script code while Figure 11 provides an example of the command and options used to operate one Ubertooth One sniffer. When sniffing is complete, each tool is terminated and the resulting capture files are stored on the Host Machine.

```
root@gimli:gimli# ubertooth-btle -f -U0 -A37 -qble.pcap
```

Figure 11. Example command and options used to capture BLE traffic

1.4.4 Preprocessor.

After passive sniffing is complete, the Wi-Fi packet captures are parsed and organized in preparation for the classifier. This is accomplished with a script written in Python using Pyshark, a Python wrapper for parsing packets with Wireshark dissectors (See Appendix X). The time, size, source, and destination for each 802.11 data packet are extracted from the captures and the resulting 4-tuples are stored in comma-separated values (CSV) files. All other 802.11 packet types and packets with a source or destination not in the list of Wi-Fi devices found during reconnaissance do not include information used by the classifier and are not saved. The new 4-tuples are stored in two files per device: one file in which each 4-tuple has a source address of the device and one file in which each 4-tuple has a destination address of the device. For example, the Belkin Motion Sensor (w_{10}) will have two files per day, one in which every 4-tuple represents a packet from the sensor to another device in the Wi-Fi device list and one file in which every 4-tuple represents a packet from a device in the list to the Motion Sensor.

1.4.5 MAC Tracker.

The MAC tracker unit tracks when devices are in the smart home based off Wi-Fi packets sent from that device. It is implemented within the preprocessing script (see

Appendix X), operates at the same time as data preprocessing, and utilizes every 802.11 packet that has a source MAC address of a device in the Wi-Fi device list. As the packet capture is parsed, the tracker keeps a list of the first and last time a packet was sent from a device within the smart home. If the tracker observes no packets sent from a device for five minutes it marks the device as no longer in the smart home and records the arrival and departure times into a CSV file. Five minutes was chosen as the time to identify a device as no longer present after trial and error showed that it was an appropriate amount of time to account for device idleness. This value is easily changed within the script and can be altered to provide more confidence that the device is away from the home.

During testing it was observed that some packets caused the tool to report erroneous times. These packets either had a corrupt time value or the frame number was incorrect. Periodically an epoch timestamp would be saved within the packet with a negative value. Figure 12 shows an example packet with a corrupted epoch time. These erroneous time values caused the MAC Tracker to provide incorrect results. Other times a subsequent packet would have a timestamp that should have appeared much later in the capture indicating that the frame number was incorrect. Figure 13 shows an example of four packets with successive frame numbers, but the times were off. This issue was overcome by comparing a new packet's time value with the previous packet's time value and if the difference in time was greater than two seconds, the new packet was ignored. The tracker provides the user with the number and time of invalid packets to facilitate further investigation.

```

▼ Frame 1863119: 10 bytes on wire (80 bits), 10 bytes captured (80 bits)
  Encapsulation type: IEEE 802.11 Wireless LAN (20)
  Arrival Time: Aug 22, 2017 15:29:38.-00020000 EDT
  ► [Expert Info (Note/Sequence): Arrival Time: Fractional second -00020000 is invalid,
    [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: -1503430178.000020000 seconds
    [Time delta from previous captured frame: 0.037440000 seconds]
    [Time delta from previous displayed frame: 0.037440000 seconds]
    [Time since reference or first frame: 34603.217113000 seconds]
    Frame Number: 1863119
    Frame Length: 10 bytes (80 bits)
    Capture Length: 10 bytes (80 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: wlan]
  ► IEEE 802.11 Clear-to-send, Flags: .....

```

Figure 12. Encrypted packet used in MAC tracker showing corrupted timestamp

No.	Time	Source	Destination
356687	2017-08-25 08:38:37	Apple_33:34:f8	Calix_73:d1:1a
356688	2017-08-25 13:46:04	Apple_33:34:f8	Calix_73:d1:1a
356689	2017-08-25 13:46:04	Apple_33:34:f8	Calix_73:d1:1a
356690	2017-08-25 08:38:38	Apple_33:34:f8	Calix_73:d1:1a

Figure 13. Encrypted packets used in MAC tracker showing sequential frame numbers but wrong times

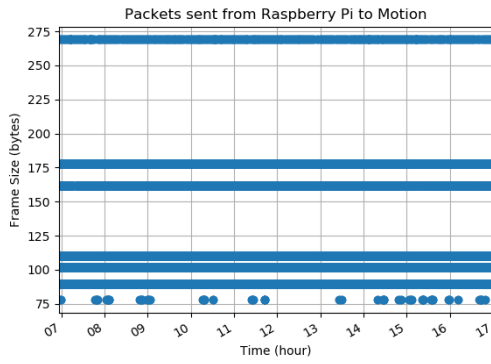
1.4.6 Classifier.

This section describes the training and operation of the classifier used to classify devices and identify events within the smart home. There are three components to the classifier: (i) the Wi-Fi classifier trainer, (ii) the Wi-Fi classifier, and (iii) the BLE classifier. Each of these components are implemented separately and are described below.

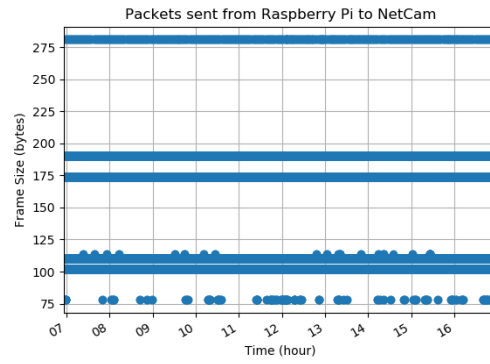
1.4.6.1 Wi-Fi Classifier Trainer.

Training the Wi-Fi classifier was accomplished using traffic captured over two ten-hour days in which each Wi-Fi and BLE device is activated according to the Classifier Training Event Log (see Appendix X). Events were chosen that represent a real user

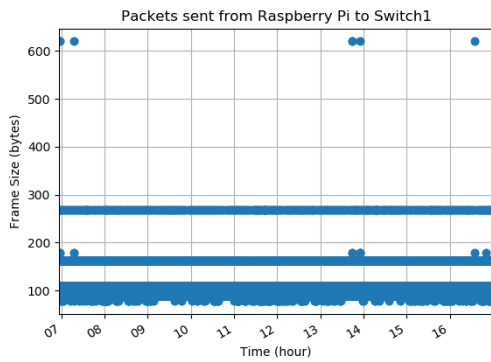
living and activating devices throughout a normal day in a smart home. The traffic from the smart home was captured and preprocessed as described above. For Wi-Fi traffic, the classifier is trained to classify devices into one of three types (outlet, sensor, or camera) and identify device events (outlet event or motion event). The CSV files created during the preprocessing stage are used by the classifier training script (see Appendix X) to provide the user two scatter plots per device (one depicting packets sent from a device and one depicting packets sent to a device). Figure 14 shows the plots for packets sent from the Raspberry Pi to each Wi-Fi device, while Figure 15 shows the plots for packets sent from each device to the router. The x-axis of these plots represents time in seconds, while the y-axis represents packet size in bytes. These plots are used as a graphical representation of the traffic to help determine trends and patterns in packet sizes for devices and events.



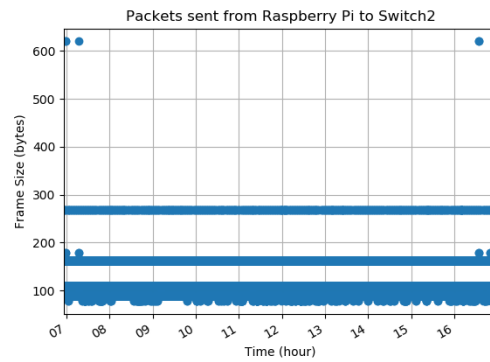
(a)



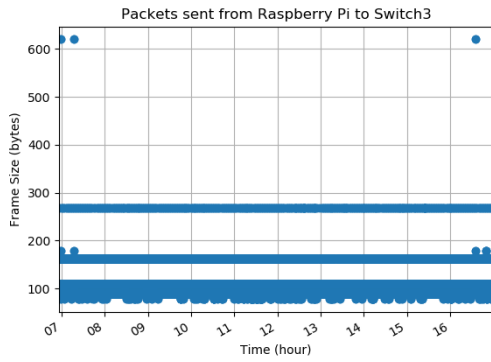
(b)



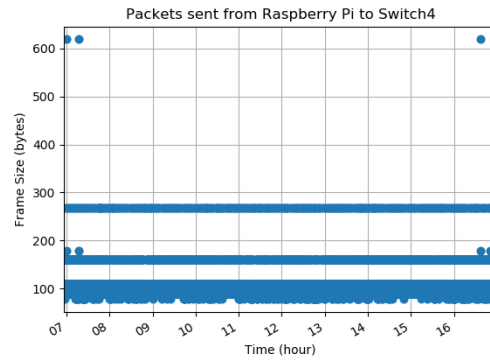
(c)



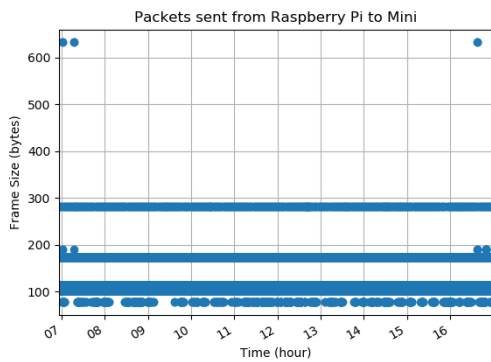
(d)



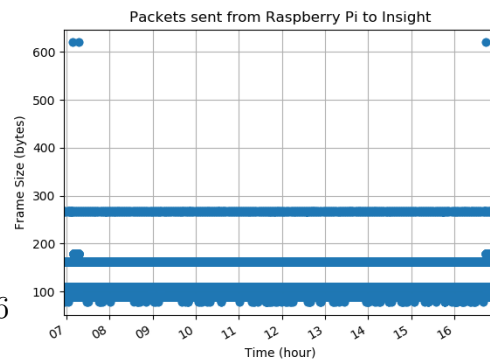
(e)



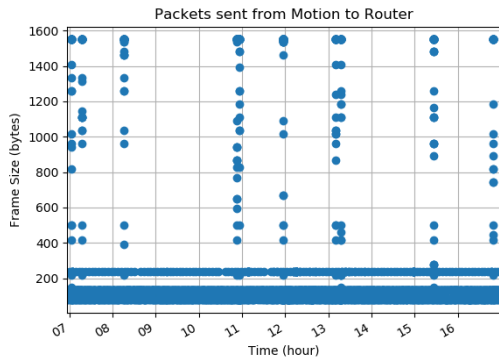
(f)



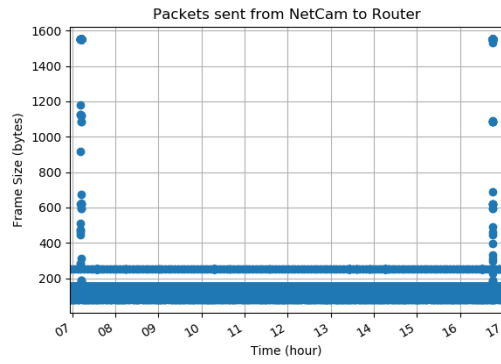
(g)



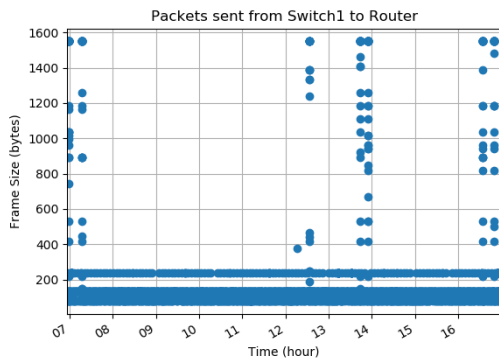
(h)



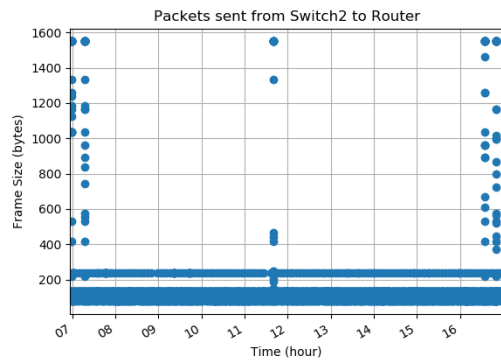
(a)



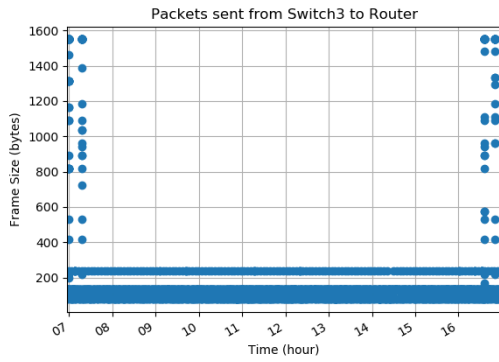
(b)



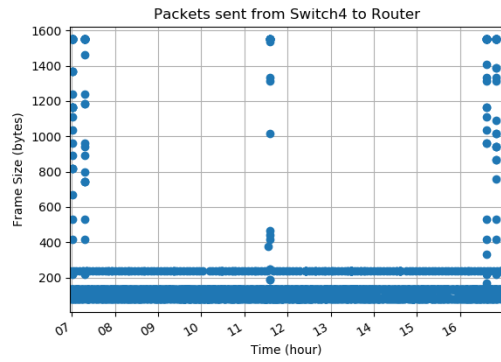
(c)



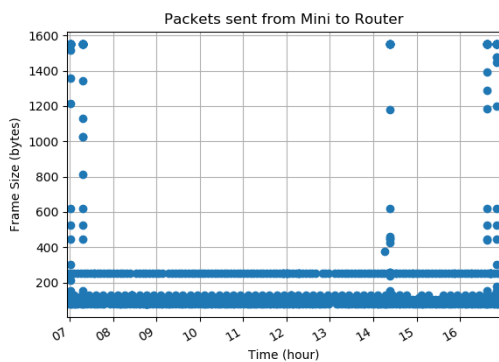
(d)



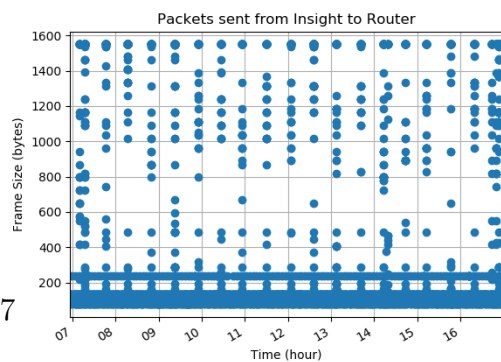
(e)



(f)



(g)



(h)

After observing the plots created in training, it was discovered that the Raspberry Pi communicates with each type of device in a unique way and, therefore, traffic from the Raspberry Pi to the devices can be used to classify devices. Figures 16, 17, and 18 show three plots from Figure 14 (packets from the Raspberry Pi to the NetCam, Motion, and Switch1) zoomed in on the unique packet traffic that helped generate the classification criteria. Figure 19 lists the criteria established to classify devices using the plots. This criteria is described as follows: if the device receives a packet from the Raspberry Pi between 619 and 632 bytes it is an outlet, if the device is not an outlet and receives packets of 269 bytes it is a sensor device, and if the device is not an outlet and receives packets of size 281 bytes it is a camera device.

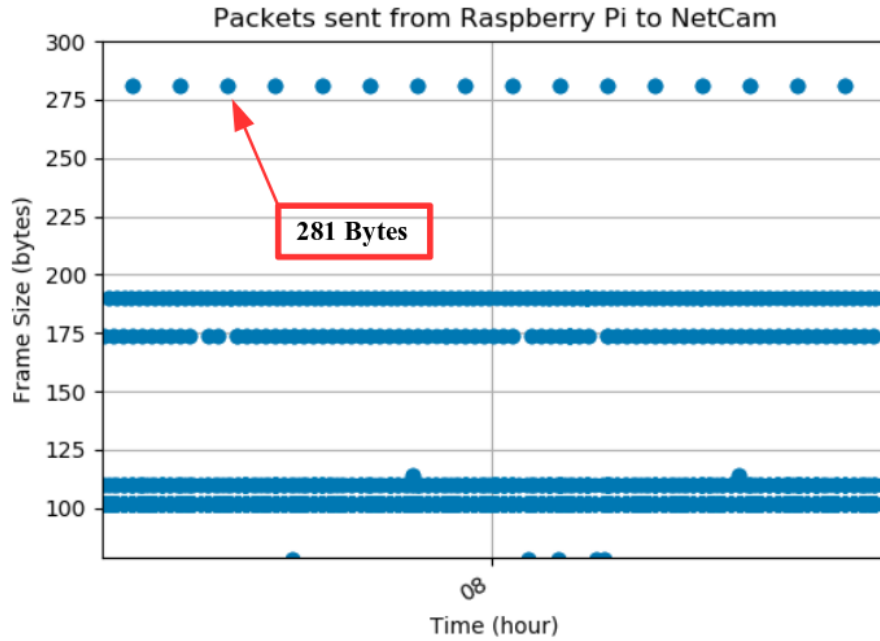


Figure 16. Figure 14(b) zoomed in on unique packet traffic used to classify camera devices

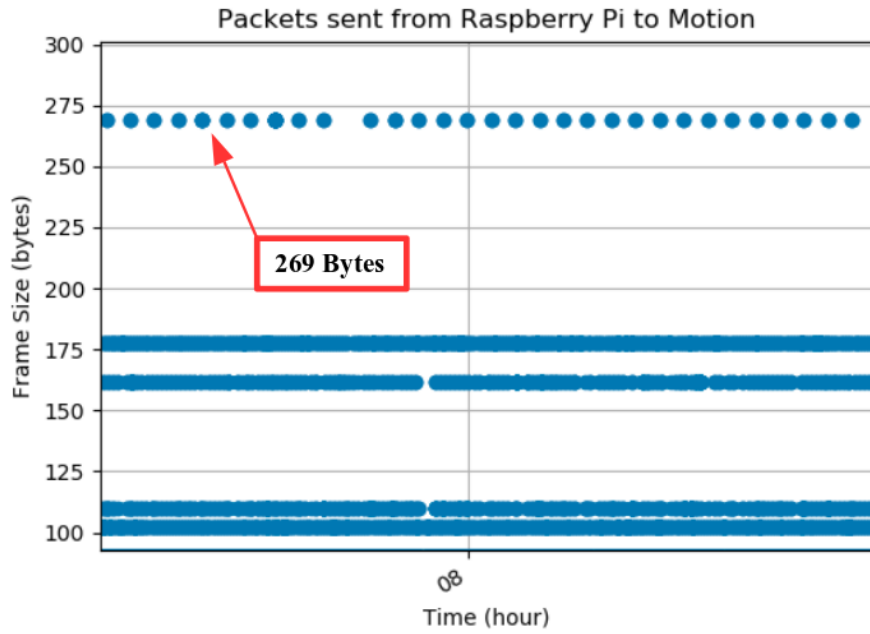


Figure 17. Figure 14(a) zoomed in on unique packet traffic used to classify motion devices

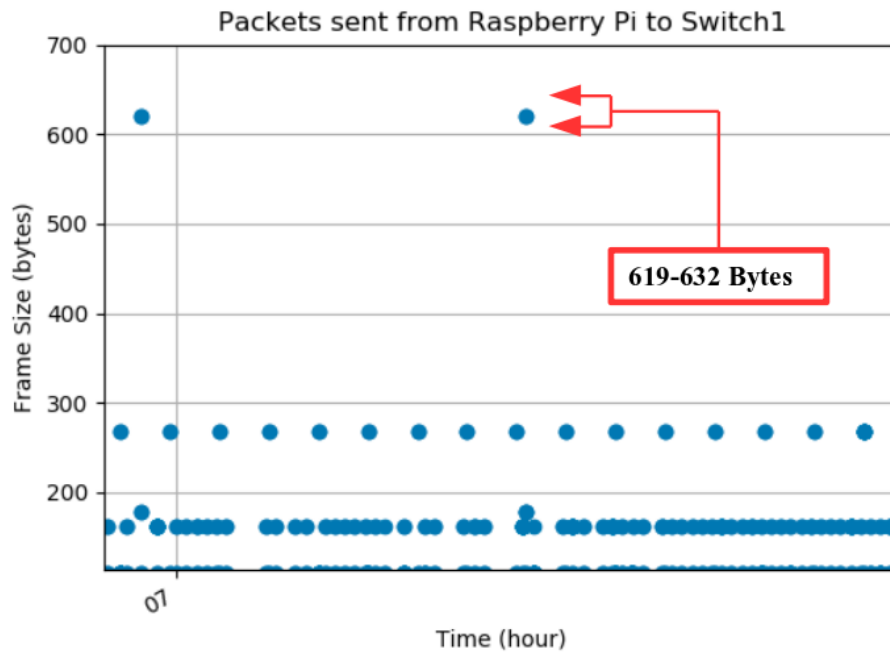


Figure 18. Figure 14(c) zoomed in on unique packet traffic used to classify outlet devices

Device Type	Device Classification Criteria
Outlet	$619 \text{ bytes} \leq FSize_{incoming} \leq 632 \text{ bytes}$
Sensor	$if \text{ device} \neq \text{outlet} \text{ and } FSize_{incoming} = 269 \text{ bytes}$
Camera	$if \text{ device} \neq \text{outlet} \text{ and } FSize_{incoming} = 281 \text{ bytes}$

Figure 19. Criteria used to classify devices

Similarly, comparing event logs to the training plots in Figure 14 revealed that the Raspberry Pi sends a packet with a frame size of 619, 620, or 632 bytes to an outlet every time an event occurs. Figure 20 shows the training plot depicting packets sent from the Raspberry Pi to the Mini outlet zoomed in on two events that helped generate the event identification criteria for outlets. At times multiple packets are sent in a minute due to retransmission, therefore, CITIoT only identifies one event per device per minute. This sample interval provides enough precision for this work. Also, by correlating event logs with the plots in Figure 15 it was observed that the sensors and cameras send a burst of packets to the router every time an event occurs. Figures 21 and 23 show two training plots depicting packets sent from the NetCam and Motion to the router focused around the unique packet traffic that helped generate the event identification criteria for these devices. The bursts sent by the sensor or camera may span a two-minute period so to avoid CITIoT flagging two events (one per minute), the tool only identifies one event per device per two minutes. Figure 25 lists the criteria established to identify events. This criteria is described as follows: if the device is a camera and the total frame size sent to the router over a minute is greater than 100,000 bytes a camera event occurred, if the device is a sensor and the total frame size sent to the router in a minute is greater than 10,000 bytes a motion event occurred, and if the device frame size received from the Raspberry Pi is between

619 and 632 bytes then an outlet event occurred.

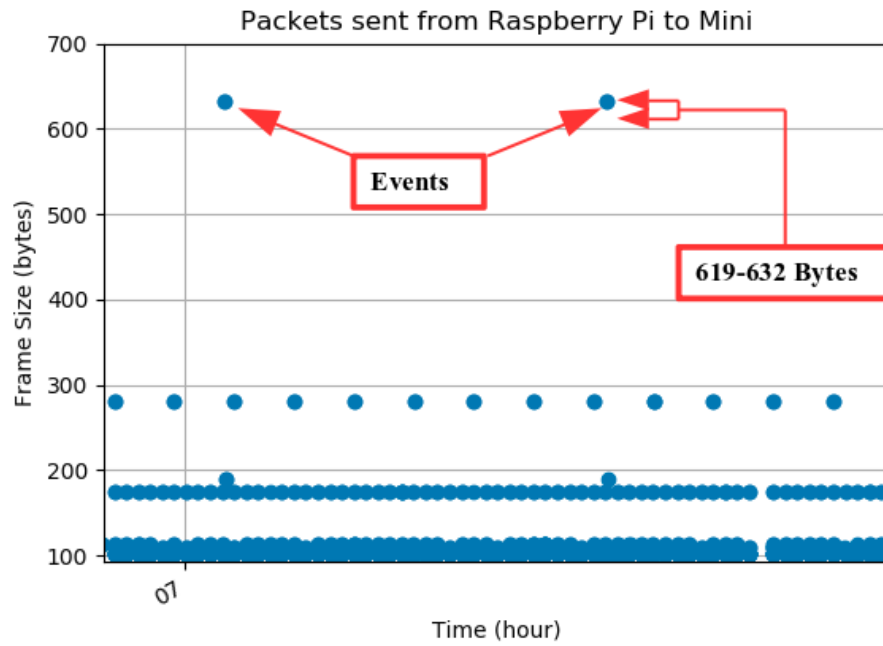


Figure 20. Figure 14(g) zoomed in on unique packet traffic used to identify outlet events

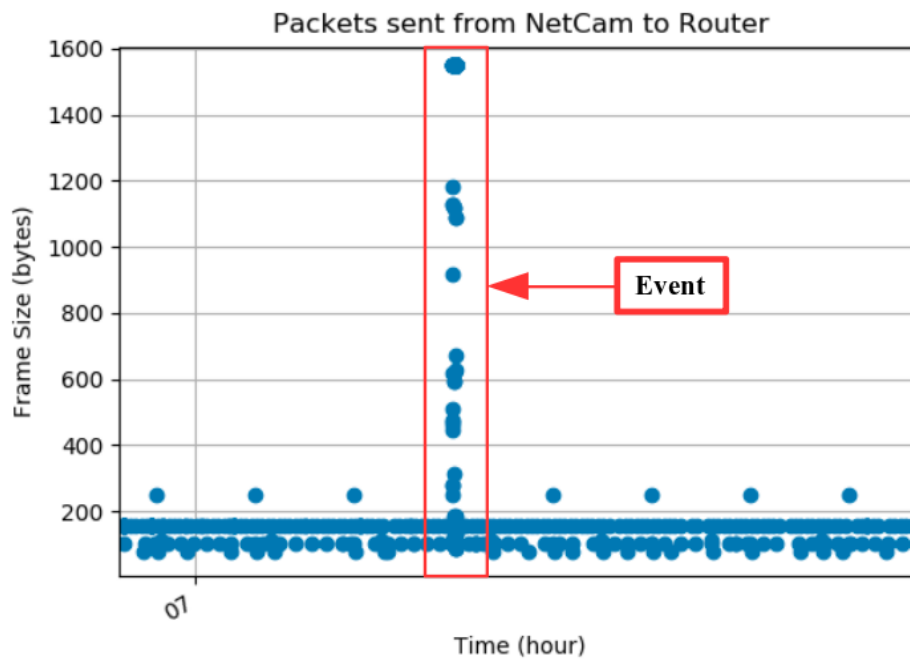


Figure 21. Figure 15(b) zoomed in on unique packet traffic used to identify camera events

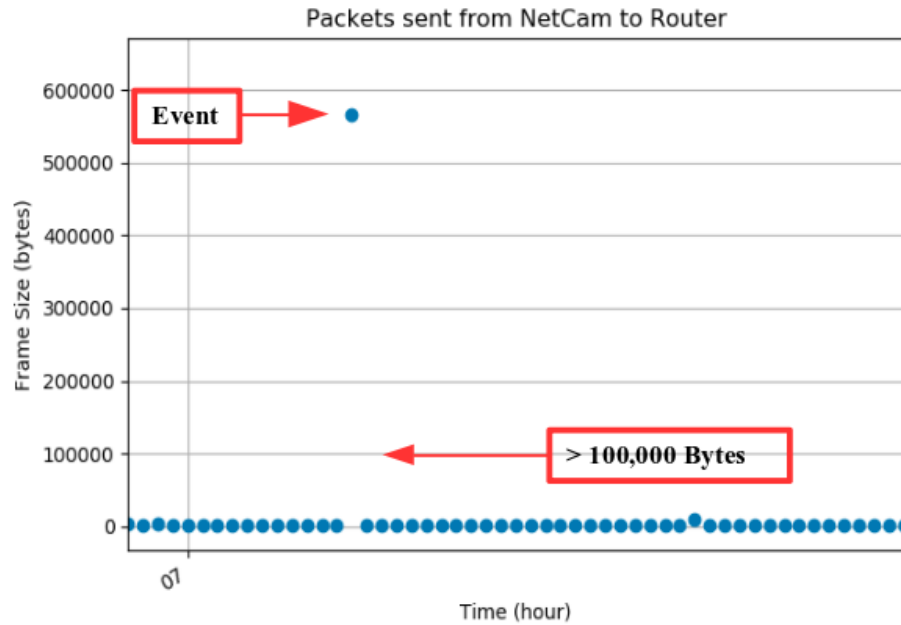


Figure 22. Figure 15(b) with one minute cumulative frame size zoomed in on unique packet traffic used to identify camera events

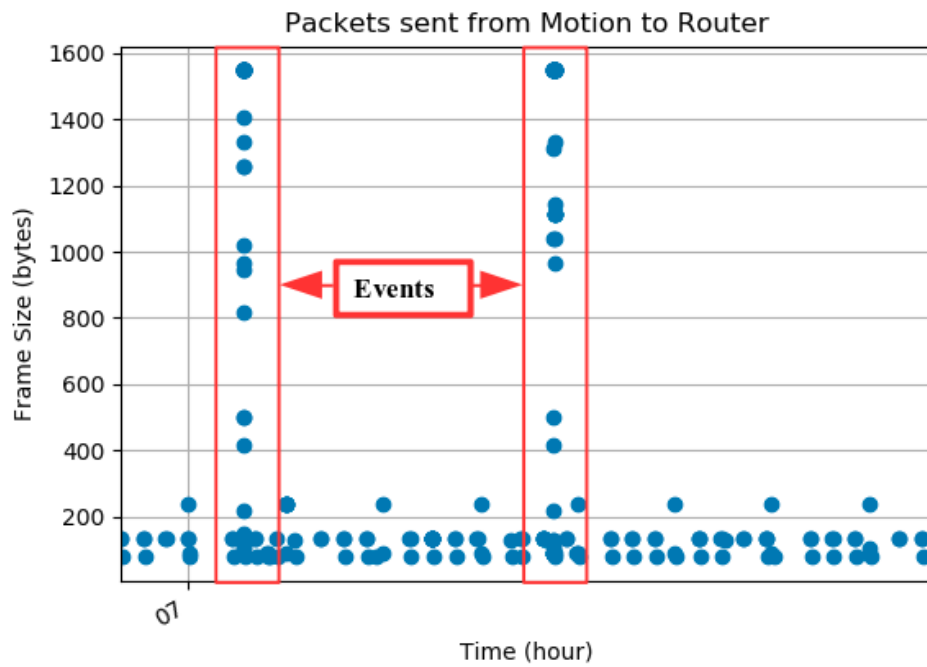


Figure 23. Figure 15(a) zoomed in on unique packet traffic used to identify motion events

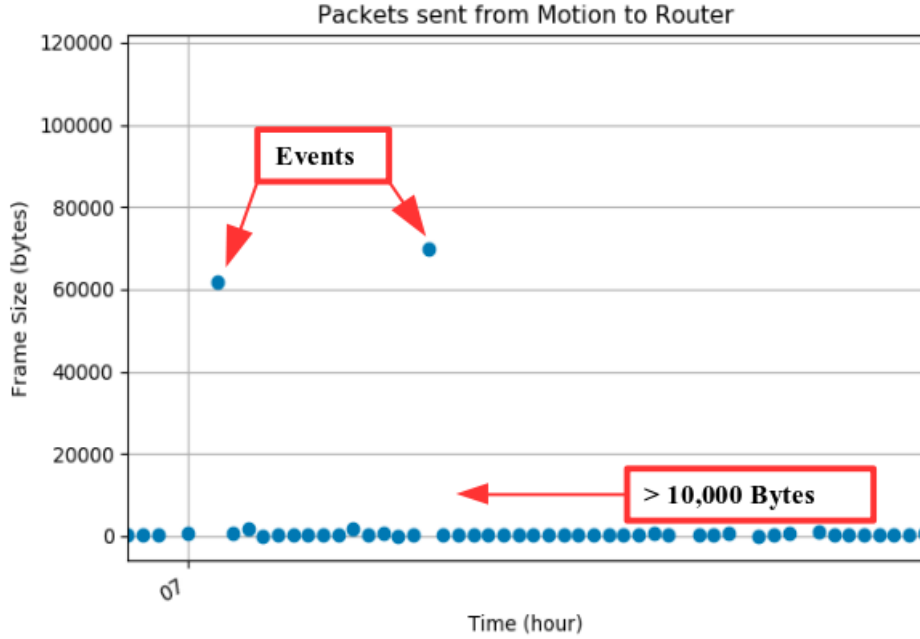


Figure 24. Figure 15(a) with one minute cumulative frame size zoomed in on unique packet traffic used to identify motion events

Device Type	Event Identification Criteria
Outlet	$if\ device = outlet\ and\ 619\ bytes \leq FSize_{Incoming} \leq 632\ bytes$
Sensor	$if\ device = sensor\ and\ \sum_t^{t+60s} FSize_{Outgoing} > 10,000\ bytes$
Camera	$if\ device = camera\ and\ \sum_t^{t+60s} FSize_{Outgoing} > 100,000\ bytes$

Figure 25. Criteria used to identify events

Further investigation of each characteristic Wi-Fi packet exchange was accomplished to determine why patterns exist in these packets. It was discovered that the Raspberry Pi sends a 269 or 281 byte Hypertext Transfer Protocol (HTTP) SUBSCRIBE packet to subscribe to events from a given device. The NetCam device is the only one to receive a 281 byte SUBSCRIBE packet and Figure 26 shows that a timestamp option accounts for the packet length difference. The Raspberry Pi sends

a 619, 620, or 632 byte HTTP POST request to activate an outlet. Figure 27 shows an example packet for each of the different sized POST packets. These packets revealed that Switch4 only recieved a 619 byte POST packet because the **IP!** (**IP!**) address had one character less in the final octet resultig in the HTTP host address being one byte less than other devices. The POST packet sent to the Mini outlet was 632 bytes with the additional 12 bytes coming from **TCP!** (**TCP!**) options that were not included in other devices. Every time the camera observes motion after one minute of idleness it sends the user a camera snapshot via email. Every time a sensor observes motion after one minute of idleness it sends a notification to the user. To do this, the device creates a secure connection to Belkin’s cloud service hosted by Amazon EC2 Cloud and then sends the notification to the Belkin Application. Decrypting this traffic is beyond the work presented in this thesis.

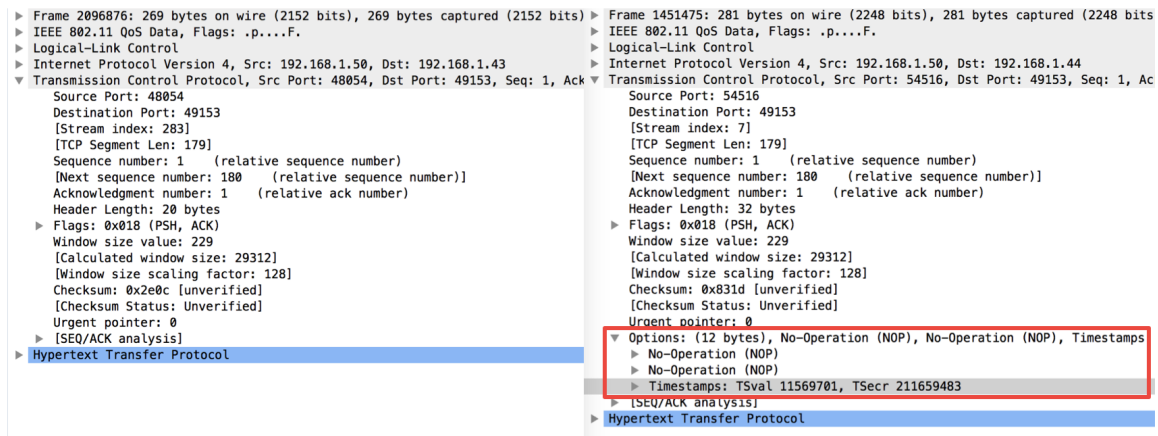


Figure 26. Decrypted SUBSCRIBE packets from Raspberry Pi to the NetCam and Motion devices depicting difference in frame length


```

▶ Frame 33578: 619 bytes on wire (4952 bits), 619 bytes captured (4952 bits)
▶ IEEE 802.11 QoS Data, Flags: .p....F.
▶ Logical-Link Control
▶ Internet Protocol Version 4, Src: 192.168.1.50, Dst: 192.168.1.7
▶ Transmission Control Protocol, Src Port: 47322, Dst Port: 49153, Seq: 1,
▼ Hypertext Transfer Protocol
  ▶ POST /upnp/control/basicevent1 HTTP/1.1\r\n
    SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"\r\n
    Content-Type: text/xml; charset="utf-8"\r\n
  Host: 192.168.1.7:49153\r\n
  Connection: close\r\n

▶ Frame 20785: 620 bytes on wire (4960 bits), 620 bytes captured (4960 bits)
▶ IEEE 802.11 QoS Data, Flags: .p....F.
▶ Logical-Link Control
▶ Internet Protocol Version 4, Src: 192.168.1.50, Dst: 192.168.1.41
▶ Transmission Control Protocol, Src Port: 57186, Dst Port: 49153, Seq: 1,
▼ Hypertext Transfer Protocol
  ▶ POST /upnp/control/basicevent1 HTTP/1.1\r\n
    SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"\r\n
    Content-Type: text/xml; charset="utf-8"\r\n
  Host: 192.168.1.41:49153\r\n
  Connection: close\r\n

▶ Frame 7570150: 632 bytes on wire (5056 bits), 632 bytes captured (5056 bits)
▶ IEEE 802.11 QoS Data, Flags: .p....F.
▶ Logical-Link Control
▶ Internet Protocol Version 4, Src: 192.168.1.50, Dst: 192.168.1.51
▼ Transmission Control Protocol, Src Port: 58466, Dst Port: 49153, Seq: 1, Ac
  Source Port: 58466
  Destination Port: 49153
  [Stream index: 1590]
  [TCP Segment Len: 530]
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 531 (relative sequence number)]
  Acknowledgment number: 1 (relative ack number)
  Header Length: 32 bytes
  ▶ Flags: 0x018 (PSH, ACK)
  Window size value: 229
  [Calculated window size: 29312]
  [Window size scaling factor: 128]
  Checksum: 0xab56 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▼ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
    ▶ No-Operation (NOP)
    ▶ No-Operation (NOP)
    ▶ Timestamps: TSval 3759416, TSecr 4294949569
    ▶ [SEQ/ACK analysis]

```

Figure 27. Decrypted POST packets from Raspberry Pi to the Switch4, Switch2, and Mini depicting differences in frame length

1.4.6.2 Wi-Fi Classifier.

While considered one unit, the classifier operates differently for Wi-Fi and BLE traffic and is implemented with two different scripts (one for each protocol). For Wi-Fi traffic, the classifier is operated via a script (see Appendix X) which utilizes each of the CSV files created during data preprocessing and the criteria found during training to classify devices and identify events. First, the classifier uses traffic destined to devices to classify device type. If the traffic meets one of the criteria the device type is set accordingly, otherwise the type is unknown. Second, based on the device type determined in the first step, traffic is tested against the event criteria to see when events occur. The device types are stored in a CSV file listing the MAC address and type of device. The time, source, and destination for each event identified are stored in a CSV file.

1.4.6.3 BLE Classifier.

For BLE traffic, a script (see Appendix X) is used to parse the packet captures created during passive sniffing for Advertising Indication (`ADV_IND`), Scan Response (`SCAN_RESP`), and Connect Request (`CONNECT_REQ`) packets. The `ADV_IND` and `SCAN_RESP` packets provide advertisement information, such as device name, used by the classifier to classify devices. Instead of classifying devices into categories like in Wi-Fi, the classifier relies on the name provided within these packets. The discovered device name is stored along with the device BLE MAC address in a CSV file. Devices only exchange information after a `CONNECT_REQ` packet, therefore, these packets correspond to device events. The classifier searches for `CONNECT_REQ` packets, use the device name found in the first step, and stores the time, source, and destination for each event into a CSV file. Only device events that match BLE devices found in reconnaissance are stored. Similar to Wi-Fi sensors and cameras, a single BLE event

may cause devices to send multiple `CONNECT_REQ` packets that span a two-minute time period. Therefore, CITIoT identifies one event per device per two-minutes (i.e., the `CONNECT_REQ` packets sent during the second minute are ignored). This may cause the device to miss two events that occur less than a minute apart, but this interval provides enough precision for this work.

1.4.7 Network Mapper.

The Network Mapper creates a graphical representation of how Wi-Fi devices are connected within the smart home using the size of 802.11 data packets passed between devices. The Network Mapper unit is implemented within the Wi-Fi classifier script (see Appendix X) and operates as the classifier parses each CSV file created by the data preprocessing tool. The frame size of each packet sent from one device to another is combined and stored in a new CSV file. For example, if the Raspberry Pi sends three packets of 500 bytes each to an outlet then a three-tuple is written to a CSV that includes the Raspberry Pi, the outlet, and the the cumulative number of bytes sent, 1500 bytes. This CSV file is then passed to a script written in R-Studio using the `FILLIN` library. Figure 28 provides a sample network map created by the script—nodes represent devices and edges depict the cumulative frame size sent between connected devices (i.e., thicker lines represent more data sent between devices).

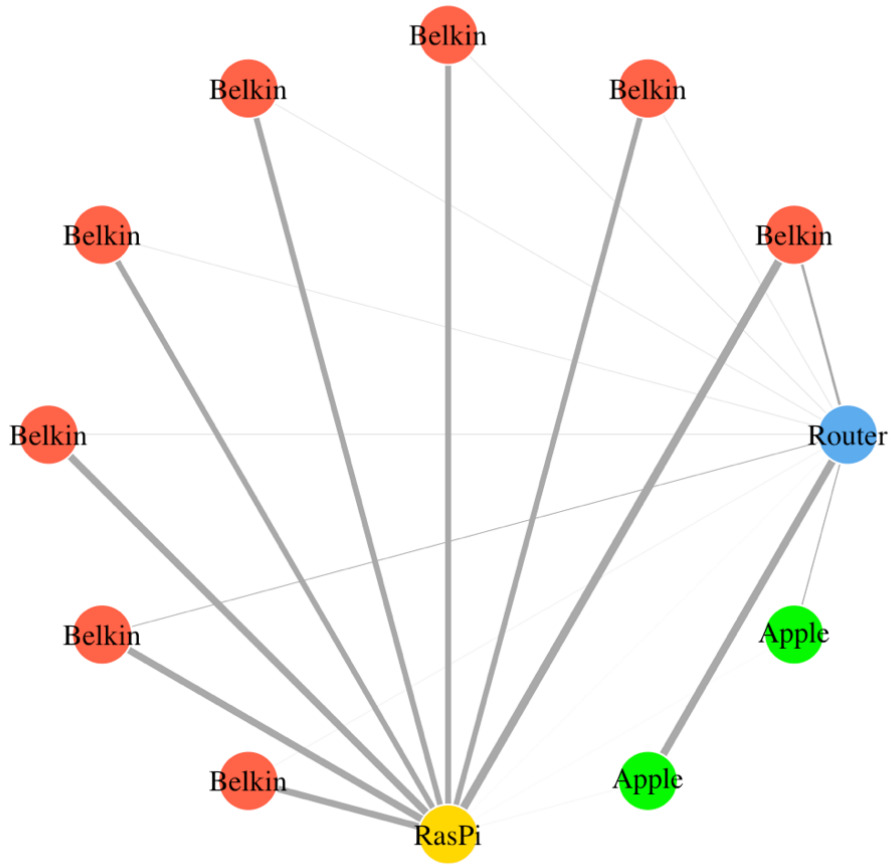


Figure 28. Network mapping of smart home architecture

1.4.8 Security.

Security of the tools presented above is not key to this work, but still worth discussion. The tool's interaction with the smart home and any other Wi-Fi or BLE devices is completely passive. The sniffers do not associate with the smart home's access point nor do they transmit any data that is detectable. Smart home operation is completely unaffected by the tools presented.

1.5 Mitigation of Internet of Things Leakage (MIoTL)

The MIoTL tool provides methods for mitigating data leakage from IoT devices in smart homes. As shown in Figure 29, MIoTL has two components which operate within the SHAA to negate capabilities provided by the CITIoT tool using the concept of chaff presented in Chapter 2. The first component, Device Shadow, spoofs device traffic to make it hard to classify devices and identify device events. The second component, MAC Shadow, spoofs traffic coming from a user’s device to make it difficult to track when devices are present in the smart home. The tool operates on a Raspberry Pi 3B with the Kali X operating system. The Raspberry Pi was used because the tool is always running and the Pi provides low power consumption and constant connectivity on the network.

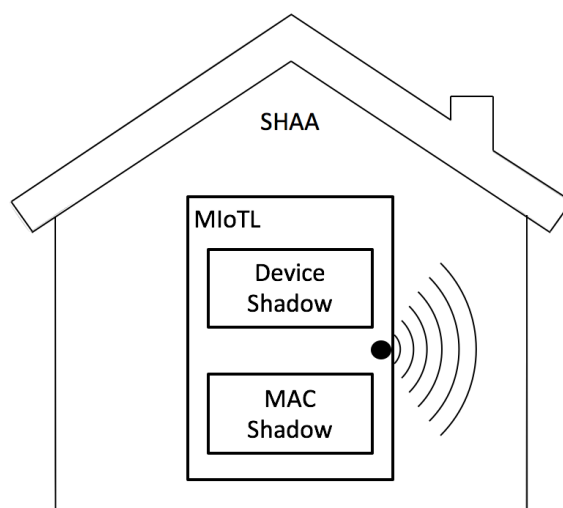


Figure 29. Diagram of MIoTL tool components

1.5.1 Device Shadow.

The Device Shadow script is presented in Appendix X and was written with Python using the Scapy network tool to randomly spoof packets sent between devices in the smart home. Three different packet groups are randomly sent: (i) a

packet of 620 bytes sent from the Raspberry Pi to every other device in a random order; (ii) a series of packets totaling 10,000 bytes sent from each device in a random order to the router; and (iii) a series of packets totaling 100,000 bytes sent from each device in a random order to the router. The component randomly sends one of these packet groups at a random interval between ten and fifteen minutes. The random order and interval are used to make it hard to differentiate real traffic from spoofed traffic sent from the tool. Packets are sent to and from each device to make it hard to identify devices. For example, per the above criteria, a packet of size 620 bytes sent from the Raspberry Pi to a device indicates that the device is an outlet. This component spoofs packets of 620 bytes from the Raspberry Pi to each device, regardless of type, so the classifier tool would incorrectly classify a sensor as an outlet. The same concept is used in spoofing events.

1.5.2 MAC Shadow.

The Mac Shadow script is shown in Appendix X and was written with Python using Scapy to spoof packets sent from a device to a controller. To accomplish this, the script first checks to see if the device is present in the smart home every five minutes. If a device is not in the smart home, then, at a random interval between 0 and 1 seconds, it sends ten packets with a spoofed source MAC address of the device to the controller. The script checks to see if the device is present to make sure packets are sent on behalf of the device only when the device is not part of the network (i.e., the user is away from the home). This occurs to ensure the Address Resolution Protocol (ARP) table is not changed while the device is on the network.

Checking if the device is on the network proved to be difficult as the Apple iPhone's network card goes into a low power mode when the phone is not in use and does not respond to ping messages. To check if the device is on the network, the script

accomplishes a series of pings that work together to wake up the network card and get a response from the phone. First, an ICMP ping is used then an ARP request is sent ten times. If the device responds to one of the pings then it is present. Documentation on how the iPhone's network card operates was not readily available so trial and error was used and the ARP request method was found to be the most consistent to detect devices.

1.6 Design Summary

This chapter describes the individual components of the SHAA, CITIoT, and MIoTL systems. The design is a unique approach to creating a smart home testbed that can be used to analyze IoT data leakage and test mitigation methods.

To do...

- ☑ 1 (p. 3): might want to move the part about the logs to Ch4.
- ☑ 2 (p. 27): Show plots with each device
- ☑ 3 (p. 27): explain why the characteristic packet sizes exist
- ☑ 4 (p. 27): Add device list to btleparser script
- ☑ 5 (p. 31): On RasPi so that home all of the time, low power consumption, etc

Bibliography

1. Bluetooth SIG. *Specification of the Bluetooth System Core Version 4.2*, 2010. [Last accessed on August 30, 2017], www.bluetooth.com/specifications/bluetooth-core-specification.
2. devbobo. “homebridge-platform-wemo,” 2017. [Last accessed on August 30, 2017], Available at npmjs.com/package/homebridge-platform-wemo/.
3. Raspberry Pi Foundation. “Raspberry Pi 3 Model B Specifications,” 2016. [Last accessed on August 30, 2017], Available at raspberrypi.org/products/raspberry-pi-3-model-b/.
4. nfarina. “Homebridge,” 2015. [Last accessed on August 30, 2017], Available at github.com/nfarina/homebridge/.