

# **BEHAVIOUR MODELLING AND ANOMALY DETECTION IN SMART-HOME IOT DEVICES**



# **BEHAVIOUR MODELLING AND ANOMALY DETECTION IN SMART-HOME IOT DEVICES**

by

**Sandesh MANGANAHALLI JAYAPRAKASH**

in partial fulfilment of the requirements for the degree of

**Master of Science in Computer Science**  
Specialisation: Cyber Security

at the Delft University of Technology,  
Faculty of Electrical Engineering, Mathematics and Computer Science,  
to be defended publicly on Thursday July 11, 2019 at 3:00 pm.

Student Number: 4724127

Project duration: November 2018 to July 2019

Thesis committee:

Dr. Ir. S.E. Verwer,	Assistant Professor,	Cyber Security group, TU Delft
Dr. Ir. J.C.A van der Lubbe,	Associate Professor,	Cyber Security group, TU Delft
Dr. Ir. Asterios Katsifodimos,	Assistant Professor,	Web Information Systems group, TU Delft
Ir. Azqa Nadeem,	PhD Student,	Cyber Security group, TU Delft

An electronic version of this dissertation is available at  
<http://repository.tudelft.nl/>





# ABSTRACT

The usage of Internet of Things (IoT) devices has been exponentially increasing and their security is often overlooked. Hackers exploit the vulnerabilities present to perform large scale attacks as well as to obtain privacy-sensitive information. Resource constraints combined with a lack of incentives for manufacturers makes it harder to implement security solutions part of these devices.

This thesis aims at developing a system that monitors the behaviour of these IoT devices. Network traffic is captured and analysed as part of a network middle-box to model the behaviour of an IoT device. This traffic shows the interactions of the IoT device with other devices and hosts. By modelling the normal behaviour of a device, we can detect anomalies exhibited.

Denial of Service attack was performed to evaluate the effectiveness of state machines in detecting anomalies. To verify the validity of state machines built based on network traffic in a laboratory setup, a test environment with a different setting was used. Traffic was captured from a smart home setting and used to validate the state machines.

We show that state machines can be effectively used to model the behaviour of IoT devices at the packet level and can also be used to uniquely identify commands issued from smartphone to IoT device. They can also effectively distinguish attack traffic from normal traffic.



# CONTENTS

<b>Preface</b>	<b>11</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	2
1.2 Proposed Solution . . . . .	2
1.3 Dataset . . . . .	2
1.4 Motivation for choosing IOT devices . . . . .	3
1.5 Motivation for choosing Philips Hue . . . . .	3
1.6 Research Questions . . . . .	6
1.7 Research Scope . . . . .	7
1.8 Contributions . . . . .	7
1.9 Summary of Results . . . . .	7
1.10 Report Outline . . . . .	8
<b>2 Background and Literature Survey</b>	<b>9</b>
2.1 Definitions . . . . .	9
2.1.1 Internet of Things . . . . .	9
2.1.2 Intrusion Detection System . . . . .	9
2.1.3 State Machines . . . . .	10
2.1.4 SPIN Device . . . . .	11
2.2 Related Work . . . . .	11
2.2.1 Homogeneous Devices . . . . .	11
2.2.2 Heterogeneous Devices . . . . .	17
2.2.3 Other related work . . . . .	20
2.2.4 Observations . . . . .	20
2.2.5 Research Gaps . . . . .	22
<b>3 Data Exploration</b>	<b>25</b>
3.1 Experimental Setup . . . . .	25
3.2 Data Collection . . . . .	25
3.3 Data Filtering . . . . .	27
3.4 Observations from the Data . . . . .	28
3.5 Features . . . . .	28
3.5.1 Selected Features . . . . .	30
3.5.2 Other Features Considered . . . . .	31
3.6 Summary . . . . .	34

<b>4</b>	<b>Methodology</b>	<b>35</b>
4.1	State Machine Learning Module . . . . .	35
4.1.1	Input Format. . . . .	35
4.1.2	Parameters to learning module . . . . .	36
4.1.3	Output Format. . . . .	37
4.2	Representation of State Machines. . . . .	37
4.3	Initial State Machines: Traffic between mobile app and Hue . . . . .	37
4.3.1	Case1: All traffic was considered . . . . .	37
4.3.2	Case2: Different Data with some filtering . . . . .	38
4.4	Final State Machines: Traffic between Mobile app and Hue . . . . .	41
4.4.1	Inspection . . . . .	41
4.5	Ordering of TCP stream and its significance on Background Traffic by Hue . . . . .	45
4.5.1	TCP packets re-ordering and sequence generation. . . . .	48
4.6	Final State Machines: Background Traffic generated by Hue . . . . .	50
4.7	Extension to another device: IKEA lights . . . . .	52
4.7.1	Traffic between smart phone and IKEA lights . . . . .	52
4.7.2	Background Traffic generated by IKEA lights . . . . .	52
4.8	Baseline: N-Grams . . . . .	53
4.9	Answers to research questions addressed in this chapter . . . . .	53
4.10	Summary . . . . .	54
<b>5</b>	<b>Results: State Machines representing the normal behaviour</b>	<b>55</b>
5.1	State Machines representing Traffic between Mobile app and Hue . . . . .	55
5.2	State machines representing Background Traffic generated by Hue . . . . .	59
5.3	State Machines representing traffic generated by IKEA lights . . . . .	63
5.3.1	Traffic between smartphone and IKEA lights. . . . .	63
5.3.2	Background traffic generated by IKEA lights . . . . .	67
5.4	Answers to research questions addressed in this chapter . . . . .	69
5.5	Summary . . . . .	70
<b>6</b>	<b>Application of State Machines on Attack Traffic and Real World Data</b>	<b>71</b>
6.1	Case Study 1: Attack Traffic . . . . .	71
6.1.1	Attack type and tools used . . . . .	71
6.1.2	IoT device as victim . . . . .	72
6.1.3	IoT device as source of attack . . . . .	75
6.2	Case Study 2: Real World Data . . . . .	76
6.2.1	Setup. . . . .	76
6.2.2	Hypothesis. . . . .	76
6.3	Summary . . . . .	79
<b>7</b>	<b>Evaluation</b>	<b>81</b>
7.1	Tools used. . . . .	81
7.2	(Semi)online evaluation. . . . .	83
7.3	Offline Evaluation. . . . .	83
7.4	Evaluation Metrics: Attack Traffic . . . . .	84
7.5	Evaluation Metrics: Baseline with Attack Traffic. . . . .	84



---

7.6	Evaluation Metrics: Real World data. . . . .	86
7.6.1	Using Absolute Packet Size. . . . .	86
7.6.2	Using Threshold on Packet Size and mismatching transitions . . . .	86
7.6.3	False Positive Analysis . . . . .	86
7.7	Evaluation Metrics: Baseline with Real World Traffic . . . . .	89
7.8	Research Questions answered in this chapter . . . . .	90
7.9	Summary . . . . .	90
<b>8</b>	<b>Limitations and Future Work</b>	<b>93</b>
8.1	Data Related . . . . .	93
8.2	Approach Related . . . . .	94
8.3	Evaluation Related . . . . .	95
8.4	Interaction with other devices . . . . .	95
<b>9</b>	<b>Conclusion</b>	<b>97</b>



# ACKNOWLEDGEMENTS

I would like to first thank my supervisor Dr.ir. S.E. (Sicco) Verwer of the Cyber Security and Intelligent Systems department at TU Delft, for guiding me constantly.

I would like to thank my daily supervisor Azqa Nadeem (Ph.D. student and junior lecturer), for helping me throughout my journey with this thesis. She constantly steered me in the right direction when I was faced with a fair number of obstacles. Even though she provided me with useful feedback and suggestions, she saw that I never lost the originality of my work.

I would like to extend my thanks to Dr.ir. Elmer Lastdrager, Research Engineer at SIDN labs for providing me with the SPIN device which was very helpful with collecting the data in this thesis.

Finally and most importantly, I would like to express my heartfelt gratitude and thanks to my parents and friends for providing me the emotional support during my most difficult time of writing my thesis, believing in my hard work. This dream would not be possible without them.

*Dedicated to Friends and Family.*

*Sandesh Manganahalli Jayaprakash  
Delft, July 2019*



# 1

## INTRODUCTION

Internet of things(IoT) refers to the "things" that are interconnected to each other for the purpose of monitoring and exchanging information[44]. These can be anything, ranging from smart locks which let you control access to the home from your mobile phone, to smart refrigerators which can monitor the contents of fridge and place orders online based on the status. It has found its place in smart homes, health care and recently as autonomous vehicles. The devices we use on a daily basis can be connected to the internet, equipped with sensors or actuators to enable them to operate autonomously [29] thereby qualifying as an IOT device. As per the report of Cisco [9], the number of IoT devices has outnumbered World Population and is increasing exponentially. Around 50 Billion devices are expected to be connected to the internet by 2020 making a ratio of 6.58 devices per person. Owing to the availability of cheap hardware and open source software, developing these devices has been made simpler [29].

While these devices have made life easier, they raise many privacy and security concerns. A report by HP highlights the security issues that are prevalent in IoT devices <sup>1</sup>. Insecure Web interface, Weak Passwords, Insecure Network Services, Lack of Secure Update Mechanism are some of the crucial weakness of IoT devices as per OWASP top 10 issues related to IoT devices <sup>2</sup>. Various attacks have been performed exploiting the weak security of these devices. For instance, a report by BBC <sup>3</sup> warned about how attackers were able to obtain a live feed from Baby Monitors. Distributed Denial Of Service attack is the most prevalent attack performed by hackers by abusing these insecure IOT devices. Mirai Botnet, which was comprised of an estimated 100.000 insecure IoT devices was used to perform Denial of Service attack on DNS operator Dyn <sup>4</sup>. Popular websites such as Github, Netflix, Amazon, Twitter were affected because of this attack. Shortly after the attack, the source code of Mirai Botnet was made publicly accessible on Github

---

<sup>1</sup><http://d-russia.ru/wp-content/uploads/2015/10/4AA5-4759ENW.pdf>

<sup>2</sup>[https://www.owasp.org/index.php/OWASP\\_Internet\\_of\\_Things\\_Project](https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project)

<sup>3</sup><https://www.bbc.com/news/technology-30121159>

<sup>4</sup><https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/>

5.

Various cryptographic techniques have been proposed specifically for securing IOT devices [17] [16] [35] [34]. While these techniques might help in ensuring confidentiality and authenticity in network traffic, they cannot be used to protect against various attacks that these IoT devices could be a victim of [26]. Additional security measures to monitor such devices is required.

### 1.1. PROBLEM STATEMENT

Given the security issues and challenges in IoT devices, this thesis aims at providing a monitoring system that could be part of network middleboxes. This system first learns the normal behaviour of an IOT device and tries to find deviations from it. As IOT devices have relatively less functionality, our hypothesis is, the behaviour of such devices could lead us to a smaller, interpretable state machine. For this, all the network traffic from/to IoT device in question is considered. The problem statement addressed in this thesis is as follows:

Developing a method to learn the behaviour of IoT devices to perform anomaly detection using state machines.

### 1.2. PROPOSED SOLUTION

We propose a method to build state machines that can not only explain the traffic between IOT device and smartphone controlling it but also the interaction of IoT device with other hosts. To achieve this, we look at the network traffic generated by IoT devices in the TCP/IP protocol stack to learn its normal behaviour. Traffic will be captured while performing various operations on the IOT device, as well as when the IoT device is idle. To aid in capturing the traffic, SPIN device(a smart router) would be used. Further details about it is provided in Chapter 3.

These state machines are then compared against attack traffic to see how well they can classify attack traffic from normal traffic.

### 1.3. DATASET

Data used for this work is composed of various pcap files which contain the packets originated from/destined to IoT device under consideration. In addition to this, these pcap files also contain traffic generated by other devices in the network. Data used for this projects takes only network traffic generated by IoT device in the TCP/IP protocol stack. Communication using other protocols such as Zigbee, Bluetooth etc. has been considered out of the scope of this project and can be an interesting aspect for future work.

Figure 1.1 shows an example of data used for this project on Wireshark<sup>6</sup>(A tool used to capture and analyse network traffic.). Here the traffic has been filtered based on MAC

<sup>5</sup><https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/akamai-mirai-botnet-threat-advisory.pdf>

<sup>6</sup><https://www.wireshark.org/>

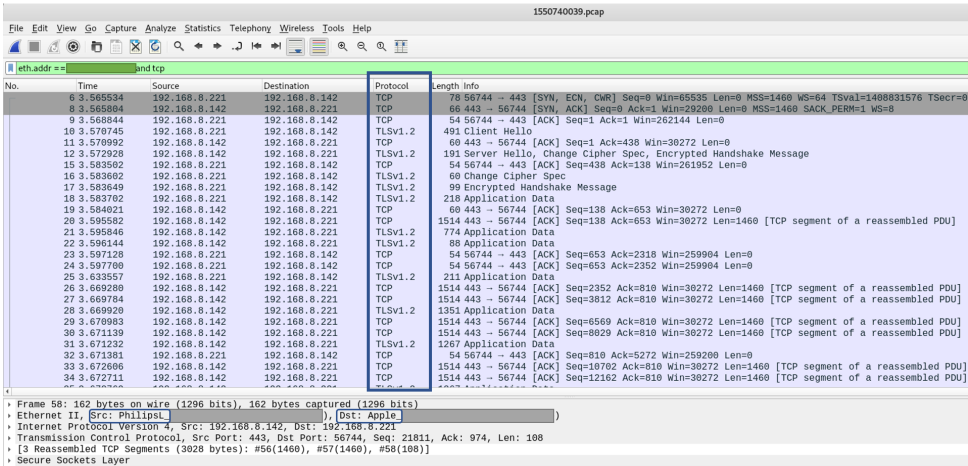


Figure 1.1: Screenshot from wireshark showing an example of the data used for this project. MAC address has been blurred for privacy reasons. It shows the traffic exchanged between Philips Hue and Apple smartphone. Highlighted protocol section shows that they use TLS over TCP for communication.

address of IoT device. The figure shows traffic exchanged between Philips Hue and Apple smartphone that was used to control it. Wireshark was able to find the name of the manufacturer of the device based on first few bytes of MAC address. This can be observed in the figure where it is highlighted. The protocol field in Figure 1.1 also shows that TLS over TCP is used for communication between the Philips Hue and smartphone. Details about how it is collected and filtered have been described in Section 3.2.

## 1.4. MOTIVATION FOR CHOOSING IOT DEVICES

Often laptops/PCs and smartphones are equipped with anti-virus software that monitors the devices for any kind of infections and security issues. IoT devices come with a challenge of limited resources to host such software. For this reason, we need a way through which these devices can be monitored from an external source such as gateway routers or network middleboxes.

IoT devices often are equipped with limited functionality. Our hypothesis is that these IoT devices would communicate with less number of endpoints for operations such as automatic updates, pings and for receiving commands from a smartphone. Therefore the data is more structured and predictable to represent and explain in the form of a state machine.

## 1.5. MOTIVATION FOR CHOOSING PHILIPS HUE

Given the range of IoT devices available, it was not possible to include all types of IoT devices if we aim at explaining the behaviour of the device at the packet level. For this reason, it was decided to focus on one device - Philips Hue.

Acar *et al.*[2] in their work look at the encrypted network traffic generated by IoT devices. They plot the number of bytes exchanged between the IoT device and smartphone when the smartphone was used to control and perform a certain operation on the IoT device. By only looking at the number of bytes exchanged per second, they were able to identify when commands were issued by smartphone to IoT device. Figure 1.2 shows the Rate(Bytes/Second) observed when commands are issued to Wemo insight switch to turn it On/Off. By looking at the peaks, it can be seen when some command was issued to the IOT device. In the case of Samsung SmartThings Outlet in Figure 1.3, we can see three peaks showing when a command was issued to IoT device. In Figure 1.4, although there is variation in the Rate at which bytes are exchanged, the peaks observed when a command was issued to lock/unlock the device reflected in the form of higher peaks.

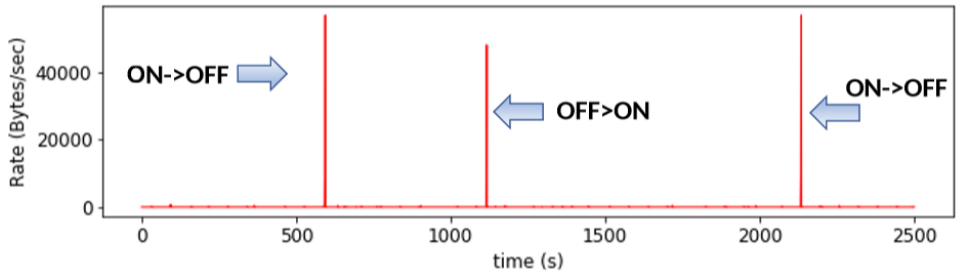


Figure 1.2: Bytes exchanged per second by the Wemo Insight Switch when commands were issued to turn it On/Off as observed by Acar *et al.*[2]. We can identify when some operation was performed by just looking at the peaks.

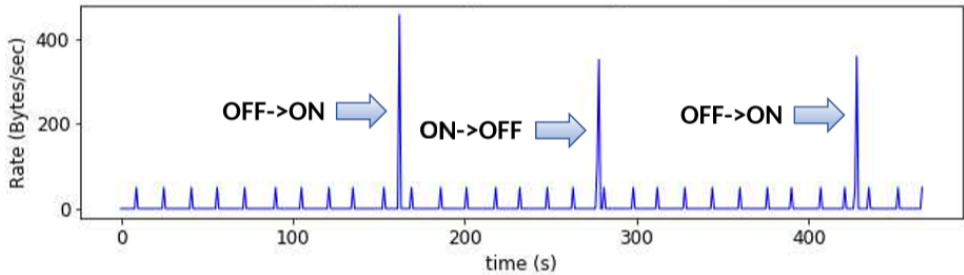


Figure 1.3: Bytes exchanged per second by the Samsung SmartThings Outlet when commands were issued to turn it On/Off as observed by Acar *et al.*[2]



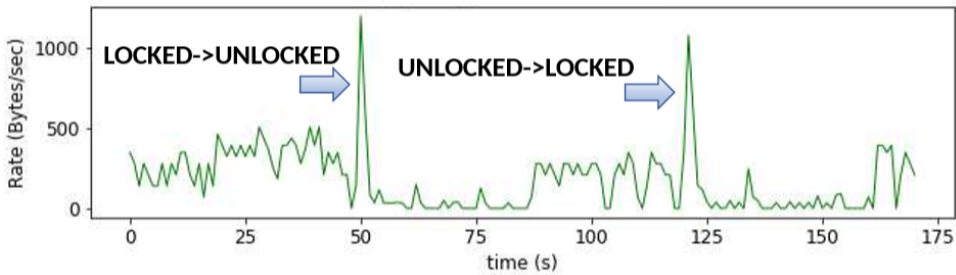


Figure 1.4: Bytes exchanged per second by August Smart Lock when commands were issued to turn it On/Off as observed by Acar *et al.*[2]

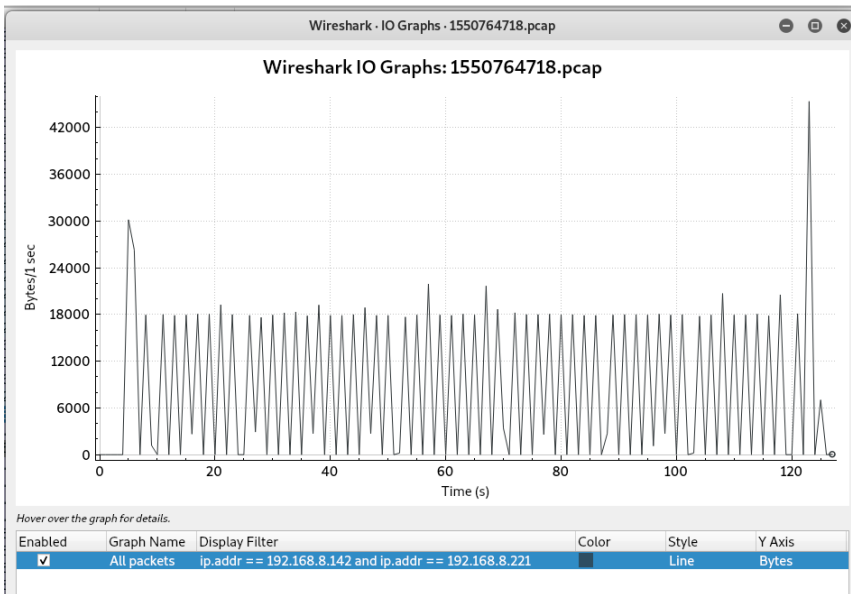


Figure 1.5: Bytes exchanged per second while controlling philips Hue from a smart phone. In this scenario, the operation of changing theme of the light was performed once in every thirty seconds for 4 times. The time window represented in this graph is for a total of two and half minutes. Unlike previous examples, here it was not straightforward to identify when certain operation was performed.

However, when traffic between Philips Hue and Mobile was captured while issuing commands, there were no clear peaks suggesting when a command was issued to the IOT device. Figure 1.5 shows the graph taken from Wireshark showing the rate at which Bytes exchanged. Here the data is filtered based on IP address of mobile phone and IOT device. A total of 4 commands were issued with an interval of 30 seconds between them. Although there is deviation with respect to bytes exchanged per second, it is not possible to directly identify when commands were issued from smartphone to IoT device.

*This graph provided the motivation to further look into network traffic generated by the*

*Philips Hue because here a large number of bytes are exchanged constantly even though fewer operations were performed.*

## 1.6. RESEARCH QUESTIONS

This work aims at developing state machines showing the network behaviour of IoT devices. The question is, how can we develop a representation of normal behaviour of the device using a state machine and validate its efficiency. All this while only looking at packet's metadata. This gives rise to our main research question.

RQ1: How can we perform anomaly detection based on network traffic generated by IoT devices using state machines?

To perform this, firstly we need to define the normal behaviour of an IOT device.

RQ1a: What are the different behaviours performed by an IoT device during its normal operation?

RQ1b: Which high-level features can be used to define the behaviour of IoT devices?

RQ1c: How can these high-level features be used to build a state machine?

Once we learn the normal behaviour, we need to look at the anomalies and see if we can detect the difference.

RQ2: How effective are state machines for detecting anomalies in network traffic generated by IoT devices?

There can be various operations performed by an IoT device. In case of smart lights, it can be actions such as turning it on/off, changing colours etc. Identifying this communication between smartphone and IOT device would be the next research question.

RQ3: Which commands issued by smartphone to IoT device can be detected using network traffic metadata?

Upon exposure to different test settings, the behaviour of an IOT device might or might not change. The next question is how well can these state machines adapt upon usage of an IOT device in a different setup.

RQ4: To what extent does the state machine adapt upon exposure to different test setting?

Previous research questions addressed a single IoT device. The next research question is about extension of the methodology to another IoT device.

RQ5: How can the approach be extended to represent behaviour of another IoT device?

## 1.7. RESEARCH SCOPE

This project considers the traffic generated by IoT devices in the TCP/IP protocol stack, specifically using TCP. Communication performed using UDP is not considered. All the traffic from the device is generated using Zigbee, Bluetooth and other such protocols are considered out of the scope of this project. In the case of Philips Hue, Zigbee was used for communication between Hue bridge and individual lights. This communication is out of the scope of this project. The traffic exchanged between the Hue bridge and smartphone as well as external hosts using TCP protocol were considered in this project.

**Assumptions held in this project:** The hardware and software of central router has not been compromised. All the traffic using TCP/IP protocol stack is routed only through a central router.

## 1.8. CONTRIBUTIONS

With regard to IoT devices, this thesis offers the following contributions.

- Proposal of a methodology to build state machines representing normal behaviour of an IoT device.
- Demonstrate that the methodology can be used to explain why a certain packet is marked as anomalous.
- A technique to perform re-ordering of TCP packets when they are received out of order and divide them into different conversations/sessions.
- We demonstrate the applicability of the state machines in a real-world setup where traffic is collected in an unsupervised setting.
- A comparison of the performance of state machines with a baseline (N-Grams).
- Extension of the approach to define normal behaviour of another IoT device.

## 1.9. SUMMARY OF RESULTS

We were able to build state machines representing the normal behaviour of an IOT device using network traffic generated by Philips Hue lighting system. Based on these state machines, we were able to identify the commands exchanged between smartphone and the device for actions such as Changing Colours, Turning On/Off and Changing Themes. A simple Denial of Service attack was performed to generate attack traffic. This traffic was then used to check the effectiveness of state machines in identifying attack traffic from normal traffic. State machines were able to identify attack traffic with an Accuracy of 99.94 %.

Traffic was captured in a smart home setting where the device was used in an unsupervised setting. This was when the device was in regular usage in someone's home for a duration of one week. State Machines were able to explain 97.82 % of the traffic generated in this scenario. The methodology was also used to model the behaviour of IKEA smart lights using state machines.

### 1.10. REPORT OUTLINE

Background knowledge and definitions pertaining the concepts used in this thesis have been explained in Chapter 2. It is combined with description of existing work in the field of anomaly detection of IoT devices and the observations made. Chapter 3 contains the details related to input data that is used in this thesis. It is followed by description of methodology used to build state machines from input data in Chapter 4. State machines that were obtained as a result of application of methodology are discussed in 5. Experimental setup and details regarding attack traffic and real world details are part of Chapter 6. Methodology used to evaluate the performance and the resulting metrics are discussed in Chapter 7. Chapter 8 consists of Limitations of this project and details about how it could be improved in future work. The report concludes in Chapter 9.

# 2

## BACKGROUND AND LITERATURE SURVEY

This chapter includes background information required to better understand the concepts used in this thesis. It is followed by a description of related work and observations made across them. The chapter ends with some research gaps observed.

### 2.1. DEFINITIONS

This section includes definitions and meaning of some keywords used throughout this work.

#### 2.1.1. INTERNET OF THINGS

Internet of things is a network of devices connected to the Internet. Unlike conventional laptops, personal computers connected to the internet, these devices have constrained memory and processing capabilities. These are heterogeneous devices each with different functionalities and requirements. Examples of such devices are Smart TV, Smart Refrigerator, Smart Lights, Smart Locks. In this thesis, Philips Hue Smart lights and IKEA smart lights are considered.

#### 2.1.2. INTRUSION DETECTION SYSTEM

Intrusion detection is the process of monitoring to find any possible threats. An intrusion detection system (IDS) is a software or a device which monitors the events to detect intrusions [29].

They can be classified into two types based on their placement strategy: Host-based Intrusion detection system and Network-based Intrusion detection system. In the case of host-based IDS, an IDS is placed inside each host and it monitors all the activities of that system. In Network-based IDS, it monitors a particular segment of network and checks for any possible intrusions.

IDSs use three types of methodologies to find intrusions: Signature-based, Anomaly

Based and Stateful protocol analysis. In the case of Signature-based, the observed characteristics of a system are compared with characteristics of known malicious entities. The advantage of such an approach is it involves very less false positives. A disadvantage of such an approach is it cannot detect unknown attacks and it requires labelled data. In Anomaly-based IDS, known behaviour of devices is learnt and any deviation from such behaviour is marked as anomalous. Advantage of such an approach is that it can detect unknown attacks and it doesn't require labelled data. Disadvantage include a high number of false positives. Stateful protocol analysis usually involves deep packet inspection. Here, the normal behaviour of an application protocol is compared with observed behaviour to find deviations [29]. It is similar to Anomaly-based approach, however, here the normal behaviour is based on predefined standards set by vendors or industry experts in that area <sup>1</sup>.

Intrusion detection systems can be placed inline or in a passive mode. In case of inline deployment, all the traffic to the device flows through the IDS. In passive deployment, IDS works on a copy of the actual traffic flowing to the device.

### 2.1.3. STATE MACHINES

A state machine, otherwise called as Finite Automata (FA) is useful when we try to model patterns in the behaviour of any system. It consists of an initial state of a system, followed by certain (optional) intermediate states and final states. The system moves from one state to another when certain events occur. Each event is referred to as 'symbol' and is represented as a transition from one state to another.

A formal definition of a state machine from the book "Introduction to theory of computation" by Michael Sipser [39] is as follows:

A finite automaton is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  where

- $Q$  is a finite set called the *states*,
- $\Sigma$  is a finite state called the *alphabet*
- $\delta: Q \times \Sigma \rightarrow Q$  is the *transition function*
- $q_0 \in Q$  is the *start state* and,
- $F \subseteq Q$  is the set of *accepted states*.

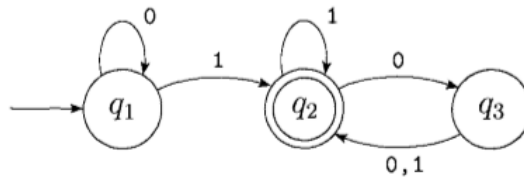


Figure 2.1: A state machine with three states  $q_1, q_2, q_3$  and transitions between them.

<sup>1</sup><https://sites.google.com/site/idpsinfo498/home/common-detection-methodologies/stateful-protocol>

Figure 2.1 gives an example of a state machine [39]. In the above example,  $q_1$  is the initial state,  $q_2$  is the final state. Transitions are represented by arrow marks between the states. Let us assume the state machine represents a system which moves from state to state upon receiving inputs. In the above example,  $q_1$  is the initial state of the system. Upon receiving 1 as input, the system moves from  $q_1$  to  $q_2$ . However, upon receiving an input of 0, the system remains in the same state.

#### 2.1.4. SPIN DEVICE

SPIN stands for Security and Privacy in In-home networks and is focused on the protection of IoT devices in the home network. The system [12] works on network level (analyses network traffic), and hence can be used to monitor different types of IOT devices. Its features include:

- It can be easily deployed in a home network by replacing home router with the plug and play SPIN device.
- It preserves privacy since all the processing and operations are handled within the home network.
- Provides complete configuration control to the user. For example, user can control which devices to monitor, which traffic to block etc
- It provides a graph showing the devices in the network and the hosts it is communicating with. User can block certain traffic, or in case of known malicious hosts, they are automatically blocked by SPIN device. Figure 2.2 shows a screenshot of network monitoring graph taken from the documentation of SPIN<sup>2</sup>.

## 2.2. RELATED WORK

In this section, a brief description of various works performed in the field of Intrusion Detection Systems for Internet of Things is presented.

### 2.2.1. HOMOGENEOUS DEVICES

An IoT system can have all devices which are similar in their structure and working mechanism. In such a scenario, all devices in the system should adhere to same working model. A best example of such a system would be wireless sensor networks, in which all sensors perform same set of actions and have similar hardware and software components. These devices are made accessible by internet thus exposing them to attackers. Multiple approaches have been taken to detect anomalies in such a system. They have been categorised based on the technique on which an element is considered as malicious or anomaly.

#### GENERALISED APPROACH/ARCHITECTURE

Some authors provide a generalised approach or architecture which can be implemented to monitor IoT devices.

<sup>2</sup><https://valibox.sidnlabs.nl/pages/userguide.html>

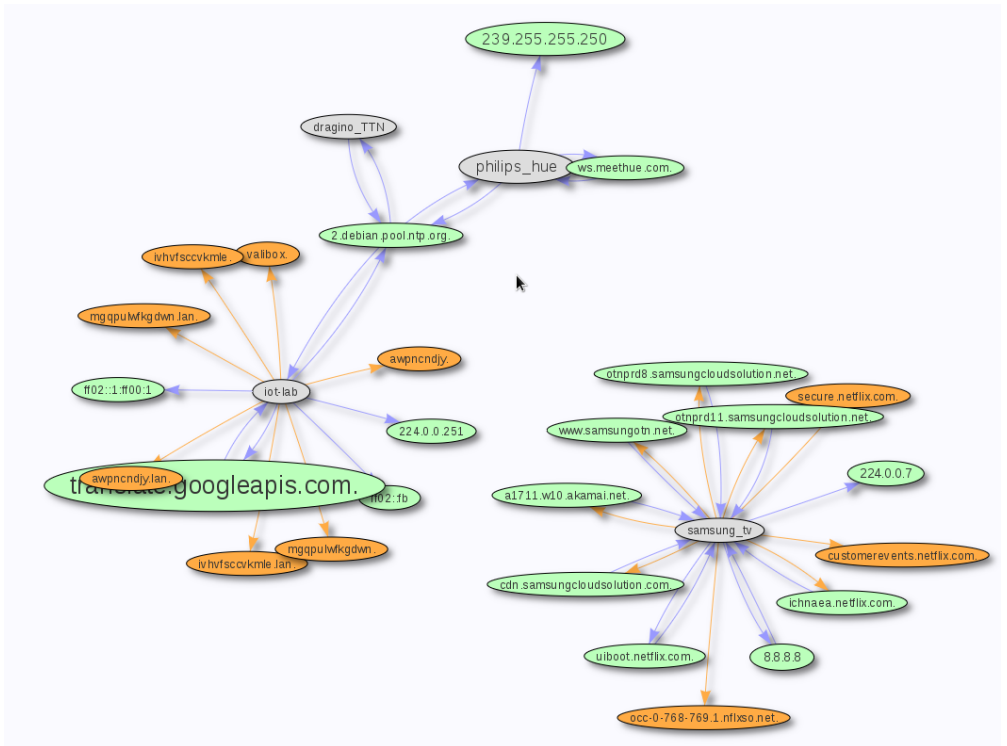


Figure 2.2: Screenshot of network monitoring graph of SPIN device. Grey nodes represent the devices in the network. Green nodes represent hosts with which the device is communicating. Orange nodes represent DNS queries performed by the device.



The paper by Thanigaivelan *et al.* [44] provides an overview of a generalised approach for anomaly detection. It shares the load of detection between nodes and centralised root node which is border router. Each node monitors its one hop neighbours and sends a control message called DPO(Distress Propagation Object) to its parent in case it finds its neighbour anomalous. The system is divided into three parts in each node. Monitoring and Grading Subsystem (MGSS) gathers the details of communication, analyses and grades the nodes. Reporting Subsystem(RSS) takes care of sends DPO messages to its parent node. Isolation Sub-system (ISS) handles dropping/allowing packets from other nodes based on the content of a repository which is shared between all three sub-systems. MGSS and RSS are part of Network Layer(L3) whereas ISS is part of Link layer(L2). The final decision of whether a node is anomalous or not is performed by edge router. Most resource intensive tasks are implemented as part of edge router.

The work of Butun *et al.* [5] highlights the security issues that arise when data collected by large number of IOT devices is uploaded to cloud and processed as big data. A brief survey on types of existing approaches that deal with anomaly detection for wireless sensor networks is provided. A comparison of existing IDSs along with their detection technique and architecture is also included. Additionally, applicability of these approaches for IoT as well as Cloud is discussed. While some approaches are lightweight and applicable for IoT devices, they are not suitable for data heavy cloud systems. On the other hand, some approaches which are efficient on cloud, can be resource heavy and not applicable for IoT devices. A generic requirements for an Anomaly detection methodology is provided which would satisfy both the requirements of IoT as well as Cloud.

#### SIGNATURE BASED

Known malicious patterns are used as a reference to detect anomalies in case of Signature based approaches.

The proposed solution by Kasinathan *et al.*[14] integrates DoS protection and IDS system a part of network manager of EBBITS project [46]. The DOS manager receives alerts from IDS when there are signs of an intrusion. It later combines information(Packet dropping rate, interference rate) from other components in the network manager to verify these alerts from IDS. The proposed IDS is a network based IDS and it consists of multiple components called IDSP(IDS probe) . These IDSPs are included in the network and they all are connected to a centralised IDS. This communication link is wired so that the detection module is resistant against DOS attacks. In the implementation phase, only preliminary work has been completed. They use an open-source signature based IDS called Suricata [21] and the integration of IDS onto network manager of ebbits has been ignored.

#### PROTOCOL BASED

In some papers, authors use specific characteristics of the protocols used by IOT devices to detect malicious activities.

Wallgren *et al.*[48] provide an overview of few technologies used in IOT. They then demonstrate well known attacks in WSNs on the IOT device networks running on 6LoWPAN<sup>3</sup> and RPL as routing protocol. Simulated attacks were performed to observe the behaviour

<sup>3</sup>Compressed IPv6 to enable communication between resource constrained devices to the Internet

of these devices and seen if RPL protocol can counter these attacks without involvement of any IDS. It was observed that RPL could not handle many of these attacks. However, the "self healing" mechanism of RPL was able to handle Hello Flood attack with the help of the Constrained Application Protocol(CoAP)[38]. (A 'Hello' message is used when a new node joins the network. In 'Hello Flood' Attack, an attacker introduces himself as a neighbour to many nodes.) A lightweight heartbeat protocol is presented as a centralised IDS. In this protocol, an echo message is sent from Border router to all devices in its network. The authors assume that IPSec is implemented and claim that IPSec is mandatory in IPv6. Since IPSec is implemented, the infected node will not be able to distinguish normal messages from these echo messages. So this can be used to detect selective forwarding attack. In selective forwarding attack, malicious nodes selectively forward packets to create issues with routing path. This attack combined with Hello Flood can be used to create distortion in the traffic exchanged between nodes. A malicious node can advertise itself as neighbour to multiple nodes, but upon receiving the packets to be forwarded to other nodes, they are dropped.

### THRESHOLD BASED

In threshold based approaches, certain aspects of the nodes and system are monitored and if they are beyond a certain threshold, then it is marked as anomalous or an intrusion.

Raza *et al.*[33] aims and designing and implementing an Intrusion Detection System for IOT devices that communicate using IPv6. Primarily, their work is on detection of sink-holing, spoofed or altered information and selective-forwarding attacks. The system is divided broadly into two parts. The resource heavy modules are placed in the Border Router and lightweight modules are placed in nodes. In addition to intrusion detection, it consists of a mini-firewall to prevent attacks externally from internet. To avoid wrong rank propagation by malicious nodes, a node is considered faulty if the number of disagreements it has with other nodes is above a threshold. To deal with selective forwarding attacks, the router keeps track of last message received from a node and if it is above a threshold, the node is marked as not online. Router will be assigned same number of ip addresses as the number of nodes in the network to ensure attacker is unaware of differentiating messages from other nodes and the router.

Cho *et al.*[7] presents a centralised approach in 6LoWPAN network to detect botnet attacks. The detector would be placed inside 6LoWPAN gateway(router). It monitors all the nodes of which data from the packet would be stored in the gateway. The monitor consists of 4 modules: Control field check module calculates the sum of TCP control field. Packet length Check calculates average packet length during a connection. Activity check module, counts the number of connections maintained by a sensor node. Bot Analysis module collects details from all these 3 modules. If these values for a particular node is lesser or greater than average value of other nodes by a certain threshold, then it raises an alert. To reduce overhead in gateway, they recommend usage of sampling methods.

Misra *et al.*[28] create a routing protocol called DLSR to defend server against DDoS attacks in a Wireless Mesh Network. Detection and prevention of DDoS attacks require heavy sampling of network data [25]. Each node is assumed to have a sampling budget, which is the maximum number of packets that can be sampled per unit time. To

accomplish optimal sampling for each node, Learning Automata is used as an intelligent system. DDoS detection of their work consists of three phases: In DDoS detection phase, server analyses the incoming traffic and if it is beyond a certain threshold, a special message called DALERT is sent to all the nodes. In the next phase, Attack identification phase, all the nodes monitor their traffic and based on its findings, if any host is found to be sending substantially high requests than others, it creates Attacker Information Packet (AIP) with the identity of malicious node to other nodes. Upon reception of AIP, all other nodes move on to the next phase, which is DDoS defence phase. In this phase, all nodes drop the packets of malicious nodes.

The proposed method by Pongle *et al.*[31] is on detection of wormhole attacks in IOT devices. It is a hybrid approach, in which centralised components are placed in 6BR(Border Router) and distributed modules are placed in individual nodes. It makes use of Received signal strength and distances to detect attacks. The central node has the capability of computing distance between nodes based on received signal strength. The assumption held here is that in the first five minutes, no attack is committed and all nodes make note of their (legitimate) neighbours. Whenever a node gets a new neighbour, it notifies the border router about it through a special message. If the distance is more than transmission range, then a wormhole attack is detected and it triggers the monitoring algorithm to approximate the location of infected device.

Lee *et al.*[20] propose an IDS which monitors the power consumption of devices to detect attacks. Behaviour of nodes during transmission/receiving is taken into consideration to model energy consumption of a Nodes. The approach takes into account the energy consumption of a node during waiting period for a channel, sending a message, receiving a message and receiving the acknowledgment. Using these, it computes the total energy. This energy prediction is further calculated for Route over routing scheme and Mesh Under Routing scheme considering various probabilities and parameters. During testing phase, the energy consumption is sampled every 0.5 seconds and if the energy consumption of a node is higher than predicted energy consumption by a certain threshold, then it is marked as malicious.

#### SEQUENCE BASED

In the work of Summerville *et al.* [43], contents of packets are considered as a long sequence of bytes. This sequences are then modelled into windows with specific stride length. Then, each window will be treated as a sequence of n-grams. Features in the feature vector can have same value of n (of n-grams) or different values. Each feature will have the count of number of times the n-grams match the bit pattern for each window of bytes. This matching is performed with the help of a method that uses bit-wise AND operation along with a increment function that works based on certain conditions. Bit patterns to be matched are binary values but contain don't care positions. So, bits at these positions will not have any effect on matching function. Since this uses counts of occurrences, normal behaviour of the system can be implemented in the form of a look-up table. Values in this table can be used to determine whether a packet or a specific window in a packet is anomalous or not.

Their approach in the implementation and evaluation involved the following procedure. First, the most common type of n-grams(in their case bi-grams) were plotted on a histogram. From this, most occurring bi-grams were chosen and they were generalised us-

ing don't care positions to include more cases. It is then evaluated, and if the system fails in detecting certain attacks, additional feature vectors with the same  $n$ (of  $n$ -grams) or different  $n$  are added and evaluated.

## 2

### NEURAL NETWORKS

Canedo *et al.*[6] use Neural networks placed in Gateways of IOT to detect anomalies. Their data consisted of Device ID, Sensor value and Delay between transmissions. A value of 1.00 is predicted when the reading are normal, and a different value is predicted with it is anomalous.

### SVM BASED

Zhang *et al.*[49] propose a simple light weight SVM based outlier detection for homogenous network of sensor node. SVM used here is based on approach of Laskov *et al.*[19], where resource heavy quadratic optimisation problem of SVM is converted into linear optimisation problem. They achieve this by modification of geometric hyper sphere into quarter sphere centered at origin. The assumption here is that nodes in WSN are spatially and temporally correlated. Each node monitors the sensor measurements of its immediate neighbours. The normal behaviour for each node is modelled based on previous its measurements as well as measurements of the nodes' neighbours with a fixed window size. This is repeated after a fixed time interval. If a new measurement is not within the quarter-sphere of SVM, it is marked as anomalous.

### OTHER MACHINE LEARNING ALGORITHMS

Proposed solution by Lun *et al.*[22] involves anomaly detection on network traffic to detect DDoS attacks. Outliers are marked as anomalous with the help of Gaussian Mixture Model. Gaussian mixture constitutes of multiple Gaussian distributions. Each data point should ideally belong to one of these distributions(Components) with a high probability. In case of anomalies, this probability is very low or close to zero.

Their model [50] uses the three layer architecture of internet mentioned in [1] which is Application layer, Network Layer and Perception layer. The IDS is situated at the Network layer. PCA is used for dimensionality reduction and Softmax regression as classifier. KNN is also used as a classifier and compared with softmax regression with respect to efficiency. Experimentation was performed on KDD CUP 99 Data Set which is a labelled dataset used in IDSs and consists of 41 features. PCA is used and top 3, 6 and 10 dimensions are chosen and compared. It was found that the model was more efficient when 10 dimensions were used. It was also observed that KNN had better accuracy than softmax regression, but KNN took substantially higher time than softmax for classification on same dataset.

Souza *et al.*[42] propose usage of K-means algorithm to find outliers. They make usage of Mahout [30] open source machine learning library of Apache to deal with large amount of data associated with big-data. As part of learning algorithm, multiple clusters are created with their corresponding centroid. During testing phase, if the euclidian distance of an instance from any of the centroids is greater than its radius, then it is marked as an outlier. The algorithm is made part of IOT middleware called LinkSmart.

### 2.2.2. HETEROGENEOUS DEVICES

An IOT system can also have different types of devices each with different functionalities and are made up of different components. A good example would be of a smart home network where there can be multitude of devices. These devices are accessible by internet so that the user can control them and monitor them. Several authors have worked on mechanisms to detect anomalies of devices in such a system. Their work has been categorised based on the detection technique and a brief summary is provided here.

#### GENERALISED APPROACH/ARCHITECTURE

In case of Kalis [26], IDS is implemented as part of smart router. It is a network based IDS that doesn't target specific protocol or application and thus can cater to the demand of handling heterogeneous IOT devices. It is a knowledge driven IDS, meaning, it uses the features of network and rules out those attacks that are not possible in a particular type of network. It is adaptable because it can handle mobility of devices. This knowledge collection is done autonomously without user intervention and can also detect mobility of devices within the network. Architecture of the system consists of following components: Communication system speaks with the devices. It has sub components to deal with different protocols. Data source logs the traffic which is obtained by communication system component on a sliding window basis. Knowledge base stores the features of monitored devices. "Modules" component, consists of Sensing modules and detection module. Sensing modules monitor dynamic changes in the network. Detection module analyses traffic and detects attacks. In their approach, for each attack, there is a separate detection module. How these detection modules work and how they can detect unknown attacks hasn't been described. This issue has also been highlighted in [13].

Sivaraman *et al.*[41] suggest employing an external party called Security Management Provider(SMP) to safeguard IOT devices. The role of SMP can be taken by ISP or home router vendor or some other entity. This entity can offer "security as a service" to the end user. This work is more like a demonstration of importance of SMP for IOT security and privacy. For philips hue lighting system, since the communication between app and the device is encrypted, it is easy to obtain unauthorised access. SMP can invoke access control to ensure only legitimate clients use the system. To deal with roaming, a mobile app is installed on user's device to send heartbeat messages of its external ip address to include it into its access control list. For smoke-alarm, it was observed that the communication was encrypted. But depending on the servers it talks to, it was noted that the device was also logging to an external server, in addition to notifications in mobile. With their system, it provided users with an option to block logging events into remote server.

#### SIGNATURE BASED

Authors of this paper by Habibi *et al.*[10] propose a white-listing based approach to monitor and detect attacks in smart home IOT devices. They use VirusTotal<sup>4</sup> to check if an url is malicious. Its API is used to check the validity of DNS response thereby avoiding DNS poisoning attacks.

It has two modes of operation. In Real-time validation mode, any communication of a device with a new destination not in the device's whitelist, triggers a query to VirusTotal

---

<sup>4</sup><https://www.virustotal.com>

to obtain a report. If the report says it is not malicious, the request is allowed. In Maximum throughput mode, each new destination is verified against blacklist. If it is not found, it is added to whitelist. The auditor module periodically monitors the whitelist and checks with virustotal if any entry is malicious, and if yes, will add that to blacklist. One interesting thing worth mentioning in this work is, their approach works even when a device gets a firmware update and changes its behavior.

### PROTOCOL BASED

In the paper by Sivanathan *et al.*[40] authors propose detection of malicious activity in smart home IOT devices. They employ this by connecting gateway in the home network to SDN controller in cloud. Detection is performed by a module called analysis engine which communicates with SDN controller. Based on certain rules implemented in SDN controller, analysis engine gets the mirrored traffic of home gateway. Using their methodology, they were able to detect remote unauthorised access to IOT devices. This occurs when a malware in a device is already present in the network, tries to scan devices in local network through standard Simple Service Discovery Protocol (SSDP). It is followed by enabling port forwarding with the help of Universal Plug-n-Play (UPnP). This will allow an attacker to access the device from the internet. They employ their mechanism at flow level and not at packet level but still achieve good results.

### SPECIFICATION BASED

Serror *et al.*[37] propose specification based system which is centrally installed in a router to secure smart IOT devices. Each device will have a specification of required communication details. This can be collected from various sources and it is referred to as communication rules. For efficient filtering, they propose the usage of SDN. They also propose usage of flexible pattern matching using existing methodologies such as P4 and eBPF [4] [23]. Anomaly detection is performed based on monitoring of these devices mainly on packet characteristics and not deep packet inspection. It is also aided by usage of machine learning algorithms.

### NEURAL NETWORKS

Meidan *et al.*[24] use unsupervised learning approach to detect mirai and Bashlite botnet infections. They learn normal behaviour of devices by learning deep auto-encoder for each device. These autoencoders capture behavioural snapshots. During testing phase, if the model is not able to reconstruct a snapshot, then it is marked as anomalous. Normal behaviour of 9 devices is first captured to learn the behaviour of these devices. They were then infected with Botnets and observed if they can detect this activity. To deal with false positives, they used majority voting among instances in a moving window. Mirsky *et al.*[27] provide an online intrusion detection system using neural networks. It uses an ensemble of Neural networks called auto encoders and learns the normal behaviour of the system using unsupervised learning. The proposed system is centralised and can be installed in home routers. To avoid complexity associated with deep neural networks, each autoencoder in the system consists of three layers with a maximum of seven visible neurons at each layer. The system is evaluated on surveillance camera set. Various attacks were performed to attack the availability and integrity of the camera

system. Detection capability of this system is compared with offline algorithms and signature based IDS was used as baseline. Normalised RMSE errors from first  $k$  encoders is fed to a final (called output layer) encoder which output an Anomaly score.

### SVM BASED

The IDS proposed by Nobakht *et al.*[29] aims at securing smart IOT devices in home network. It works at network level, thereby making it not specific to any individual device or protocol. Attack they tried to detect was unauthorised access to a smart IOT device. IDS would be placed at controller of SDN architecture and all the monitoring would be done by an external entity having security expertise, making it a Security as service (SaaS) system. Whenever a new device is added to the network, it is added to Device manager component of IDS. Sensor element module captures the traffic between the monitored device and rest of the network. Useful features are extracted in Feature Extractor module. The detection unit module can have any machine learning algorithm to detect attacks. They demonstrate attack on Philips Hue lighting system. Because of weak security measures, they were able to obtain unauthorised access to the system. First, they capture the traffic in case of regular access to lighting system from a mobile app, next they capture the traffic between device running attack script and lighting system. Parameters which were different in both the cases were taken as features for Detection. SVM was used as a ML algorithm for detection of attacks.

Paper by Bhunia *et al.*[3] uses SDN in combination with machine learning algorithm SVM to detect attacks in IOT devices. They use a hierarchical model, where IOT devices are connected to SDN enabled switches, which are then connected to cluster SDN controller and then connected to master SDN controller. Cluster SDN controllers inform master controller in case of significant changes in the behaviour. Anomaly detection is performed at cluster SDN controller. The machine learning algorithm is provided with flow level and packet level features of IOT devices. It is also provided with known behaviour of various attacks. When an anomalous behaviour is observed, they can either restrict the flow rate of the device, or block the device or blacklist the source of attack. The system was tested with three types of attacks. In first case, IOT device is used as a target of TCP flooding, in second case, IOT device is used as source of ICMP flood attack, in third case, two IOT devices were used as source to perform DDoS attack on a target. They were able to detect attacks within few seconds, and were able to counter these attacks. It was observed that non-linear version of SVM provided better results than linear SVM.

### COMBINATION OF MULTIPLE ALGORITHMS

Doshi *et al.*[8] use multiple machine learning algorithms for detection of DDoS attacks originating from smart devices in a home network. Detection system is part of Middle-box such as a router/switch. For normal traffic, interaction is done with smart devices for ten minutes and pcap files are recorded. Attacks were simulated from a virtual machine running kali linux as source of attack, destination is a Raspberry pi running web server. Both stateful and stateless features were extracted. Attack data and benign data were combined and they used machine learning algorithms to detect malicious packets. Algorithms used were KNN, Linear kernel SVM, Decision tree, Random forest and Neural

networks. Among these, linear SVM performed poorly and Neural networks performed the best in terms of detection.

### 2.2.3. OTHER RELATED WORK

In this section a brief description of some Intrusion detection systems which are not designed specifically for IOT devices, but can still be relevant for this thesis is provided.

This paper by Sekar *et al.*[36] combines Specification based techniques along with Anomaly detection schemes. This enables them to combine the strengths of both the approaches and provides a scheme which can detect Novel attacks, as well as reducing false positives. Evaluation was performed based on its performance on its detection of probing and DOS attacks. The specifications are obtained based on description of protocols and are modelled into Extended Finite State Automata (EFSA). To deal with improper classification of packets, an abstract specifications of protocol are used. In such a case, the state machine accepts a super-set of these abstract specifications. One prime benefit claimed by authors is it simplifies feature selection. It limits the number of possible combinations of parameters by translating parameters of a sequence of packets into properties associated with each transition in state machine. For this, statistical properties of packet sequences are mapped to statistical properties of transitions of state machine. Each individual transition in a trace will have two types of properties. First, is about whether a transition is taken, and second, values of state variables in that transition. Based on this, they can find out about frequency of a transition and common values for state variables. For detection, the same statistics used for learning are computed and are compared with values of training data. An alarm is raised if the difference is above a threshold.

Torres *et al.* [45] try to learn the behaviour of botnets using LSTM networks, which is a special type of recurrent neural networks. They use the learnt profile to detect the traffic of unknown botnets. Behaviour of a connection is learnt as a sequence of states. By aggregating flows based on source-destination ip address, port and protocol, the behaviour of flow is encoded into a sequence of characters. Optimum length of this sequence of states and sampling is computed based on training data. Evaluation was performed by training with labelled data of a botnet. Labelled data of a different botnet with different distribution was used as test data.

### 2.2.4. OBSERVATIONS

While each work employs unique approach, there has been some similarities across most of the papers. In this section, some observations made in the process of literature survey are mentioned.

- Various machine learning algorithms have been implemented and analysis is performed predominantly at packet level.
- Majority of the authors perform unsupervised learning approach followed by simulated attack detection.
- There is an assumption in all the works that the software and hardware of router is not compromised.



- Another assumption, specifically with the systems having centralised detection mechanism is that all the traffic from the devices is routed through central network router.
- The usage of Software Defined Networking, so as to have a trusted third party monitoring devices has been suggested in multiple works.

#### Homogeneous Systems:

- A single model is learnt for a single device in the system and verified whether this model is followed across all the devices.
- Work on detection mechanism has been performed from as far as 2005[22].
- In Homogeneous systems, detection system can be placed in multiple configurations, each with its own advantages and disadvantages.
  - Centralised: In this configuration, detection module is usually placed in Border Router or at an external third party using Software Defined Networking (SDN).
  - Distributed: Here, each node monitors other nodes in the network for anomalies.
  - Hybrid: This is a combination of Centralised and Distributed configurations. Certain modules are placed in the router and some modules are implemented in the individual nodes.
- Attacks that most authors have focused on: Routing attacks, DoS attack on the individual nodes in the system.

#### Heterogeneous Systems:

- For Heterogeneous systems, a separate model is learnt for each device and verified.
- There has been a trend observed and it is worth noting that most of the research performed has been quite recent with majority of the papers being published in 2017-2018.
- Detection mechanisms are implemented mostly in a central system. Typically border router or at a separate third party using SDN.
- Attacks that most authors have focused on: Botnet infections, Denial Of Service attacks originating from the devices in the system to external world, Unauthorised Access.

### 2.2.5. RESEARCH GAPS

- 1 Interpretability: Limited work has been done on explaining why something is marked as anomalous.
- 2 Most of the detection mechanisms are performed at packet level leading to large number of false positives.
- 3 Many propose usage of SDN and suggest a third party providing Security as a Service. But this arises privacy concerns.
- 4 Limited work has been done to devise a generic Intrusion Detection scheme as they have aimed at detecting specific types of attacks.

In this thesis, research gap 1 which concerns interpretability and research gap 3 which concerns privacy have been focused on.

Table 2.1: Features considered and attacks performed in some of the papers part of literature survey.

Paper	Features used	Attacks Detected
Nobakht <i>et al.</i> [29]	Packet size, Inter-packet time interval	Unauthorised Access
Doshi <i>et al.</i> [8]	Packet size, Inter-packet time interval, Protocol	Denial of Service
Bhunia <i>et al.</i> [3]	no. of sent requests, no. of failed authentication attempts, source of requests, bandwidth consumption, device usage at different time periods	Denial of Service
Meidan <i>et al.</i> [24]	Packet Size, Packet Count, inter-packet time interval	mirai and Bashlite botnet infections
Mirsky <i>et al.</i> [27]	Packet size, Packet count, Inter-packet time interval	Denial Of Service, Man in the middle, Reconnaissance, Botnet Malware
Habibi <i>et al.</i> [10]	Whitelist and blacklist of domains	Raspberry Pi spoofs MAC address of IOT device and pings multiple domains outside the whitelist.
Koroniotis <i>et al.</i> [15]	Number of Bytes, Number of Packets, Number of Connections. All aggregated at various levels	Botnet from simulated IOT device using Kali Linux
Acar <i>et al.</i> [2]	timestamp,direction, packet length for each packet in a sequential form	Privacy Leakage
Serror <i>et al.</i> [37]	Communication rules: Ip address, port numbers, direction	-
Bhunia <i>et al.</i> [3]	no. of sent requests, no. of failed authentication attempts, source of requests, bandwidth consumption, device usage at different time periods	Denial Of Service
Sivaraman <i>et al.</i> [41]	Generalised Architecture	Unauthorised Access using access control rules



# 3

## DATA EXPLORATION

*This chapter begins with discussion about the experimental setup used to capture the data. It is followed by additional details of data collection, initial observations and feature selection.*

### 3.1. EXPERIMENTAL SETUP

As described in the previous sections, this thesis makes use of SPIN device (Valibox) by SIDN labs to capture network traffic which serves as our data. The experimental setup consists of SPIN device, Philips Hue Bridge with light, a smart phone running on IOS operating system and laptop running linux operating system. The home router was replaced by SPIN device<sup>1</sup>, Hue bridge was connected to it by means of a LAN cable. Laptop and smart phone were connected to SPIN device by means of a WiFi connection. This has been depicted in the Figure 3.1.

### 3.2. DATA COLLECTION

Post the setup mentioned in the previous section, the network traffic was captured in the form of pcap file and analysed. Since the SPIN device runs on openWRT<sup>1</sup>, which is a Linux based operating system, it could be accessed from the laptop by means of ssh<sup>2</sup>. tcpdump<sup>3</sup> and netcat<sup>4</sup> were used to capture and transfer pcap from the SPIN device to laptop.

Multiple pcaps were captured for each type of operation that can be performed on the Philips Hue lighting system. Some examples include capturing packets for a duration of two and half minutes and one of the following operation was performed for every 30 seconds.

---

<sup>1</sup><https://openwrt.org/>

<sup>2</sup><https://www.ssh.com/ssh/command/>

<sup>3</sup><https://www.tcpdump.org/>

<sup>4</sup><http://netcat.sourceforge.net/>

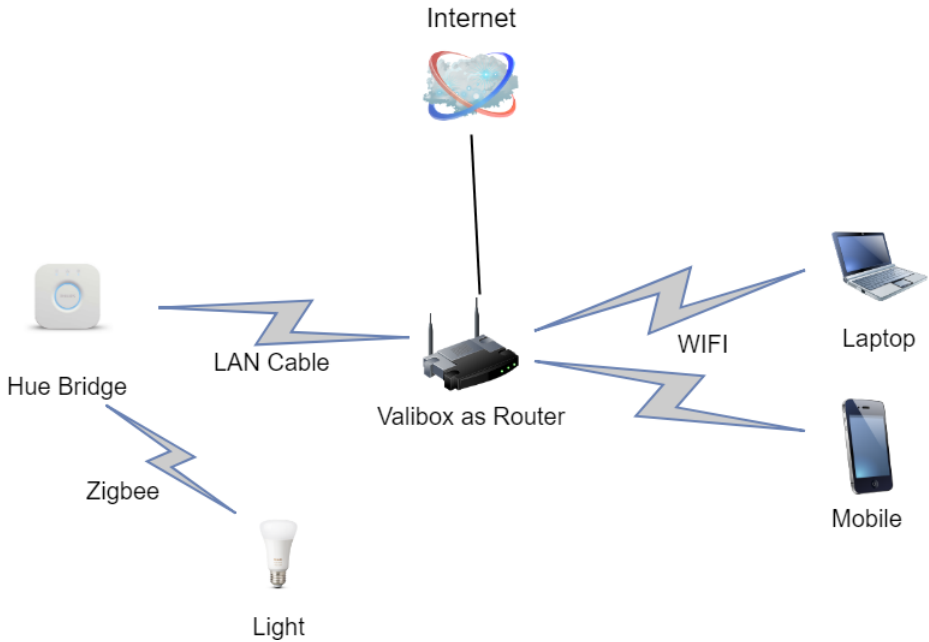


Figure 3.1: Experimental setup used to capture the traffic.

- Turning the Philips Hue light on/off. Figure 3.2 shows a screenshot taken from a mobile while performing this operation. Note that the app offers the option to perform this operation over three different places, only one of them is used. This is because the setup used to collect traffic in this project involves only one light.
- Changing colours of light as can be seen in Figure 3.3. The app allows us to choose colour of the light over a range of options.
- Changing themes that are offered by Philips hue. Figure 3.4 shows the interface provided by the app to change various themes available.

Multiple pcaps were captured while these operations were performed for a duration of 2 minutes to 10 minutes, where the time interval between them was 30 seconds. In some cases, the app was kept open in the smart phone without providing any instructions, just to capture the background traffic between app and the device.

It was also noticed that Philips Hue bridge was communicating with multiple other hosts with various protocols. For this, some pcaps were captured for duration of 2-3 hours to obtain the background information. It was also observed that there were few hosts to which the bridge was sending and receiving large number of packets in the first 3-5 minutes after connecting the bridge to power supply and internet. Multiple pcaps were also captured to accommodate this case.

It is important to note that in case of Philips Hue lighting system, the Hue bridge communicates with light using Zigbee protocol. In this thesis, this communication is

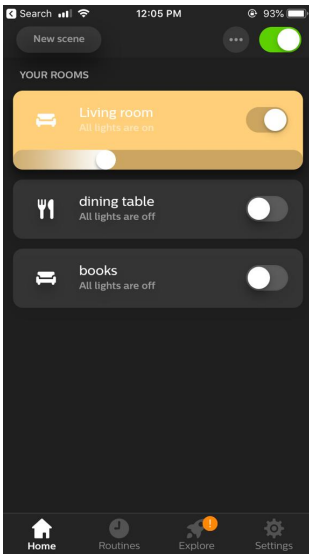


Figure 3.2: Screenshot of the app taken while performing On/Off operation on Hue Lights. Only one of the three is highlighted because the experimental setup used here involves only one light.

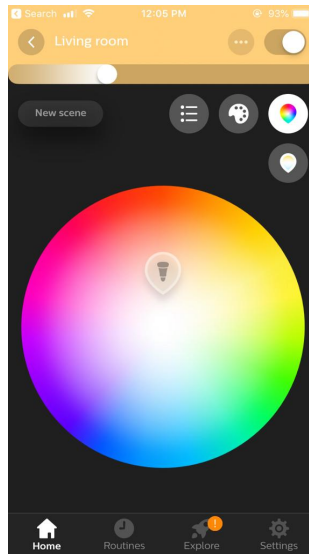


Figure 3.3: Screenshot of the app taken while choosing the colour of light from a colour palette. The cursor in the center of colour palette represents the current colour of lights.

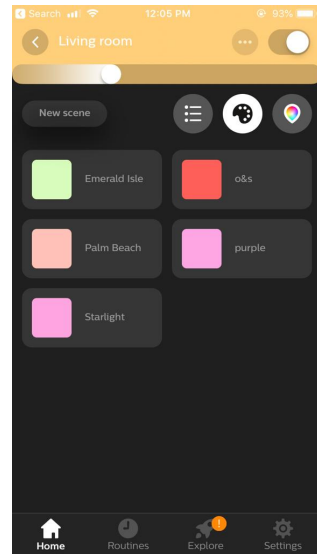


Figure 3.4: Screenshot of the app taken while choosing the theme of lights. In this picture, user can choose one of the five available options. Here none of the themes is selected and it is in default settings.

ignored. Only communication between the mobile containing app to control the system and the Hue bridge is considered. Throughout this report, any reference to the words Hue or Hue bridge would refer to the Hue bridge which communicates with the mobile app.

### 3.3. DATA FILTERING

The pcaps captured were also having other background traffic generated by the mobile phone and PC, to exclude them, all the traffic was filtered based on MAC address of Hue Bridge, and only those traffic sent/received from it is used for analysis.

Figure 3.5 shows the example of a pcap file containing data captured for this project. The screenshot is of Wireshark - a tool used to capture and analyse network traffic<sup>5</sup>, the traffic is filtered based on MAC address using Wireshark's filter.

Additionally, empty TCP Acknowledgements have also been filtered out. This is because the order in which these packets arrive is not always same. Communication between hosts is considered as a sequence of packets in this project. This sequence will get affected upon usage of empty acknowledgement packets. Hence they have been ignored.

<sup>5</sup><https://www.wireshark.org/>

No.	Time	Source	Destination	Protocol	Length	Info
738	46.889937	192.168.8.142	52.49.36.240	TCP	66	56164 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=8
739	46.911363	52.49.36.240	192.168.8.142	TCP	60	80 → 56164 [SYN, ACK] Seq=0 Ack=1 Win=20803 Len=0 MSS=1460 SACK_PERM=1 WS=256
740	46.915938	192.168.8.142	52.49.36.240	TCP	60	56164 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
741	46.911933	192.168.8.142	52.49.36.240	HTTP	1248	POST /NotificationService/RequestHandler.aspx HTTP/1.1 (application/cb-encrypted)
742	46.924993	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [ACK] Seq=1 Ack=1195 Win=29440 Len=0
744	47.067794	52.49.36.240	192.168.8.142	HTTP	813	HTTP/1.1 200 OK (application/cb-encrypted)
745	47.067915	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [FIN, ACK] Seq=760 Ack=1195 Win=29440 Len=0
746	47.068095	192.168.8.142	52.49.36.240	TCP	60	56164 → 80 [ACK] Seq=1195 Ack=760 Win=30720 Len=0
747	47.068898	192.168.8.142	52.49.36.240	TCP	60	56164 → 80 [FIN, ACK] Seq=1195 Ack=761 Win=30720 Len=0
748	47.074540	192.168.8.142	192.168.8.1	DNS	80	Standard query 0x0007 AAAA www.ecdinterface.philips.com
749	47.077995	192.168.8.1	192.168.8.142	DNS	251	Standard query response 0x0007 AAAA www.ecdinterface.philips.com CNAME dp.dc1.philips.com CNAME dc1-elb-52-101498
750	47.078283	192.168.8.142	192.168.8.1	DNS	88	Standard query 0x0008 A www.ecdinterface.philips.com
751	47.080375	192.168.8.1	192.168.8.142	DNS	338	Standard query response 0x0008 A www.ecdinterface.philips.com CNAME dp.dc1.philips.com CNAME dc1-elb-52-101498
752	47.086849	192.168.8.142	52.49.36.240	TCP	66	56165 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=8
753	47.089893	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [ACK] Seq=761 Ack=1196 Win=29440 Len=0
754	47.108741	52.49.36.240	192.168.8.142	TCP	66	80 → 56165 [SYN, ACK] Seq=0 Ack=1 Win=26884 Len=0 MSS=1460 SACK_PERM=1 WS=256
755	47.108975	192.168.8.142	52.49.36.240	TCP	60	56165 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
756	47.109289	192.168.8.142	52.49.36.240	HTTP	1466	POST /DevicePortallCPRequestHandler/RequestHandler.aspx HTTP/1.1 (application/cb-encrypted)
757	47.116778	192.168.8.142	216.239.35.12	NTP	90	NTP Version 4, client
758	47.117129	192.168.8.142	216.239.35.4	NTP	90	NTP Version 4, client
759	47.122163	216.239.35.12	192.168.8.142	NTP	90	NTP Version 4, server
760	47.131238	52.49.36.240	192.168.8.142	TCP	54	80 → 56165 [ACK] Seq=1 Ack=1413 Win=29952 Len=0
761	47.135679	52.49.36.240	192.168.8.142	TCP	1514	80 → 56165 [ACK] Seq=1 Ack=1413 Win=29952 Len=1460 [TCP segment of a reassembled PDU]
762	47.135796	52.49.36.240	192.168.8.142	HTTP	602	HTTP/1.1 200 OK (application/cb-encrypted)
763	47.135884	52.49.36.240	192.168.8.142	TCP	54	80 → 56165 [FIN, ACK] Seq=2089 Ack=1413 Win=29952 Len=0
764	47.136093	52.49.36.240	192.168.8.142	TCP	60	56165 → 80 [ACK] Seq=1413 Ack=1461 Win=32120 Len=0
765	47.136123	192.168.8.142	52.49.36.240	TCP	60	56165 → 80 [ACK] Seq=1413 Ack=2089 Win=35040 Len=0
766	47.136379	192.168.8.142	52.49.36.240	TCP	60	56165 → 80 [FIN, ACK] Seq=1413 Ack=2910 Win=35040 Len=0
767	47.151592	216.239.35.4	192.168.8.142	NTP	90	NTP Version 4, server
768	47.158990	52.49.36.240	192.168.8.142	TCP	54	80 → 56165 [ACK] Seq=2010 Ack=1414 Win=29952 Len=0

Frame 769: 991 bytes on wire (7920 bits), 991 bytes captured (7928 bits) on interface  
 Ethernet II, Src: GuangLia\_02[...], Dst: PhilipsL\_0[...]  
 Internet Protocol Version 4, Src: 52.49.36.240, Dst: 192.168.8.142  
 Transmission Control Protocol, Src Port: 80, Dst Port: 56163, Seq: 4519, Ack: 2823, Len: 937  
 [2 Reassembled TCP Segments (2397 bytes): #728(1460), #729(937)]  
 Hypertext Transfer Protocol  
 Media Type

Figure 3.5: Screenshot from wireshark showing the data used for this project. MAC address has been blurred for privacy reasons.

### 3.4. OBSERVATIONS FROM THE DATA

Based on some preliminary analysis of the traffic generated by the Hue Bridge, following were some of the observations made.

- It communicates with the mobile app through TCP protocol. Every-time the app is opened, a TLS session over TCP is created and the TCP session ends when the user leaves the app. This was triggered by sending a packet with SYN-Flag from app to bridge, and when the user leaves the app, a packet with FIN flag is sent to close the session.
- It also performed multiple POST and GET calls with various hosts but it was not always periodic. Figure 3.6 shows statistics and details related to these HTTP calls. It can be seen from the figure that Philips Hue performs these HTTP calls to domains such as fds.dc1.philips.com, www.ecdinterface.philips.com, dcp.dc1.philips.com, diagnostics.meethue.com.
- The bridge used NTP, SSDP, DNS and IGMP protocol on top of UDP with various hosts which was periodic in most of the cases. Figure 3.7 lists the protocols used by Philips Hue as seen on Wireshark.

### 3.5. FEATURES

Multiple features were considered initially but it was later concluded that the better solution would be to use minimum number of features. This was mainly because the algorithm used in this thesis is State machines, more number of features would lead to more combinations of transitions which would lead to more states and bigger state machines. To make state machines easy to understand, it was important to choose fewer features



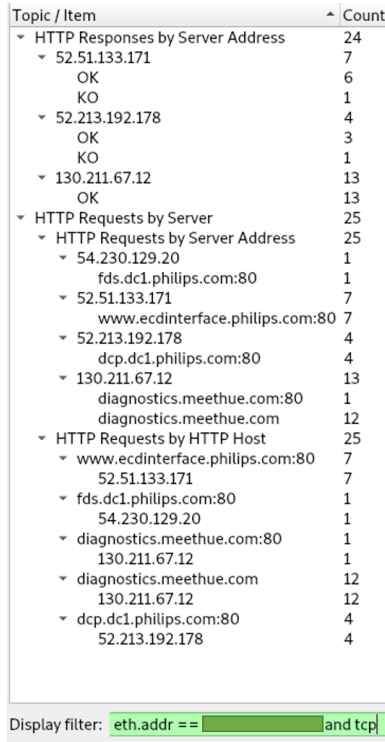


Figure 3.6: Screenshot from wireshark showing various stats related to HTTP calls performed by IOT device from one of the pcaps captured.

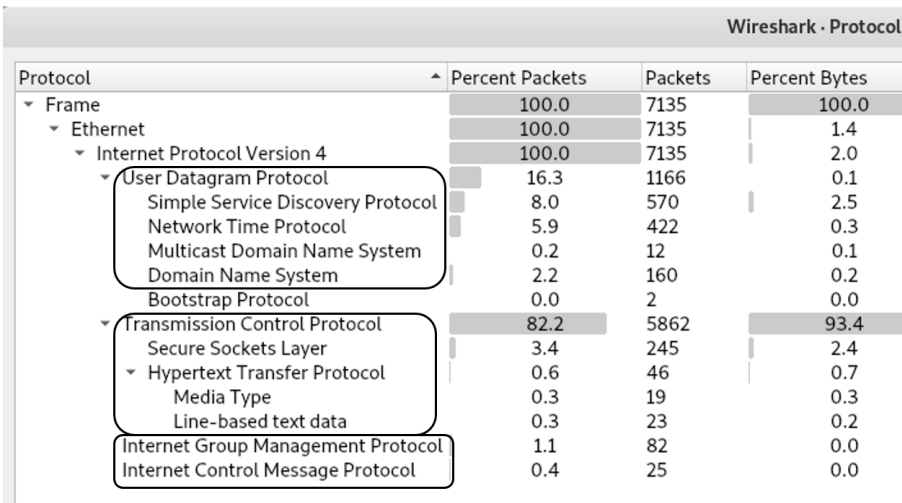


Figure 3.7: Various protocols used by Philips Hue as seen on Wireshark.

which capture the behaviour of the device as much as possible.

One of the important aspect of this thesis was to explain every packet that is sent/received from the IOT device, preferably in an online manner. In order to accomplish this, every packet had to be considered as a transition in state machine. For this, only packet-level features were considered and not flow level features. In case of flow level features, multiple packets will be aggregated based on some criteria and features will be extracted to represent these flows. Packet-level features are considered in this thesis because it provides us with fine-grained information. As we are using state machines which is a sequential model, aggregating the packets into flows will make us miss the temporal aspect of the data.

### 3.5.1. SELECTED FEATURES

Features on which the emphasis was laid were Protocol, Packet Size and inter-packet time interval. These three were predominantly used by multiple authors in the papers which are discussed in Literature Survey as seen in Table 2.1.

#### TCP FLAGS

As observed in the previous section, the Hue bridge communicates with various hosts using TCP protocol and in many cases, these sessions start and end in a reasonable time window. A TCP session was created between smartphone and Philips Hue everytime the app was opened. This session was closed when user leaves the app. So one of the features that was used was TCP flags in those cases where the bridge communicates using TCP protocol.

#### PACKET SIZE

Based on the literature in previous chapter, it was observed that packet size and inter-packet time interval was used prominently in case of stateless features. It was also observed that, with many hosts, the Hue bridge communicates with a fixed packet size. In this thesis, packet size was used as second feature .

As discussed in previous section, Philips Hue communicated with multiple hosts using HTTP. Figures 3.9, 3.11, 3.13, 3.15 represent the variation in packet size when Hue is communicating with these hosts. It could be seen that there is a regular variation in packet size. This highlights the importance of packet size as a feature.

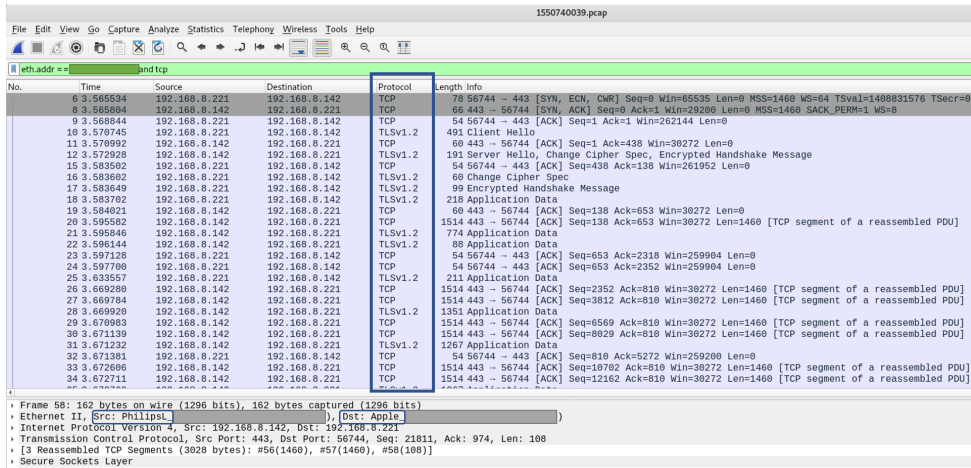
#### HTTP REQUESTS AND RESPONSES

As seen from Figure 3.6, Philips Hue communicates with some hosts using HTTP. Figure 3.5 shows example of three such HTTP post calls followed by their responses. So it was decided to include details related to it as part of the features. In case of HTTP requests from IOT device, the domain of GET/POST call was used as the feature. For HTTP responses, the response code was used as feature.

#### TLS HEADER MESSAGES

As seen in the Figure 3.8, Philips Hue uses TLS on top of TCP to communicate with mobile app of smartphone. The message field of TLS is used as one of the feature. This

could be messages such as 'Client Hello', 'Server Hello', 'Change Cipher Spec', 'Encrypted Handshake', etc. More details about TLS messages can be found here <sup>6</sup>.



No.	Time	Source	Destination	Protocol	Length	Info
6	3.565534	192.168.8.221	192.168.8.142	TCP	78	56744 → 443 [SYN, ECH, CWR] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1488831576 TSecr=0
8	3.565884	192.168.8.142	192.168.8.221	TCP	66	443 → 56744 [SYN, ACK] Seq=9 Ack=1 Win=29226 Len=0 MSS=1460 SACK_PERM=1 WS=8
9	3.569844	192.168.8.221	192.168.8.142	TCP	54	56744 → 443 [ACK] Seq=1 Ack=1 Win=252144 Len=0
10	3.570745	192.168.8.221	192.168.8.142	TLShv.2	491	Client Hello
11	3.570992	192.168.8.142	192.168.8.221	TCP	60	443 → 56744 [ACK] Seq=1 Ack=438 Win=30272 Len=0
12	3.572928	192.168.8.142	192.168.8.221	TLShv.2	191	Server Hello, Change Cipher Spec, Encrypted Handshake Message
15	3.583502	192.168.8.221	192.168.8.142	TCP	54	56744 → 443 [ACK] Seq=438 Ack=138 Win=261952 Len=0
16	3.583602	192.168.8.221	192.168.8.142	TLShv.2	60	Change Cipher Spec
17	3.583649	192.168.8.221	192.168.8.142	TLShv.2	99	Encrypted Handshake Message
18	3.583702	192.168.8.221	192.168.8.142	TLShv.2	218	Application Data
19	3.584021	192.168.8.142	192.168.8.221	TCP	60	443 → 56744 [ACK] Seq=138 Ack=653 Win=30272 Len=0
20	3.595582	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=138 Ack=653 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
21	3.595846	192.168.8.142	192.168.8.221	TLShv.2	774	Application Data
22	3.596144	192.168.8.142	192.168.8.221	TLShv.2	88	Application Data
23	3.597128	192.168.8.221	192.168.8.142	TCP	54	56744 → 443 [ACK] Seq=653 Ack=2318 Win=259904 Len=0
24	3.597780	192.168.8.221	192.168.8.142	TCP	54	56744 → 443 [ACK] Seq=653 Ack=2352 Win=259904 Len=0
25	3.633557	192.168.8.221	192.168.8.142	TLShv.2	211	Application Data
26	3.669280	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=2352 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
27	3.669784	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=3812 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
28	3.669920	192.168.8.142	192.168.8.221	TLShv.2	1351	Application Data
29	3.670983	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=6509 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
30	3.671139	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=8029 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
31	3.671232	192.168.8.142	192.168.8.221	TLShv.2	1267	Application Data
32	3.671381	192.168.8.221	192.168.8.142	TCP	54	56744 → 443 [ACK] Seq=810 Ack=5272 Win=259200 Len=0
33	3.672006	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=18702 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]
34	3.672711	192.168.8.142	192.168.8.221	TCP	1514	443 → 56744 [ACK] Seq=12162 Ack=810 Win=30272 Len=1460 [TCP segment of a reassembled PDU]

\* Frame 58: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface 0  
 \* Ethernet II, Src: Philips [redacted], Dst: Apple [redacted]  
 \* Internet Protocol Version 4, Src: 192.168.8.142, Dst: 192.168.8.221  
 \* Transmission Control Protocol, Src Port: 443, Dst Port: 56744, Seq: 21811, Ack: 974, Len: 108  
 \* [3 Reassembled TCP Segments (3028 bytes): #56(1460), #57(1460), #58(108)]  
 \* Secure Sockets Layer

Figure 3.8: TLS is used for communication between Philips Hue and Apple smart phone.

### 3.5.2. OTHER FEATURES CONSIDERED

Several other features were initially considered but later not used because of various reasons. In this section we discuss about these features and the reason for not including them.

#### INTER-PACKET TIME INTERVAL

Similar to packet size, Inter-packet time interval was also used by multiple authors in literature. Upon visual inspection of traffic, it was seen that the communication between Hue and smart phone was not periodic. It was also seen that various hosts with which Hue communicated was not periodic either. Figures 3.10, 3.12, 3.14, 3.16 shows inter-packet time interval when Hue is communicating with these hosts. It can be seen that the Jitter is low during first few packets and is higher in the later packets. This shows that more communication occurs in the first few seconds and the frequency decreases over time. These graph also symbolise that the inter-packet time interval is not periodic. Furthermore, interpacket time interval is also affected by network delays and might vary from location to location. For these reasons, interpacket time interval has not been considered as part of feature set in this project.

<sup>6</sup>[https://www.ibm.com/support/knowledgecenter/en/SSFKSJ\\_7.1.0/com.ibm.mq.doc/sy10660\\_.htm](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm)

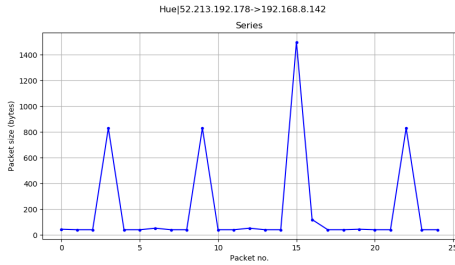


Figure 3.9: Variation in Packet size during a HTTP call and response between Hue and dcp.dc1.philips.com

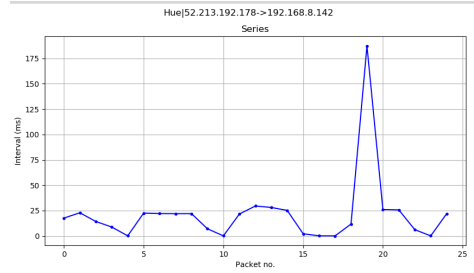


Figure 3.10: Interpacket time interval during a HTTP call and response between Hue and dcp.dc1.philips.com

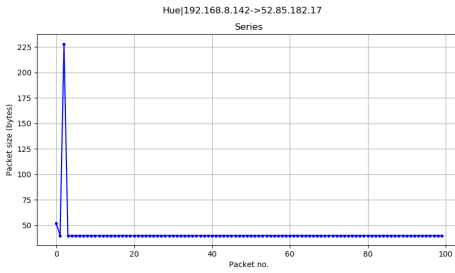


Figure 3.11: Variation in Packet size during a HTTP call and response between Hue and fds.dc1.philips.com

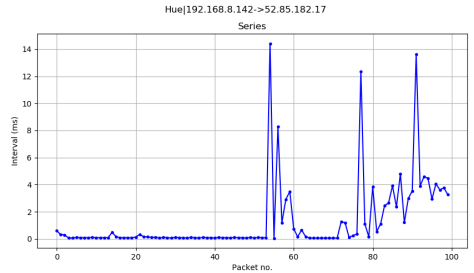


Figure 3.12: Interpacket time interval during a HTTP call and response between Hue and fds.dc1.philips.com

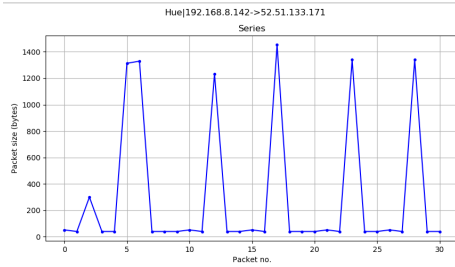


Figure 3.13: Variation in Packet size during a HTTP call and response between Hue and www.ecdinterface.philips.com

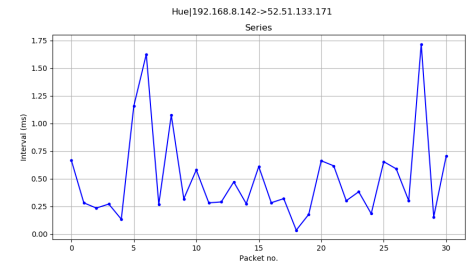


Figure 3.14: Interpacket time interval during a HTTP call and response between Hue and www.ecdinterface.philips.com

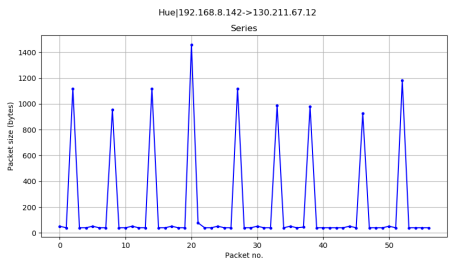


Figure 3.15: Variation in Packet size during a HTTP call and response between Hue and diagnostics.meethue.com

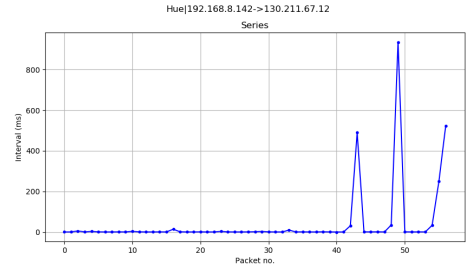


Figure 3.16: Interpacket time interval during a HTTP call and response between Hue and diagnostics.meethue.com

## PORT NUMBERS

It was observed that Hue Bridge always communicated with the mobile app using TCP port 443. It was also observed this was the same case with other hosts. Hence port numbers was not used.

## IP ADDRESS

Upon inspecting the data, it was realised that IP address will not be a useful feature. For instance, in Figure 3.6 it can be seen that Philips Hue communicated with multiple hosts using HTTP calls and responses. It was observed that the Hue was performing the same HTTP call with different ip address over time. If IP address is used as feature, these conversations would be considered as different from each other, but the underlying communication is same in both the cases.

Figures 3.17 and 3.18 shows example of a HTTP Post call made in two different pcap files by Philips Hue. It can be seen that even though the type of call performed in same in both the cases, IP address is different.

Time	Source	Destination	Protocol	Length	Info
325.48.797726	192.168.8.142	52.213.192.178	TCP	60	54156 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=0
326.48.819956	52.213.192.178	192.168.8.142	TCP	58	80 → 54156 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460
327.48.828240	192.168.8.142	52.213.192.178	TCP	60	54156 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
328.48.828463	192.168.8.142	52.213.192.178	TCP	572	54156 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29200 Len=518 [TCP segment of a r...
329.48.842859	52.213.192.178	192.168.8.142	TCP	54	80 → 54156 [ACK] Seq=1 Ack=519 Win=28016 Len=0
330.48.842859	192.168.8.142	52.213.192.178	HTTP	1313	POST /DcpRequestHandler/Index.aspx HTTP/1.1 (application/cb-encrypted)
331.48.865485	52.213.192.178	192.168.8.142	TCP	54	80 → 54156 [ACK] Seq=1 Ack=1783 Win=31600 Len=0
332.48.871723	52.213.192.178	192.168.8.142	HTTP	845	HTTP/1.1 200 OK (application/cb-encrypted)
333.48.871852	52.213.192.178	192.168.8.142	TCP	54	80 → 54156 [FIN, ACK] Seq=792 Ack=1783 Win=31600 Len=0
334.48.872610	192.168.8.142	52.213.192.178	TCP	60	54156 → 80 [ACK] Seq=1783 Ack=792 Win=30958 Len=0
335.48.872557	192.168.8.142	52.213.192.178	TCP	60	54156 → 80 [FIN, ACK] Seq=1783 Ack=793 Win=30958 Len=0

```

Frame 330: 1318 bytes on wire (10544 bits), 1318 bytes captured (10544 bits)
Ethernet II, Src: Phillips..., Dst: GuangLia...
Internet Protocol Version 4, Src: 192.168.8.142, Dst: 52.213.192.178
Transmission Control Protocol, Src Port: 54156, Dst Port: 80, Seq: 519, Ack: 1, Len: 1264
[2 Reassembled TCP Segments (1782 bytes): #328(518), #330(1264)]
Hypertext Transfer Protocol
  POST /DcpRequestHandler/Index.aspx HTTP/1.1\r\n
  Host: dcp.dc1.phillips.com:80\r\n
  [truncated]Authorization: \r\n
  Content-Length: 1264 \r\n
  Content-Type: application/CB-Encrypted; cipher=AES\r\n
  Connection: close\r\n
  \r\n
  Full request URI: http://dcp.dc1.phillips.com:80/DcpRequestHandler/Index.aspx
  [HTTP request 171]
  [Response in frame: 332]

```

Figure 3.17: Hue communicating with one of the hosts to perform a HTTP Post call. Request URL has been highlighted in the picture.

No.	Time	Source	Destination	Protocol	Length	Info
286	15.201079	192.168.8.142	54.76.81.242	TCP	66	57387 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=8
288	15.222872	54.76.81.242	192.168.8.142	TCP	58	80 → 57387 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460
289	15.223160	192.168.8.142	54.76.81.242	TCP	60	57387 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
290	15.22397	192.168.8.142	54.76.81.242	TCP	572	57387 → 80 [PSH, ACK] Seq=1 Ack=1 Win=29200 Len=518 [TCP segment of a r
291	15.245513	54.76.81.242	192.168.8.142	TCP	54	80 → 57387 [ACK] Seq=1 Ack=519 Win=30016 Len=0
292	15.271003	192.168.8.142	54.76.81.242	HTTP	1414	POST /DcpRequestHandler/index.aspx HTTP/1.1 (application/cb-encrypted)
294	15.268926	54.76.81.242	192.168.8.142	TCP	54	80 → 57387 [ACK] Seq=1 Ack=1879 Win=32640 Len=0
295	15.275106	54.76.81.242	192.168.8.142	HTTP	845	HTTP/1.1 200 OK (application/cb-encrypted)
296	15.275249	54.76.81.242	192.168.8.142	TCP	54	80 → 57387 [FIN, ACK] Seq=792 Ack=1879 Win=32640 Len=0
297	15.275741	192.168.8.142	54.76.81.242	TCP	60	57387 → 80 [ACK] Seq=1879 Ack=792 Win=30050 Len=0
298	15.286618	192.168.8.142	54.76.81.242	TCP	60	57387 → 80 [FIN, ACK] Seq=1879 Ack=793 Win=30050 Len=0

\* Frame 292: 1414 bytes on wire (11312 bits), 1414 bytes captured (11312 bits) on interface 0  
 \* Ethernet II, Src: Phillips (08:00:0c:27:1d:1d), Dst: Guanglia (08:00:0c:27:1d:1d)  
 \* Internet Protocol Version 4, Src: 192.168.8.142, Dst: 54.76.81.242  
 \* Transmission Control Protocol, Src Port: 57387, Dst Port: 80, Seq: 519, Ack: 1, Len: 1360  
 \* [2 Reassembled TCP Segments (1878 bytes): #290(518), #292(1360)]  
 \* Hypertext Transfer Protocol  
 POST /DcpRequestHandler/index.aspx HTTP/1.1  
 Host: dcp.dc1.phillips.com:80  
 [truncated]Authorization:  
 Content-Length: 1360  
 Content-Type: application/CB-Encrypted; cipher=AES  
 Connection: close  
 [Full request URI: http://dcp.dc1.phillips.com:80/DcpRequestHandler/index.aspx]  
 [HTTP request 171]  
 [Response in frame: 295]

Figure 3.18: Hue communicating with one of the hosts to perform a HTTP Post call to the same URL as the previous picture. Notice that IP address is different in both the cases.

### IP FLAGS

IP flags were also considered, but it was later observed in the data that the flag was either not set to anything or it was set to DF (Dont fragment).

## 3.6. SUMMARY

Focus of this chapter has been completely on the data used for this project. Experimental setup used to capture network traffic generated by IOT device was discussed. Tools and methodology to obtain this traffic from router to local machine is then explained. Various operations performed on the IOT device were visualised. Format of the data collected and filtering approach used is then discussed. It is followed by some preliminary observations that were made by observing the data captured. Based on these observations and inputs from literature survey, features that could be useful to represent the behaviour of IoT devices are then mentioned.

# 4

## METHODOLOGY

*This chapter contains the description of the state machine learning module and their representation. It is followed by a discussion of initial state machines learnt from the network traffic of Smart Home devices. Their weaknesses are discussed, followed by more interpretable state machines which can be used as the final result.*

### 4.1. STATE MACHINE LEARNING MODULE

This thesis uses *Flexfringe*, a passive automata learning package[47] to build state machines from input data. Flexfringe works with both labelled and unlabeled data. Present focus is to learn the normal behaviour of an IoT device and hence input to this module will be unlabelled (All the traffic captured from IoT device is assumed to be from its normal behaviour and the device is not infected).

It starts by constructing a tree which exactly represents the input data. Each transition of the tree represents an entry in the input. It then iteratively merges pairs of states to create a deterministic and generic state machine. A merger of two states will be evaluated based on two criterion: *Consistency* and *Score*. When is a merge consistent and what is the score of that merge is based on a heuristic. This can either be controlled with the help of a parameter of Flexfringe or users can define their own heuristic. More details about state machine learning module and heuristics defined in flexfringe can be found as part of this work [11].

#### 4.1.1. INPUT FORMAT

Input format to the state machine learning module is based on Abbadingo competition[18]. The first line of input file contains the number of sequence of input symbols followed by number of unique symbols. Each subsequent line contains the following: The label of the sequence of symbols, followed by number of symbols in the sequence and rest of the line contains actual sequence of symbols. Figure 4.1 gives an example of such input file.

In the first line of Figure 4.1, 3 represents number of sequences, 7 represents number

```

3 7
1 5 S60 SA54 A54 FA54 FA60
1 6 S60 SA54 A54 A54 FA54 FA60
1 7 S60 SA54 A54 A54 A54 FA54 FA60

```

Figure 4.1: Example of an input file that can be given to Flexfringe[47] to obtain a state machine.

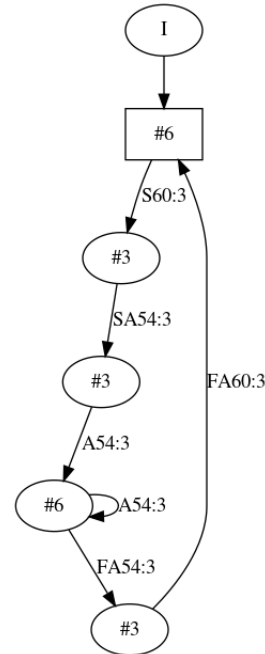


Figure 4.2: State machine that is obtained based on input specified in Figure 4.1

of unique symbols. Next two lines start with a label of 1 along with number of entries in the sequence, followed by actual sequence of symbols. Here, a label of 1 is used for all the traces. Since it is unlabelled data, any label can be used with the sequences, as long as all of them share the same label.

#### 4.1.2. PARAMETERS TO LEARNING MODULE

Flexfringe comes with a range of parameters, many of them were explored, but in the end 4 of them were considered for this thesis and rest remained with default values. As discussed in the previous section, one of the important aspects of state merging is based on a heuristic. The parameters "heuristic\_name" and "data\_name" are used to specify this. The heuristic *overlap* was used for this project. In *overlap*, Consistency is based on overlapping outgoing transitions for two states. The score represents the number of overlapping outgoing transitions. This was chosen because the training data is not labelled and emphasis has been more on transitions in state machines and not much on their probability. Other heuristics available as part of flexfringe works either with Probabilistic Deterministic Finite Automaton or labelled data. More details about heuristic could be found as part of this work [11].

The second set of parameters considered were "state\_count" and "symbol\_count". These two numbers specify the minimum number of occurrences of any state/symbol. Any state/symbol whose occurrences is below this threshold is merged with any other state



without verifying Consistency and Score.

### 4.1.3. OUTPUT FORMAT

The final output by state machine learning algorithm is in the form of a graphviz dot file<sup>1</sup> or a PNG file. Each state contains its number of occurrences. Transitions between states are the symbols or elements of input file. More details about how these state machines can be read are discussed in next section.

## 4.2. REPRESENTATION OF STATE MACHINES

In this section, a brief explanation of how state machines are represented and how they can be read is provided.

In continuation with the previous chapter, Protocol related information and Packet Size were chosen as features. Each packet exchanged between hosts is marked by a transition in the state machine. Figure 4.2 shows an example of a state machine that is obtained as output from *Flexfringe*[47]. Initial state is represented in the shape of a square. All the intermediate states are represented in the form of an oval. In that figure, entries such as #6, #3 that are present inside each state represent the number of occurrences of that state. Arrow marks represent transitions between states. Each transition marks a packet sent/received. Transitions in the above example are in the form "TP:N". Where T represents TCP flag of that packet, P represents Packet Size and N represents the number of occurrences of that symbol.

### 4.3. INITIAL STATE MACHINES: TRAFFIC BETWEEN MOBILE APP AND HUE

Initial focus was to concentrate only on communication between the app which controls the lighting system with the Hue bridge (which here is a TCP connection). The idea was to get good, readable state machines for this and then extend the work to other traffic.

#### 4.3.1. CASE I: ALL TRAFFIC WAS CONSIDERED

In this scenario, all the traffic between Hue bridge and the mobile app was considered. The input traffic was composed of pcaps which captured the turning "On/Off" actions on Hue multiple times. It was filtered based on IP address of Mobile and MAC address of Hue. The hypothesis was that, since it is a TCP session, the state machine should start with a SYN packet and should end with a FIN or FIN-ACK packet. In between this, there should be some looping behaviour representing the actions which were performed on Hue with fixed time intervals between them.

State machine in Figure 4.3 that was obtained using the pcaps that were captured to learn the behaviour of Hue while performing "On/Off" operation multiple times on lights. It could be seen that the initial few transitions represent the creation of a TCP session as we can see, the first three transitions from the initial state are "S00", "SA00", "A00" as highlighted in the figure. We can also see some looping behaviour in the lower part of state machine. However, we do not see the termination of TCP session clearly in this

<sup>1</sup><https://www.graphviz.org/doc/info/lang.html>

state machine. The obtained state machine also consists of a loop to its initial state with a long list of symbols. The reason is that there isn't much evidence for these symbols to create a state or to merge them into an existing state. This caused the state machine learning module to create a transition from the root state to itself to have less number of total states in the final result. This has been highlighted in Figure 4.3.

From this, we could infer that, although the initial part of state machine is reasonable, the way it creates a loop to the root node with many symbols is not convincing. It was also not possible to explain all the transitions in the state machine.

### 4.3.2. CASE2: DIFFERENT DATA WITH SOME FILTERING

In this scenario, all the traffic between Hue bridge and the mobile app was considered. The input traffic was composed of pcaps which captured the "changing colour" on Hue multiple times. In addition to filtering mentioned in the previous case, packets representing "TCP Re-transmissions" were omitted. The hypothesis was same as the previous case.

State machine in Figure 4.4 that was obtained using the pcaps that were captured to learn the behaviour of Hue while performing "Changing Colours" operation multiple times on lights. As seen in previous case, it was clear that there was a TCP session created in the beginning based on highlighted transitions. It could be seen that the final two symbols to the first state represent the closing of TCP session because they represent a TCP packet with FIN-ACK flag set. Another drawback which we had of the previous case was of a loop with many symbols pointing from initial state to itself. However, it is not present in this case.

While it was possible to identify the creation and termination of TCP session, we still are not aware of what the intermediate transitions represent in this state machine. We are also not able to identify the packets representing the commands issued by smart phone to IoT device. Another issue is that this works in very specific cases and depends on the input. Different set of pcaps were captured performing different actions on Hue, the resulting state machines were similar to the previous case and not like this case.

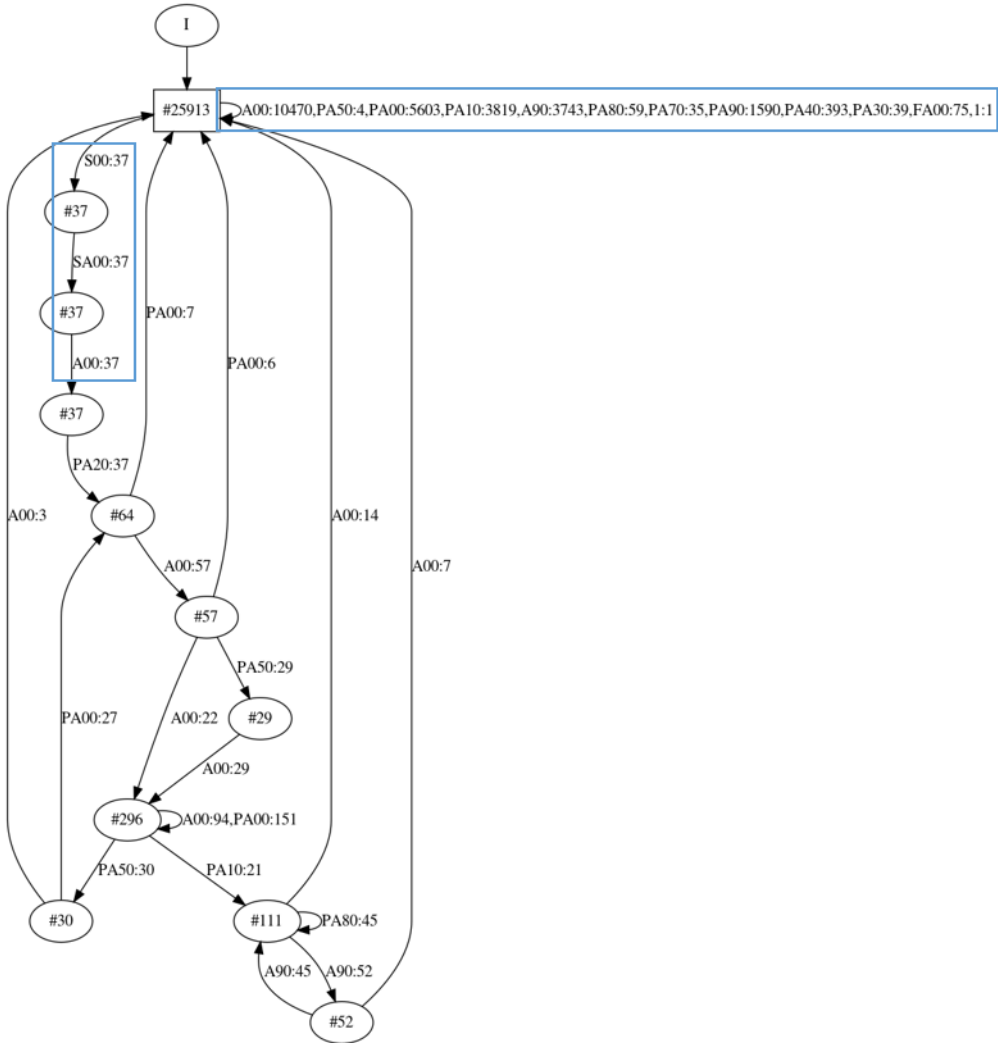


Figure 4.3: Initial state machine representing the traffic between Mobile App and Hue

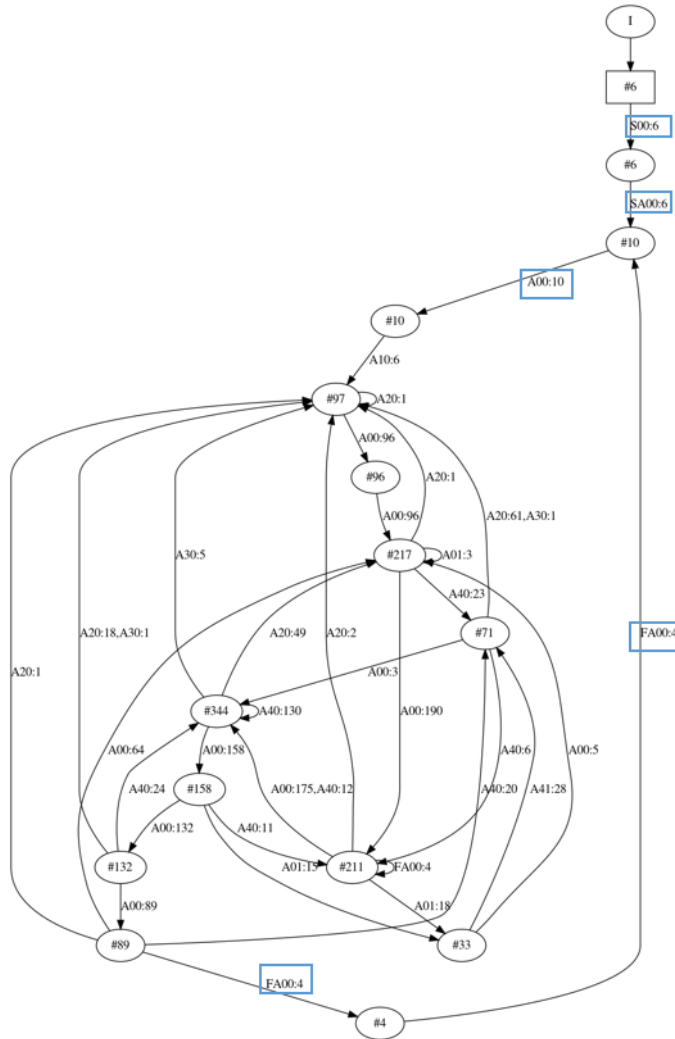


Figure 4.4: Initial state machine representing the traffic between Mobile App and Hue while Changing Colours command was issued. Highlighted packets signify creation and termination of a TCP session.

## 4.4. FINAL STATE MACHINES: TRAFFIC BETWEEN MOBILE APP AND HUE

Continuing from previous section, it was observed that the state machines were not interpretable and differed with different input data. To deal with this issue, certain steps were taken. Firstly, Packets with empty TCP acknowledgements were removed in this stage because the order in which these arrive differ every time. Secondly, manual inspection of pcaps was performed.

### 4.4.1. INSPECTION

As mentioned in Chapter 3.2, while recording pcaps, actions were performed every 30 seconds. Few such smaller pcaps were taken and analysed. Initial few packets demonstrated the initiation of TCP session and then TLS session with various packets depicting TLS handshake as seen in Figure 4.5.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.533673	192.168.8.221	192.168.8.142	TCP	78	57837 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1421928741
6	0.555954	192.168.8.142	192.168.8.221	TCP	66	443 → 57837 [SYN, ACK] Seq=0 Ack=1 Win=29208 Len=0 MSS=1460 SACK_PERM=1
9	0.591668	192.168.8.221	192.168.8.142	TCP	54	57837 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
10	0.591786	192.168.8.221	192.168.8.142	TLSv1.2	491	Client Hello
11	0.592127	192.168.8.142	192.168.8.221	TCP	60	443 → 57837 [ACK] Seq=1 Ack=438 Win=38272 Len=0
12	0.594455	192.168.8.142	192.168.8.221	TLSv1.2	191	Server Hello, Change Cipher Spec, Encrypted Handshake Message
13	0.595534	192.168.8.221	192.168.8.142	TCP	54	57837 → 443 [ACK] Seq=438 Ack=138 Win=261952 Len=0
14	0.599297	192.168.8.221	192.168.8.142	TLSv1.2	60	Change Cipher Spec
15	0.599370	192.168.8.221	192.168.8.142	TLSv1.2	99	Encrypted Handshake Message

\* Frame 15: 99 bytes on wire (792 bits), 99 bytes captured (792 bits)  
 \* Ethernet II, Src: Apple ( ), Dst: Philips ( )  
 \* Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 \* Transmission Control Protocol, Src Port: 57837, Dst Port: 443, Seq: 444, Ack: 138, Len: 45  
 \* Secure Sockets Layer

Figure 4.5: Creation of TLS session between Philips Hue and the smartphone.

To make the state machine more readable, these packets were labelled with the appropriate TLS action they were performing.

It was then observed that there was a certain query (in the form of a single packet) from Mobile to Hue in the initial stage of connection, it was followed by a bigger response from Hue (comprising multiple packets). This could probably be checking for availability of lights (Since a single Hue bridge can be used to control multiple light bulbs). This can be seen in Figure 4.6.

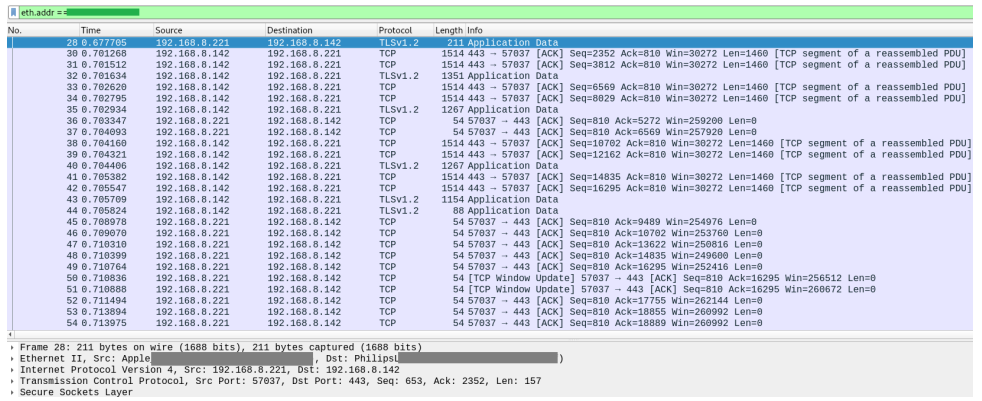


Figure 4.6: Query from smart phone to Hue followed by response in the form of multiple packets from Hue in the initial phase of the connection.

It was also noted that there was a another query which happened multiple times, throughout the duration of entire connection and it triggered relatively smaller response from hue (comprising 2-3 packets). This has been highlighted in Figure 4.7.

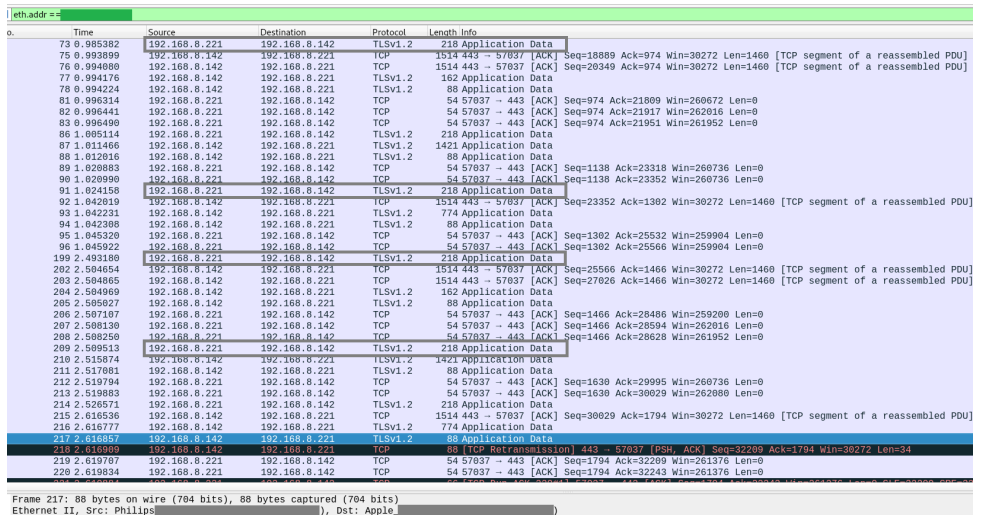


Figure 4.7: Packets depicting command from smartphone to perform some action followed by response from Hue. This type of conversation occurs multiple times in the data.

Ignoring these packets, when other packets were inspected, it was possible to uniquely identify the packets sent from Mobile App to Hue and the corresponding response to perform various actions. This conversation can be seen in Figure 4.8.

No.	Time	Source	Destination	Protocol	Length	Info
367	31.822999	192.168.8.221	192.168.8.142	TLSv1.2	272	Application Data
368	31.831955	192.168.8.142	192.168.8.221	TLSv1.2	647	Application Data
369	31.832516	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data

▶ Frame 367: 272 bytes on wire (2176 bits), 272 bytes captured (2176 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: Philips ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 57037, Dst Port: 443, Seq: 2122, Ack: 36671, Len: 218  
 ▶ Secure Sockets Layer

Figure 4.8: Query from smartphone to Hue followed by response occurring multiple times.

These conversations between the smartphone and Hue were considered as separate sequences and given as input to state machine learning module.

Using all this knowledge, and filtering, a state machine was learnt using the pcaps which captured "Changing colours" action on hue lights shown in Figure 4.9.

Following things can be inferred about the traffic based on state machine generated.

- 1 This part of state machine represents the TLS session initiation between Hue and the mobile app when user opens the app.
- 2 This symbol represents the initial query from Mobile app to Hue. All the transitions that follow this symbol are the responses from Hue to Mobile app.
- 3 It represents the query from App to Hue which happens multiple times throughout the connection. Similar to previous case, all the symbols after this represent the reply from Hue.
- 4 A command sent from mobile app to Hue.
- 5 Response from Hue to the app for the previous command.
- 6 Any response from Hue to App includes this packet as its final reply.
- 7 After any of these query-responses between App and Hue, when the user leaves the app, it sends a packet with FIN-ACK to close the session.

**Unlike the state machines in previous case, here it was possible to identify all the transitions that were part of state machine.**

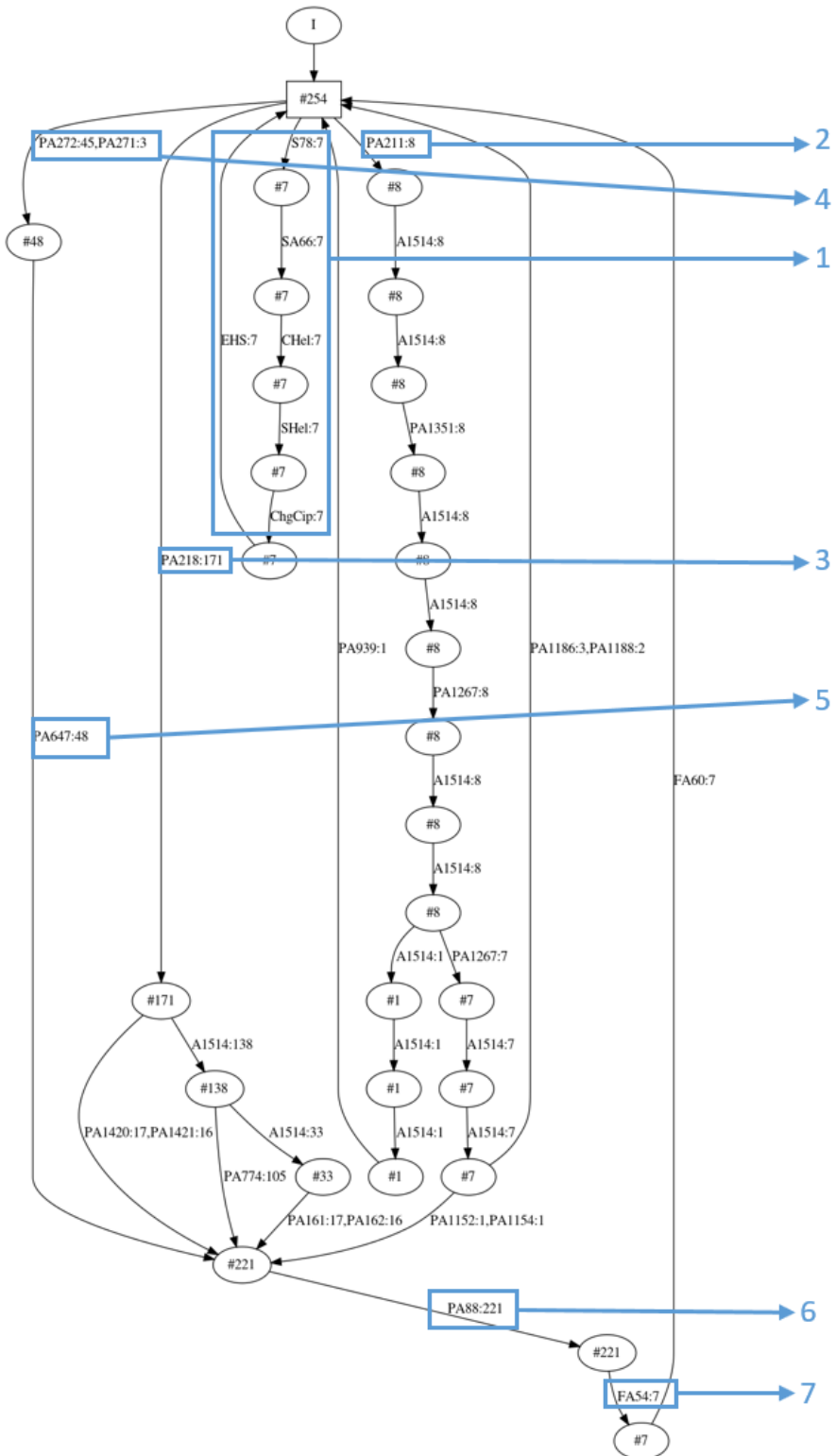
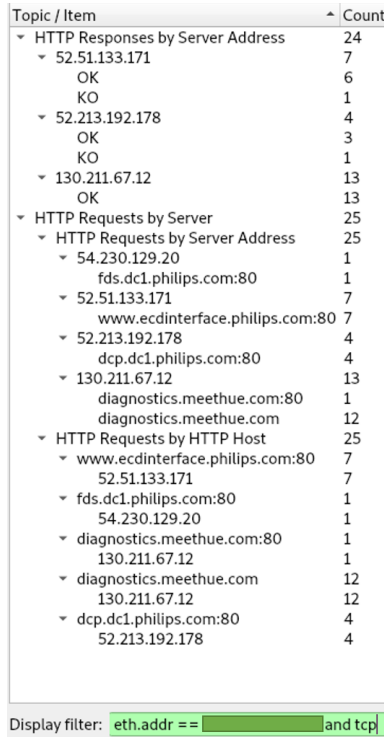


Figure 4.9: State machine depicting the traffic captured while performing the action of "Changing colours" of light. Each state contains the number of times it has occurred. A symbol/transition "S78" means it is a packet with TCP flag SYN and size of 78 bytes



## 4.5. ORDERING OF TCP STREAM AND ITS SIGNIFICANCE ON BACKGROUND TRAFFIC BY HUE

Once readable state machine was obtained for traffic between Hue and mobile app, focus now was on background traffic generated by Hue. This comprised mainly of various HTTP calls by Hue to certain domains of Philips, followed by uploading or downloading of data. This can be seen in Figure 4.10. It lists the hosts with which Philips hue was communicating using HTTP in one of the pcaps captured during this thesis. To make state machines more readable, packets which represent performing GET/POST calls are marked with the domain to which these calls were made. The response from these calls is marked with HTTP status. Traffic from multiple pcaps were used as input and gave the following state machine in Figure 4.12.



Topic / Item	Count
HTTP Responses by Server Address	24
52.51.133.171	7
OK	6
KO	1
52.213.192.178	4
OK	3
KO	1
130.211.67.12	13
OK	13
HTTP Requests by Server	25
HTTP Requests by Server Address	25
54.230.129.20	1
fds.dc1.philips.com:80	1
52.51.133.171	7
www.ecdinterface.philips.com:80	7
52.213.192.178	4
dcp.dc1.philips.com:80	4
130.211.67.12	13
diagnostics.meethue.com:80	1
diagnostics.meethue.com	12
HTTP Requests by HTTP Host	25
www.ecdinterface.philips.com:80	7
52.51.133.171	7
fds.dc1.philips.com:80	1
54.230.129.20	1
diagnostics.meethue.com:80	1
130.211.67.12	1
diagnostics.meethue.com	12
130.211.67.12	12
dcp.dc1.philips.com:80	4
52.213.192.178	4

Display filter: eth.addr == [redacted] and tcp

Figure 4.10: Screenshot from wireshark showing various stats related to HTTP calls performed by IOT device from one of the pcaps captured.

State machine in Figure 4.12 could be used to infer that there were TLS sessions created along with HTTP calls and responses. A TCP session starts by an exchange of SYN and SYN-ACK packet. Termination is marked by FIN-ACK packets <sup>2</sup>. **The ordering in state machines in Figure 4.12 violates TCP protocol with respect to ordering of packets.**

<sup>2</sup>[http://telescript.denayer.wenk.be/~hcr/cn/idoceo/tcp\\_connection.html](http://telescript.denayer.wenk.be/~hcr/cn/idoceo/tcp_connection.html)

eth.addr ==	Time	Source	Destination	Protocol	Length	Info
	710.46.711952	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [SYN] Seq=0 Win=29208 Len=0 MSS=1460 SACK_PERM=1 WS=8
	711.46.733833	52.49.36.240	192.168.8.142	TCP	66	80 → 56163 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460 SACK_PERM=1 WS=256
	712.46.734112	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [ACK] Seq=1 Ack=1 Win=29208 Len=0
	713.46.741878	192.168.8.142	52.49.36.240	HTTP	314	POST /DevicePortalIICRequestHandler/RequestHandler.ashx HTTP/1.1 (application/cb-encrypted)
	719.46.763550	52.49.36.240	192.168.8.142	TCP	54	80 → 56163 [ACK] Seq=1 Ack=261 Win=28160 Len=0
	720.46.765207	52.49.36.240	192.168.8.142	TCP	1514	80 → 56163 [ACK] Seq=1 Ack=261 Win=28160 Len=1460 [TCP segment of a reassembled PDU]
	721.46.765313	52.49.36.240	192.168.8.142	HTTP	305	HTTP/1.1 401 Unauthorized (text/html)
	722.46.765519	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [ACK] Seq=261 Ack=1461 Win=32128 Len=0
	723.46.765625	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [ACK] Seq=261 Ack=1892 Win=35948 Len=0
	724.46.766679	192.168.8.142	52.49.36.240	HTTP	1327	POST /DevicePortalIICRequestHandler/RequestHandler.ashx HTTP/1.1 (application/cb-encrypted)
	725.46.805196	192.168.8.142	224.0.0.22	IGMPv3	68	Membership Report / Join group 239.255.255.250 for any sources
	726.46.819688	52.49.36.240	192.168.8.142	HTTP	1311	HTTP/1.1 200 OK (application/cb-encrypted)
	727.46.820699	192.168.8.142	52.49.36.240	HTTP	1243	POST /DevicePortalIICRequestHandler/RequestHandler.ashx HTTP/1.1 (application/cb-encrypted)
	728.46.850780	52.49.36.240	192.168.8.142	TCP	1514	80 → 56163 [ACK] Seq=3059 Ack=2823 Win=32288 Len=1460 [TCP segment of a reassembled PDU]
	729.46.850935	52.49.36.240	192.168.8.142	HTTP	991	HTTP/1.1 200 OK (application/cb-encrypted)
	730.46.851235	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [ACK] Seq=2823 Ack=5456 Win=43800 Len=0
	731.46.852281	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [FIN, ACK] Seq=2823 Ack=5456 Win=43800 Len=0
	732.46.874854	52.49.36.240	192.168.8.142	TCP	54	80 → 56163 [FIN, ACK] Seq=5456 Ack=2824 Win=32288 Len=0
	733.46.874336	192.168.8.142	52.49.36.240	TCP	66	56163 → 80 [ACK] Seq=2824 Ack=5457 Win=43800 Len=0
	734.46.876439	192.168.8.142	192.168.8.1	DNS	86	Standard query response 0x0006 AAAA www.ecdinterface.philips.com
	735.46.879460	192.168.8.1	192.168.8.142	DNS	251	Standard query response 0x0005 AAAA www.ecdinterface.philips.com CNAME dp.dci.philips.com CNAME
	736.46.880690	192.168.8.142	192.168.8.1	DNS	88	Standard query 0x0006 A www.ecdinterface.philips.com
	737.46.886453	192.168.8.1	192.168.8.142	DNS	338	Standard query response 0x0006 A www.ecdinterface.philips.com CNAME dp.dci.philips.com CNAME dci
	738.46.889037	192.168.8.142	52.49.36.240	TCP	66	56164 → 80 [SYN] Seq=0 Win=29208 Len=0 MSS=1460 SACK_PERM=1 WS=8
	739.46.911363	52.49.36.240	192.168.8.142	TCP	66	80 → 56164 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460 SACK_PERM=1 WS=256
	740.46.911338	192.168.8.142	52.49.36.240	TCP	66	56164 → 80 [ACK] Seq=1 Ack=1 Win=29208 Len=0
	741.46.911933	192.168.8.142	52.49.36.240	HTTP	1248	POST /NotificationService/RequestHandler.ashx HTTP/1.1 (application/cb-encrypted)
	742.46.934993	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [ACK] Seq=1 Ack=1195 Win=29440 Len=0
	744.47.087794	52.49.36.240	192.168.8.142	HTTP	613	HTTP/1.1 200 OK (application/cb-encrypted)
	745.47.067915	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [FIN, ACK] Seq=760 Ack=1195 Win=29440 Len=0
	746.47.068895	192.168.8.142	52.49.36.240	TCP	66	56164 → 80 [ACK] Seq=1195 Ack=760 Win=30720 Len=0
	747.47.069030	192.168.8.142	52.49.36.240	TCP	66	56164 → 80 [FIN, ACK] Seq=1195 Ack=761 Win=30720 Len=0
	748.47.074240	192.168.8.142	192.168.8.1	DNS	66	Standard query 0x0007 AAAA www.ecdinterface.philips.com
	749.47.077695	192.168.8.1	192.168.8.142	DNS	251	Standard query response 0x0007 AAAA www.ecdinterface.philips.com CNAME dp.dci.philips.com CNAME
	750.47.078283	192.168.8.142	192.168.8.1	DNS	88	Standard query 0x0008 A www.ecdinterface.philips.com
	751.47.086375	192.168.8.1	192.168.8.142	DNS	338	Standard query response 0x0008 A www.ecdinterface.philips.com CNAME dp.dci.philips.com CNAME dci
	752.47.086949	192.168.8.142	52.49.36.240	TCP	66	56165 → 80 [SYN] Seq=0 Win=29208 Len=0 MSS=1460 SACK_PERM=1 WS=8
	753.47.098083	52.49.36.240	192.168.8.142	TCP	54	80 → 56164 [ACK] Seq=761 Ack=1196 Win=29440 Len=0
	754.47.108741	52.49.36.240	192.168.8.142	TCP	66	80 → 56165 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 MSS=1460 SACK_PERM=1 WS=256
	755.47.108975	192.168.8.142	52.49.36.240	TCP	66	56165 → 80 [ACK] Seq=1 Ack=1 Win=29208 Len=0
	756.47.109208	192.168.8.142	52.49.36.240	HTTP	1468	POST /DevicePortalIICRequestHandler/RequestHandler.ashx HTTP/1.1 (application/cb-encrypted)
	757.47.146226	192.168.8.142	216.239.36.1	HTTP	98	HTTP/1.1 303 See other
	758.47.147120	192.168.8.142	216.239.36.1	HTTP	98	HTTP/1.1 303 See other

Figure 4.11: Screenshot from wireshark showing some of the HTTP calls performed by Philips Hue. We would expect the state machine to represent this.

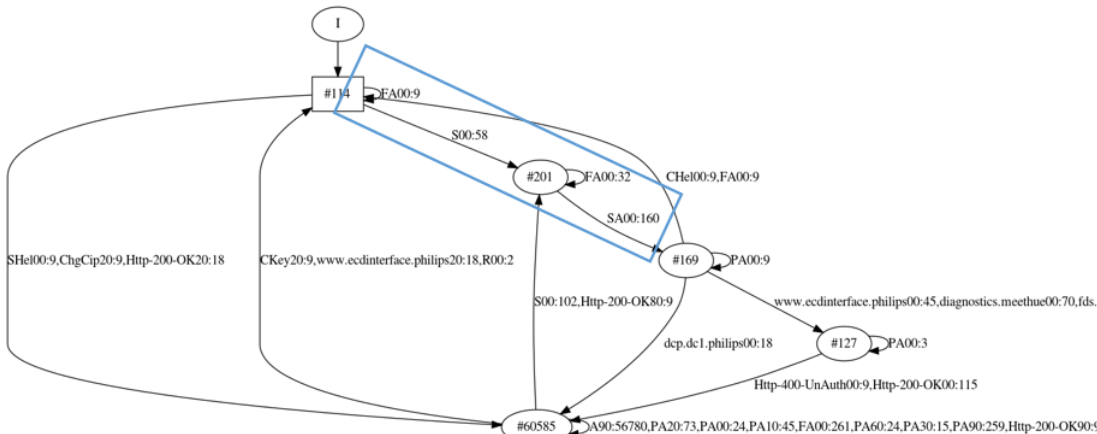


Figure 4.12: Initial state machine representing background traffic generated by Hue while interacting with various hosts. Highlighted part shows that the state machine doesn't comply TCP protocol ordering.

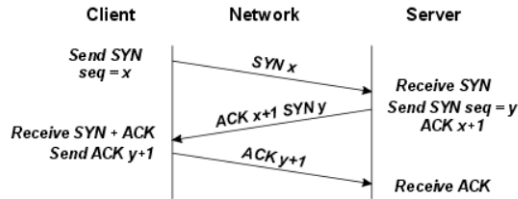


Figure 4.13: Opening a TCP session. A SYN packet initiates the creation, followed by SYN-ACK packet from the second host and an ACK from the first host.

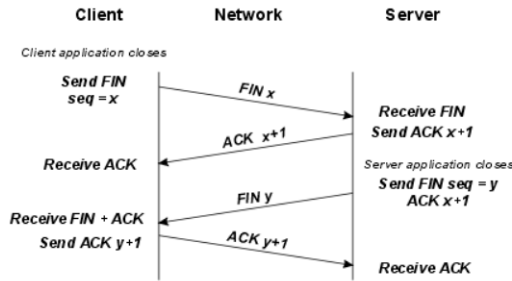


Figure 4.14: Terminating a TCP session. A FIN packet initiates termination, followed by a FIN response with acknowledgements to terminate the session.<sup>2</sup>

Time	Source	Destination	Protocol	Length	Info
970.87.416060	192.168.8.142	192.168.8.221	TLSv1.2	1420	Application Data
971.87.416409	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
972.87.419133	192.168.8.221	192.168.8.142	TCP	54	56748 → 443 [ACK] Seq=16471 Ack=227432 Win=260736 Len=0
973.87.419217	192.168.8.221	192.168.8.142	TCP	54	56748 → 443 [ACK] Seq=16471 Ack=227466 Win=260736 Len=0
994.89.588681	192.168.8.221	192.168.8.142	TCP	54	56748 → 443 [FIN, ACK] Seq=16471 Ack=227466 Win=262144 Len=0
995.89.588796	192.168.8.221	192.168.8.142	TCP	78	56749 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=1173
997.89.589274	192.168.8.142	192.168.8.221	TCP	66	443 → 56749 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK
999.89.589631	192.168.8.142	192.168.8.221	TCP	60	443 → 56748 [FIN, ACK] Seq=227466 Ack=16472 Win=30272 Len=0
1003.89.594957	192.168.8.221	192.168.8.142	TCP	54	56749 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
1005.89.595913	192.168.8.221	192.168.8.142	TCP	54	56748 → 443 [ACK] Seq=16472 Ack=227467 Win=262144 Len=0
1006.89.602526	192.168.8.221	192.168.8.142	TLSv1.2	491	Client Hello
1007.89.602809	192.168.8.142	192.168.8.221	TCP	60	443 → 56749 [ACK] Seq=1 Ack=438 Win=30272 Len=0
1008.89.604743	192.168.8.142	192.168.8.221	TLSv1.2	191	Server Hello, Change Cipher Spec, Encrypted Handshake Message
1009.89.606429	192.168.8.221	192.168.8.142	TCP	54	56749 → 443 [ACK] Seq=438 Ack=138 Win=261952 Len=0
1010.89.611897	192.168.8.221	192.168.8.142	TLSv1.2	66	Change Cipher Spec
1011.89.613161	192.168.8.221	192.168.8.142	TLSv1.2	99	Encrypted Handshake Message
1012.89.613236	192.168.8.221	192.168.8.142	TLSv1.2	218	Application Data

Figure 4.15: Screenshot from wireshark highlighting the TCP packets received out of order.

For instance, in Figure 4.12, initial state has a self-loop for a packet with TCP flag FIN-ACK. In the first three transitions from the initial state, there was a packet with SYN flag which was followed by a loop depicting a packet with FIN flag and then a packet with SYN-ACK flag. This has been highlighted in Figure 4.12. This should not be the normal behaviour of a TCP session. Ideally, a TCP session should begin with a SYN packet followed by SYN-ACK and it should terminate with a FIN-ACK packet from either direction as shown in Figures 4.13 and 4.14.

**Upon further investigation, it was found that some of the packets were received out of order.** Figure 4.15 illustrates one such example. Here, before a response to closing a session could arrive, there was another request to initiate a new session. Although these occurred less often, they were having an impact on the resulting state machine. To deal with these irregularities, TCP sequence re-ordering was performed. The approach used to achieve this is described in next section.

### 4.5.1. TCP PACKETS RE-ORDERING AND SEQUENCE GENERATION

As discussed, there were few packets which reached out of order when the traffic was captured at the router. To deal with this, the exact sequence in which traffic was generated at the hosts need to be recreated. Additionally, there might be multiple conversations between hosts over a period of time. We need to devise a way to regenerate individual conversations. For solving both these problems, TCP sequence number and acknowledgement numbers were used. Figure 4.16 shows pseudocode of the methodology used to achieve this goal.

```

input : Stream of packets

For each packet when it has TCP layer:
  #identify first packet in a conversation
  if (TCP Acknowledgement Number is Zero):
    increment conversation number of that ip address
    if TCP flags contain SYN or FIN:
      next_sequence_number_expected = Seq Number of this packet + 1
    else:
      if (Packet has Padding)
        next_sequence_number_expected = Seq Number of this packet + length of this packet TCP payload - length of TCP padding
      else:
        next_sequence_number_expected = Seq Number of this packet + length of this packet TCP payload
  else:
    if (TCP Acknowledgement Number mapped to a previously seen ip address)
      Get the entry of a datastructure corresponding to that ip address
      append metadata of this packet inside the datastructure corresponding to this ip address

    else if TCP Sequence Number mapped to a previously seen ip address:
      Get the entry of a datastructure corresponding to that ip address
      if TCP flags contain SYN or FIN:
        next_sequence_number_expected = Seq Number of this packet + 1
      else:
        if (Packet has Padding)
          next_sequence_number_expected = Seq Number of this packet + length of this packet TCP payload - length of TCP padding
        else:
          next_sequence_number_expected = Seq Number of this packet + length of this packet TCP payload

    update the datastructure to hold the sequence number of future packet related to that ip address
    append metadata of this packet inside the datastructure corresponding to this ip address

```

Figure 4.16: Pseudocode representing the implementation used to perform tcp packet reordering and separation of conversations.

***The goal is to create a mapping, such that every packet belongs to an entry in the form of IP-address:conversation-number. All the packets in such mapping are in the order in which they were generated at sender.***

It was important to identify ***the first packet of a conversation*** upon arrival. This can be identified based on its acknowledgement number. Every TCP session begins with a random sequence number called Initial Sequence Number (ISN), and an Acknowledgement number of zero<sup>3</sup>. So when there is a packet with acknowledgement number of zero, an entry is created with its IP-address and conversation number of 1 and this packet is mapped to that entry. To identify next packet that would belong to this conversation, the next sequence number is stored in a different data structure. The sequence number of packet that is expected next is calculated as follows:

- For all the packets: Sequence number of current packet + TCP payload length.

<sup>3</sup><http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/>

- For those packets with SYN or FIN flag set, it is further incremented by 1<sup>4 5</sup>.
- In case of packets with padding, length of the padding is subtracted from TCP payload length.

**Upon arrival of a packet in the same direction as the first packet of a conversation:**

Sequence number of the packet is checked against all the entries in the data structure, to see if the next sequence number of any conversation matches it. If there is a match, it is appended in the data structure and Sequence number of packet that is expected next is updated.

**Upon arrival of a packet in the opposite direction as the first packet of a conversation:**

The packet's acknowledgement number is checked against next sequence number of all the conversations. If matched, it is appended to the data structure. In this case, sequence number of packet that is expected next is not updated.

Figure 4.17 shows an example of how the output looks before applying this procedure. Out of order packets, such as those shown in Figure 4.15 have been highlighted in this figure. Here, all the packets exchanged with an IP address by Hue belong to same conversation.

Figure 4.18 shows an example of how the output looks after applying this procedure to the same data as before. It is important to note that here different conversations have been separated.

```
130.211.67.12:1 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:60 S:66 FA:54 SA:66
diagnostics.meethue Http-200-OK FA:60 FA:54 S:66 SA:66 diagnostics.meethue
Http-200-OK FA:60 S:66 FA:54 SA:66 diagnostics.meethue Http-200-OK S:66 SA:66 PA:60
diagnostics.meethue Http-200-OK FA:54 FA:60 FA:54 FA:60 S:66 SA:66
diagnostics.meethue Http-200-OK FA:54 FA:60 S:66 SA:66 diagnostics.meethue
Http-200-OK FA:54

52.51.133.171:1 => S:66 SA:66 www.ecdinterface.philips Http-400-UnAuth PA:395
www.ecdinterface.philips Http-200-OK www.ecdinterface.philips Http-200-OK PA:991
FA:60 S:66 FA:54 SA:66 www.ecdinterface.philips Http-200-OK FA:54 FA:60 S:66 SA:66
www.ecdinterface.philips Http-200-OK PA:602 FA:54 FA:60 S:66 SA:66
www.ecdinterface.philips Http-200-OK FA:54 FA:60 S:66 SA:66 www.ecdinterface.philips
Http-200-OK FA:54 FA:60
```

Figure 4.17: Output *before* TCP sequence re-generation. Each key is in the form of IP-address:conversation-number and value consist of metadata(TCP Flag:Packet Size) of packets belonging to the conversation. Each IP address is associated with all the packets exchanged between Hue and the host of that IP address in that pcap. Hence here the conversation number is always 1. Highlighted parts show the packets received out of order.

<sup>4</sup>[http://www.cs.miami.edu/home/burt/learning/Csc524.032/notes/tcp\\_nutshell.html](http://www.cs.miami.edu/home/burt/learning/Csc524.032/notes/tcp_nutshell.html)

<sup>5</sup><https://osqa-ask.wireshark.org/questions/61819/tcp-sequence-number-calculation>

```

130.211.67.12:1 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:60 FA:54
130.211.67.12:2 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:60 FA:54
130.211.67.12:3 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:60 FA:54
130.211.67.12:4 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:54 FA:60
130.211.67.12:5 => S:66 SA:66 PA:60 diagnostics.meethue Http-200-OK FA:54 FA:60
130.211.67.12:6 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:54 FA:60
130.211.67.12:7 => S:66 SA:66 diagnostics.meethue Http-200-OK FA:54

52.51.133.171:1 => S:66 SA:66 www.ecdinterface.philips Http-400-UnAuth PA:395
www.ecdinterface.philips Http-200-OK www.ecdinterface.philips Http-200-OK
PA:991 FA:60 FA:54
52.51.133.171:2 => S:66 SA:66 www.ecdinterface.philips Http-200-OK FA:54 FA:60
52.51.133.171:3 => S:66 SA:66 www.ecdinterface.philips Http-200-OK PA:602
FA:54 FA:60
52.51.133.171:4 => S:66 SA:66 www.ecdinterface.philips Http-200-OK FA:54 FA:60
52.51.133.171:5 => S:66 SA:66 www.ecdinterface.philips Http-200-OK FA:54 FA:60

```

Figure 4.18: Output *after* TCP sequence re-generation. Each key is in the form of IP-address:conversation-number and value consists of metadata of packets belonging to the conversation. Each new conversation is recognised and separated.

## 4.6. FINAL STATE MACHINES: BACKGROUND TRAFFIC GENERATED BY HUE

After performing TCP sequence regeneration mentioned in the previous section, these new sequences were provided as input to state machine learning algorithm and a state machine was obtained.

Figure 5.5 shows the state machine obtained from most of the background traffic from Hue. Each chain of transitions represents a conversation between Philips hue and some host.

Following are the inferences that can be made based on the state machine.

- 1 Packets signifying the creation of TCP session.
- 2 Packets signifying the termination of TCP session.
- 3 This and its subsequent transitions represent the packets exchanged while creating a TLS session.

More details about every transition in the state machine will be explained as part of next chapter.



## 4.7. EXTENSION TO ANOTHER DEVICE: IKEA LIGHTS

Once the state machines for the behaviour of Philips Hue lighting system were learnt, the work was then extended to another lighting system. IKEA lights work similar to Philips Hue lighting system. It consists of a bridge which controls the lights. To control the bridge, commands can be issued by user through a Mobile app or a Remote. In this work, we consider the traffic generated by IKEA light when controlling them through Mobile app. The connections and methodology used to capture the traffic remained the same.

### 4.7.1. TRAFFIC BETWEEN SMART PHONE AND IKEA LIGHTS

It was observed that IKEA bridge communicates with the Mobile app using DTLS protocol, version 1.2. DTLS provides communication privacy on top of UDP, which is similar to how TLS works on top of TCP. Instead of TCP flags, which was used as a feature in previous case, the type of DTLS message was used as a feature along with packet-size. Following is the approach used to identify these messages.

In order to identify the type of message, whether it is part of initial key exchange, or application data being exchanged, raw DTLS payload of the packet from scapy was used. From the documentation<sup>6</sup> and wireshark, following inference was made about the structure of DTLS packet header and is shown in Figure 4.20.

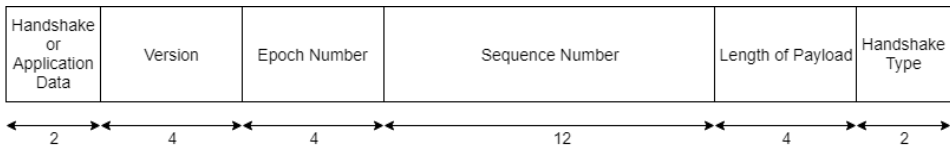


Figure 4.20: DTLS header format.

First two bytes specify whether the packet contains handshake message or application data. Next four bytes contains the DTLS version used. Epoch number is used in conjunction with sequence number, it starts with zero and is incremented by 1, every time there is a change in cipher state between hosts. Next 12 bytes are used to specify sequence number of the packet. In the case of DTLS, it starts with zero and is incremented. It is followed by 4 bytes which signify the length of payload. Last two bytes specify the handshake type (Client Hello, Server Hello, Client Key Exchange etc.) that is part of this packet.

### 4.7.2. BACKGROUND TRAFFIC GENERATED BY IKEA LIGHTS

Traffic generated by IKEA bridge apart from its interaction with the mobile app was analysed. It was observed that similar to Philips Hue, IKEA bridge also communicated with hosts performing HTTP calls, followed by some response using TCP. Methodology used in previous study with background traffic generated by Hue was applied to this traffic

<sup>6</sup><https://tools.ietf.org/html/rfc6347>

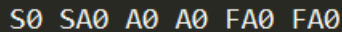


and state machines were obtained. Details regarding those state machine will be discussed in next chapter.

## 4.8. BASELINE: N-GRAMS

To compare the results of this thesis, N-Grams was used as a baseline. N-Grams captures sequences of length N. It iterates over the training data as a sliding window. N-Grams is used as the baseline measure here because similar to state machine, N-Grams also accounts for the sequence of events occurred.

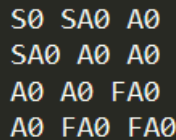
Consider the following trace in Figure 4.21.



S0 SA0 A0 A0 FA0 FA0

Figure 4.21: A simple sequence from which we can generate N-grams.

To see how N-Grams work, consider a value of N=3 in N-grams, we would get the following trigrams(3-grams) shown in Figure 4.22.



S0 SA0 A0  
SA0 A0 A0  
A0 A0 FA0  
A0 FA0 FA0

Figure 4.22: Trigrams obtained from the sequence in Figure 4.21.

Probabilities of such sequences are calculated. In the test data, if there are any sequences which never occur/rarely occur in training data, then it is marked as anomalous.

N-Grams work with categorical data[32] and since packet size is a continuous data, we need a way to discretise it. Percentiles is one way to convert continuous variables into discrete variables. So in this project, with the baseline, instead of using absolute value of packet size as a feature, we use its percentiled value. All other features remain similar to those provided for state machine learning module as they are categorical.

## 4.9. ANSWERS TO RESEARCH QUESTIONS ADDRESSED IN THIS CHAPTER

Two state machines were used to define the behaviour of IoT device during its normal operation. One to represent the traffic exchanged between IoT device and mobile and one to represent the background traffic.

Results of this chapter can be used to answer the following research questions:

RQ1b: Which high-level features can be used to define the behaviour of IoT devices?

High-level features such as Packet size, TCP Flag, domain of HTTP call, HTTP response code, TLS messages were used to describe the behaviour of IoT device using state machines in this thesis. As IKEA lights use DTLS protocol, instead of TCP flags, which was used as a feature in Philips Hue, the type of DTLS message was used as a feature along with packet-size.

RQ1c: How can these high-level features be used to build a state machine?

Each packet in the conversation between hosts was considered as a transition in state machine. One of the protocol specific information such as TCP Flag, domain of HTTP call, HTTP response code, TLS messages, DTLS message was concatenated with Packet size to create the transitions that could be used for state machines.

## 4

### 4.10. SUMMARY

This chapter started with the description of state machine learning module that was used to build state machines in this project. It was later used to build state machines explaining the traffic generated between Philips Hue and Mobile app controlling it. Challenges were encountered when dealing with background traffic generated by Hue because of packets received out of order. They were solved with the help of TCP sequence reordering to replicate the order of packets in which they were generated by end hosts. Once this problem was tackled, it was possible to generate interpretable state machines explaining the background traffic generated by Philips hue while communicating with multiple hosts over a period of time. To extend the methodology for another device, traffic generated from IKEA lights was used. N-Grams was used as a baseline in this thesis.

# 5

## RESULTS: STATE MACHINES REPRESENTING THE NORMAL BEHAVIOUR

*This chapter discusses the details about state machines that we obtain upon usage of methodology in previous chapter. It starts with a discussion about state machines representing traffic between Hue and Mobile App, followed by background traffic. Extension of the methodology to IKEA lights and the resulting state machines are also discussed.*

### 5.1. STATE MACHINES REPRESENTING TRAFFIC BETWEEN MOBILE APP AND HUE

As state machine explaining packets captured while changing colours of light was obtained in the previous chapter, traffic captured while performing other actions were combined with this traffic. In each pcap file, only one type of action (Turning lights On/Off or Changing Colours or Changing Themes) was performed and the time of the action was noted.

Figure 5.1 shows the packets recorded while performing "Changing Theme" of Philips Hue light. It can be seen that the highlighted packets match the transitions 5, 6 and 7 from the state machine in Figure 5.4.

No.	Time	Source	Destination	Protocol	Length	Info
152	7.990740	192.168.8.221	192.168.8.142	TCP	54	TCP Window Update] 53118 → 443 [ACK]
153	7.100288	192.168.8.221	192.168.8.142	TCP	66	TCP Window Update] 53118 → 443 [ACK]
154	7.110206	192.168.8.221	192.168.8.142	TLSv1.2	274	Application Data
155	7.150028	192.168.8.142	192.168.8.221	TCP	60	443 → 53118 [ACK] Seq=60762 Ack=1513
156	7.150248	192.168.8.142	192.168.8.221	TLSv1.2	653	Application Data
157	7.150335	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
158	7.153375	192.168.8.221	192.168.8.142	TCP	54	53118 → 443 [ACK] Seq=1513 Ack=61361
159	7.153513	192.168.8.221	192.168.8.142	TCP	54	53118 → 443 [ACK] Seq=1513 Ack=61395

▶ Frame 154: 274 bytes on wire (2192 bits), 274 bytes captured (2192 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: PhilipsL ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 53118, Dst Port: 443, Seq: 1293, Ack: 60762, Len: 220  
 ▶ Secure Sockets Layer

Figure 5.1: Packets exchanged while *Changing Themes* of Philips Hue. Empty Acknowledgements have to be ignored.

Figure 5.2 shows the packets recorded while performing "Changing Colours" of Philips Hue light. It can be seen that the highlighted packets match the transitions 1, 2 and 7 from the state machine in Figure 5.4.

eth\_addr ==

No.	Time	Source	Destination	Protocol	Length	Info
467	60.118049	192.168.8.221	192.168.8.142	TLSv1.2	272	Application Data
468	60.139002	192.168.8.142	192.168.8.221	TLSv1.2	647	Application Data
469	60.139682	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
470	60.141899	192.168.8.221	192.168.8.142	TCP	54	57042 → 443 [ACK] Seq=3050
471	60.142450	192.168.8.221	192.168.8.142	TCP	54	57042 → 443 [ACK] Seq=3050
472	60.183199	192.168.8.221	192.168.8.142	TLSv1.2	272	Application Data
473	60.205288	192.168.8.142	192.168.8.221	TLSv1.2	647	Application Data
474	60.205831	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
475	60.208004	192.168.8.221	192.168.8.142	TCP	54	57042 → 443 [ACK] Seq=3268
476	60.208074	192.168.8.221	192.168.8.142	TCP	54	57042 → 443 [ACK] Seq=3268

▶ Frame 467: 272 bytes on wire (2176 bits), 272 bytes captured (2176 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: PhilipsL ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 57042, Dst Port: 443, Seq: 2832, Ack: 43934, Len: 218  
 ▶ Secure Sockets Layer

Figure 5.2: Packets exchanged while *Changing Colours* of Philips Hue. Empty Acknowledgements have to be ignored.

Figure 5.3 shows the packets recorded while performing "Changing Colours" of Philips Hue light. It can be seen that the highlighted packets match the transitions 3 and 4 and 7 from the state machine in Figure 5.4.

No.	Time	Source	Destination	Protocol	Length	Info
495	31.143542	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6386
500	31.664974	192.168.8.221	192.168.8.142	TLSv1.2	259	Application Data
501	31.674953	192.168.8.142	192.168.8.221	TLSv1.2	638	Application Data
502	31.676430	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
503	31.676634	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6591
504	31.678503	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6591
515	31.913061	192.168.8.221	192.168.8.142	TLSv1.2	218	Application Data
518	31.921031	192.168.8.142	192.168.8.221	TCP	1514	443 → 56764 [ACK] Seq=95276
519	31.921207	192.168.8.142	192.168.8.221	TCP	1514	443 → 56764 [ACK] Seq=96736
520	31.921311	192.168.8.142	192.168.8.221	TLSv1.2	162	Application Data
522	31.922757	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6755
523	31.922844	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6755
524	31.927251	192.168.8.142	192.168.8.221	TLSv1.2	88	Application Data
525	31.932034	192.168.8.221	192.168.8.142	TCP	54	56764 → 443 [ACK] Seq=6755

▶ Frame 500: 259 bytes on wire (2072 bits), 259 bytes captured (2072 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: PhilipsL ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 56764, Dst Port: 443, Seq: 6386, Ack: 94658, Len: 205  
 ▶ Secure Sockets Layer

Figure 5.3: Packets exchanged while *Turning the lights On/Off* on Philips Hue. Empty Acknowledgements have to be ignored.

Traffic from all the pcap files was used as input to state machine learning module described in the previous chapter. This will result in a state machine explaining packets captured while performing actions such as Turning the light On/Off, Changing Colours, Changing Themes as seen in Figure 5.4.

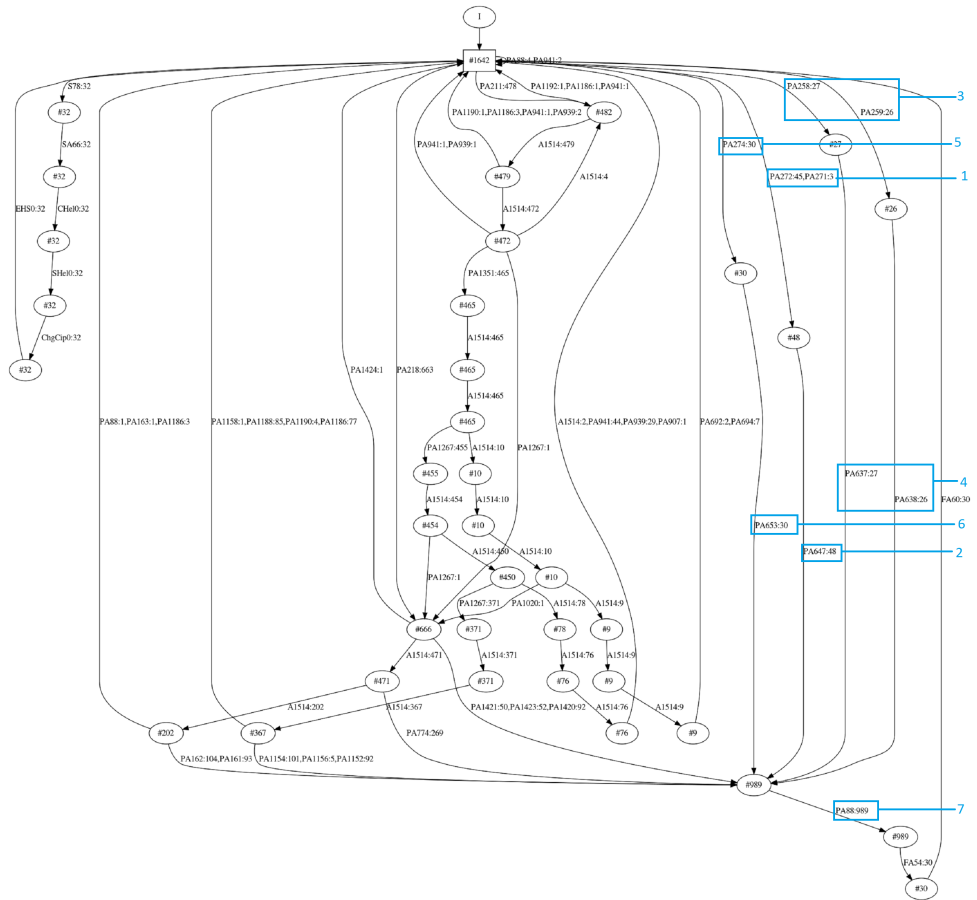


Figure 5.4: *State machine* depicting the traffic captured while performing the actions of *"Changing colours"*, *"Changing Theme"* and *"Turning On/Off"* for Hue lights. Each state contains the number of times it has occurred. A symbol/transition "S78" means it is a packet with TCP flag SYN and size of 78 bytes

Following are the different packets exchanged while performing these operations based on the state machine in Figure 5.4.

- 1 Packet sent from Mobile app to Hue to change the colour of light.
- 2 Response from Hue to App for the previous command of changing colour.
- 3 Packet sent from Mobile app to Hue to turn the light On/Off.
- 4 Response from Hue to App for the previous command of turning light On/Off.
- 5 Packet sent from Mobile app to Hue to change theme of lights.
- 6 Response from Hue to App for the previous command of changing theme.
- 7 All the responses from Hue to Mobile are followed by this packet.

It was important to verify if the actions marked in the state machine indeed represent the packets exchanged while performing these operations. To achieve this, once these state machines were learnt, the operations of "Changing colours", "Changing Theme" and "Turning On/Off" for Hue lights were performed again and time of the action was recorded.

## 5.2. STATE MACHINES REPRESENTING BACKGROUND TRAFFIC GENERATED BY HUE

This section contains the description of individual transitions that are part of state machines representing the background traffic generated by Hue.

State machine in Figure 5.5 shows the state machine obtained from most of the background traffic from Hue (few hosts were ignored for better readability. They will be included in next figure). Each chain of transitions represents a conversation between Philips hue and some host.

All the transitions before and after the marked transition represent the following:

- 1 With some hosts, Hue creates a TLS session and terminates it without exchanging any other packets.
- 2 Performs an HTTP call to `www.ecdinterface.philips.com` and obtains a reply with HTTP status 400. It retries again to obtain a response with HTTP status 200 and closes the session.
- 3 Performs an HTTP call to `www.ecdinterface.philips.com` and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.
- 4 Performs an HTTP call to `dcp.dc1.philips.com` and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.

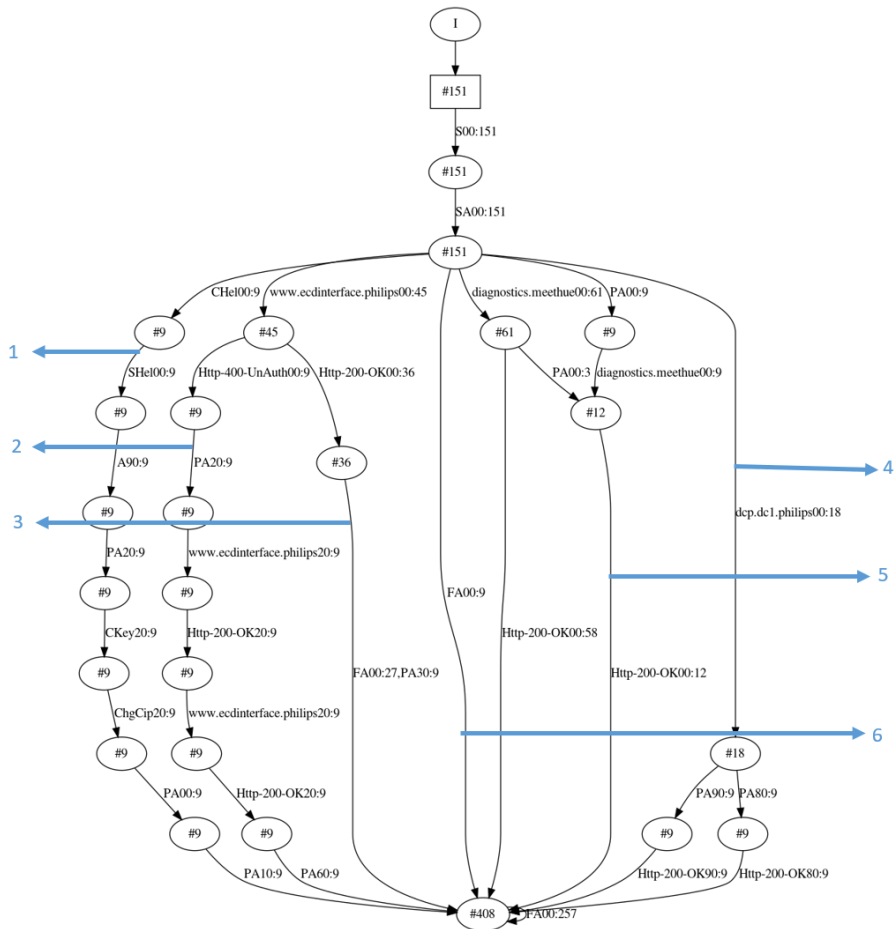


Figure 5.5: State machine explaining Background traffic *After* performing TCP reordering. Here, communication with one of the hosts ignored for better readability.

- 5 Performs an HTTP call to diagnostics.meethue.com and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.
- 6 Opens a TCP session and closes it without any operation with some hosts.



State machine in Figure 5.6 shows the state machine obtained from background traffic with all the hosts from Hue captured in this project.

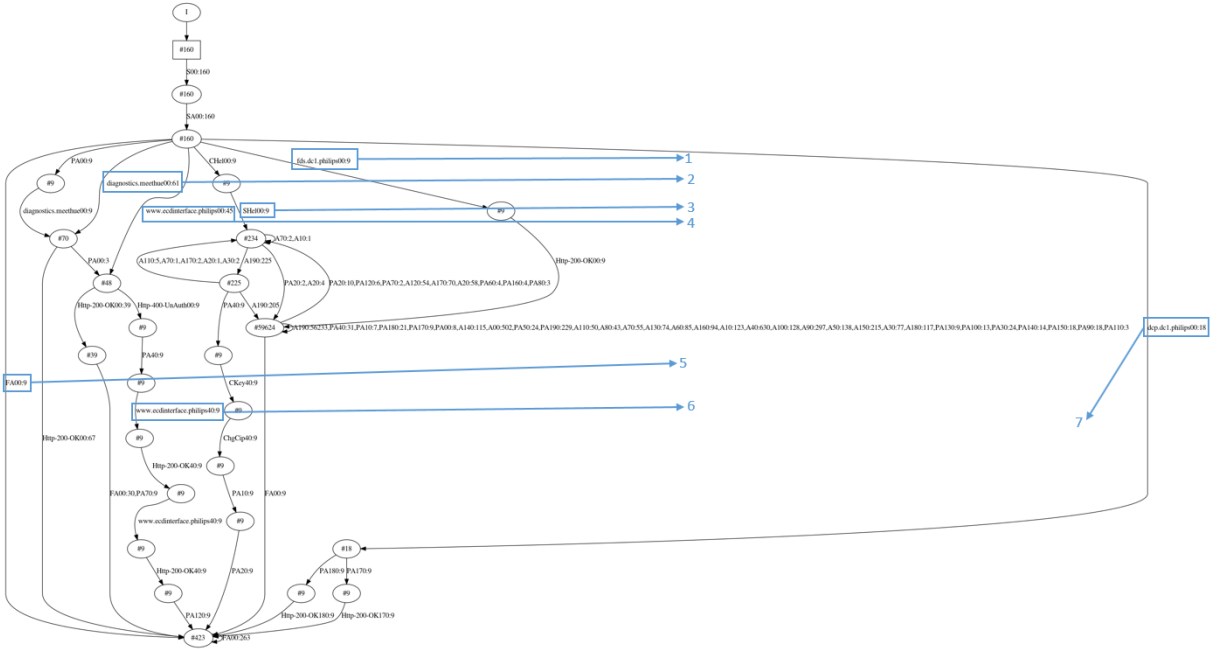


Figure 5.6: State machine explaining Background traffic with all the hosts with Philips Hue using TCP based on traffic captured in this project.

All the transitions before and after the marked transition represent the following:

- 1 Performs an HTTP call to `fds.dc1.philips.com` and obtains a reply with HTTP status 200 and response of many packets. **This communication was ignored in the previous state machine for better readability.**
- 2 Performs an HTTP call to `diagnostics.meethue.com` and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.
- 3 With some hosts, Hue creates a TLS session and closes it.
- 4 Performs an HTTP call to `www.ecdinterface.philips.com` and obtains a reply with HTTP status 400. It retries again to obtain a response with HTTP status 200 and closes the session.
- 5 Opens a TCP session and closes it without any operation with some hosts.
- 6 Performs an HTTP call to `www.ecdinterface.philips.com` and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.

- 7 Performs an HTTP call to `dcp.dc1.philips.com` and obtains a reply with HTTP status 200. Closes the session with packets containing FIN-ACK in the last state.

To validate the claims made in state machines in Figure 5.5 and Figure 5.6, we can refer to the screenshot of Wireshark in Figure 5.7 showing the details regarding HTTP calls performed by Hue as part of one of the pcaps captured. Highlighted part of the figure shows various domains such as `fds.dc1.philips.com`, `diagnostics.meethue.com`, `ecdinterface.philips.com` and `dcp.dc1.philips.com`. These were represented as 1, 2, 4 and 6 in the state machine in Figure 5.6.

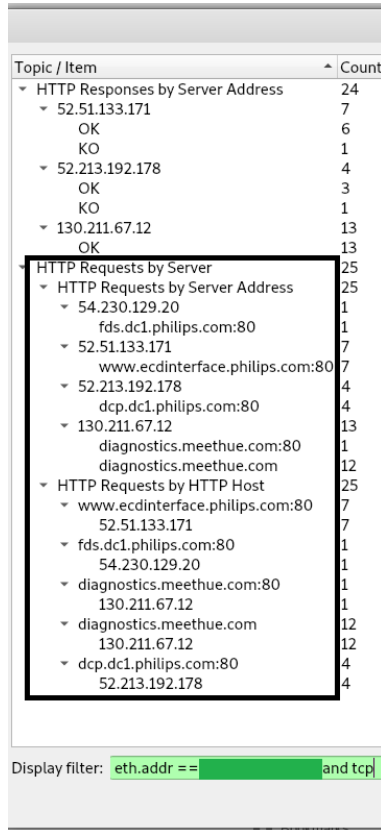


Figure 5.7: Screenshot from Wireshark showing various stats related to HTTP calls performed by IOT device from one of the pcaps captured.

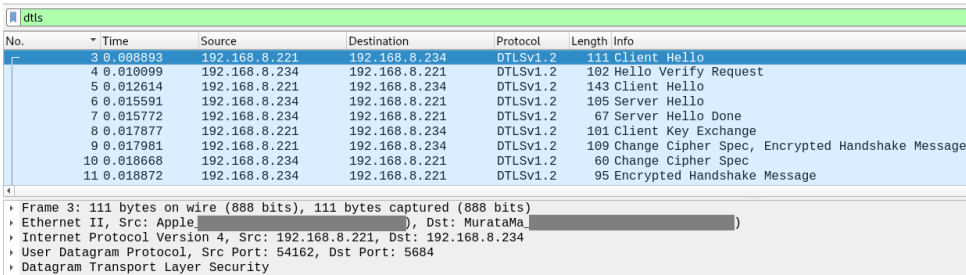
## 5.3. STATE MACHINES REPRESENTING TRAFFIC GENERATED BY IKEA LIGHTS

Once state machines representing the behaviour of Philips Hue lighting system was learnt, the work was then extended to another lighting system. IKEA lights work similarly to Philips Hue lighting system. It consists of a bridge which controls the lights. To control the bridge, commands can be issued by user through Mobile app. The connections and methodology used to capture the traffic remained the same. It was observed that IKEA bridge communicates with Mobile app using DTLS protocol, version 1.2<sup>1</sup>.

### 5.3.1. TRAFFIC BETWEEN SMARTPHONE AND IKEA LIGHTS

As discussed in previous chapter, we were able to identify the type of DTLS message exchanged to distinguish between packets corresponding to key exchange and application data.

During the inspection of pcap files, it was noticed that every time a user opens the app, a DTLS session was created by exchanging key related information as seen in Figure 5.8.

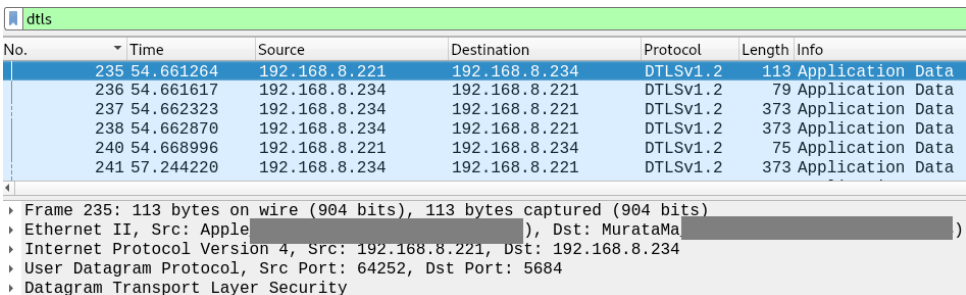


No.	Time	Source	Destination	Protocol	Length	Info
3	0.008893	192.168.8.221	192.168.8.234	DTLSv1.2	111	Client Hello
4	0.010099	192.168.8.234	192.168.8.221	DTLSv1.2	102	Hello Verify Request
5	0.012614	192.168.8.221	192.168.8.234	DTLSv1.2	143	Client Hello
6	0.015591	192.168.8.234	192.168.8.221	DTLSv1.2	105	Server Hello
7	0.015772	192.168.8.234	192.168.8.221	DTLSv1.2	67	Server Hello Done
8	0.017877	192.168.8.221	192.168.8.234	DTLSv1.2	101	Client Key Exchange
9	0.017981	192.168.8.221	192.168.8.234	DTLSv1.2	109	Change Cipher Spec, Encrypted Handshake Message
10	0.018668	192.168.8.234	192.168.8.221	DTLSv1.2	60	Change Cipher Spec
11	0.018872	192.168.8.234	192.168.8.221	DTLSv1.2	95	Encrypted Handshake Message

▶ Frame 3: 111 bytes on wire (888 bits), 111 bytes captured (888 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: MurataMa ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.234  
 ▶ User Datagram Protocol, Src Port: 54162, Dst Port: 5684  
 ▶ Datagram Transport Layer Security

Figure 5.8: Packets representing key exchange between smart phone and IKEA lights.

Time at which actions such as Turning the lights On/Off and changing brightness were recorded and packets exchanged were inspected. Figure 5.9 shows the packets exchanged when turning the lights On/Off. Figure 5.10 shows the packets exchanged when changing the brightness of lights.



No.	Time	Source	Destination	Protocol	Length	Info
235	54.661264	192.168.8.221	192.168.8.234	DTLSv1.2	113	Application Data
236	54.661617	192.168.8.234	192.168.8.221	DTLSv1.2	79	Application Data
237	54.662323	192.168.8.234	192.168.8.221	DTLSv1.2	373	Application Data
238	54.662870	192.168.8.234	192.168.8.221	DTLSv1.2	373	Application Data
240	54.668996	192.168.8.221	192.168.8.234	DTLSv1.2	75	Application Data
241	57.244220	192.168.8.234	192.168.8.221	DTLSv1.2	373	Application Data

▶ Frame 235: 113 bytes on wire (904 bits), 113 bytes captured (904 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: MurataMa ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.234  
 ▶ User Datagram Protocol, Src Port: 64252, Dst Port: 5684  
 ▶ Datagram Transport Layer Security

Figure 5.9: Packets exchanged while turning the lights On/Off.

<sup>1</sup><https://tools.ietf.org/html/rfc6347>

No.	Time	Source	Destination	Protocol	Length	Info
275	101.483545	192.168.8.221	192.168.8.234	DTLSv1.2	120	Application Data
276	101.483950	192.168.8.234	192.168.8.221	DTLSv1.2	79	Application Data
277	101.484764	192.168.8.234	192.168.8.221	DTLSv1.2	373	Application Data
278	101.489117	192.168.8.221	192.168.8.234	DTLSv1.2	75	Application Data

<ul style="list-style-type: none"> <li>▶ Frame 275: 120 bytes on wire (960 bits), 120 bytes captured (960 bits)</li> <li>▶ Ethernet II, Src: Apple_ , Dst: MurataMa )</li> <li>▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.234</li> <li>▶ User Datagram Protocol, Src Port: 64252, Dst Port: 5684</li> <li>▶ Datagram Transport Layer Security</li> </ul>						
---	--	--	--	--	--	--

Figure 5.10: Packets exchanged while changing the brightness of light.

Upon usage of methodology discussed in previous section, we obtain the state machine in Figure 5.11 representing the traffic exchanged between IKEA lights and smart phone.

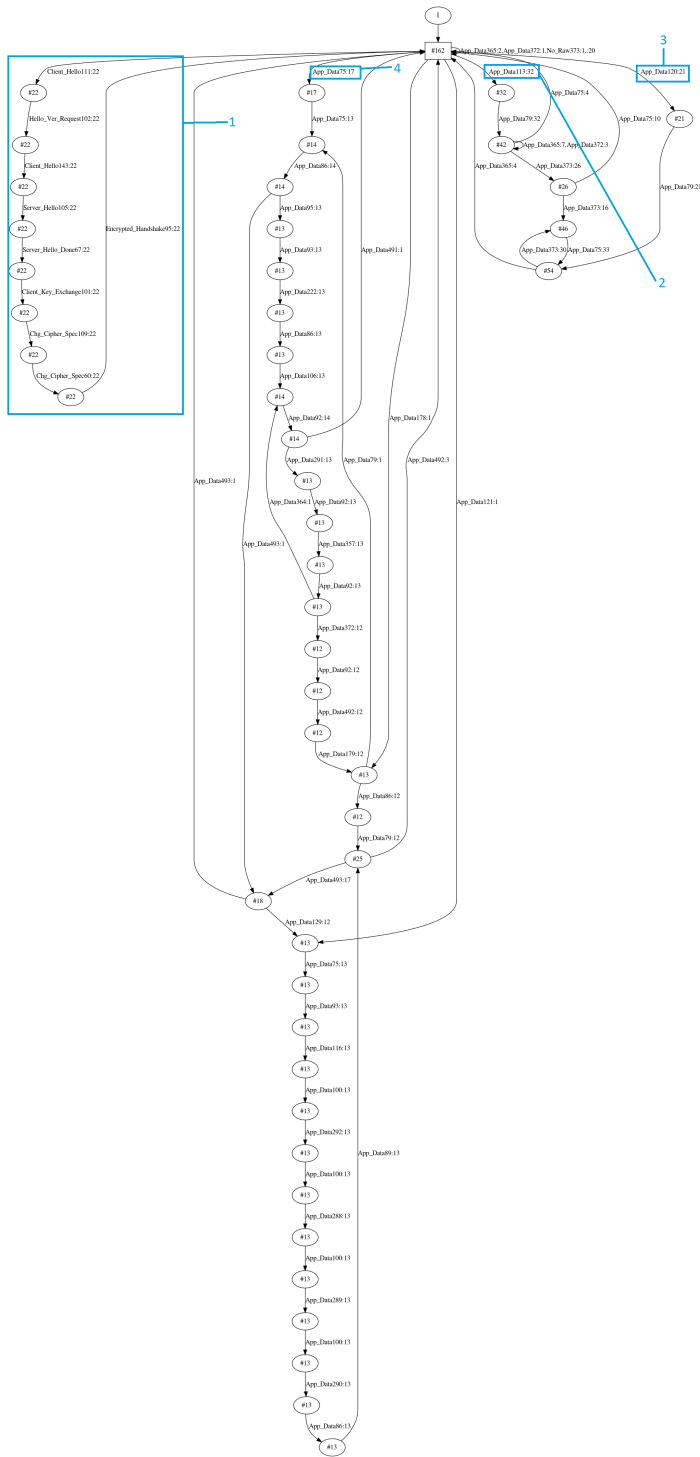


Figure 5.11: State machine explaining Background traffic generated by IKEA lights. Here, communication with one of the hosts ignored for better readability.

All the transitions after the highlighted parts in Figure 5.11 represent the following:

- 1 Packets exchanged while exchanging key related information of DTLS.
- 2 Packets exchanged while performing On/Off operation on IKEA lights.
- 3 Packets exchanged while performing Changing brightness operation on IKEA lights.
- 4 Packets exchanged after creation of DTLS session when the app is first opened.

Here, it is important to note that, unlike previous case, we were not able to identify all the transitions that are part of the state machines. Additionally, re-transmitted packets in case of UDP were not effectively handled, leading to the state machine in Figure 5.11. In future work, these aspects can be better handled.

### 5.3.2. BACKGROUND TRAFFIC GENERATED BY IKEA LIGHTS

Similar to Philips Hue, IKEA lights also communicates with external hosts to perform HTTP calls and in some cases, just creates and terminates a TCP session regularly. Additionally, IKEA lights were also seen communicating with some hosts using TLS to exchange some packets. Background traffic generated by IKEA lights with some hosts was used to generate state machine as shown in Figure 5.12.

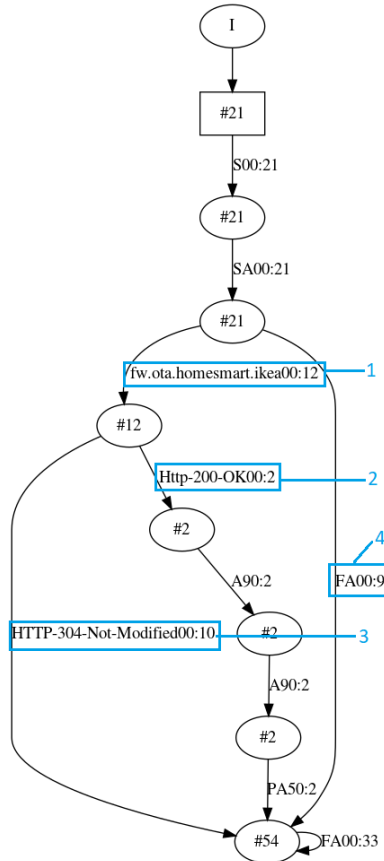


Figure 5.12: State machine explaining Background traffic generated by IKEA lights. Here, communication with one of the hosts ignored for better readability.

Highlighted parts of the state machine represent the following.

- 1 HTTP call from IKEA lights to fw.ota.homesmart.ikea.com.
- 2 A reply to the previous call with HTTP status 200 followed by some packets.
- 3 A reply to the previous call with HTTP status 304.
- 4 With some hosts, it opens and closes a TCP session without performing any action.





Figure 5.13 shows the state machine representing the background traffic with all the hosts observed as part of this thesis. While most of the details remain similar to previous state machine, **one significant difference is the communication represented by 5 in the figure**. It represents the communication between IKEA lights and some hosts where a TLS connection is established, followed by exchange of some packets and termination of session.

## 5.4. ANSWERS TO RESEARCH QUESTIONS ADDRESSED IN THIS CHAPTER

In this chapter, we discussed the details of results obtained upon implementing the methodology discussed in previous chapter.

Results of this chapter can be used to answer the following research questions:

RQ1a: What are the different behaviours performed by IoT device during its normal operation?

In the state machine representing the traffic between IoT device and mobile phone, TLS was used for communication between these two devices. It involved sending a single packet from smartphone followed by multiple packets of response from IOT device depending on the input.

In the state machine representing background traffic, Philips Hue communicated with few hosts in the form of HTTP POST/GET calls followed by uploading/downloading of data. Similar was the case with IKEA lights.

Results obtained in section 5.1 can be used to answer our fifth research question:

RQ5: Which commands issued by smartphone to IoT device can be detected using network traffic metadata?

Using state machines learnt from network metadata, the following commands issued from smartphone to IoT device were detected: the commands to turn the lights On/Off, Changing Colours, Changing themes.

In 5.3, we were able to generate the state machine explaining the background traffic generated by IKEA lights. The results in that section can be used to answer our next research question

RQ5: How can the approach be extended to represent the behaviour of another IoT device?

To represent background traffic generated by IKEA lights, no changes were made to methodology. However, to model the traffic between IKEA lights and Smartphone, it required changing the methodology to include a new protocol(DTLS).

## 5.5. SUMMARY

In this chapter, details regarding the state machines were discussed. The chapter started with description of state machines explaining the behaviour of Philips Hue. It was followed by details regarding state machines obtained to represent the behaviour of IKEA lights.

# 6

## APPLICATION OF STATE MACHINES ON ATTACK TRAFFIC AND REAL WORLD DATA

*This chapter contains the description of attack traffic and its implications on the state machines which were learnt in the previous chapter. It contains details about setup and tools used to generate attack traffic. It is followed by details of setup used to capture traffic in real-world data outside the lab environment. This is then used to validate the state machines using different evaluation methods.*

### 6.1. CASE STUDY 1: ATTACK TRAFFIC

Once state machines were learnt on the normal behaviour of the device, attack traffic was generated to see if we can detect it based on these state machines. This section contains the details regarding attack traffic and its generation.

#### 6.1.1. ATTACK TYPE AND TOOLS USED

The type of attack focused in this thesis is a Denial-Of-Service attack. Here, the attacker floods the victim with random requests. This causes victim to allocate more resources and gets overloaded, thereby unable to serve the requests of genuine users.

The attack was performed using hping<sup>1</sup> utility of Kali Linux. hping is a command line tool that is being used for security testing and auditing. It can be used to generate and analyse packets complying various protocols in TCP/IP protocol stack. Here, it was used to flood many SYN packets from attacker to victim thereby overloading the resources of victim. This attack is called a SYN-flood attack. The rate at which these packets are sent can also be configured based on requirements. This can be specified in terms of

---

<sup>1</sup><http://www.hping.org/>

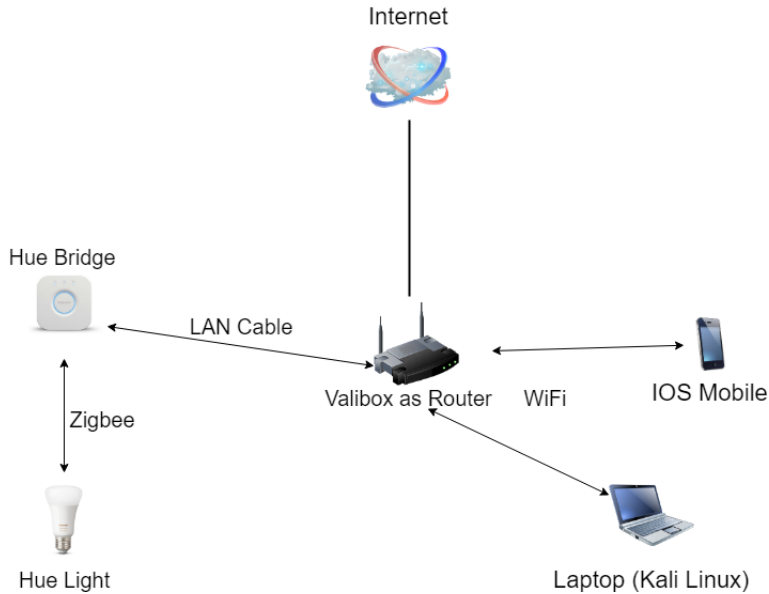


Figure 6.1: IOT device as victim. A laptop running Kali linux performs attack on IOT device till it becomes unresponsive.

interval between packets that are sent. If the parameter 'flood' is used, it sends packets with interpacket time interval as less as possible. Various details regarding parameters for this tool can be found here <sup>2</sup>. In this project, this time interval was varied to see the highest amount of time interval between packets to render Philips hue unresponsive. Upon experimentation, a duration of 8 milliseconds between packets made Philips hue unresponsive within a few seconds.

### 6.1.2. IOT DEVICE AS VICTIM

In this scenario, IoT device was made as victim of the attack and flooded with many requests from the attacker. The attacker here is a laptop running kali Linux. Experimental setup used to perform this attack was the same as the one used to capture traffic described in section 3.1. The only difference is here a Laptop running Kali Linux is used instead of a regular Linux laptop.

Once packet capturing process is started, mobile app is interacted with the lights to generate normal traffic. After this, attack is started and continued till the lights stop responding. Once the mobile app fails to control the light, the attack and packet capture process is stopped. Figure 6.2 shows an example of how this attack traffic looks in Wireshark. Figure 6.3 shows the overlap of packets between attack traffic and normal traffic.

<sup>2</sup><http://www.hping.org/manpage.html>

No.	Time	Source	Destination	Protocol	Length	Info
19590	11.706768	192.168.8.165	192.168.8.142	TCP	54	46588 → 443 [SYN] Seq=0 Win=512 Len=0
19591	11.707573	192.168.8.165	192.168.8.142	TCP	54	46589 → 443 [SYN] Seq=0 Win=512 Len=0
19592	11.707690	192.168.8.165	192.168.8.142	TCP	54	46590 → 443 [SYN] Seq=0 Win=512 Len=0
19593	11.707750	192.168.8.165	192.168.8.142	TCP	54	46591 → 443 [SYN] Seq=0 Win=512 Len=0
19594	11.707797	192.168.8.165	192.168.8.142	TCP	54	46592 → 443 [SYN] Seq=0 Win=512 Len=0
19596	11.708298	192.168.8.165	192.168.8.142	TCP	54	46593 → 443 [SYN] Seq=0 Win=512 Len=0
19597	11.708409	192.168.8.165	192.168.8.142	TCP	54	46594 → 443 [SYN] Seq=0 Win=512 Len=0
19599	11.709101	192.168.8.221	192.168.8.142	TLvSv1.2	211	Application Data
19600	11.709230	192.168.8.165	192.168.8.142	TCP	54	46595 → 443 [SYN] Seq=0 Win=512 Len=0
19601	11.709630	192.168.8.165	192.168.8.142	TCP	54	46596 → 443 [SYN] Seq=0 Win=512 Len=0
19603	11.710120	192.168.8.165	192.168.8.142	TCP	54	46597 → 443 [SYN] Seq=0 Win=512 Len=0
19604	11.710211	192.168.8.165	192.168.8.142	TCP	54	46598 → 443 [SYN] Seq=0 Win=512 Len=0
19605	11.710264	192.168.8.165	192.168.8.142	TCP	54	46599 → 443 [SYN] Seq=0 Win=512 Len=0
19607	11.711247	192.168.8.165	192.168.8.142	TCP	54	46600 → 443 [SYN] Seq=0 Win=512 Len=0
19608	11.711368	192.168.8.165	192.168.8.142	TCP	54	46601 → 443 [SYN] Seq=0 Win=512 Len=0
19609	11.712049	192.168.8.165	192.168.8.142	TCP	54	46602 → 443 [SYN] Seq=0 Win=512 Len=0
19610	11.712151	192.168.8.165	192.168.8.142	TCP	54	46603 → 443 [SYN] Seq=0 Win=512 Len=0
19611	11.712202	192.168.8.165	192.168.8.142	TCP	54	46604 → 443 [SYN] Seq=0 Win=512 Len=0
19612	11.712256	192.168.8.165	192.168.8.142	TCP	54	46605 → 443 [SYN] Seq=0 Win=512 Len=0
19613	11.712308	192.168.8.165	192.168.8.142	TCP	54	46606 → 443 [SYN] Seq=0 Win=512 Len=0
19614	11.712356	192.168.8.165	192.168.8.142	TCP	54	46607 → 443 [SYN] Seq=0 Win=512 Len=0
19615	11.713140	192.168.8.165	192.168.8.142	TCP	54	46608 → 443 [SYN] Seq=0 Win=512 Len=0
19616	11.713259	192.168.8.165	192.168.8.142	TCP	54	46609 → 443 [SYN] Seq=0 Win=512 Len=0
19617	11.713312	192.168.8.165	192.168.8.142	TCP	54	46610 → 443 [SYN] Seq=0 Win=512 Len=0
19618	11.713360	192.168.8.165	192.168.8.142	TCP	54	46611 → 443 [SYN] Seq=0 Win=512 Len=0
19619	11.714231	192.168.8.165	192.168.8.142	TCP	54	46612 → 443 [SYN] Seq=0 Win=512 Len=0
19620	11.714363	192.168.8.165	192.168.8.142	TCP	54	46613 → 443 [SYN] Seq=0 Win=512 Len=0
19621	11.714413	192.168.8.165	192.168.8.142	TCP	54	46614 → 443 [SYN] Seq=0 Win=512 Len=0
19622	11.714462	192.168.8.165	192.168.8.142	TCP	54	46615 → 443 [SYN] Seq=0 Win=512 Len=0
19623	11.714510	192.168.8.165	192.168.8.142	TCP	54	46616 → 443 [SYN] Seq=0 Win=512 Len=0
19624	11.714558	192.168.8.165	192.168.8.142	TCP	54	46617 → 443 [SYN] Seq=0 Win=512 Len=0
19625	11.715271	192.168.8.165	192.168.8.142	TCP	54	46618 → 443 [SYN] Seq=0 Win=512 Len=0
19629	11.716054	192.168.8.165	192.168.8.142	TCP	54	42262 → 443 [RST] Seq=1 Win=0 Len=0
19630	11.716154	192.168.8.165	192.168.8.142	TCP	54	46275 → 443 [RST] Seq=1 Win=0 Len=0
19631	11.716202	192.168.8.165	192.168.8.142	TCP	54	46619 → 443 [SYN] Seq=0 Win=512 Len=0

▶ Frame 19599: 211 bytes on wire (1688 bits), 211 bytes captured (1688 bits)  
 ▶ Ethernet II, Src: Apple ( ), Dst: PhilipsI ( )  
 ▶ Internet Protocol Version 4, Src: 192.168.8.221, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 50270, Dst Port: 443, Seq: 817, Ack: 3728, Len: 157

Figure 6.2: Screenshot from Wireshark showing how attack traffic looks like when performed on IoT device. Attacker, in this case, IP address 192.168.8.165 floods IoT device with multiple SYN packets and RST packets. This will result in IoT device not responding to legitimate requests.

No.	Time	Source	Destination	Protocol	Length	Info
19760	11.745235	192.168.8.165	192.168.8.142	TCP	54	46715 → 443 [SYN] Seq=0 Win=512 Len=0
19761	11.745284	192.168.8.165	192.168.8.142	TCP	54	46716 → 443 [SYN] Seq=0 Win=512 Len=0
19762	11.745352	192.168.8.142	192.168.8.165	TCP	60	443 → 46713 [SYN, ACK] Seq=0 Ack=1 Win=0
19763	11.745915	192.168.8.165	192.168.8.142	TCP	54	46717 → 443 [SYN] Seq=0 Win=512 Len=0
19764	11.746016	192.168.8.165	192.168.8.142	TCP	54	46718 → 443 [SYN] Seq=0 Win=512 Len=0
19765	11.746070	192.168.8.165	192.168.8.142	TCP	54	46719 → 443 [SYN] Seq=0 Win=512 Len=0
19766	11.746586	192.168.8.142	192.168.8.221	TCP	1514	443 → 50270 [ACK] Seq=12078 Ack=974 Win=0 Len=0
19767	11.746900	192.168.8.142	192.168.8.221	TCP	1514	443 → 50270 [ACK] Seq=13538 Ack=974 Win=0 Len=0
19768	11.747021	192.168.8.142	192.168.8.221	TLSv1.2	1267	Application Data
19769	11.747181	192.168.8.165	192.168.8.142	TCP	54	46720 → 443 [SYN] Seq=0 Win=512 Len=0
19770	11.747262	192.168.8.165	192.168.8.142	TCP	54	46721 → 443 [SYN] Seq=0 Win=512 Len=0
19771	11.747810	192.168.8.165	192.168.8.142	TCP	54	46722 → 443 [SYN] Seq=0 Win=512 Len=0
19772	11.747922	192.168.8.165	192.168.8.142	TCP	54	46723 → 443 [SYN] Seq=0 Win=512 Len=0
19773	11.747974	192.168.8.221	192.168.8.142	TCP	54	50270 → 443 [ACK] Seq=974 Ack=6648 Win=0 Len=0
19774	11.748172	192.168.8.142	192.168.8.221	TCP	1514	443 → 50270 [ACK] Seq=16211 Ack=974 Win=0 Len=0
19775	11.748326	192.168.8.142	192.168.8.221	TCP	1514	443 → 50270 [ACK] Seq=17671 Ack=974 Win=0 Len=0
19776	11.748418	192.168.8.142	192.168.8.221	TLSv1.2	1267	Application Data
19777	11.749085	192.168.8.165	192.168.8.142	TCP	54	46724 → 443 [SYN] Seq=0 Win=512 Len=0
19778	11.749805	192.168.8.165	192.168.8.142	TCP	54	46725 → 443 [SYN] Seq=0 Win=512 Len=0
19779	11.750913	192.168.8.165	192.168.8.142	TCP	54	46726 → 443 [SYN] Seq=0 Win=512 Len=0
19780	11.751039	192.168.8.221	192.168.8.142	TCP	54	50270 → 443 [ACK] Seq=974 Ack=7945 Win=0 Len=0
19781	11.751102	192.168.8.221	192.168.8.142	TCP	54	50270 → 443 [ACK] Seq=974 Ack=10865 Win=0 Len=0
19782	11.751151	192.168.8.221	192.168.8.142	TCP	54	50270 → 443 [ACK] Seq=974 Ack=12078 Win=0 Len=0
19783	11.751246	192.168.8.142	192.168.8.221	TLSv1.2	263	Application Data
19784	11.752247	192.168.8.165	192.168.8.142	TCP	54	46727 → 443 [SYN] Seq=0 Win=512 Len=0
19785	11.752329	192.168.8.165	192.168.8.142	TCP	54	46728 → 443 [SYN] Seq=0 Win=512 Len=0
19786	11.752668	192.168.8.165	192.168.8.142	TCP	54	46729 → 443 [SYN] Seq=0 Win=512 Len=0
19787	11.753138	192.168.8.165	192.168.8.142	TCP	54	46730 → 443 [SYN] Seq=0 Win=512 Len=0
19788	11.753272	192.168.8.165	192.168.8.142	TCP	54	46731 → 443 [SYN] Seq=0 Win=512 Len=0
19789	11.754452	192.168.8.165	192.168.8.142	TCP	54	46732 → 443 [SYN] Seq=0 Win=512 Len=0
19790	11.754552	192.168.8.165	192.168.8.142	TCP	54	46733 → 443 [SYN] Seq=0 Win=512 Len=0
19791	11.754603	192.168.8.221	192.168.8.142	TCP	54	50270 → 443 [ACK] Seq=974 Ack=14998 Win=0 Len=0
19792	11.754699	192.168.8.165	192.168.8.142	TCP	54	46734 → 443 [SYN] Seq=0 Win=512 Len=0
19793	11.755221	192.168.8.165	192.168.8.142	TCP	54	46735 → 443 [SYN] Seq=0 Win=512 Len=0
19794	11.755324	192.168.8.165	192.168.8.142	TCP	54	46736 → 443 [SYN] Seq=0 Win=512 Len=0

▶ Frame 19768: 1267 bytes on wire (10136 bits), 1267 bytes captured (10136 bits) on interface eth0  
 ▶ Ethernet II, Src: Philips (08:00:27:00:00:00), Dst: Apple (08:00:27:00:00:00)  
 ▶ Internet Protocol Version 4, Src: 192.168.8.142, Dst: 192.168.8.221  
 ▶ Transmission Control Protocol, Src Port: 443, Dst Port: 50270, Seq: 14998, Ack: 974, Len: 1213  
 ▶ [3 Reassembled TCP Segments (4133 bytes): #19766(1460), #19767(1460), #19768(1213)]

Figure 6.3: Screenshot from wireshark showing overlap between traffic sent from Apple smart phone and Attack traffic.

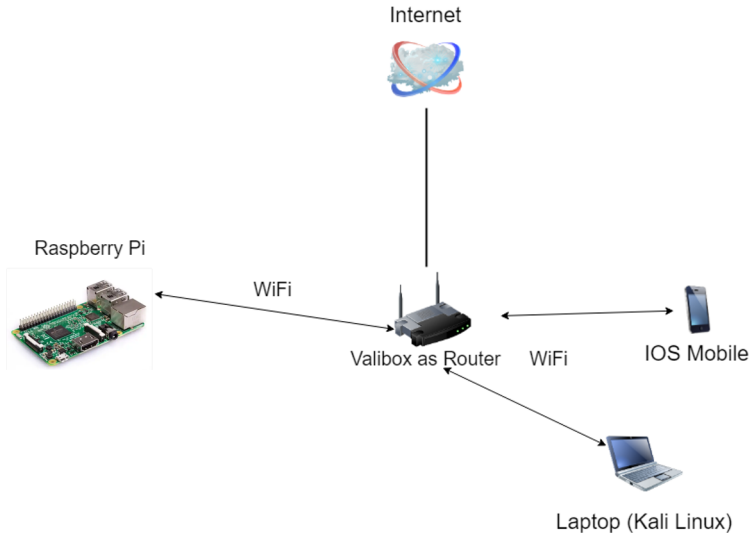


Figure 6.4: Illustration of attack from IOT device on Raspberry Pi. Laptop running Kali Linux spoofs MAC address of IOT device.

### 6.1.3. IOT DEVICE AS SOURCE OF ATTACK

The approach here used was similar to the one used in Doshi *et al.*[8]. It is a simulated attack where, instead of infecting the IOT device, a laptop running Kali Linux is used as the attacker. This laptop will spoof the MAC address of IoT device. This can also be seen in Figure 6.6, which is a screenshot taken from Wireshark showing attack traffic. In highlighted parts of the figure, we can see that the source MAC address is seen as something that belongs to a Philips device. Since the logic to build the system filters traffic of IoT device based on MAC address, the attack traffic would be considered as the traffic originating from IoT device. A raspberry pi running apache web server would be the victim.

First, the apache server on raspberry pi is accessed from mobile phone to check if it is up and running. After this, the attack is initiated from the laptop and continued until the server becomes unresponsive. Figure 6.5 provides a visual explanation of attack traffic generated in the form of a state machine. As can be seen, the attack starts with a large number of SYN packets and some responses from victim in the form of SYN-ACK. Once this is increased, the attacker starts sending Reset packets as long as the victim replies with SYN-ACK. And this process repeats. Figure 6.6 shows an example of how this attack traffic looks in Wireshark.

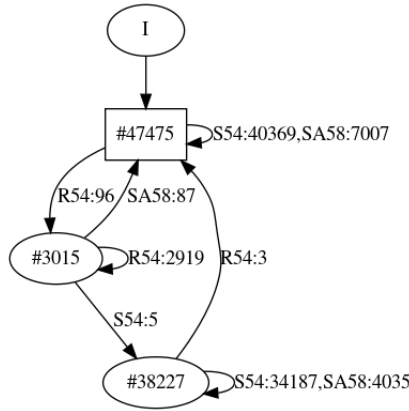


Figure 6.5: State machine representing SYN Flood attack.

## 6.2. CASE STUDY 2: REAL WORLD DATA

Details regarding traffic captured in a smart home environment is mentioned in this section. Traffic was captured for a week when the device was actually in use in a home network.

### 6

#### 6.2.1. SETUP

The setup to capture was similar to the approach used in 3. However, there were two significant differences:

- The traffic was captured while several users performed action on the device and it was not in a controlled environment.
- The setup used to capture traffic while learning state machines included the Hue bridge and one light along with it. In this setup, it consisted of a Hue bridge but with three different types of lights.
- Training data involved controlling the IoT device with a smartphone running IOS operating system. Here, android phone and an IOS smartphone were used to control the device.

This setup has been illustrated in Figure 6.7

#### 6.2.2. HYPOTHESIS

Because of the differences, some hypotheses were made concerning false positives that could be encountered.

- Some of them would arise because of the commands issued to control other two types of lights which were not initially part of training data.
- There could be difference in the type of communication between IOT device with an IOS smartphone in comparison to IoT device with Android smartphone.



No.	Time	Source	Destination	Protocol	Length	Info
271	6.878194	192.168.8.165	192.168.8.222	TCP	54	2247 → 80 [SYN] Seq=0 Win=512 Len=0
272	6.878264	192.168.8.165	192.168.8.222	TCP	54	2248 → 80 [SYN] Seq=0 Win=512 Len=0
273	6.878334	192.168.8.165	192.168.8.222	TCP	54	2249 → 80 [SYN] Seq=0 Win=512 Len=0
274	6.878403	192.168.8.165	192.168.8.222	TCP	54	2250 → 80 [SYN] Seq=0 Win=512 Len=0
275	6.878487	192.168.8.165	192.168.8.222	TCP	54	2251 → 80 [SYN] Seq=0 Win=512 Len=0
276	6.878558	192.168.8.165	192.168.8.222	TCP	54	2252 → 80 [SYN] Seq=0 Win=512 Len=0
277	6.878630	192.168.8.165	192.168.8.222	TCP	54	2253 → 80 [SYN] Seq=0 Win=512 Len=0
278	6.878699	192.168.8.165	192.168.8.222	TCP	54	2254 → 80 [SYN] Seq=0 Win=512 Len=0
279	6.878769	192.168.8.165	192.168.8.222	TCP	54	2255 → 80 [SYN] Seq=0 Win=512 Len=0
280	6.878901	192.168.8.165	192.168.8.222	TCP	54	2256 → 80 [SYN] Seq=0 Win=512 Len=0
281	6.879042	192.168.8.165	192.168.8.222	TCP	54	2257 → 80 [SYN] Seq=0 Win=512 Len=0
282	6.879218	192.168.8.165	192.168.8.222	TCP	54	2258 → 80 [SYN] Seq=0 Win=512 Len=0
283	6.879340	192.168.8.165	192.168.8.222	TCP	54	2259 → 80 [SYN] Seq=0 Win=512 Len=0
284	6.879434	192.168.8.165	192.168.8.222	TCP	54	2260 → 80 [SYN] Seq=0 Win=512 Len=0
285	6.879648	192.168.8.222	192.168.8.165	TCP	58	80 → 2056 [SYN, ACK] Seq=0 Ack=1 Win=
286	6.879802	192.168.8.222	192.168.8.165	TCP	58	80 → 2057 [SYN, ACK] Seq=0 Ack=1 Win=
287	6.879949	192.168.8.165	192.168.8.222	TCP	54	2261 → 80 [SYN] Seq=0 Win=512 Len=0
288	6.880094	192.168.8.165	192.168.8.222	TCP	54	2262 → 80 [SYN] Seq=0 Win=512 Len=0
289	6.880307	192.168.8.165	192.168.8.222	TCP	54	2263 → 80 [SYN] Seq=0 Win=512 Len=0
290	6.880458	192.168.8.165	192.168.8.222	TCP	54	2264 → 80 [SYN] Seq=0 Win=512 Len=0
291	6.880717	192.168.8.165	192.168.8.222	TCP	54	2265 → 80 [SYN] Seq=0 Win=512 Len=0
292	6.880834	192.168.8.165	192.168.8.222	TCP	54	2266 → 80 [SYN] Seq=0 Win=512 Len=0
293	6.880905	192.168.8.165	192.168.8.222	TCP	54	2267 → 80 [SYN] Seq=0 Win=512 Len=0
294	6.880972	192.168.8.165	192.168.8.222	TCP	54	2268 → 80 [SYN] Seq=0 Win=512 Len=0
295	6.881084	192.168.8.165	192.168.8.222	TCP	54	2269 → 80 [SYN] Seq=0 Win=512 Len=0
296	6.881182	192.168.8.165	192.168.8.222	TCP	54	2270 → 80 [SYN] Seq=0 Win=512 Len=0
297	6.881251	192.168.8.165	192.168.8.222	TCP	54	2271 → 80 [SYN] Seq=0 Win=512 Len=0
298	6.881440	192.168.8.165	192.168.8.222	TCP	54	2272 → 80 [SYN] Seq=0 Win=512 Len=0
299	6.881570	192.168.8.165	192.168.8.222	TCP	54	2273 → 80 [SYN] Seq=0 Win=512 Len=0
300	6.881690	192.168.8.165	192.168.8.222	TCP	54	2274 → 80 [SYN] Seq=0 Win=512 Len=0
301	6.881792	192.168.8.165	192.168.8.222	TCP	54	2275 → 80 [SYN] Seq=0 Win=512 Len=0
302	6.881983	192.168.8.165	192.168.8.222	TCP	54	2276 → 80 [SYN] Seq=0 Win=512 Len=0
303	6.882140	192.168.8.165	192.168.8.222	TCP	54	2277 → 80 [SYN] Seq=0 Win=512 Len=0
304	6.882242	192.168.8.165	192.168.8.222	TCP	54	2278 → 80 [SYN] Seq=0 Win=512 Len=0
305	6.882360	192.168.8.165	192.168.8.222	TCP	54	2279 → 80 [SYN] Seq=0 Win=512 Len=0

▶ Frame 271: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)  
 ▶ Ethernet II, Src: Philips Hue ( ), Dst: Raspberr  
 ▶ Internet Protocol Version 4, Src: 192.168.8.165, Dst: 192.168.8.222  
 ▶ Transmission Control Protocol, Src Port: 2247, Dst Port: 80, Seq: 0, Len: 0

Figure 6.6: Attack traffic when IOT is used as source of attack. Notice that source Ethernet address of Wireshark shows that it is originating from Philips Hue and destination is Raspberry Pi. This is because of spoofing MAC address by the attacker.

- It was not a controlled simulation. It was unsupervised and involved multiple devices over a longer duration. This could give rise to some false positives.
- The gap between traffic collection of training data and this real-world data was around 3-4 months. There could be some minor changes in the way system behaves because of possible updates.

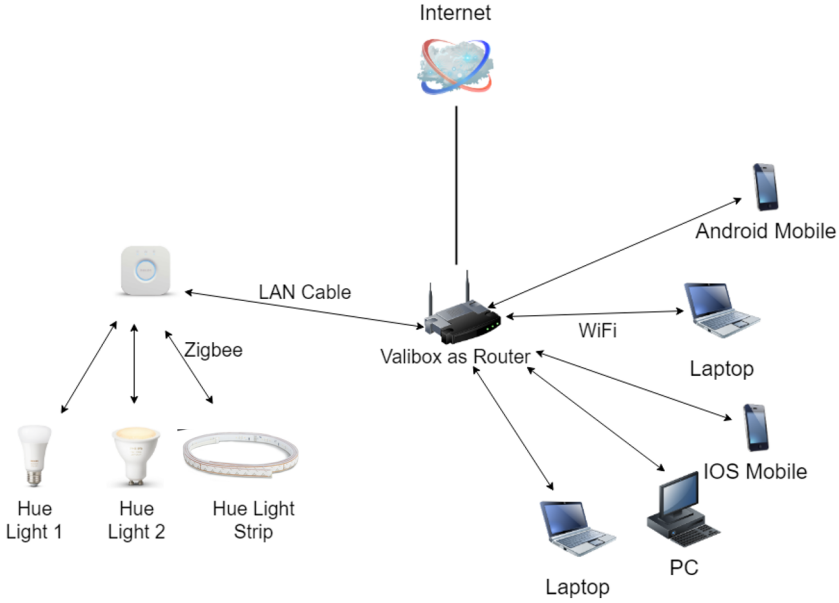


Figure 6.7: Experimental setup used to capture the traffic in a real life environment.

6

eth.addr == [redacted] and tcp

No.	Time	Source	Destination	Protocol	Length	Info
213	133.239243	192.168.8.144	192.168.8.142	TLSv1.2	258	Application Data
214	133.248850	192.168.8.142	192.168.8.144	TLSv1.2	637	Application Data
215	133.249461	192.168.8.142	192.168.8.144	TLSv1.2	88	Application Data

▶ Frame 213: 258 bytes on wire (2064 bits), 258 bytes captured (2064 bits)  
 ▶ Ethernet II, Src: Apple [redacted], Dst: PhilipsL [redacted]  
 ▶ Internet Protocol Version 4, Src: 192.168.8.144, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 58126, Dst Port: 443, Seq: 2787, Ack: 45782, Len: 204  
 ▶ Secure Sockets Layer

Figure 6.8: Traffic captured at smart home highlighting source host as Apple smart phone and destination as Philips Hue. Note that the packets exchanged here are same as the packets in Figure 5.3. This means that turning the lights On/Off was performed.

eth.addr == [redacted] and tcp

No.	Time	Source	Destination	Protocol	Length	Info
142	66.167630	192.168.8.105	192.168.8.142	TLSv1.2	258	Application Data
143	66.178780	192.168.8.142	192.168.8.105	TLSv1.2	637	Application Data
144	66.191433	192.168.8.142	192.168.8.105	TLSv1.2	88	Application Data
145	66.194145	192.168.8.105	192.168.8.142	TCP	54	44592 → 443 [ACK] Seq: [redacted]

▶ Frame 142: 258 bytes on wire (2064 bits), 258 bytes captured (2064 bits)  
 ▶ Ethernet II, Src: SamsungE [redacted], Dst: PhilipsL [redacted]  
 ▶ Internet Protocol Version 4, Src: 192.168.8.105, Dst: 192.168.8.142  
 ▶ Transmission Control Protocol, Src Port: 44592, Dst Port: 443, Seq: 2828, Ack: 42910, Len: 204  
 ▶ Secure Sockets Layer

Figure 6.9: Traffic captured at smart home highlighting source host as Samsung smartphone and Philips Hue. Note that the packets exchanged here are same as the packets in Figure 6.8. This shows that the commands issued by android and IOS smartphones could be same.

### 6.3. SUMMARY

Details regarding the setup used to capture attack traffic are presented. IOT device was initially considered as victim of Denial of Service attack and traffic was captured during the execution of that attack. In second case, IOT device was considered as source of attack. Instead of infecting the device in question, a laptop running kali linux was used to perform attack. To impersonate IoT device, this laptop spoofed the MAC address of IoT device. A state machine describing the attack has been included in this chapter.

This chapter also contains the details regarding setup used to capture traffic from a smart home environment. Several hypothesis regarding how this traffic might be slightly different from the traffic generated as part of training data in this project are then discussed.

Given the network traffic while performing attack and from real-world data, it can be used to evaluate the efficiency of state machines in identifying them. Details regarding the methodology used to perform such evaluation and corresponding results are presented as part of next chapter.



# 7

## EVALUATION

*This chapter contains a description of the procedure used to verify test traffic against the learnt state machine. It is followed by various metrics used to evaluate the performance of the system. Evaluation is performed on Attack traffic from DoS attack and on real-world traffic separately. Obtained results are discussed along with false positive analysis.*

We now have the state machines as well as the test traffic containing normal and attack traffic. Additionally, traffic was also captured in a smart home environment. In this Chapter, the methodology used to compare metadata of packets with a state machine is provided. In the second half of this chapter, this methodology is applied and resulting evaluation metrics are discussed.

### 7.1. TOOLS USED

The final output by state machine learning module is in the form of a graphviz dot file<sup>1</sup> or a PNG file. In this work, dot file is considered. *Pydot*<sup>2</sup> is a tool that can be used to parse, modify Graphviz DOT files. Output by flexfringe provides a graphviz DOT file that need to be processed in order to perform evaluation methodology mentioned in this chapter. It can be used to obtain the transitions and states in the form of a data-structure to easily parse the data. Figure 7.2 provides an example of a DOT file that can be used to create a state machine shown in Figure 7.1 with the help of state machine learning module Flexfringe[47].

Highlighted parts of the DOT file specify the following:

- 1 This part of the DOT file specifies the number assigned to each state. Each of these entries corresponds to a state in the state machine.
- 2 This part contains the label that will be placed inside the corresponding state

<sup>1</sup><https://www.graphviz.org/doc/info/lang.html>

<sup>2</sup><https://pypi.org/project/pydot/>

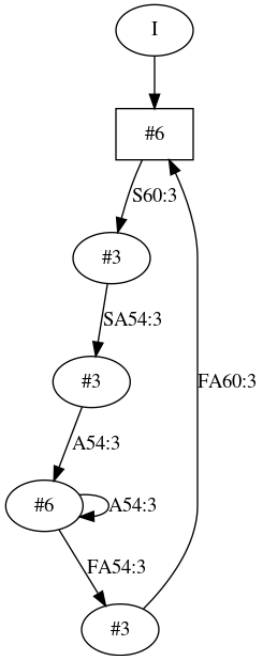


Figure 7.1: Example of a state machine obtained as from Flexfringe[47].

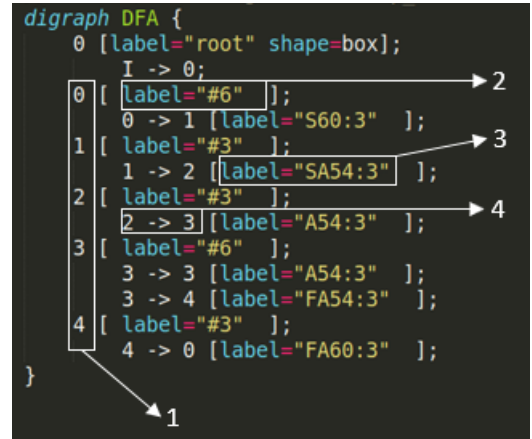


Figure 7.2: Graphviz DOT file corresponding to the state machine in Figure 7.1

## 7

- 3 This part is used to specify the label of transition.
- 4 It contains the source node and destination node of the transition.

## 7.2. (SEMI)ONLINE EVALUATION

Here, every packet is checked against the state machine as it is parsed from pcap file. Algorithm 1 provides the pseudo-code of logic used to achieve this. It is called (semi) online evaluation because details of a packet are parsed individually without storing any details related to them. However, it is performed over a pcap file which is already captured instead of packets captured in real-time.

**Result:** Number of matched and not matched inputs  
initialization;  
**for** *each packet in pcap file* **do**  
    check if the packet is sent from/to IOT device ;  
    input = details of packet in required format ;  
    **if** *ip\_addr is never seen before* **then**  
        | cur\_state = ip\_addr\_state[ip\_addr] = root\_node ;  
    **else**  
        | cur\_state = ip\_addr\_state[ip\_addr] ;  
    **end**  
    **for** *each transition from cur\_state* **do**  
        | check if input matches one of the transition ;  
    **end**  
    **if** *no match* **then**  
        | increment not\_matched\_count ;  
    **else**  
        | increment matched\_count ;  
        | cur\_state = get\_next\_state(cur\_state, transition) ;  
    **end**  
**end**

**Algorithm 1:** Pseudo-code used to traverse the state machine and compare against packets in a pcap file

For each IP address, the latest state where it is currently present is recorded. Every time a new packet arrives, this state is updated if there is a match. If it is the first packet, then the current state of the IP address would be the root node. Starting from the root node, if there is a transition corresponding to the input symbol, make a transition and now this new state is the current state for that IP address. If there is no match, then an appropriate response is displayed.

In order to handle the situation where packets from multiple hosts are mixed with each other, a separate data-structure was used to keep track of latest state depicting the most recent packet received/sent from that IP address.

## 7.3. OFFLINE EVALUATION

In this case, details from pcap are converted into an input file, similar to the way inputs were given to state machine learning module. Every conversation performed by Hue is considered as an input to the evaluation module and checked if the entire sequence matches with the state machine. If there is a mismatch in the packet among even one

packet of the conversation, entire conversation is marked as not-matched. Algorithm 2 provides the pseudo-code of logic used to achieve this.

```

Result: Number of matched and not matched inputs
initialization;
for each sequence in the input file do
    for each input in the sequence do
        for each transition from cur_state do
            | check if input matches one of the transition ;
        end
        if no match then
            | increment not_matched_count ;
        else
            | increment matched_count ;
            | cur_state = get_next_state(cur_state, transition) ;
        end
    end
end

```

**Algorithm 2:** Pseudo-code used to traverse the state machine and compare against packets in a pcap file

For the attack traffic, the entire sequence of packets coming from the attacker signifying everything from beginning to end of traffic is considered as a single sequence.

#### 7.4. EVALUATION METRICS: ATTACK TRAFFIC

As discussed in previous chapter, Denial of Service attack was performed on/from IoT device and traffic was captured along with normal traffic. Evaluation mechanism explained in the previous section was used on the traffic which was captured when IoT device was used as victim of DoS attack.

Since the data consists of normal traffic as well as attack traffic, an evaluation metric that takes both positives and negatives into consideration was required. For this purpose, accuracy was used.

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalPositives + TotalNegatives} \quad (7.1)$$

Using this as a measure, **an Accuracy of 99.94% was obtained** in detection of attack traffic from normal traffic.

#### 7.5. EVALUATION METRICS: BASELINE WITH ATTACK TRAFFIC

As mentioned in Chapter 4, N-Grams was used as a baseline in this thesis. Instead of using absolute packet size, its value in terms of percentiles was used. Other features remained the same as those provided to state machine learning module. Probability was assigned to each N-Gram that appeared in training data. For any N-gram in Test data, if the probability of its occurrence is below a threshold, then it is marked as anomalous. Table 7.1 shows accuracy achieved upon usage of a various number of percentiles for packet size, a threshold on the probability of occurrence and value of N in N-grams.



Table 7.1: Accuracy obtained based on number of percentiles used for packet size, thresholds defined for probability of occurrence of a sequence below which it is marked as anomalous and size of sequences(N in N-grams).

Length of Sequences	No. of Percentiles	Threshold(on probability)	Accuracy
2	10	0.1	99.71%
2	10	0.01	99.71%
2	30	0.1	99.61%
2	30	0.01	99.61%
2	30	0.01	99.61%
2	50	0.01	99.61%
2	50	0.01	99.61%
3	10	0.1	100%
3	10	0.01	100%
3	30	0.1	100%
3	30	0.01	100%
3	30	0.01	100%
3	50	0.01	100%
3	50	0.01	100%
4	10	0.1	100%
4	10	0.01	100%
4	30	0.1	100%
4	30	0.01	100%
4	30	0.01	100%
4	50	0.01	100%
4	50	0.01	100%

We considered values less than 5 for N in N-grams because longer traces would lead to sequences that might occur less frequently. Additionally, with some hosts, the IOT device opens and closes a TCP session without exchanging any packets. This would give us traces of length 4.

## 7.6. EVALUATION METRICS: REAL WORLD DATA

Data collected in a home network is used to evaluate the state machines. Whenever a packet matches a transition in state machine, it is marked as True Positive. When there is a mismatch, it is marked as a false positive. This is under the assumption that the IOT device is not infected by any malware and represents normal behaviour. Upon manual inspection in Wireshark, there weren't any traces of infection.

Since the assumption is that data only positives constitute the traffic, Precision or Positive predictive value is used. It is the ratio of total number of true positives to the sum of true positives and false positives. The following formula represents this

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (7.2)$$

When the Offline evaluation was applied to the data, two types of evaluation were performed:

### 7.6.1. USING ABSOLUTE PACKET SIZE

In this case, absolute number of packet size from the test traffic was used to create metadata of the packet. All other features were used in the same manner in which they were used in training data. Metadata was then used to see if there is any matching transition corresponding to that packet. This is more a stricter version.

Upon evaluation from the pcaps captured over a week, A precision of 75.13% was obtained. That would mean, out of all the packets generated by the IOT device, 24.87% of the packets were marked as anomalous even though they were from benign traffic.

### 7.6.2. USING THRESHOLD ON PACKET SIZE AND MISMATCHING TRANSITIONS

The hypothesis was that usage of a stricter approach in terms of checking for absolute packet size and number of packets would have led to a large number of false positives in the previous case. To negate this effect, **a buffer for packet size was used**. For each packet in test data, all other features were compared against the state machine, except in case of packet size, if the packet size in test data is within the range of b bytes from packet size specified in the state machine, then it is still counted as true positive. (where b represents the buffer for packet size)

Alongside, **a buffer was used for mismatching transitions**. In each trace, if the number of transitions that are not matched is less than t, then it is marked as true positive. (Here t represents the threshold for mismatching transitions)

Upon using a **threshold of 15 bytes for packet size, and 3 in terms of transitions, a precision of 97.82 % was obtained**. Table 7.2 provides the precision obtained upon usage of various thresholds.

### 7.6.3. FALSE POSITIVE ANALYSIS

In the previous section, it was inferred that there were still some false positives in the network traffic captured from real-world traffic. This was even after imposing threshold.

Table 7.2: Precision obtained based on thresholds defined for packet size and mismatching transitions.

Mismatching Transition	Buffer for packet size(in bytes)	Precision
2	25	95.23%
2	20	95.23%
2	15	82.53%
3	25	97.82%
3	20	97.82%
3	15	97.82%
3	10	84.72%
4	25	97.82%
4	20	97.82%
4	15	97.82%

These false positives were manually inspected to find the cause of these alarms.

Upon usage of the threshold, there were no false positives in the traffic exchanged between IoT device and smartphone.

Following were the traces where there was an alarm raised.

**Background Traffic:** Philips Hue communicated with various hosts over HTTP and this constituted background traffic. False positives raised here are:

- HTTP Post call to [www.diagnostics.meethue.com](http://www.diagnostics.meethue.com) generated a response in the form of few packets(2-3) in training data. However, in traffic captured from real-world data, there were few instances where this resulted in a response of 30-35 packets. These instances caused the majority of false positives in this data. This has been highlighted as 1 in Figure 7.3.
- HTTP call to [www.ecdinterface.philips.com](http://www.ecdinterface.philips.com) always resulted in a HTTP response with status 200. In real-world traffic, during a few instances, a response of 400 was obtained. These traces raised a false alarm. This has been highlighted as 3 in Figure 7.3.
- HTTP response to call made to [dcp.dc1.philips.com](http://dcp.dc1.philips.com) involved more number of packets than what was seen in training data. This has been highlighted as 2 in Figure 7.3.

**Implications:** Upon observing the false positives, we can draw certain implications. In terms of background traffic, the state machines don't remain valid if the response to an HTTP call is significantly different than what was seen in training data. Minor changes can be accounted with the help of thresholds on Packet sizes and number of mismatching transitions.

However, there were no false positives in traffic exchanged between Mobile App and Hue upon usage of threshold on packet size and mismatching transitions. Extra flexibility offered in terms of these thresholds were helpful in making false positives zero in this case.

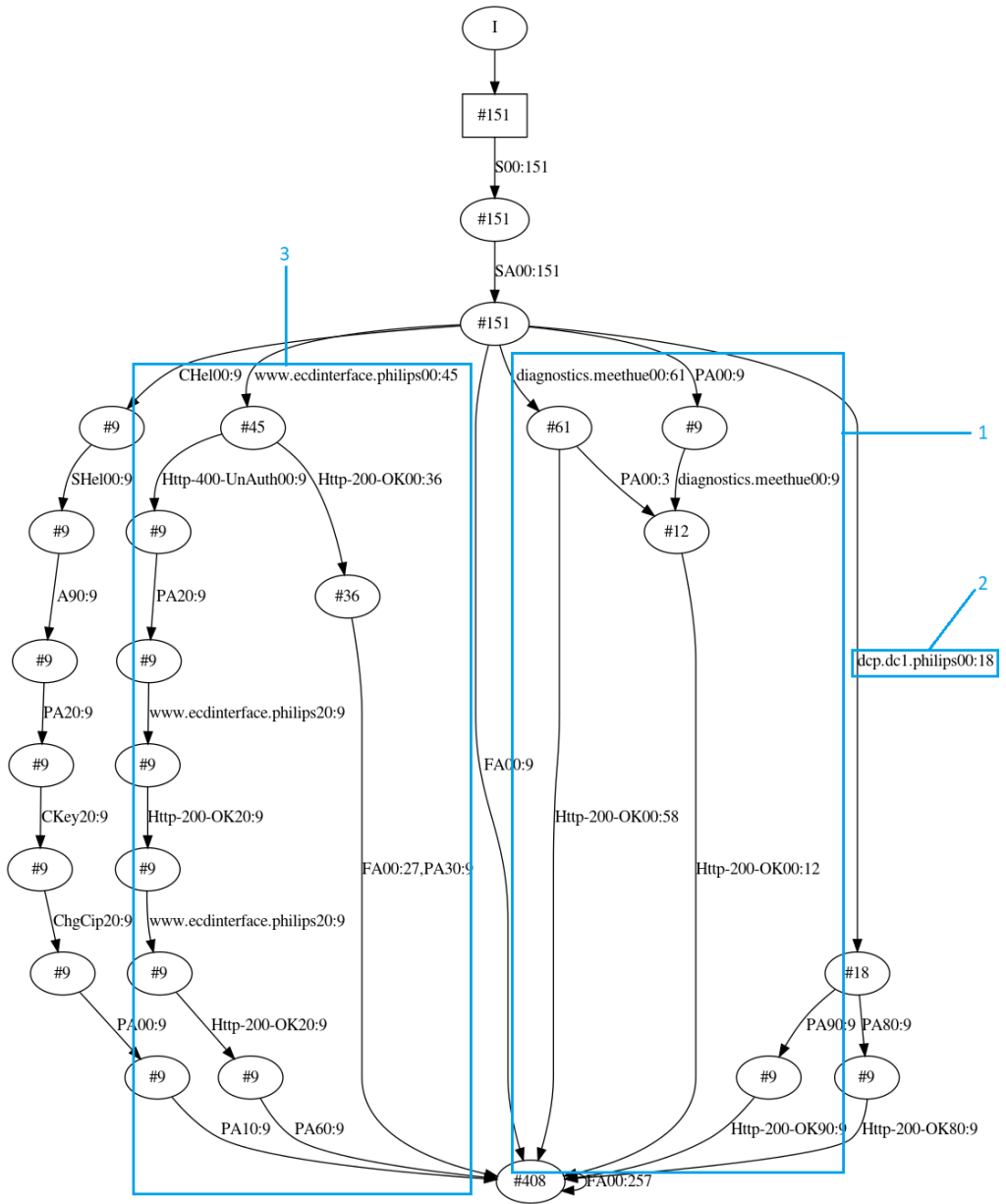


Figure 7.3: State machine representing background traffic generated by philips Hue. Highlighted parts represent the conversations where false positives were raised.

## 7.7. EVALUATION METRICS: BASELINE WITH REAL WORLD TRAFFIC

Traffic captured in real-world settings were used to evaluate the performance of N-grams which formed our baseline.

Similar to section 7.5, the performance of N-grams was measured upon various parameters. As discussed in the previous section, we assumed that the IOT device was not infected so precision was used to evaluate the performance.

Table 7.3 provides the precision achieved upon usage of different parameters. It can be seen that, as the number of percentiles increased, precision decreased.

Table 7.3: Precision obtained based on number of percentiles used for packet size, thresholds defined for probability of occurrence of a sequence below which it is marked as anomalous and size of sequences(N in N-grams).

Length of Sequences	No. of Percentiles	Threshold(on probability)	Precision
2	10	0.1	92.87%
2	10	0.01	92.87%
2	10	0.001	92.87%
2	30	0.1	86.15%
2	30	0.01	86.15%
2	30	0.001	86.15%
2	50	0.1	83.68%
2	50	0.01	83.68%
3	10	0.1	86.24%
3	10	0.01	86.24%
3	30	0.1	74.33%
3	30	0.01	74.33%
3	50	0.01	70.88%
3	50	0.01	70.88%
4	10	0.1	76.06%
4	10	0.01	76.06%
4	30	0.1	63.97%
4	30	0.01	63.97%
4	50	0.1	59.31%
4	50	0.01	59.31%

Another observation that can be made from the table is, as the length of sequences increases, precision decreases.

With the best possible parameters: Length of sequences as 2, Number of percentiles as 10 and Threshold as 0.1, we were able to obtain a precision of 92.87% using N-Grams on real-world data.

## 7.8. RESEARCH QUESTIONS ANSWERED IN THIS CHAPTER

Attack traffic and traffic from real world were considered in this chapter. The performance of our state machines is then evaluated. The results from this chapter answer the following research questions:

RQ2: How effective are state machines for detecting anomalies in network traffic generated by IOT devices?

State machines were able to successfully detect traffic of Denial of Service attack. Upon evaluation, a high value of accuracy was obtained. While there were some false positives when we used state machines with real world data, these false positives were significantly lowered upon usage of a threshold and resulted in a precision of 97.86%.

By using a test setting, which was different from the setup used to build state machines, we were able to answer the following research question:

RQ4: To what extent does the state machine differ upon exposure to different test setting?

Upon exposure to different test setting, state machines were still able to capture 97.82% of the traffic.

## 7.9. SUMMARY

This chapter started with evaluation methodology of semi-online evaluation and offline evaluation. Pseudo-code explaining how it was implemented is then mentioned. These evaluation methodologies were used on Attack traffic from Chapter 6 and real-world traffic from Chapter ?? separately. Accuracy of 99.94% was obtained when evaluated on Attack traffic. A precision of 75.13% was obtained on real-world traffic using absolute packet size as one of the features. Upon relaxation of this requirement, by including a buffer of for packet size and mismatching transitions, precision increased to 97.82%. The remaining traces which were marked as false positives were analysed and reason for marking it as false positive has been the slight difference in the behaviour of IoT device during training data and in real-world data.

Table 7.4: Performance of N-Grams and State Machines with Attack Traffic and Real World data upon usage of best set of parameters observed as part of this thesis.

Algorithm	Type of Test Data	Metric Used	Value
State Machines	Attack Traffic	Accuracy	99.94 %
N-Grams	Attack Traffic	Accuracy	100 %
State Machines	Real World Data	Precision	97.82 %
N-Grams	Real World Data	Precision	92.87 %

N-Grams as a baseline was used to identify attack traffic, as well as with real-world traffic. Although N-Grams performed better than State machines in identifying attack traffic, state machines was better than N-Grams with real-world traffic. Table 7.4 provides the values of evaluation metrics obtained using the best set of parameters observed as part of this thesis.

Although N-grams have slightly higher accuracy than State machines in detecting attack traffic, the key strength of using state machines for this thesis is explainability:

N-grams consider sequences of packets with fixed length. In case of state machines, a complete TCP session could be explained in a single trace. It is also easier to interpret the behaviour of a device using TCP sessions in state machines than stating the frequency of sequences of packets with fixed length.





# 8

## LIMITATIONS AND FUTURE WORK

*This chapter contains the limitations of the thesis and how it can be extended for future work.*

### 8.1. DATA RELATED

#### **Limitations:**

While collecting data, operations to control IOT device were performed manually. This restricts the amount of traffic that was captured.

As discussed in Chapter 3, only those packets which used TCP were considered to build state machines in case of Philips Hue lighting system. Communication using UDP was not considered.

#### **Future Work:**

Data Collection can be automated to collect data over longer durations using tools such as Appium<sup>1</sup>. This would help to achieve more training data and provide exact timestamps at which certain operations were performed.

It was observed in the data that the Hue used protocols such as NTP, SSDP etc. Packets related to these protocols can also be used to learn the normal behaviour of IoT devices. Philips Hue uses Zigbee protocol to communicate between the Hue bridge and individual lights as illustrated in Figure 8.2. This traffic was not considered as part of this project. There have been instances where vulnerabilities in Zigbee protocol were used to gain unauthorised access<sup>2</sup>. Metadata from traffic generated by this protocol can be combined with the features as part of this thesis. That would enable to detect such kind of attack and provides a complete view of working of such IoT devices.

---

<sup>1</sup><http://appium.io/>

<sup>2</sup><https://eyalro.net/project/iotworm/>



Figure 8.1: Hue bridge uses Zigbee to communicate with lights whereas it uses TCP to communicate with smart phone controlling it.

## 8.2. APPROACH RELATED

**Limitations:** Packet level features were used for this project. This could lead to more false positives.

In scenarios where TCP is used, retransmitted and out of order packets were handled in this thesis. However, in the case of UDP, there needs to be a better approach for identifying the retransmitted and out of order packets.

**Future Work:** Flow level features could also be used to capture normal behaviour. Packet level and flow level features can be combined to learn normal behaviour.

DTLS on top of UDP was encountered while working with IKEA lights. It was observed that, even Wireshark was unable to identify retransmitted packets. Usage of parameters such as Sequence number, epoch number could be used to better identify re-transmitted and out of order DTLS packets.

In Chapter 6, attack traffic of Denial of Service attack was used to build a state machine. It used the same metadata that was used to build the state machines representing normal behaviour of IOT device. This could further be extended. Traffic can be captured while performing various attacks and can be given as input to the module in this project. It will help in visualising and explaining various attacks and their behaviour without making any changes to the existing module of this project.

As mentioned before, background traffic generated by Philips Hue consists of HTTP GET/POST calls made by the device. In some instances, these HTTP calls and their responses are comprised of more than one packet. Figure provides an example of one such instance where contents of POST calls is part of two packets. In the figure, packet no. 292 is highlighted with its content labelled as a POST call by Wireshark. However, packet no. 290 is also part of this POST call and they have been divided into two TCP segments. This can be seen in the highlighted part of screenshot.

In the current approach, these two packets are considered as two transitions in the state machine. In future work, these two packets can be merged into a single transition in state machine. This would give smaller state machine. Upon usage of a threshold on packet size, as mentioned in Chapter 7 would also lead to less number of false positives.

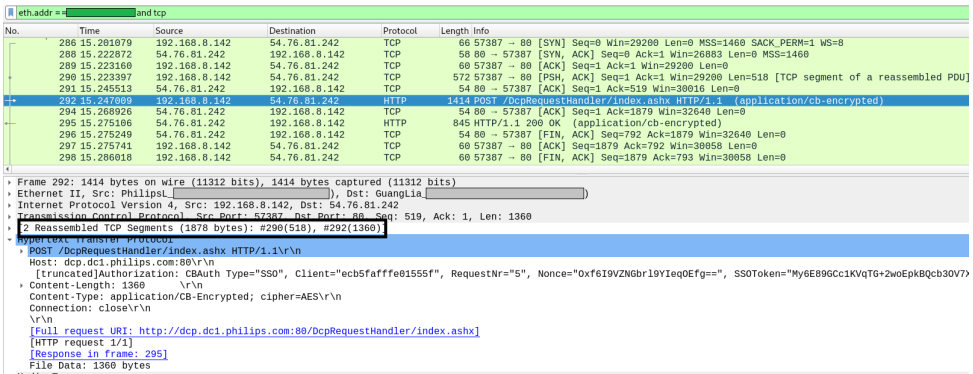


Figure 8.2: Screenshot from wireshark showing a POST call which is made up of two TCP segments.

### 8.3. EVALUATION RELATED

**Limitations:** While evaluating with IOT device as source of attack, instead of infecting the IOT device, MAC address of the device was spoofed and a simulated attack was performed.

**Future Work:** For future work, it could be infected by an actual malware. The approach used by Meidan *et al.* [24] to infect IOT device with Mirai Botnet could be a good starting point.

Missing states and additional states are not part of the current evaluation methodology in this project. A threshold can be set, upto which these missing or additional states will be tolerated.

Current evaluation involves reading the state machines without making any changes to it. A programming language such as Java can be used to exploit the advantage of multi-threaded programming feature present there. This will speed up the evaluation as traces can be evaluated parallelly. It can also be used to implement online evaluation where each packet can be checked if it matches a transition in the intermediate state of state machine or if it matches the transition from the initial state simultaneously. Python is used as the programming language for evaluation in this project.

In section 7.6.2 of evaluation chapter, a threshold was used to handles mismatching transitions and slight changes in packet sizes. Various values for these thresholds were experimented and the one which provides least amount of false positives were chosen. It would be more useful if we can have a method that calculates this threshold automatically based on the data.

### 8.4. INTERACTION WITH OTHER DEVICES

**Future Work:** Devices such as Google Home, Alexa could be configured to control Philips Hue. Here, only smart phone was used to control IoT device. It would be interesting to see the difference between commands issued by Google home/Alexa and smart phone, if any.



# 9

## CONCLUSION

IoT devices are getting increasingly popular, but they face a multitude of security issues<sup>1</sup>. These insecure IOT devices are exploited by hackers, for instance, in the form of Botnets to perform large scale Denial Of Service attacks. Additionally, they raise privacy concerns as they are used in home environment.

This project starts with learning the behaviour of these IoT devices and tries to explain it at packet level. It uses state machines to achieve this. Type of work performed in this project only involves passively listening to network traffic generated by IoT devices. For this reason, there will be no overload on individual IOT devices and can be built on top of existing network infrastructure. Network middle-boxes such as routers can be used to deploy this work.

Answers for the research questions posed in section 1.6 have been answered in the following manner:

RQ1: How can we perform anomaly detection based on network traffic generated by IoT devices using state machines?

To achieve this, traffic from IoT device was captured during its normal operation. This was used to create a state machine explaining the normal behaviour of IoT device. To validate this, traffic was captured at a smart home environment involving multiple devices. This traffic was used to validate the fit between state machines and actual behaviour of IoT device. Furthermore, Denial of Service attack was performed. Traffic from this attack was used to see whether the state machines can effectively detect anomalies from benign traffic.

RQ1a: What are the different behaviours performed by an IoT device during its normal operation?

<sup>1</sup><http://d-russia.ru/wp-content/uploads/2015/10/4AA5-4759ENW.pdf>

The search for an answer to this question started with the manner in which traffic was collected. Initial focus was on the traffic in TCP/IP protocol suite exchanged between IOT device with all the other hosts. Traffic was captured while issuing various commands from the smartphone. Duration and time at which the traffic was captured were also varied. It was observed that the IOT device communicated with multiple hosts during its operation. This remained true even when the device was idle. With respect to UDP, it polled some hosts. For instance, it used NTP to obtain time-related data from certain hosts. However, the focus was on TCP, because using this protocol, IOT device communicated with the smartphone controlling it. It was also using TCP, the IOT device performed certain HTTP POST/GET calls followed by uploading/downloading data.

RQ1b: Which high-level features can be used to define the behaviour of IoT devices?

Initial set of features considered were protocol, packet size and inter-packet time interval. It was observed in the literature that these three protocols were predominantly used as stateless features. Upon performing some analysis in the training data, it was decided to discard inter-packet time interval. Other features such as port numbers, IP Address, IP Flags were discarded. This was based on the distribution in values of these features in the dataset.

To aid better explanation of each packet, TLS header message was used in those packets which were communicated using TLS. For the same purpose, in case of HTTP calls performed by IOT device, domain name of the call and response code of that call were used as features. In all the remaining cases, TCP flags were used as the feature. In case of IKEA lights, as DTLS was used for communication between smartphone and the lights, DTLS message in the packets was used instead of TCP flags.

One important thing to highlight is that here the sequence in which packets are exchanged is used as an implicit feature to define the behaviour. This is because the arrival of each packet marks a transition for one state to another in the state machine.

RQ1c: How can these high-level features be used to build a state machine?

High-level features such as Packet Size, TCP Flag, TLS Headers, HTTP Call domains and response code, DTLS message were decided to use for constructing state machines as part of the previous research question. To build these state machines, arrival or departure of each packet was considered as a transition. Packet size was combined with one of the other 4 features and this constituted a symbol of state machine that creates a transition. As each packet's arrival marks a transition, it was important to handle packets received out of order. Since the packets were communicated using TCP protocol, TCP stream reordering was performed before it was used to generate state machines. Sequence Number and Acknowledgement number of TCP packets were used to identify and reorder the out of order packets and also to divide the communication between two hosts into multiple conversations.

Each packet was arranged in the form of PS (P represents protocol-related information, S

specifies packet size). Each conversation between Hue and any other hosts was then separated by newline character. It is then given as an input in Abbadingo competition[18] format to state machine learning module *Flexfringe*[47]. Output of this module is in the form of graphviz dot file which can be converted to PNG format.

RQ2: How effective are state machines for detecting anomalies in network traffic generated by IoT devices?

Once the state machines were learnt from network traffic of IoT device, the focus was then on its performance to identify anomalous network traffic. To achieve this, Denial of Service attack was performed on IoT device to render the device unresponsive. Traffic was captured which covered normal operation as well as attack traffic. Direction of the attack was reversed to simulate the same attack but with IoT device as victim. To simulate the attack, MAC address of IoT device was spoofed from a laptop running kali Linux. Test data containing attack traffic and normal traffic were used to see if the attack traffic matches the behaviour specified by state machines. Accuracy was used as a measure to find the effectiveness of the detection mechanism. Upon calculation, an Accuracy of 99.94% was obtained.

The state machines learnt also highlights that network traffic can still be used to deduce important information even if it is encrypted. In this project, only by looking at the metadata of packets sent/received from IoT device and smartphone, actions performed by the device could be deduced. This answers our fifth research question.

RQ3: Which commands issued by smartphone to IoT device can be detected using network traffic metadata?

Using state machines learnt from metadata of network traffic exchanged between the Philips Hue and Smart Phone, we were able to identify when and which actions were performed by IoT device. These actions include:

- Issue of command by Smart Phone to IoT device to turn the lights On/Off.
- Response from IoT device to smartphone upon receiving the command to turn lights On/Off.
- Issue of command by Smart Phone to IoT device to change colours of light.
- Response from IoT device to smartphone upon receiving the command to change colours of light.
- Issue of command by Smart Phone to IoT device to change the theme of lighting.
- Response from IoT device to smartphone upon receiving the command to change the theme of lighting.

In case of IKEA lights, we were able to identify the following actions:

- Command from smart phone to IKEA lights to turn it On/Off and its corresponding replies.
- Command from smart phone to IKEA lights to change brightness and its corresponding replies.

To validate this claim, pcaps were captured while issuing these commands from smart-phone to IoT device. Time of this action was recorded. Later, these pcaps were visually inspected on Wireshark to verify if the packets exchanged indeed matched those transitions on state machine.

To answer our next research question, we set up an IoT device in a home network involving multiple devices. The traffic used to learn state machine was exchanged between Philips Hue Bridge and an IOS smartphone. However, in this setting, both Android smartphone and IOS smartphone were used to control the IOT device. While the training setup included only one type of light, there were three types of lights in this test setup.

RQ4: To what extent does the state machine differ upon exposure to different test setting?

Upon evaluation with thresholds set on packet size and mismatching transitions, a precision of 97.82% was obtained.

While the focus was on Philips Hue in the previous research questions, the methodology was then used to learn state machine of IKEA smart lights. Its results could be used to answer our next research question.

RQ5: How can the approach be extended to represent the behaviour of another IoT device?

To represent the background traffic generated by IKEA lights was used to build the state machine. The methodology for building these state machines was same as the one used for background traffic generated by Philips Hue. For modelling the traffic between smart phone and IKEA lights, methodology was extended to include DTLS protocol.

In conclusion, we were able to build a state machine that can explain the behaviour of a IoT devices at packet level. These state machines can be used to detect the actions performed by IoT device as well as to distinguish normal network traffic with any anomalous traffic.



# BIBLIOGRAPHY

- [1] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani. “Architecting the internet of things: state of the art”. In: *Robots and Sensor Clouds*. Springer, 2016, pp. 55–75.
- [2] Abbas Acar et al. “Peek-a-Boo: I see your smart home activities, even encrypted!”. In: *arXiv preprint arXiv:1808.02741* (2018).
- [3] Suman Sankar Bhunia and Mohan Gurusamy. “Dynamic attack detection and mitigation in IoT using SDN”. In: *Telecommunication Networks and Applications Conference (ITNAC), 2017 27th International*. IEEE, 2017, pp. 1–6.
- [4] Pat Bosshart et al. “P4: Programming protocol-independent packet processors”. In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.
- [5] Ismail Butun, Burak Kantarci, and Melike Erol-Kantarci. “Anomaly detection and privacy preservation in cloud-centric Internet of Things”. In: *Communication Workshop (ICCW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2610–2615.
- [6] Janice Cañedo and Anthony Skjellum. “Using machine learning to secure IoT systems”. In: *Privacy, Security and Trust (PST), 2016 14th Annual Conference on*. IEEE, 2016, pp. 219–222.
- [7] Eung Jun Cho, Jin Ho Kim, and Choong Seon Hong. “Attack model and detection scheme for Botnet on 6LoWPAN”. In: *Asia-Pacific Network Operations and Management Symposium*. Springer, 2009, pp. 515–518.
- [8] Rohan Doshi, Noah Apthorpe, and Nick Feamster. “Machine Learning DDoS Detection for Consumer Internet of Things Devices”. In: *arXiv preprint arXiv:1804.04159* (2018).
- [9] Dave Evans. “The internet of things: How the next evolution of the internet is changing everything”. In: *CISCO white paper* 1.2011 (2011), pp. 1–11.
- [10] Javid Habibi et al. “Heimdall: Mitigating the Internet of insecure things”. In: *IEEE Internet of Things Journal* 4.4 (2017), pp. 968–978.
- [11] Christian Hammerschmidt. “Learning Finite Automata via Flexible State-Merging and Applications in Networking”. PhD thesis. University of Luxembourg, Luxembourg, 2017.
- [12] Cristian Hesselman et al. *SPIN: a user-centric security extension for in-home networks*. Tech. rep. Technical Report SIDN-TR-2017-002. SIDN Labs, 2017.
- [13] Philokypros Ioulianos et al. “A Signature-based Intrusion Detection System for the Internet of Things”. In: *Information and Communication Technology Form* (2018).

- [14] Prabhakaran Kasinathan et al. “Denial-of-Service detection in 6LoWPAN based Internet of Things”. In: *2013 IEEE 9th international conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE. 2013, pp. 600–607.
- [15] Nickolaos Koroniotis et al. “Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset”. In: *Future Generation Computer Systems* (2019).
- [16] Thomas Kothmayr et al. “DTLS based security and two-way authentication for the Internet of Things”. In: *Ad Hoc Networks* 11.8 (2013), pp. 2710–2723.
- [17] Thomas Kothmayr et al. “Poster: Securing the internet of things with DTLS”. In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM. 2011, pp. 345–346.
- [18] K Lang. “Evidence driven state merging with search”. In: *Rapport technique TR98-139, NECI* 31 (1998).
- [19] Pavel Laskov et al. “Intrusion detection in unlabeled data with quarter-sphere support vector machines”. In: *Praxis der Informationsverarbeitung und Kommunikation* 27.4 (2004), pp. 228–236.
- [20] Tsung-Han Lee et al. “A lightweight intrusion detection scheme based on energy consumption analysis in 6LowPAN”. In: *Advanced Technologies, Embedded and Multimedia for Human-centric Computing*. Springer, 2014, pp. 1205–1213.
- [21] MultiMedia LLC. *Suricata*. URL: <https://suricata-ids.org/> (visited on 11/24/2018).
- [22] Wei Lu and Issa Traore. “An unsupervised approach for detecting DDoS attacks based on traffic-based metrics”. In: *Communications, Computers and signal Processing, 2005. PACRIM. 2005 IEEE Pacific Rim Conference on*. IEEE. 2005, pp. 462–465.
- [23] Steven McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture.” In: *USENIX winter*. Vol. 46. 1993.
- [24] Yair Meidan et al. “N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders”. In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22.
- [25] Diego M Mendez, Ioannis Papapanagiotou, and Baijian Yang. “Internet of things: Survey on security and privacy”. In: *arXiv preprint arXiv:1707.01879* (2017).
- [26] Daniele Midi et al. “Kalis—A System for Knowledge-Driven Adaptable Intrusion Detection for the Internet of Things”. In: *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE. 2017, pp. 656–666.
- [27] Yisroel Mirsky et al. “Kitsune: an ensemble of autoencoders for online network intrusion detection”. In: *arXiv preprint arXiv:1802.09089* (2018).
- [28] Sudip Misra et al. “An adaptive learning routing protocol for the prevention of distributed denial of service attacks in wireless mesh networks”. In: *Computers & Mathematics with Applications* 60.2 (2010), pp. 294–306.

- [29] Mehdi Nobakht, Vijay Sivaraman, and Roksana Boreli. “A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow”. In: *Availability, Reliability and Security (ARES), 2016 11th International Conference on*. IEEE. 2016, pp. 147–156.
- [30] Sean Owen and Sean Owen. “Mahout in action”. In: (2012).
- [31] Pavan Pongle and Gurunath Chavan. “Real time intrusion and wormhole attack detection in internet of things”. In: *International Journal of Computer Applications* 121.9 (2015).
- [32] Okko Räsänen and Unto K Laine. “A method for noise-robust context-aware pattern discovery and recognition from categorical sequences”. In: *Pattern Recognition* 45.1 (2012), pp. 606–616.
- [33] Shahid Raza, Linus Wallgren, and Thiemo Voigt. “SVELTE: Real-time intrusion detection in the Internet of Things”. In: *Ad hoc networks* 11.8 (2013), pp. 2661–2674.
- [34] Shahid Raza et al. “Lithe: Lightweight secure CoAP for the internet of things”. In: *IEEE Sensors Journal* 13.10 (2013), pp. 3711–3720.
- [35] Shahid Raza et al. “Secure communication for the Internet of Things—a comparison of link-layer security and IPsec for 6LoWPAN”. In: *Security and Communication Networks* 7.12 (2014), pp. 2654–2668.
- [36] Ramasubramanian Sekar et al. “Specification-based anomaly detection: a new approach for detecting network intrusions”. In: *Proceedings of the 9th ACM conference on Computer and communications security*. ACM. 2002, pp. 265–274.
- [37] Martin Serror et al. “Towards In-Network Security for Smart Homes”. In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM. 2018, p. 18.
- [38] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The constrained application protocol (CoAP)*. Tech. rep. 2014.
- [39] Michael Sipser. *Introduction to the Theory of Computation*. Vol. 2. Thomson Course Technology Boston, 2006.
- [40] Arunan Sivanathan et al. “Low-cost flow-based security solutions for smart-home IoT devices”. In: *Advanced Networks and Telecommunications Systems (ANTS), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1–6.
- [41] Vijay Sivaraman et al. “Network-level security and privacy control for smart-home IoT devices”. In: *Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on*. IEEE. 2015, pp. 163–167.
- [42] Alberto MC Souza and José RA Amazonas. “An outlier detect algorithm using big data processing and internet of things architecture”. In: *Procedia Computer Science* 52 (2015), pp. 1010–1015.
- [43] Douglas H Summerville, Kenneth M Zach, and Yu Chen. “Ultra-lightweight deep packet anomaly detection for Internet of Things devices”. In: *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*. IEEE. 2015, pp. 1–8.

- [44] Nanda Kumar Thanigaivelan et al. "Distributed internal anomaly detection system for Internet-of-Things". In: *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*. IEEE. 2016, pp. 319–320.
- [45] Pablo Torres et al. "An analysis of recurrent neural networks for botnet detection behavior". In: *Biennial Congress of Argentina (ARGENCON), 2016 IEEE*. IEEE. 2016, pp. 1–6.
- [46] Viliam Vajda et al. "The EBBITS Project: An Interoperability platform for a Real-world populated Internet of Things domain". In: *Proceedings of the International Conference Znalosti (Knowledge), Technical University of Ostrava, Czech Republic*. 2011, pp. 317–320.
- [47] Sicco Verwer and Christian A Hammerschmidt. "flexfringe: a passive automaton learning package". In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2017, pp. 638–642.
- [48] Linus Wallgren, Shahid Raza, and Thiemo Voigt. "Routing Attacks and Countermeasures in the RPL-based Internet of Things". In: *International Journal of Distributed Sensor Networks* 9.8 (2013), p. 794326.
- [49] Yang Zhang, Nirvana Meratnia, and Paul Havinga. "Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks". In: *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*. IEEE. 2009, pp. 990–995.
- [50] Shengchu Zhao et al. "A Dimension Reduction Model and Classifier for Anomaly-Based Intrusion Detection in Internet of Things". In: *Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence & Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2017 IEEE 15th Intl*. IEEE. 2017, pp. 836–843.