REPORT STATUS DECLARATION FORM

Title: _____

_____

_____

Academic Session: _____

I _____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in

Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____

(Author's signature) (Supervisor's signature)

Address:

_____

_____ _____

_____

Supervisor's name

Date: _____ **Date**: _____

UNIVERSAL IOT DEVICES DISCOVERY SYSTEM USING PROTOCOL-INDEPENDENT NETWORK FLOWS CHARACTERISTICS

BY

LAI YAN LEUNG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

MAY 2018

DECLARATION OF ORIGINALITY

I declare that this report entitled "**UNIVERSAL IOT DEVICES DISCOVERY SYSTEM USING PROTOCOL-INDEPENDENT NETWORK FLOWS CHARACTERISTICS**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Mr. Aun Yichiet for giving me the opportunity to take part in this IoT devices discovery project. IoT has been my passion and interest and this project means a lot to me. I'm extremely grateful for this chance to help further improve the technology.

I would also like to thank Ch'ng Liu Yin for being with me at all times, giving me supportive advice and helping me stay strong through tough times during the process of completing this project. Finally, I would like to thank my parents for all the support they provide to aid me in the project.

ABSTRACT

This project is to develop an IoT device discovery system for academic purposes. It will provide students insights on network discovery concepts as well as methods and technology required to design it. This project aims to overcome cumbersome network management when dealing with IoT devices on multiple platforms. The cumbersome process roots from the heterogeneity nature of current IoT devices. Each IoT device supports a limited amount of platform. To craft an IoT ecosystem which best suits the requirements of a user, it is highly probable that more than one platform is used. This is because some platforms contain devices with functionalities which is not found in devices on another platform. For users to have a brief understanding about the number of devices or the number of a certain type of devices that exists in the network, users will need to access native platform application and combine the data obtained. The solution is a universal device discovery system which works across multiple platforms. The proposed solution requires knowledge about data packets for analysis on the network flow characteristics as well as programming languages for the construction of discovery and classification algorithms. Discovery of IoT devices is based on selected statistical features extracted from data packets being transmitted by devices within the network. These features are then combined to create a unique classification model to be used by a machine learning software to perform the classification. The result of the system is an accurate device classification model based on network flow characteristics. The result proves that, with further development, it is possible to produce a fully automated, self-updating device discovery system without requiring to periodically update the database.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
RTT	Round Trip Time
TP	True Positive
FP	False Positive

1.0 INTRODUCTION

1.1 Problem Statement

At the current stage of IoT development, IoT devices are getting increasingly accessible by the mass for different purposes such as research project, business entities or households. This is mainly due to the mass production of IoT devices which makes the cost for IoT devices being much cheaper. The participation of technological companies such as Broadlink, Logitech and Nest in the production of household IoT end devices has greatly expanded the market. However, all these end devices require a platform for them to communicate with each other. In the current market, big players in the technological industry such as Google, Amazon and XiaoMi have come up with their very own platform for example Google Home, Amazon Alexa and XiaoMi Home. However, in order to work in a particular platform, the end devices have to be designed to support the platform.

An ideal IoT implementation is achieved by having one IoT platform which all devices can be compatible with. However, in the real world, this ideal does not exist with the current state of technology. To fully automate a home, especially by doing it yourself, it often requires more than one platform. This is because not all IoT devices support the same platform and not all platforms are supported by the IoT devices which meets the requirements of the user. A problem arises where existing IoT end devices can only be discovered based on their inherent supported protocols. For example, a XiaoMi Air Purifier can only be discovered by XiaoMi MiJia. There are also IoT devices which supports multiple platforms such as the Sonoff Switch which supports both Google and Amazon. This makes it difficult for users to manage their IoT devices if each of their devices support different platforms. The current technology requires users to discover devices on per platform basis. This means that to discover a device on the Google ecosystem, users will need to access the Google Home application and to discover devices on the Amazon ecosystem, access the Alexa application so on and so forth. This can be very tedious to do. When users attempt to move their devices to a new environment, for example a friend's house, it will be difficult to find out which ecosystem is available unless they ask about it. Furthermore, network analysis can be challenged by the heterogeneity of the IoT devices. Network statistics from the router does not have much information such as the device type. It is hard for users to identify how many devices

of the same type currently exists in the network. Without this information, it will inhibit the ability to optimize the network performance based on the type and number of devices connected to it.

1.2 Project Scope

In this project, I would like to propose a universal IoT device discovery system using protocol-independent network flows characteristics. Besides regular discovery, the proposed system will also be able to classify devices up to the device-type level. For example, the system is able to detect and tell that which device a bulb is, and which is a socket. This is achieved by capturing data packets sent by IoT devices to determine the type of the respective device. This is done by analyzing the data captured by the system and match it with a set of pre-defined metrics based on several different attributes of the data packet. The accuracy of device discovery will depend on the amount of data received and the degree of definition of metrics on the system. The system can only detect and identify devices with metrics that already exist within the system. The device type that are currently supported for the project includes a bulb, an IR blaster, media player and socket. In this project, it is assumed that the machine learning techniques used does not employ self-learning capability. Hence, an update in implementation by the device manufacturer that might alter the values of features used in this project will render the device undetectable until the machine learning software is being trained again after the new implementation. This can be achieved by future works and development. Finally, for the purpose of this project, this system is designed to work with the three major platforms available in the IoT market which is XiaoMi Home, Google Home and Amazon Alexa.

1.3 Project Objectives

The main objective of this project is to produce a universal IoT device discovery system that is able to work across major smart home IoT platforms namely XiaoMi Home, Google Home and Amazon Alexa. Being cross-platform means that the system is capable of discovering and identifying devices regardless of which platform it does or does not support as long as it is one of the platforms supported by the system. These platforms were selected mainly due to its popularity and its large user base with each of them owning a huge piece of the market share. By working with these platforms, we are able to provide our system for the greatest number of users in the market.

One of the sub-objectives of this project is to synthesize existing IoT devices discovery and fingerprinting methods. Methods adopted is the most crucial part of the system as it directly impacts the capability and accuracy of the system to discover and identify IoT devices. In order to select the best method, the system is installed with several methods during the testing. Network packets are then collected and fed into the system to be processed. A result of the device discovered and identified is produced. This result is used to select the method that best suits this application.

Next, this project aims to simulate and collect network packets from devices running on three popular IoT protocols, namely XioaMi, Alexa and Google. Simulation is done by collecting IoT devices which supports these platforms which were set up in a physical environment. A packet sniffer is then used to capture packets being transmitted and received by the devices. The packets will be used to train the system for discovery and classification. The number of packets captured will have an impact on the accuracy of the system. Hence, it is important to ensure that sufficient number of packets are collected to train the system to produce a high accuracy metric.

This project also intends to identify useful features for IoT devices discovery and classification. These features will be able to allow the system to classify IoT devices based on the type of devices instead of their platform. For example, the system is able to discover and recognize a device to be a light or an air purifier based on the features identified. Similarly, the number of features affects the accuracy of the system. A large number of features implies a more accurate discovery and classification.

Finally, this project aims to evaluate the effectiveness of the protocol independent features in detecting cross-protocols IoT devices. This evaluation is crucial to determine the accuracy of the system in discovering and identifying IoT devices. This evaluation also allows us to fine tune the system by removing features that are less effective and adding features that are more effective.

1.4 Contributions

The success of this project will realize a better IoT devices monitoring system which is a universal IoT devices discovery and classification system using protocol-independent network flows characteristics. The proposed system will be able to help users easily manage their IoT devices by allowing them to discover devices across multiple platform. With device classification feature, network management and optimization can be done more easily. Instead of discovering devices using 2 separate application, devices discovery can be done on 1 single system. This will allow users to know how many IoT devices exists in the network at a glance. Furthermore, discovered devices can be identified up to its device-level and allow users to further determine which device type has what number of devices. These statistics can be very useful for network optimization and helps users know what network performance to anticipate.

In terms of device migration, the proposed system is able to help users migrate to an unfamiliar environment such as a new building. Without prior knowledge of what platform exists within the new environment, the system helps users identify platforms being used within the environment. This is achieved by sampling data being transmitted by IoT devices that has already existed in the network and identify which platform does that device belong to. Sampling all devices in the environment will result in the detection of all existing platform in the environment. This way, users can immediately know which platform integration process to be used.

Finally, this system is able to aid in network performance troubleshooting especially in large enterprise network where there exists a huge number of IoT devices. This can be especially useful if IoT device manufacturers only provide device model as the device name. A scenario where a performance bottleneck is identified to be one of the IoT device within the network, it is hard to identify which device is it based merely on information provided by the router such as IP address, MAC address and device name. This system specifically identifies the respective MAC address as a device type so that users know which device within the building is the culprit and is able to quickly remove it from the network. Furthermore, users can also know in which area to look for the faulty device and troubleshoot it instead of looking for the whole building. For example, when a building consists of several types of IoT device, if the system identifies the faulty device to be an air purifier, users immediately know that they should only inspect air purifiers

instead of also inspecting other devices such as smart bulbs and smart sockets which can be a very tedious task to do.

1.5 Proposed Approach

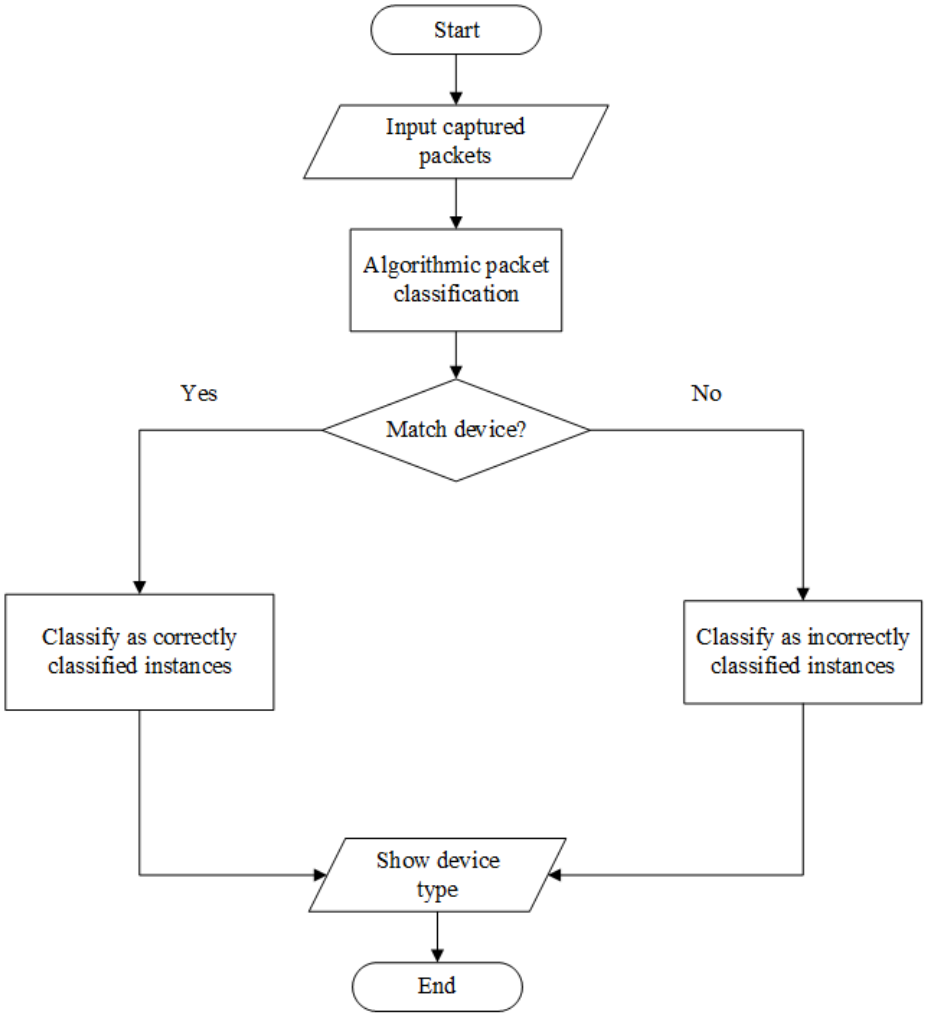

Figure 1-5-1: System work flow

The work flow of the proposed system is as illustrated in figure 1-5-1. The system works by feeding it packets captured from the network. The packets will then go through packet classification process done by a selected classification algorithm. The results will then be matched with existing devices to determine its device type. If there are no matching device type, the device database will have to be updated with the new device type. Finally, the system will show what device type does the input data packets belong to. At the current stage of development, the system is able to accurately classify devices that I own.

1.6 Report organization

The report is organized to make it easy to understand using a step-by-step approach. The first chapter highlights the field of study as well as problem and solutions this project can solve. It is then followed by providing a brief introduction of how the system will work. The following chapter highlights related works done by researchers with respect to solve similar problem.

Further sections of the paper detail the information about the system design, how it is being developed and how it works. Detailed explanation on the system will be provided in the sequence of the proposed work flow so that it is easy to understand. The paper then shows the results obtained from the project. Finally, the paper wraps up with a conclusion obtained from conducting the project.

2.0 LITERATURE REVIEW

In the current world where everything starts to be automated, Internet of Things emerges as an inevitable trend not only among the industry but also among households. As of now, an IoT system may consists of more than tens of nodes which may start to make the system hard to manage.

It is crucial to make the system easily manageable if we wish to use it efficiently. To achieve it, device identification and classification methods adapted by the system becomes one of the determining factor. A lot of efforts have been put by researchers across the globe to realize this by proposing their respective method of device identification and classification.

This study will discuss about previous related works and analyses the strength and weaknesses of each of them. This will be able to aid with the development of improved methods in the future.

2.1 Application Fingerprinting

In the world of information technology, fingerprinting refers to the process in which all information about a device available in the network is obtained and analysed to determine the hardware and software that exists in the network. This is usually done by network engineers to better monitor the network.

Currently there are two types of fingerprinting being practiced, namely active fingerprinting and passive fingerprinting. The first method is done by sending challenges to target hosts and analyse the replies from the host (Bartlett, Heinemann and Papadopoulos, 2007). Tools that can be utilized to perform active fingerprinting includes Nmap. Nmap contains a growing database of up to 2200 popular services port scanning being one of the most important function. When reply packets are received by Nmap, Nmap determines the type of services by comparing the packet with existing data on the database.

Another tool used for application fingerprinting is AMAP which also contains a database of known data signatures to determine the type of service (Rana, 2014). The differences between AMAP and Nmap is that AMAP is able to scan for services running on non-standard ports.

The second method is called a passive fingerprinting because it does not get in contact with the host. Instead, it captures packets being sent by a certain host in the network. One passive fingerprinting tool includes ETTERCAP which is most widely used to determine operating system and open ports. ETTERCAP examines packet headers of the packets sent by a host and also matches it with data signature stored on a database (Ko, Kang and Sim, 2007).

Besides the above types of fingerprinting, hybrid fingerprinting is currently being studied by many in terms of usability. This method combines active and passive fingerprinting by using Nmap and ETTERCAP simultaneously. This method allows application fingerprinting to be much more flexible. This method of hybrid fingerprinting is Hybrid Application Detection (HAD) which involves a two-phase process.

In this method, the purpose of phase 1 is to build a signature database. The process can be summarized as shown in figure 2-1. The first step collects results from tools such as NMAP ETTERCAP and AMAP and import it into a file for later use. Before importing the

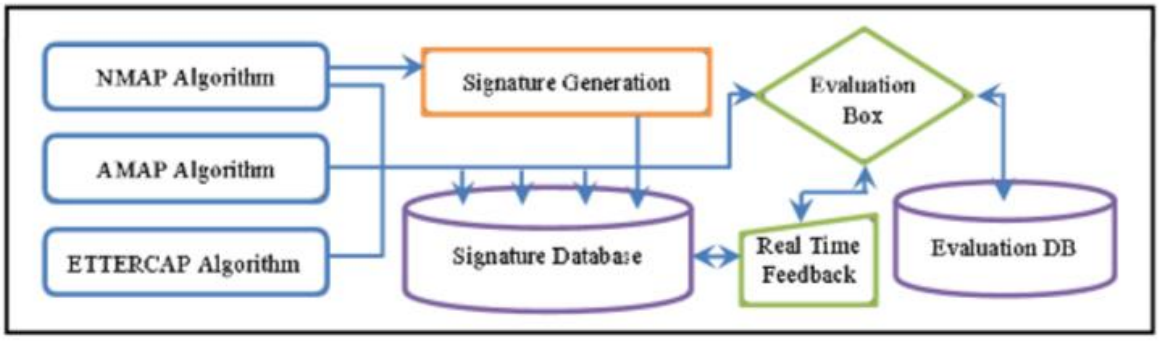

Figure 2-1-1: Phase 1 process.

results, only strings crucial for the process will be extracted and evaluated by conducting a real-time feedback for accuracy comparison. The next step generates signature for each extracted string. Finally, these signatures will be stored on the database. The next phase consists of evaluation process where accuracy of the system is being verified through benchmarking by performing stress tests on the tools. Sample sets and signature stored on the database is being matched with the requirement of an exact match and an inexact match. Exact matching involves direct lookup while inexact match involves algorithm which defines the condition in which a match is found. The system is being tested in comparison with NMAP, AMAP and ETTERCAP. The accuracy of this

project is defined by the formula (# hits / # runs) * 100. From the results, HAD performed better than all 3 fingerprinting methods with an accuracy of 94.06% in detecting service only and an 87.13% in detecting service plus version (Ghanem and Belaton, 2013).

In a nutshell, application fingerprinting is a decent device discovery and identification method as it is able to provide high accuracy in device identification due to its massive database of data signature. However, it is not able to automatically update its database. This downside happens when network devices undergoes software update and changed their data signature. In such cases, the device will fail to be identified as the new data signature does not exist on the database.

2.2 Universal Plug and Play (UPnP)

UPnP is a protocol that aims to allow UPnP devices to search for each other and connects seamlessly. It works on a basic principle which considers announcement and discovery as essentials in service discovery. This is a huge plus for unmanaged networks as it largely improves the manageability of the network. UPnP is designed for all peer-to-peer (P2P) network connectivity of all PCs, wireless devices and smart appliances. For this to work, the fundamental requirement is that it requires all devices to be installed with all UPnP protocols before using it. UPnP is built on an open, internet-based communication standard for example SSDP, XML and HTTP (Jin, et al., 2014).

Among all communication standards supported by UPnP, XML has a large advantage over the other standards because it can contain detailed information about the device. One major drawback is that it requires adequate memory and processing power to support it. Simple Service Discovery Protocols was included in UPnP to easily discover devices within a network without any prior knowledge of it. It works by using HTTP to search for services available within the network and advertising the availability of the service (Jin, et al., 2014).

UPnP is a very powerful method in determining services and devices without prior knowledge. However, a few requirements had to be met such as sufficient storage space and processing power. Pre-installation of UPnP is also required on all involved devices. As UPnP relies heavily on multicast for advertising services, it is not scalable as the network will be saturated with advertisement traffics of UPnP when the amount of UPnP device reached a large scale.

2.3 Efficient Probing of Heterogeneous IoT Networks

IoT networks often exhibit the nature of being heterogeneous with each device originating from different manufacturer, different operating system, and naturally different network technology is used. This poses a huge challenge in term of device discovery and identification. The fact that most IoT networks consists of large number of devices makes this issue even worse. To solve this issue, an approach utilising Round Trip Time (RTT) measurements with variable network size and probing speed is proposed.

In this approach, differentiation of IoT devices can be performed by the scanner using RTT based on the network technology used. The first step involves fast probing of an entire selected IP address range. To do this, it is reasonable to use a high scan rate technique. This is able to provide an insight on what to expect of the scanned network. The following step involves probing with variable network sizes. The purpose of this step is to find out what network technologies used by the devices in the network which had been discovered in the first step. This step is done by dividing the discovered devices into smaller sub-sets and perform a slow scan on each of the sub-sets. As IEEE 802.15.4 standards only yield a maximum of 250kbit/s speed while IEEE 802.11 standards have a whooping 11Mbit/s, it is obvious that devices using IEEE 802.15.4 standards will result in a longer RTT for data packets larger than 128 bytes compared to IEEE802.11 due to its smaller physical layer service data unit (PSDU) of 127 bytes compared to 1280 bytes of MTU on the latter. Finally, a slow re-probing of the network is performed on address range identified in the previous step. This is done as a precautionary step to identify devices that are probably missed out by the first step.

This approach has been tested on IEEE 802.11 only networks, IEEE 802.15.4 only networks and a mix of these 2 networks as a representation of the heterogeneous nature of IoT. IEEE 802.11 only networks proved to be tolerable on probing size. The network is probed using inter-probing time (T) of 10ms, 20ms and 80ms with light and normal background traffic. From the results obtained, this approach successfully discovered 100% of the devices with the network (Metongnon, Ezin and Sadre, 2017).

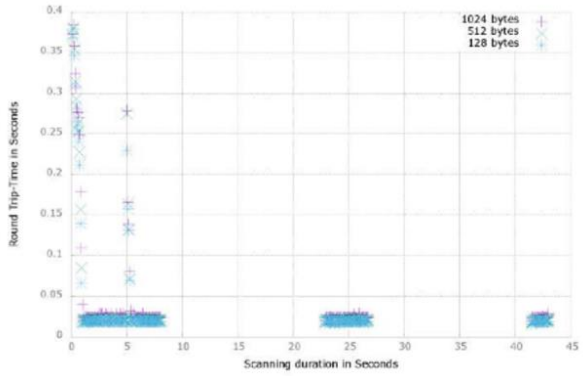

Figure 2-3-1: RTT for IEEE 802.11 normal background, T=80ms (Metongnon, Ezin and Sadre, 2017. p. 1055)

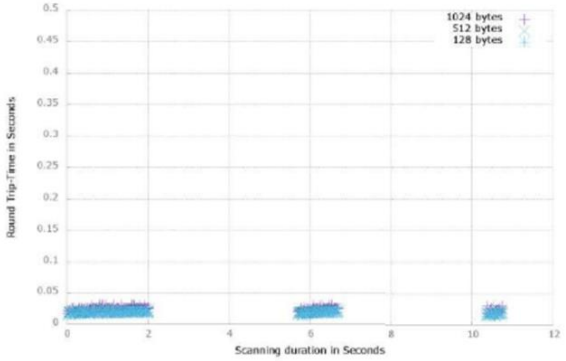

Figure 2-3-2: RTT for IEEE 802.11 normal background, T=20ms (Metongnon, Ezin and Sadre, 2017. p. 1055)

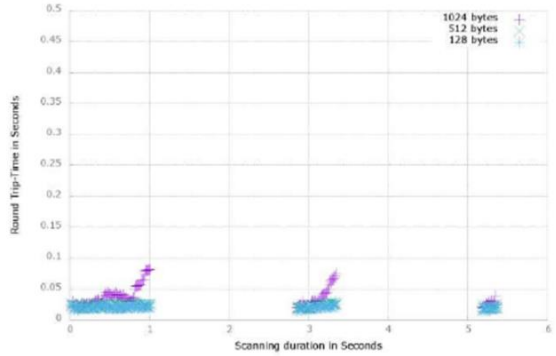

Figure 2-3-3: RTT for IEEE 802.11 normal background, T=10ms (Metongnon, Ezin and Sadre, 2017, p.1055)

Based on the figures above, probe size of 1024 bytes are required to queue when sent at T=10ms.

In networks with IEEE 802.15.4 devices, similar tests are being carried out yielding the results in the figures below.

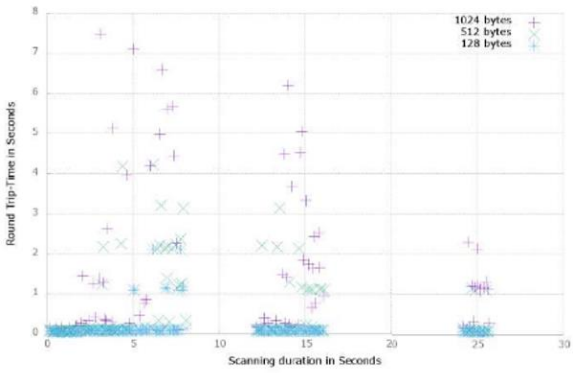

Figure 2-3-4: RTT for IEEE 802.15.4; normal background; T=80ms
(Metongnon, Ezin and Sadre, 2017, p.1056)

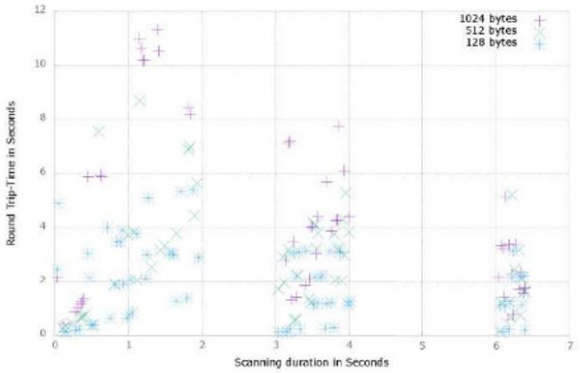

Figure 2-3-5: RTT for IEEE 802.15.4; normal background; T=20ms
(Metongnon, Ezin and Sadre, 2017, p.1056)

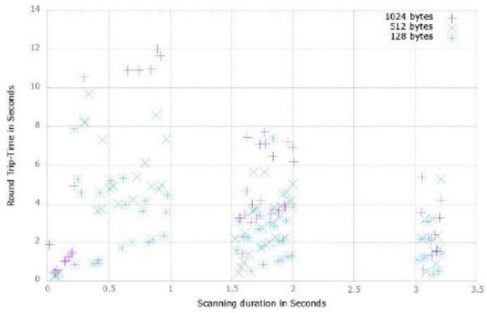

Figure 2-3-6: RTT for IEEE 802.15.4; normal background; T=10ms
(Metongnon, Ezin and Sadre, 2017, p.1056)

As observed from figures 2-2-4, 2-2-5 and 2-2-6, IEEE 802.15.4 devices exhibits very high RTT with large fluctuations for all probe sizes when probing intervals T=20ms and 10ms. It is also observed that the probing suffers from quite a number of losses. However, when T=80ms, the probe successfully discovers more than 95% of devices.

In a mixed network where IEEE 802.11 and 802.15.4 devices exists, this approach is used to perform probing. The first step using high speed probing of T=10ms and size of 32 bytes yields

the result in figure 2-2-7. As the probe size is not that large, there is little concern about it. As for the low inter-probing time, it achieved the desired result of a low scan period.

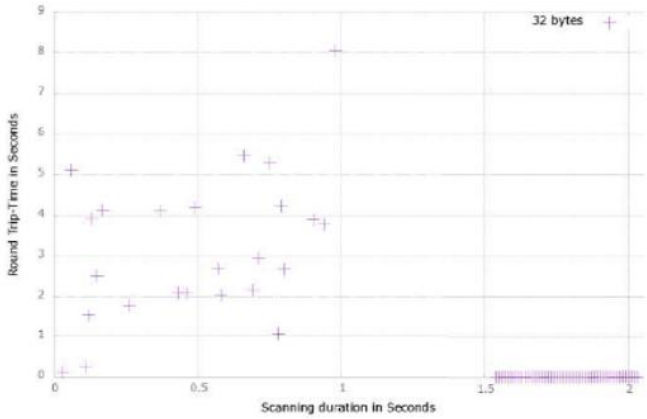

Figure 2-3-7: Mixed network; Normal background; T=10ms
(Metongnon, Ezin and Sadre, 2017, p.1057)

Based on the results, it can be seen that the first half of the network consists of high number of IEEE 802.15.4 devices as there are a lot of fluctuations and loss while the second half has no problem with the probes implying that it might contain a high number of IEEE 802.11 devices. A second scan is conducted by randomly selecting small number of devices and perform slow probes on these devices. The scan is done using different probe size of 32 bytes and 512 bytes with probe speed of T=80ms.

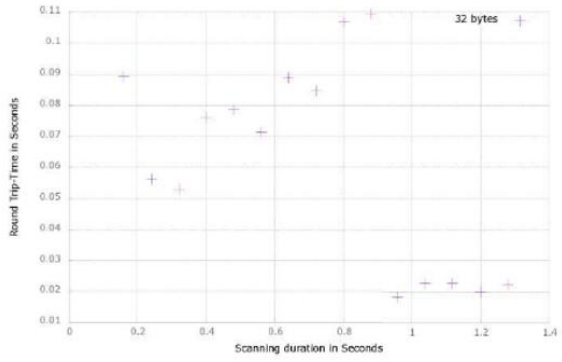

Figure 2-3-8: RTT for selected devices; Normal background T=80ms; 32 bytes
(Metongnon, Ezin and Sadre, 2017, p.1057)

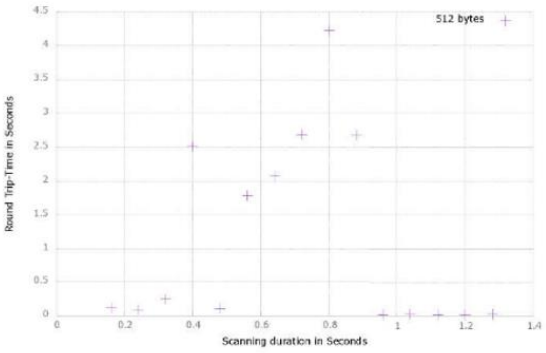

Figure 2-3-9: RTT for selected devices; Normal background; T=80ms; 512 bytes
(Metongnon, Ezin and Sadre, 2017, p.1057)

Based on figures 2-2-9, it is observed that probe size of 512 bytes caused a large increase in RTT which implies that these are IEEE 802.15.4 devices. Finally, the last step to rescan address range containing IEEE 802.15.4 devices are performed using slow speed probing with small probes of 32 bytes. This scan yields the result in figure 2-2-10. It is shown that the scan was able to discover 96% of the IEEE 802.15.4 devices. This indicates that results of slow scan are achieved without needing to perform a slow scan on the entire network.

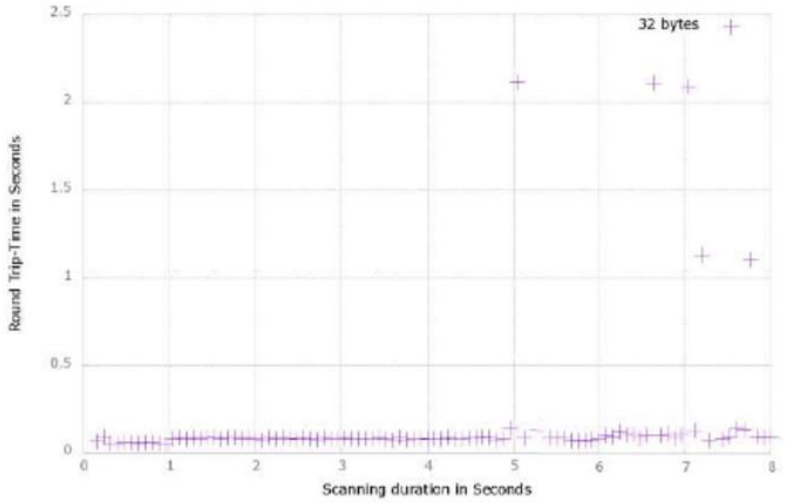

Figure 2-3-10: RTT for mixed network; normal background; T=80ms; 32 bytes
(Metongnon, Ezin and Sadre, 2017, p.1057)

In conclusion, this approach has successfully performed fast probing on networks consisting of high speed and low speed devices without needing to sacrifice on probing speed as long as the appropriate probe size is selected. This approach aces in high discovery of devices that exists in the network as well as active response to newly added devices. Although it does classify devices based on their network technology, it does not perform more specific classifications such as operating system or services running on devices.

2.4 IoT Eye

IoT Eye is a two phase IoT device auto-discovery protocol. Phase one is a high-speed TCP scanning of IP segment of active hosts within the network. In this phase, a high-speed scan is performed to discover active hosts along with their respective open ports on the network. This is done by combining Zero-Copy Socket and TCP SYN probe. Phase two consists of equipment fingerprint identification and data interaction. Based on open ports on the active host, the system is capable of learning details about the device such as device type, operating system info, brand and supported services by deploying further data interactions via corresponding protocols. Since the system uses high-speed IP address scanning and active protocol fingerprinting, the system is able to detect devices once the device is exposed within the shortest time period.

In order to have an accurate real-time equipment fingerprinting, PI-AC algorithm is used. PI-AC is a multi-pattern matching algorithm which supports parallel matching and incremental modification of matching automata. Due the heterogeneous nature of IoT, keyword sets used to generate matching automata becomes very large because it is being continuously updated to ensure accurate device detection. The system also includes a quick emergency response rule in order to detect devices that expose themselves suddenly in the network. This involves quick responses towards keyword set updates. This is done my adopting a PI-AC algorithm that supports dynamic incremental modification of model built on Aho-Corasick (AC) algorithm.

The first step of IoT Eye involves a network probe using TCP SYN scan. The usual TCP port scan is good for determining status of IP host whether it is alive or dead as well as open ports on each host. As TCP is connection-oriented which involves a three-way handshake to establish a connection, it proves to be a reliable protocol to be used for network scan. However, it is tedious to repeatedly establish a TCP connection. This results in great delay in the discovery of live hosts in the network. Hence, a high-speed TCP SYN scan is introduced. Using this method, a host is

determined to be alive as long as the scanner sends a SYN message to the host and it replies with a SYN+ACK message. In this method, there is no requirement for a complete establishment of the TCP connection. This allows for a much faster network probe.

To achieve a much faster data processing rate, zero-copy socket is used. In most operating system, architecture is designed in a hierarchical method. This makes copy operation very cumbersome. Copy operations are performed by having the kernel read data from the memory, transfer the read data to an application which is operating in a user-level, then the application transfers the same data back to the socket which is in kernel-level and finally the data is passed from the kernel to the network interface in order to be sent out a physical link. This process is illustrated in figure 2.0.

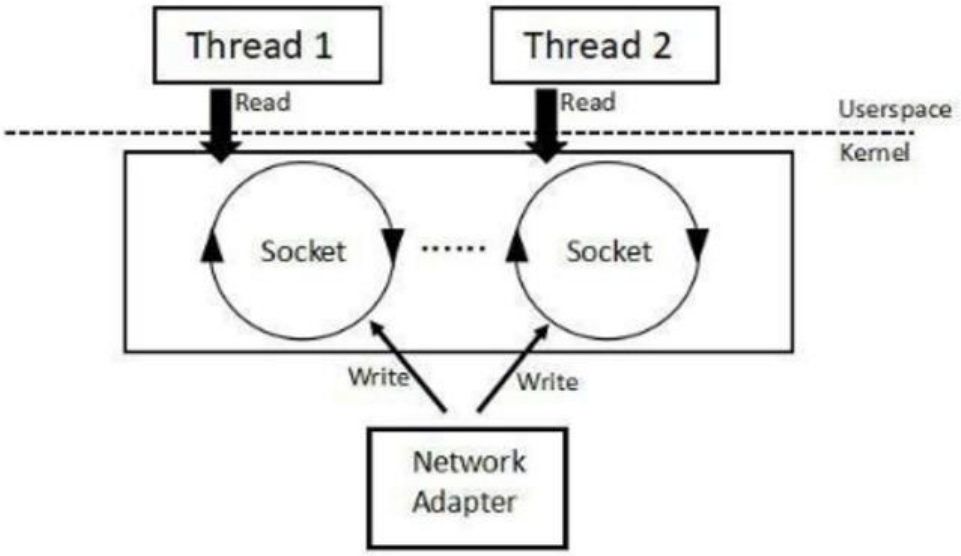

Figure 2-4-1: Copy process of NIC data

From this process, it can be seen that the user-level application is an extra intermediate medium that does not contribute much to the purpose of the process. Zero-copy socket solves this issue by avoiding redundant system calls. It is achieved by skipping system kernel and access storage hardware directly.

PI-AC algorithm takes the initial characters of each string from a pattern set and constructs an index array. Each string in the pattern set is divided into subsets according to their first character and only characters of each subset is taken. This allows the elimination of repeated characters in

the array. To check if the string already existed previously, the character is converted into its relevant integer and used to access the array element of that integer. If the element is non-null, this means that this character had already existed, and other pending characters will be inserted as a sub-tree of the initial character. If the element is a null, this means other pending characters is new and is inserted into the array. All other pending characters will be added as a sub-tree of the new character. PI-AC algorithm is designed to perform parallel pattern matching. This implies that all matching tasks runs simultaneously, and each data being processed does not affect the or interfere with each other. This allows for high-speed data processing which contributes to the emergency update response of the IoT Eye.

From the above methods, IoT Eye managed to complete a single port scan on 131072 IPs within 16.33s while multi-port scans took 534.62s. Based on this result, it is considered that IoT Eye is able to complete a port scan within a significantly short period of time. In terms of pattern-matching, PI-AC algorithm is able to achieve a total of 50000 pattern matching in 185.92s. This is a very short period of time considering the large number of patterns being matched. From the result, IoT Eye proved to be a highly efficient and reliable system to conduct automatic device discovery in real time (Shen, et al., 2017).

2.5 Automatic Discovery and Classifications of IoT

Currently in the advancement of IoT technology, many devices are being manufactured for many different distinct goals that would result in different modes of connection in reality (Pedro & Luis, 2017). This is because every mode of connection has its own advantages and disadvantages which if properly used can be able to effectively achieve its goals. This is also true for the many different platform that exists within the IoT industry namely Alljoyn, IoTivity, Apple HomeKit and Google Brillo. To achieve the sole purpose of IoT which is to (Pedro & Luis, 2017) allow communication among device within a network such as a local area network (LAN) and the Internet, a simple application is developed to perform automatic discovery and classification of IoT devices.

The primary function of the application is to analyze captured data packets and classify these packets to its respective device and device type. For example, the application is able to tell that a captured packet belongs to a SmartLife smart bulb. To achieve this, the process if broken down into 3 phases – capture, analyze, classify.

During the first phase, data packets exchanged by devices within the network is being captured by a HTTP proxy server placed at the access point. This proxy server will then allow the data exchanged to be viewed including those with SSL certificate. The proxy server is also responsible of extracting communication files in the form of XML or JSON as input data for the system. Figure 2-5-1 shows the network topology setup for the project.

Figure 2-5-1: Network Topology (Pedro & Luis, 2017)

In the analysis phase, four different algorithms, which is the TF-IDF Table, Levenshtein Algorithm, Synonyms Match and Multi-property Matching is selected for the classification process. During this phase, all data collected will be fed to the algorithm to perform classification based on its device type. Finally, the application displays all devices discovered according to its device type.

This approach is being tested using devices already on the database and devices that does not exist in the database. Table 2-5-1 shows the comparison of average precision in the algorithms applied. According to the table, it can be observed that Levenshtein algorithm is good in performing classifications on devices already on the database. Whereas Multi-property Matching algorithm aced the precision in classification of devices not on the database.

Algorithm	Devices already on database	Devices not on database
Direct Match	0.83	0.39
Levenshtein Algorithm	0.70	0.46
TF-IDF Table	0.68	0.42
Synonyms Match	0.66	0.48
Multi-property Matching	-	0.91

Table 2-5-1: Comparison of average precision for device classification among selected algorithms

2.6 Comparison

In this section, all discussed methods are compared against one another based on attributes related to our works. Each of these attributes play an important role in the proposed system. Table 2.0 shows the comparison of the results obtained from the above studies.

Attributes / Methods	Scan Speed	Classification Level	Network Size
Application Fingerprinting	Normal	Service	Normal
UPnP	Normal	Device	Small
TCP Probing	Fast	Network Technology	Large
IoT Eye	Fast	Device	Large
Automatic Discovery and Classifications of IoT	Normal	Device	Medium

Table 2-6-1: Comparison of results

3.0 METHODS

3.1 Design Specification and Overview

The realization of the project involves the four standard phases which includes environmental setup, data collection, training process and evaluation. Each phase is crucial in determining the quality of outcome of the project. The fundamental concept of the project is to produce a device classification model which is able to differentiate and identify IoT devices at the device level within a network. This is typically achieved by differentiating a combination of features that are unique to a certain device such as network behavior, payload size and protocol used.

A brief work flow of the model starts by obtaining datasets by capturing communication traffic being exchanged between devices within the network. These data are then processed into useful information via data processing. A set of data packet features or attributes are being identified and used for feature extraction of the corresponding data packet. The dataset is then split into training data and testing data respectively. The training data is to be fed to a machine learning software which performs data classification. The classified data will then yield a model to identify IoT devices within a network. The work flow of the model is illustrated in figure 3-1-1.

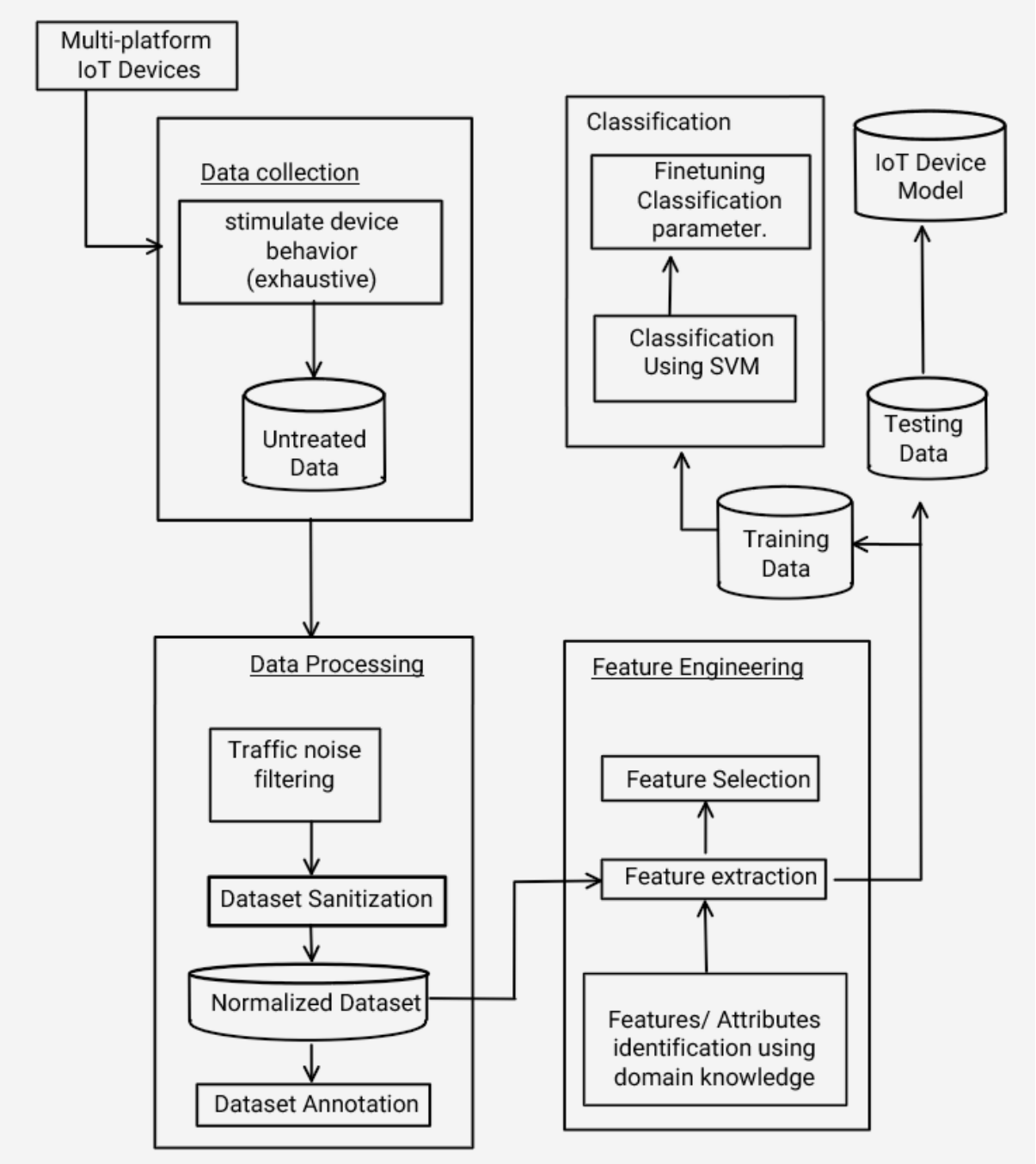

Figure 3-1-1: Flow chart

IoT environment has to be setup prior to the data collection process. To achieve this, a network of IoT devices are being setup in a room. The IoT devices are being setup using a star topology to ease the data collection process. The network topology is illustrated in figure 3-1-2. A star network topology will guarantee all IoT devices which wishes to communicate with each other

or access the Internet must have their packets passed through the central node. This will ensure no communication packets are missed during data collection.

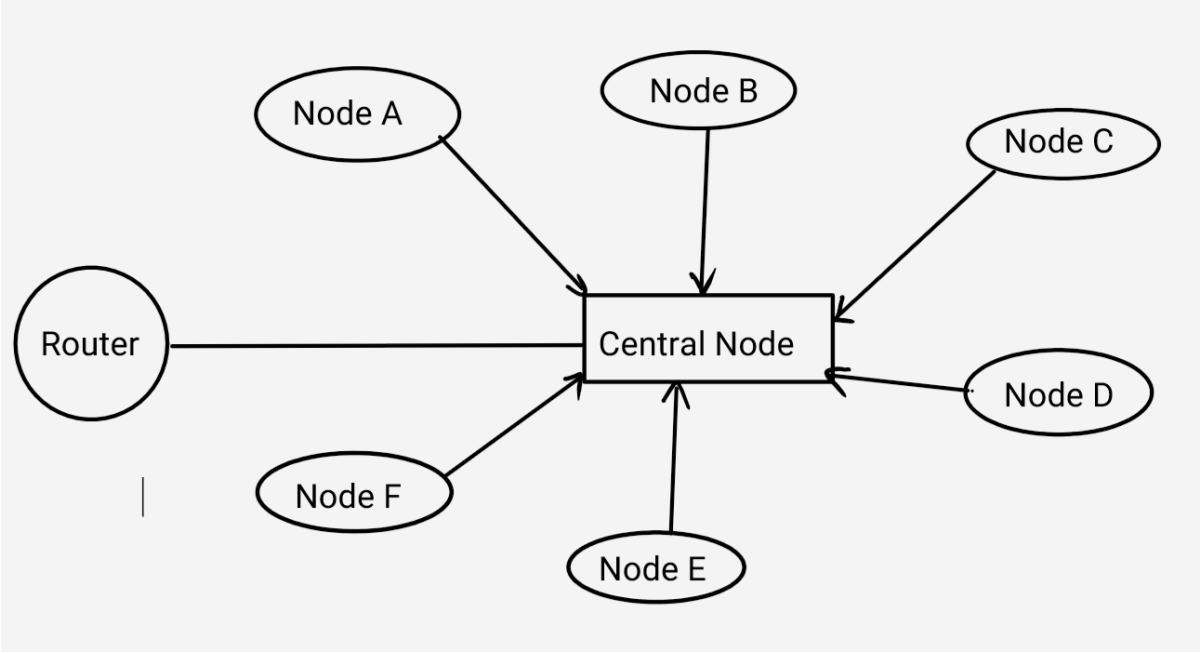

Figure 3-1-2: Network topology

After the environment is being setup, data collection is performed at the central node. Wireshark is the software used to perform packet capture due to its large amount of predefined protocol which eases the process of packet analysis in the later stage. During data collection, each device is being stimulated to generate data packets and communicate among devices. This is done by triggering a node to communicate with another node to perform an action. Before each device is being stimulated, packet capture is started. After the desired action has been performed by the devices, the packet capture is stopped. The same process is being repeated for all devices and platforms available. The resulting captured packet will include all background traffic and the traffic required for feature engineering. The packets captured is then filtered by the IP address of the relevant devices to produce a clean dataset without any traffic noise. These packets are then labelled based on the devices involved for storing.

The next step is the feature engineering. In this step, the datasets acquired is being analysed to identify combination of features unique to each devices and platform. This process is done based on the domain knowledge available to the analyser. Features to take note includes packet transmission frequency, packet time, payload size, protocol used and so on. After the features had

been identified, a feature selection is performed so that only feasible features are used to build the classification model during the next step.

With the available set of features, device classification can now be performed. This involves training a machine learning software, Weka and have it classified the devices based on a certain model. First of all, Weka is configured with classification parameters which will be used to differentiate the devices. Weka classifies the devices using the following steps:

1. Weka takes input data packets and put it through a classification algorithm.
2. The classification algorithm decides if the input packets belongs to a specific device
3. If yes, Weka successfully classifies the packet.
4. If no, Weka indicates it as incorrectly classified instances.
5. Finally, Weka will show the type of devices being classified and its respective accuracy matrix.

The process is illustrated in figure 3-1-3.

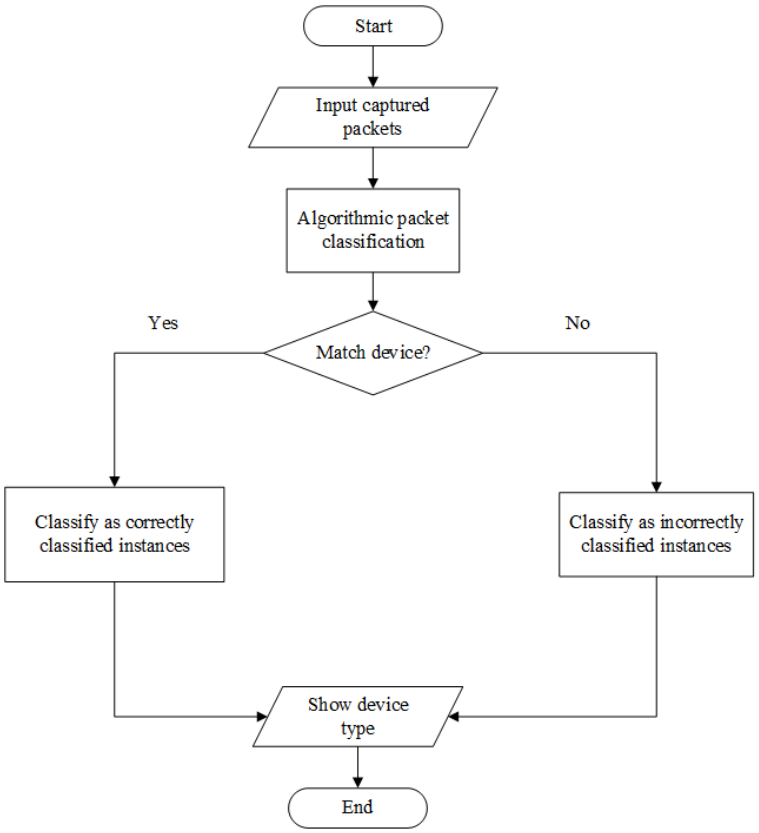

Figure 3-1-3: Pattern matching flow

Finally, the model is being evaluated using the same method with testing data captured previously using Wireshark. The machine learning software performs a pattern matching based on the constructed feature model and classifies the testing data into its respective device and platform. The model is said to be accurate if the classification results matches the expected outcome. Else, finetuning of the classification parameters is done by altering the feature model to improve the accuracy of the model.

3.2 Issues and Challenges

Each of the stages comes with its own issues and challenges. Most of it is due to the requirement of utilizing a mobile hotspot created by the laptop for the other IoT devices to be connected to. The issue a rise during data collection when a node is triggered to communicate with another node. Communication within the local network was not an issue, but with unknown reasons, the communication involving Internet access was not possible, making it unable to generate the packet required for the dataset. NAT was suspected to be the reason. However, it was not verified. This issue has been resolved successfully by obtaining a third-party software which turns the laptop into a Wi-Fi repeater, subnetting the router network instead of creating a whole new network.

The next challenge happens due to monetary constraints in obtaining IoT devices to setup the environment. This has caused the difficulty to obtain an Echo Dot to provide the Amazon Alexa platform which is one of the most important part of the project. To resolve this issue, a previously owned Raspberry Pi is being setup with an external speaker and microphone to take over the role of Echo Dot. The Raspberry Pi is then installed with an unofficial open-source Alexa smart assistant called the EchoPi. It was able to access most of the functions of Echo Dot including communicating with other supported IoT devices.

The final issue is encountered during dataset filtering. A laptop being used as a mobile hotspot to work as a medium of communication between nodes introduces a lot of traffic noise due to background services running on Windows making it very difficult to filter. It gets even more difficult when there is no known port number used for communication between the nodes. To solve this issue, intensive analysis of the network traffic is performed to distinguish noise from the required traffic.

4.0 SYSTEM DESIGN

4.1 Data Collection

The initial phase in designing the system is the data collection phase. This phase involves two parts. The first part is to set up the required IoT integration environment. The second part involves capturing data packets being sent and received by the IoT devices within the network.

In setting up the environment, the mobile hotspot functionality of the laptop is used as the internet access point for all devices of interest. In this project, IoT devices used will be the Amazon Echo Dot, Google Home Mini, Broadlink RMPro+ smart remote, Broadlink SP2 smart socket, XiaoMi Second Generation Air Purifier and the SmartLife smart bulb. All devices have Wi-Fi installed in order to be able to connect to the Internet and the laptop.

To join all the devices to the same network, all devices are set up using its respective process. For example, use the Google Home app to connect to the Google Home Mini to the network and use the Amazon Alexa app to do the same for the Amazon Echo Dot. Similar process applies to all the other devices with manufacturer specific instructions. Once all devices are being connected to the network, it is now possible to link these devices to its respective platform. As all devices used in the project supports both Amazon Alexa and Google Home platform, these devices can be linked directly to each platform from within the platform app which is the Amazon Alexa and the Google Home app. Once these are done, setting up of the environment is now complete.

The second part requires a special software to be installed in the laptop in order to collect the data. In this project, Wireshark is used as the data packet capture software. This is because Wireshark has promiscuous mode enabled. This mode allows it to capture any data packets going through it regardless of their destination. This allows it to capture data packets not destined to be sent to the laptop. In order to only capture the packets of interest, capturing of data packets is done at the mobile hotspot network interface only. This ensures clean packet capture with minimum amount of noise introduced in the data collection process. Promiscuous mode can be enabled by accessing the packet capture option as shown in figure 4-1-1.

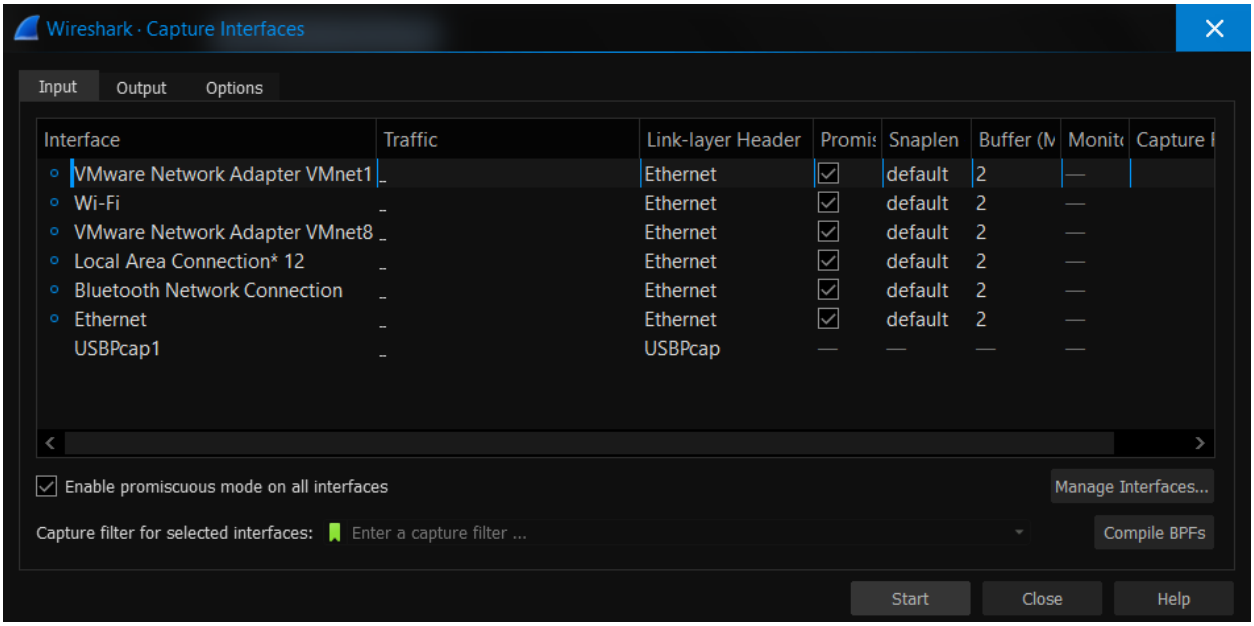

Figure 4-1-1: Enable promiscuous mode

Upon starting capture, data packet generation can be sped up by continuously execute commands to the Google Home and Amazon Alexa to perform some actions such as turning on the light or turning on the sockets depending on what device you have. This will trigger the IoT devices to generate command specific data packets which will significantly improve the accuracy of the system if it is used to perform the training. Once the number of captured data packets hit the requirement. This concludes the data collection process of the system. Figure 4-1-2 shows the data collection process using Wireshark to capture data packets being exchanged within the network.

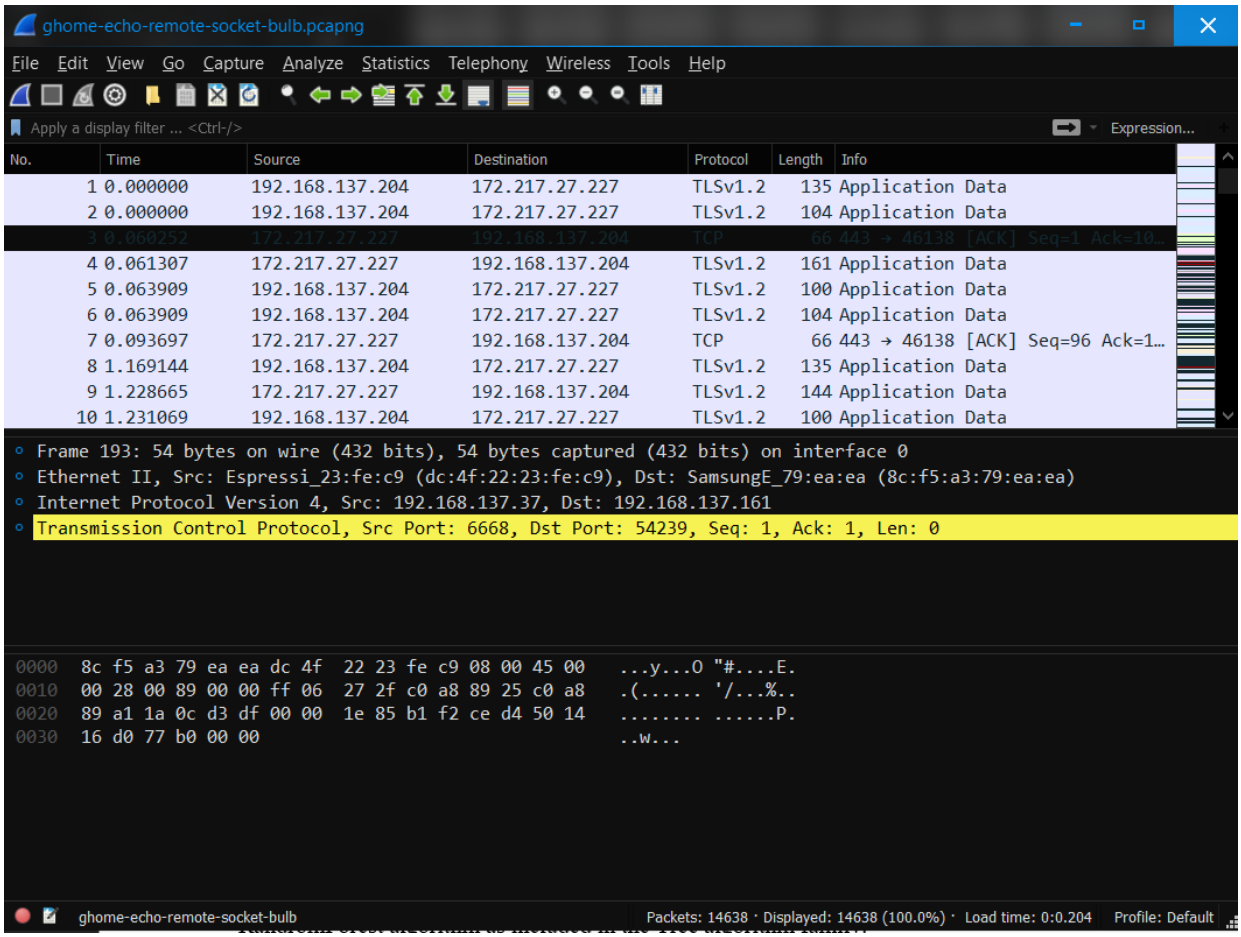

Figure 4-1-2: Data packet capturing using Wireshark

4.2 Data Processing

This process aims to produce useful dataset that can be later used as training and test materials for our machine learning model later in the final process. The first part involves extracting packets of interest from the large block of data packets being captured. The data packets are being extracted should be distinguishable from other devices. To do so, simply apply a filter on Wireshark to produce the required data packets. In the project, as each individual device is our data interest, packet filtering is done by filtering MAC address. This guarantees that all data packets to be extracted belongs to that specific device. To do so, the command “eth.addr==<Device MAC Address>” is entered into the filter field in Wireshark where <Device MAC Address> is replaced by the real MAC address of each device of interest. This is illustrated by figure 4-2-1.

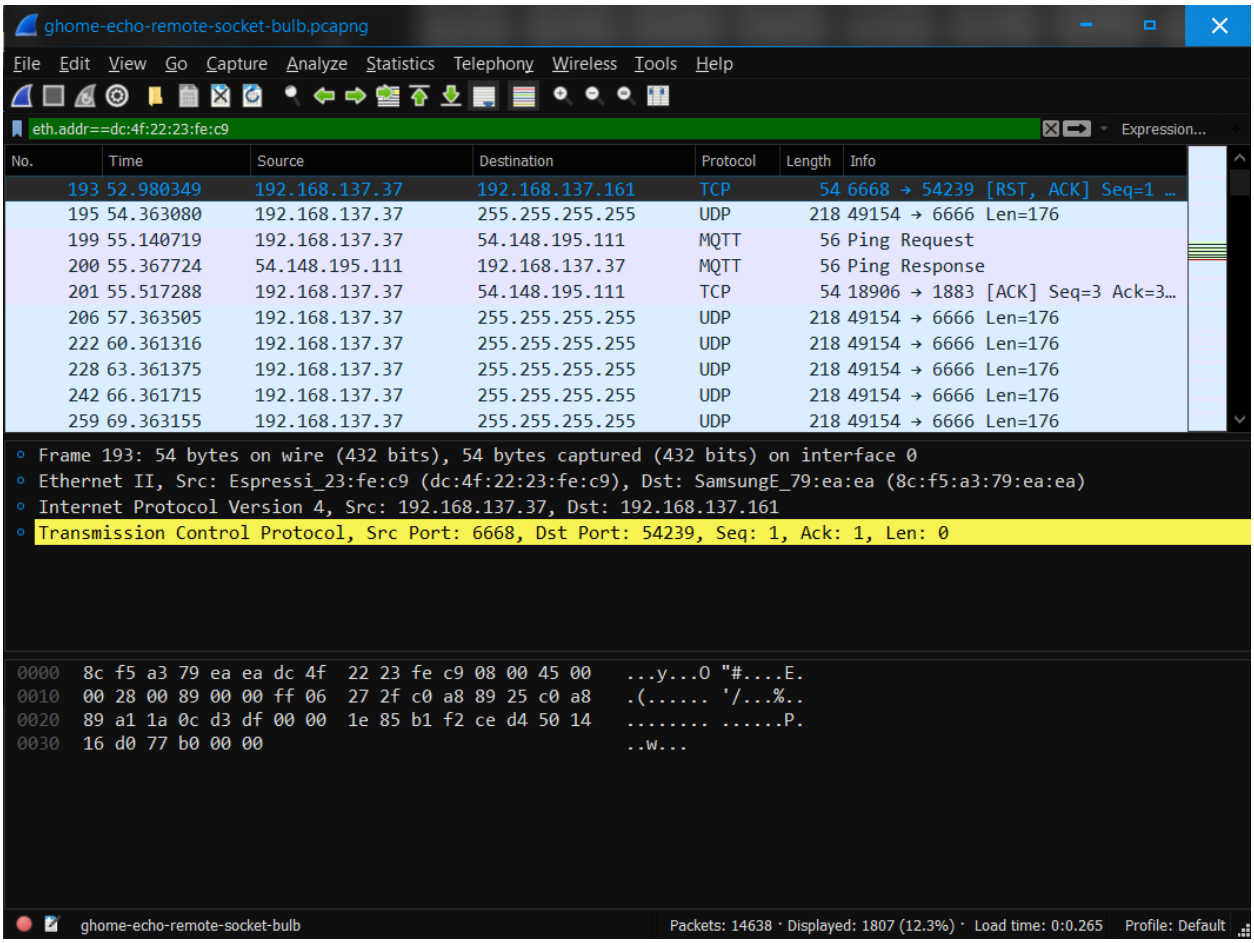

Figure 4-2-1: Device filtering process in Wireshark

In order to extract these packets of interest, it is possible to export specified packets and label it as the device name for dataset annotation. This process is illustrated in figure 4-2-2.

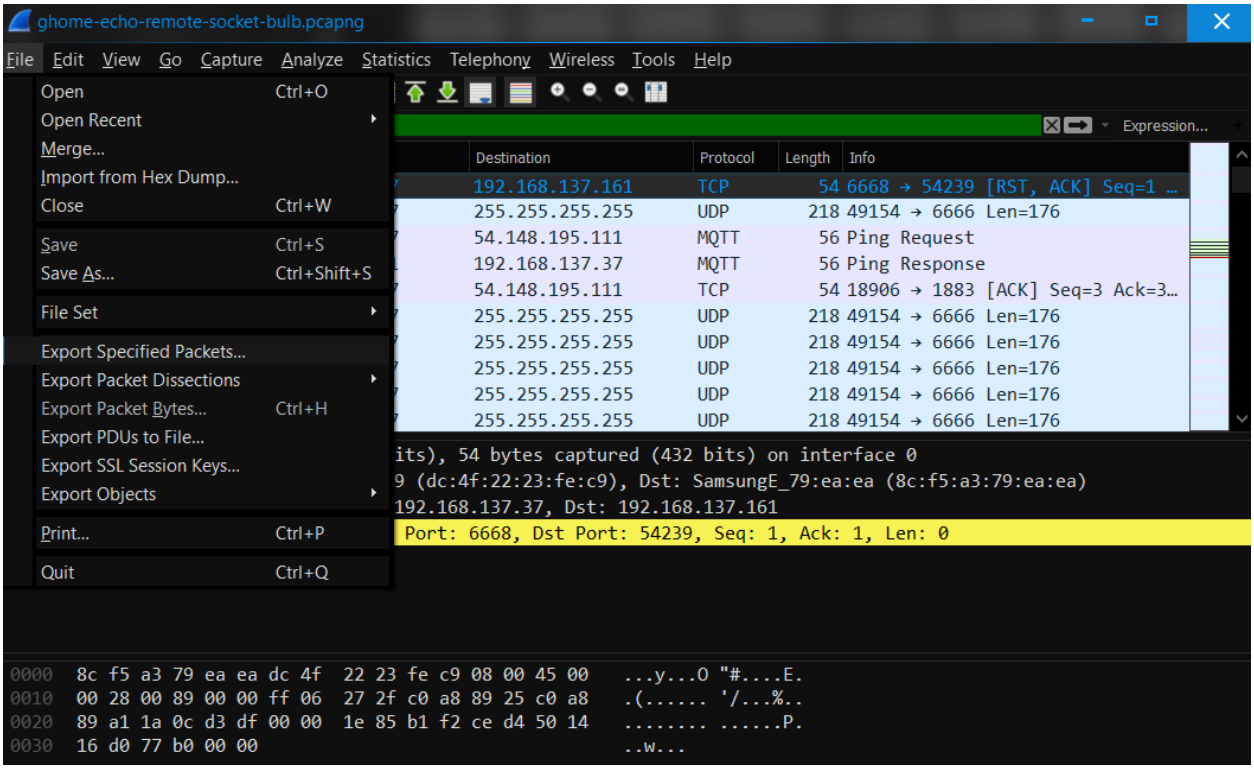

Figure 4-2-2: Exporting specified packets in Wireshark

In order to prepare the dataset for the final process, the data packets are saved as a text file in its hex dump form. This can be done using the “save as” function from within Wireshark. Each file saved is labelled with its respective device name. The data processing process is complete once all data packets that belongs to each individual device has gone through the same process. This results in six files. One .pcapng file which is the original Wireshark capture file and 5 .txt files which is the individual data packet hex dump of each device. This completes the data processing process. Figure 4-2-3 show an example of hex dump in text.

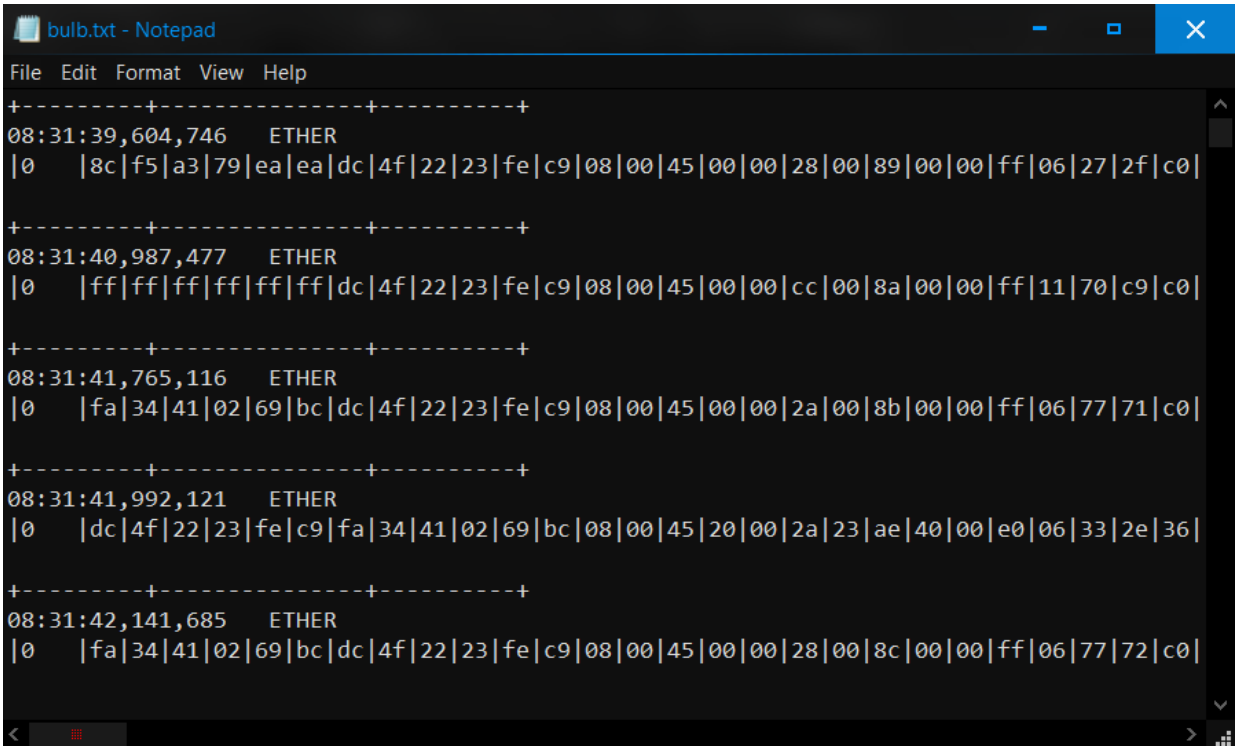

Figure 4-2-3: Wireshark packet hex dump in text.

4.3 Feature Extraction

This process involves programming of specialized software to perform the feature extraction from the dataset processed in the previous process. In this project, a few network flows statistical feature is being identified in order to be used for the machine learning software to classify each device. In this sense, these features selected must have values unique to that packet only. In the project, the features selected includes interarrival time, total packet length, packet header length and packet payload length of each packet. An additional feature which is the average payload length of packets being captured within one second is used to further distinguish the packets based on network flow feature.

The program in the project is writing in C++ using Microsoft Visual Studio as C++ is most efficient when running on a Windows laptop. The designed program is written to extract all mentioned feature values from the input dataset and output as a .csv file for use with the machine learning software. Upon extracting all feature values for all devices, these values are then

combined into one single .csv file using Microsoft Excel. It is also required to label each feature value with its respective device name as shown in figure 4-3-1

	A	B	C	D	E	F	G
1	device	IAT	total	header	payload	packets in	sMeanLoad
2	socket	0	76	20	56	1	56
3	socket	1119860	84	20	64	2	60
4	socket	20647870	132	20	112	1	112
5	socket	130670	84	20	64	2	88
6	socket	2.28E+08	76	20	56	1	56
7	socket	1292080	84	20	64	2	60
8	socket	99401710	132	20	112	1	112
9	socket	130070	84	20	64	2	88
10	socket	1.49E+08	76	20	56	1	56
11	socket	1241470	84	20	64	2	60
12	socket	49316290	132	20	112	1	112
13	socket	137080	84	20	64	2	88
14	socket	2.09E+08	76	20	56	1	56
15	socket	1354690	84	20	64	2	60
16	socket	1.86E+08	132	20	112	1	112
17	socket	128690	84	20	64	2	88
18	socket	62037590	76	20	56	1	56
19	socket	1274700	84	20	64	2	60
20	socket	1.32E+08	132	20	112	1	112
21	socket	129060	84	20	64	2	88
22	socket	1.17E+08	76	20	56	1	56
23	socket	1358070	84	20	64	2	60
24	socket	2.49E+08	76	20	56	1	56
25	socket	1286760	84	20	64	2	60
26	socket	17869460	132	20	112	1	112
27	socket	131750	84	20	64	2	88
28	socket	2.21E+08	132	20	112	1	112
29	socket	129770	84	20	64	2	88

Figure 4-3-1: combined .csv file

The resulting .csv file is then converted into a machine learning input .arff file. This can be done from within Weka, the machine learning software to be used in this project. The following steps explains the conversion process.

1. Select “Tools” from the top menu in Weka GUI Chooser.
2. Choose “ARFF Viewer”
3. In the new windows, go to “Files” and choose save as.
4. In the pop-up windows, select “.arff” from the file format drop down list.
5. Finally, provide a file name and save the file.

This process is being illustrated in figure 4-3-2.

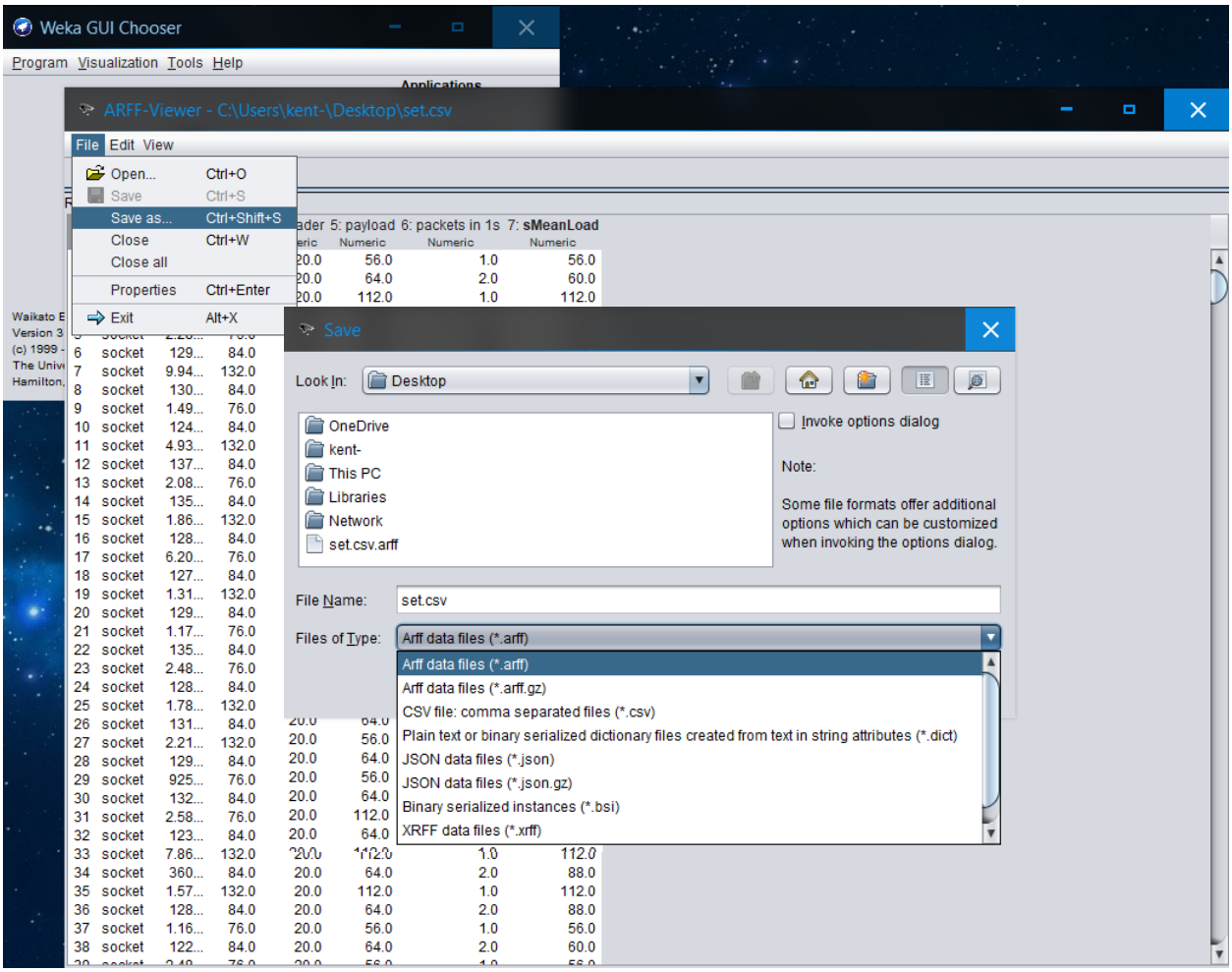

Figure 4-3-2: Conversion of .csv to .arff

4.4 Model Training

This marks the final process of producing the system. In this process, Weka is used to perform the training and testing of the classification model using the prepared dataset. The first step involves selection of classification algorithm to be used. This is crucial because these algorithms will directly impact the performance of the system in terms of time efficiency. A good algorithm will be able to perform the classification process in a short period of time. This is because time delay can be noticeable on large scale network.

In this project, RandomForest is selected to be the classification algorithm due to its accuracy. To run the training, simply import the prepared dataset in the form of .arff into Weka by

using the “Open file” button. Next, keep only the feature of interest from the attributes field. To start training and testing the model, use the classify tab from the top menu. Here, we can select the classifier to be used to perform the classification. The dataset can be set to be split into 70% for training and the remaining for testing. Upon starting the training, Weka will produce the result of the classification indicating its precision for each device as well as its average precision. Upon obtaining results, feature sets can be fine-tuned at the “Select Attributes” tab. This iteration is repeated to obtain the most essential feature set to build the feature model.

5.0 FEATURE SELECTION

This section explains the feature selection process for building the machine learning model which will be used to classify network devices into their respective types. As network traffic classification is being used to determine the device that exists within the network, features of network data packets are being observed on per packet basis in order to identify the most useful feature.

5.1 Payload Feature

From the works done by Shen, et al. (2017) and Pedro and Luis (2017), payload level inspection is used to build the machine learning model and classification is done on packet payload. At the current state of technology where security issue is a huge concern, these methods are being limited by encryption algorithm used by device manufacturers to convey information within the network from device to device. When encryption is used, payload information immediately becomes unreadable rendering their works limited to device which does not use encryption. When encryption is applied, only those devices that are able to be decrypted by standard decryption tools such as OpenSSL can be used. For this reason, this feature is excluded in the project as future data packets will adopt mode complex encryption algorithm for security purposes especially in the field of IoT.

5.2 Packet Feature

Packet features are network packet characteristics that can be used to distinguish packets from different devices which can be found in the header itself. Unlike payload feature, it is not affected by encryption as most information stored in the packet header is required to be readily processed by network routers for routing purposes. However, not all packet header information is required to build the machine learning model as many of them only provide information about network performance instead of their originating device such as roundtrip time, time to live and header checksum. The device source and destination IP and MAC address is also not used because this can be easily spoofed for security purposes like what is being implemented in Windows 10. Finally, we are left with the lengths and arrival time. From these information, statistical features can be crafted to provide a better picture of the differences of each packet sent by their respective device.

From the remaining information, it is possible to induce statistical features which will be used to build the machine learning model. From the lengths information, it is possible to find out the total packet length, the header length and the payload length. From the arrival time, it is also possible to find out interarrival times of each packet. When combined, more complex feature is possible. For example, number of packets sent or received in one second and the average payload size of packets sent or received so far within one second. As a result, we have a feature set which is comprised of interarrival time, total packet length, header length, payload length, packet count in one second and average payload length so far in one second.

5.3 Feature Improvement

In order to reduce the workload of the classification process, only essential features are being used to perform the classification. To do so, we use the "Select Attribute" feature of Weka. In this context, we use "CfsSubsetEval" as our attribute evaluator as this method attempts to identify a subset of features which are most closely related to the class used which in our case is the device type.

6.0 RESULT ANALYSIS

The accuracy of the system is defined by the average accuracy in classifying all devices within the network. For each device, the accuracy is determined by the True Positive (TP) rate in classifying the respective device. False Positive (FP) rate marks the proportion of falsely classified instances which were reported to be positive. For instance, these packets are classified as x but is actually part of y. For the purpose of this project, it is not acceptable that FP rate is high. This is because information used for network monitoring must always be true or else it will lead to ineffective network performance improvement efforts. It is also important that TP rate is high as it indicates a truly classified instance.

```
Time taken to build model: 4.31 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.42 seconds


=== Summary ===

Correctly Classified Instances        4209               93.3466 %
Incorrectly Classified Instances       300                6.6534 %
Kappa statistic                          0.8989
Mean absolute error                      0.0342
Root mean squared error                  0.1305
Relative absolute error                 15.5002 %
Root relative squared error             39.2832 %
Total Number of Instances             4509

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0.793    0.005    0.882      0.793    0.835      0.828    0.986     0.904     socket
               0.813    0.011    0.706      0.813    0.756      0.750    0.980     0.830     remote
               0.975    0.001    0.995      0.975    0.985      0.983    0.999     0.995     bulb
               0.966    0.060    0.941      0.966    0.954      0.906    0.989     0.985     ghome
               0.887    0.027    0.921      0.887    0.903      0.871    0.984     0.966     echo
               0.957    0.000    0.987      0.957    0.972      0.971    0.997     0.992     airpurifier
Weighted Avg.  0.933    0.038    0.934      0.933    0.933      0.900    0.988     0.973

=== Confusion Matrix ===

   a    b    c    d    e    f   <-- classified as
 172   43    0    1    1    0 |    a = socket
  23  113    2    1    0    0 |    b = remote
   0    2  550    2   10    0 |    c = bulb
   0    0    0 2184   75    1 |    d = ghome
   0    0    1  130 1033    1 |    e = echo
   0    2    0    2    3  157 |    f = airpurifier
```

Figure 6-0-1: Classification result

Based on figure 6-0-1, it can be observed that the selected feature set successfully obtained a classification precision of 93.4% on average. With this set of features, 4.31 seconds is used to train the model while 0.42 seconds are used to perform classification on the test set. This is considered quite high in terms of device classification.

```
=== Attribute Selection on all input data ===

Search Method:
        Best first.
        Start set: no attributes
        Search direction: forward
        Stale search after 5 node expansions
        Total number of subsets evaluated: 22
        Merit of best subset found:    0.479

Attribute Subset Evaluator (supervised, Class (nominal): 1 device):
        CFS Subset Evaluator
        Including locally predictive attributes

Selected attributes: 3,4,5,7 : 4
                      total
                      header
                      payload
                      sMeanLoad
```

Figure 6-0-2: Select attribute result

As mentioned in chapter 5, we attempt to improve the classification time by reducing the number features used to classify the data packets. According to figure 6-0-2, we ran the attribute evaluator function of Weka and found that only total packet length, header length, payload length and the average payload length so far in one second is more essential relative to our class which is the device type.

```
Time taken to build model: 3.29 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.38 seconds

=== Summary ===

Correctly Classified Instances        4104               91.018  %
Incorrectly Classified Instances       405                8.982  %
Kappa statistic                          0.8637
Mean absolute error                      0.0361
Root mean squared error                  0.1431
Relative absolute error                 16.3402 %
Root relative squared error             43.0674 %
Total Number of Instances             4509

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.447    0.002    0.924      0.447    0.602      0.632    0.988     0.837     socket
                 0.978    0.028    0.529      0.978    0.687      0.709    0.990     0.705     remote
                 0.977    0.004    0.970      0.977    0.973      0.970    0.996     0.990     bulb
                 0.958    0.070    0.932      0.958    0.945      0.889    0.980     0.971     ghome
                 0.858    0.029    0.912      0.858    0.884      0.846    0.969     0.945     echo
                 0.951    0.001    0.963      0.951    0.957      0.955    1.000     0.993     airpurifier
Weighted Avg.    0.910    0.044    0.920      0.910    0.909      0.872    0.981     0.953

=== Confusion Matrix ===

    a    b    c    d    e    f   <-- classified as
   97  118    0    0    2    0 |    a = socket
    0  136    3    0    0    0 |    b = remote
    0    1  551    2   10    0 |    c = bulb
    4    0    5 2165   81    5 |    d = ghome
    4    1    6  154  999    1 |    e = echo
    0    1    3    1    3  156 |    f = airpurifier
```

Figure 6-0-3: Classification result after removing non-useful feature

Based on figure 6-0-3, it is shown that the classification precision of the model is high enough that removing non-essential features from the model only have the slightest effect on the classification precision of the model. However, the time taken to build the model has been reduced to only 3.29 seconds while the time taken to perform classification on test set is reduced to 0.38 seconds. This might not seem much on paper but it will have a significant impact on large networks with over hundreds of IoT devices.

7.0 CONCLUSION

In the current stage of IoT development, many platforms still lack functionality that other platform can provide. For example, Amazon Alexa is able to make phone calls while Google Home is not able to do so. XiaoMi Home has support from very affordable electronics such as the Phillips Smart LED bulb which is much cheaper than Phillips Hue which supports Google Home. There is also platform such as the Apple Home that is limited to its Apple ecosystem only. Users who wish to customize the integration of IoT device to suit their needs and desires will often need to integrate more than one platform.

The adoption of multiple platform makes network management more challenging especially for power users who wish to optimize their network performance based on the devices on the network. In order to obtain device type information for IoT devices, accessing the router interface does not suffice as router interface only shows the device name and IP address. Devices with distinguishable names can be used to determine the device type. However, IoT devices often use code names as device name which makes it more indistinguishable. It is required to access each platform separately using the native application of the platform in order to accurately determine the device type of each IoT device.

The solution to this issue is to have a universal IoT devices discovery and identification system using protocol-independent network flows characteristics. This allows users to quickly identify the total number of IoT devices in the network without needing to check separate application. This makes network device management a bit easier. Using this system also provides statistics such as number of devices of a certain type that exists in the network as the system is able to classify devices based on its type such as a bulb, a media streamer or a plug.

REFERENCES

Bartlett, G., Heidemann, J. and Papadopoulos, C., 2007, October. Understanding passive and active service discovery. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (pp. 57-70). ACM.

Rana, A., 2014. What is AMap and how does it fingerprint applications. SANS Institute.

Ko K., Kang P., Sim W. (2007) Design of Hybrid Network Discovery Module for Detecting Client Applications and ActiveX Controls. In: Gervasi O., Gavrilova M.L. (eds) Computational Science and Its Applications – ICCSA 2007. ICCSA 2007. Lecture Notes in Computer Science, vol 4706. Springer, Berlin, Heidelberg

Ghanem, W.A.H. and Belaton, B., 2013, November. Improving accuracy of applications fingerprinting on local networks using NMAP-AMAP-ETTERCAP as a hybrid framework. In Control System, Computing and Engineering (ICCSCE), 2013 IEEE International Conference on (pp. 403-407). IEEE.

Jin Mitsugi, Yuki Sato, Miyuki Ozawa & Shigeya Suzuki 2014, 'An Integrated Device and Service Discovery with UPnP and ONS to Facilitate the Composition of Smart Home Applications', Paper presented at *IEEE World Forum on Internet of Things (WF-IoT),* IEEE, Fujisawa, Japan

Shen, J., Li, Y., Li, B., Chen, H. and Li, J., 2017, June. IoT Eye An Efficient System for Dynamic IoT Devices Auto-discovery on Organization Level. In Cyber Security and Cloud Computing (CSCloud), 2017 IEEE 4th International Conference on (pp. 294-299). IEEE.

Lionel Metongnon, Eugene C. Ezin, Ramin Sadre, 'Efficient Probing of Heterogeneous IoT Networks', Paper presented at *3rd International Workshop on Security for Emerging Distributed Network Technolgies,* IEEE.

Pedro R. J. & Luis Nunes, 2017, June. 'Automatic Discovery and Classification of IoT', Paper presented at *2017 12th Iberian Conference on Information Systems and Technologies (CISTI),* IEEE.