Addis Ababa University
College of Natural Sciences

Anomaly Based Peer-to-Peer Botnet Detection Using Fuzzy-Neuro Network

Tewodros Worku Osman

A Thesis Submitted to the Department of Computer Science in Partial Fulfilment for the Degree of Master of Science in Computer Science

Addis Ababa, Ethiopia

October 2020

Addis Ababa University

College of Natural Sciences

Tewodros Worku Osman

Advisor: Solomon Gizaw (PhD)

This is to certify that the thesis prepared by Tewodros Worku, titled: *Anomaly Based Peer-to-Peer Botnet Detection Using Fuzzy-Neuro Network* and submitted in partial fulfilment of the requirements for the Degree of Master of Science in Computer Science complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the Examining Committee:

	Name	Signature	Date
Advisor:	Solomon Gizaw (PhD)	_____	_____
Examiner:	Mulugeta Libsie (PhD)	_____	_____
Examiner:	Dida Midekso (PhD)	_____	_____

Abstract

Peer-to-Peer (P2P) botnets are considered as one of the most significant contributors to various malicious activities on the Internet. The denial of service attacks, spamming, keylogging, click fraud, traffic sniffing, stealing personal user information, for example credit card numbers, and social security numbers, are some of the illegal activities based on botnets. P2P botnets are networks of infected computing devices, called zombies or bots. These bots are remotely controlled and instructed by malicious entities commonly referred to as Botmasters or hackers. In recent years, lots of researchers have proposed a number of P2P botnet detection models, but due to the evolving nature of botnets, there is still a need for new techniques to identify recent botnets. Due to that, we propose a model that is able to distinguish genuine network traffic from malicious one by analyzing the network flow data using Fuzzy-Neuro Network (FNN).

The proposed model has the following components: Feature Extractor, Feature Selector, Dataset Constructor, Preprocessor, Classifier and P2P Botnet Detector. The feature extraction component extracts the network traffic-based feature vectors from the network traffic whereas the feature selection component selects vital features based on their information gain value. The next component which is the dataset constructor is used to convert the comma separated value (CSV) file into sets and help us to split the dataset as training (70%) and testing (30%) sets. Then, the major activities in the preprocessing component are data cleaning, data transformation and data reduction. Finally, the FNN classifier is utilized to classify the network traffic into P2P botnet and normal using the botnet detection module.

The feasibility of our proposed model has been validated through experiments using network traffic records acquired from two publicly available P2P botnet datasets Bot-IoT and UNSW-NB15. The datasets include both genuine and malicious network traffic. The evaluation result shows the proposed model is effective in detecting P2P botnets. Based on the evaluation results of our classifier, using Bot-IoT dataset, the model scored 100% for all evaluation metrics. Whereas, using the UNSW-NB15 dataset, the model scored highest classification accuracy of 99.9%, precision of 99.9% and recall of 100% with F-measure rate of 99.9%.

Keywords: P2P Botnet Detection, Classification, Fuzzy-Neuro Network, Anomaly Detection

Dedication

- -

To my family for their love and support.

- -

Acknowledgments

First and foremost, I would like to thank God for giving me the opportunity, knowledge and strength to undertake and complete this research work. Without His blessings, each and every success in my life would not have been possible.

Then, I would like to offer profound gratitude to my advisor, Dr. Solomon Gizaw, whose expertise was invaluable in guiding this research work. Your insightful feedback, guidance and support pushed me to sharpen my thinking and brought my work to a higher level.

Last but not least, I wish to acknowledge the prayer, great love and support of my family. They kept me going on and this research work would not have been possible without their continuous support and patience.

Table of Contents Pages

List of Tables

List of Figures

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
C&C	Command and Control
CICFlowMeter	Canadian Institute for Cybersecurity Flow Meter
CNN	Convolutional Neural Network
CSV	Comma Separated Values
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DT	Decision Tree
FCM	Fuzzy C-means
FN	False Negatives
FNN	Fuzzy-Neuro Network
FP	False Positives
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ICQ	I Seek You
IG	Information Gain
IoT	Internet of Things
IRC	Internet Relay Chat
ISS	Internet Security Systems
ML	Machine Learning
NB	Naïve Bayesian
P2P	Peer to Peer

PCAP	Packet Capture
POP3S	Post Office Protocol Version 3 Secured
RNN	Recurrent Neural Network
SSL	Secure Sockets Layer
SVM	Support Vector Machine
TFTP	Trivial File Transfer Protocol
TN	True Negatives
TP	True Positive
UDP	User Datagram Protocol
UNSW	University of New South Wales
Vc	Vectors

Chapter 1: Introduction

1.1 Background

In recent years, botnet led attacks have become serious threats on the Internet. The botnets are considered as one of the most significant contributors to various malicious activities on the Internet today [1]. Botnets are networks of infected computing devices, called bots or zombies [2]. These bots are remotely controlled and instructed to conduct criminal activities by malicious entities commonly referred to as Botmasters or hackers [3].

The botnets are used for various illegal activities such as distributed denial of service (DDoS) attacks, spamming, keylogging, identity theft, pay-per-click fraud, traffic sniffing and botnet spread [4, 5]. Botmasters are beneficial from these illegal activities by compromising network of computers without the knowledge of the owner by stealing sensitive information, perform variety of attacks on other machine(s) [4].

Botnets can be classified as centralized (Command and Control (C&C)) and decentralized (Peer-to-Peer (P2P)) [1].

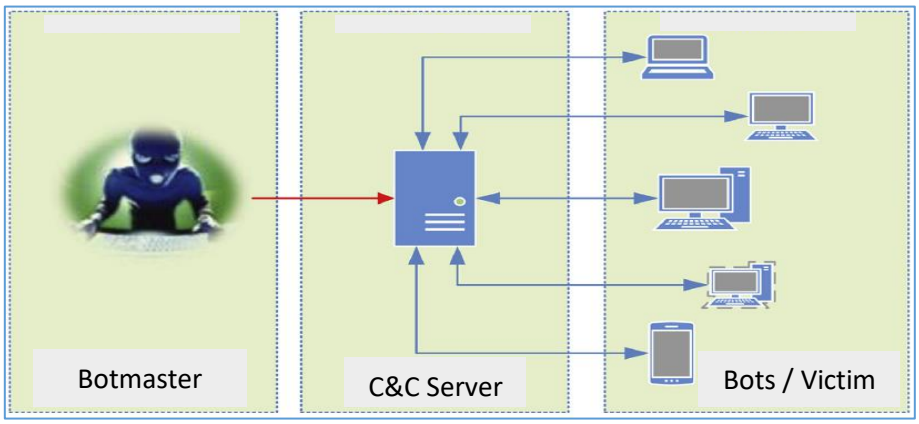

Figure 1.1: *C&C Botnet Operation*

Previously, Command and Control botnets were dominating the cybercrime world [6]. A command and control client server is established and all the bots are connected to that server. The Botmaster sends instructions to the C&C server and the bots perform malicious actions by executing commands received from the C&C server [7].

Figure 1.1 [8] illustrates the C&C botnet operation between the Botmasters and the infected hosts. The server remotely controls the bots for giving commands and queries and those commands are executed and replied by the zombies [9].

Currently, Peer-to-Peer botnet came into picture. As shown in Figure 1.2 [10], the hacker communicates with each bot and the bots can communicate each other. It is basically a decentralized structure in which every bot behaves as a client as well as a server [11]. There is a list of neighbors with every bot and the bot (behaving as a server) circulates command to every bot in its neighbor list [9]. Due to the distributed nature of P2P botnets, they are comparatively very difficult to detect [1].

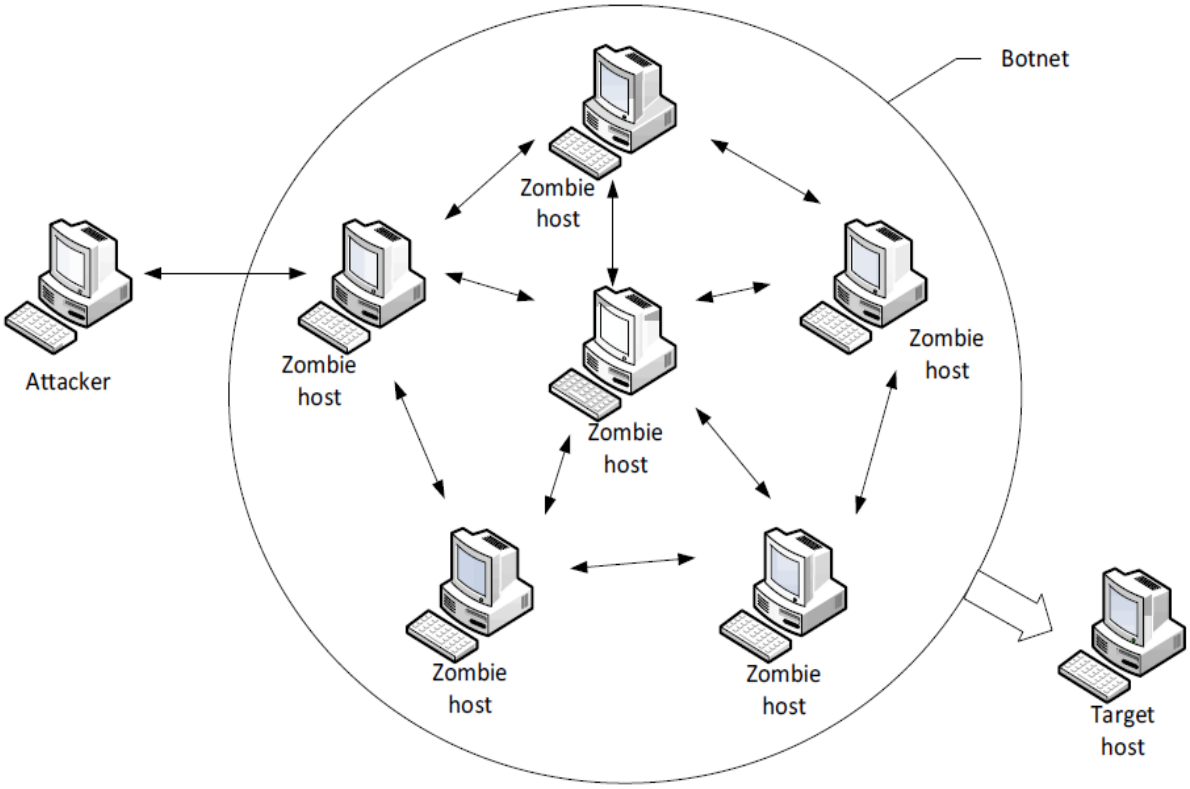

Figure 1.2: *P2P Botnet Overview*

Botnet detection techniques can be categorized as signature-based, domain name system based (DNS-based), mining-based and anomaly-based [12]. Signature-based approaches identify the known characteristics of botnets from signature database [13]. It is suitable for real-time detection, but fail to identify new types of botnets since they can only detect known threats [7].

DNS-based botnet detection is a kind of botnet detection method which utilizes DNS-related network traffics to determine whether the machines involved are infected with a botnet or not [14]. DNS based techniques work with information collected from DNS queries [12]. Whereas, data mining-based botnet detection technique aims to recognize useful patterns to discover regularities and irregularities in large datasets [15].

Anomaly-based approaches try to discover anomalous network behaviors, such as high network latency and activities on rarely used ports [12]. Anomaly-based techniques have become the main research area in botnet detection [16] due to the limitations of signature-based methods, such as the difficulty in keeping the signature database up to date, and their inability to detect previously unseen botnets [13].

The anomaly-based approaches are further classified as host-based and network-based [17]. With the host-based technique, it identifies abnormal computer usage (e.g., increased CPU usage and memory usage). The drawback is that the resource usage of all end hosts needs to be monitored which can be quite costly in terms of time [7].

In contrast, the network-based technique attempts to find patterns in network traffic flow which are distinguishable from normal flow [17]. This method analyzes network traffic looking for suspicious patterns in the behavior of multiple hosts [16].

Although researchers in [6, 8, 11, 13, 18] have proposed a number of botnet detection models, most of them were not able to fully detect the recent generation of botnets.

1.2 Motivation

Recently, P2P botnets have become the biggest threat to cyber security and have been used as an infrastructure to carry out nearly every type of cyber attack. As a result, P2P botnets are in continuous research attention as they severely attack businesses, services, and operations. Hence, it is essential to look for solutions to monitor the network so as to detect P2P botnet attacks. P2P botnet detection techniques have become the best solutions for monitoring computer networks for illegal malicious activities.

This work is motivated based on the following two issues: low detection rate and relatively low performance of existing P2P botnet detection techniques. In general, due to the evolving

nature of the botnets they cannot be dealt with by existing detection techniques and improvements are required. Therefore, our study has a meaningful significance on the cyber world.

1.3 Statement of the Problem

Today, P2P botnets have been identified as one of the most serious threats against network security due to their distributed and evolving nature [1]. This makes detection of P2P botnets more difficult as compared to centralized botnets. Based on the literature review, lots of researchers have proposed a number of P2P botnet detection methods, but due to the evolving nature of P2P botnets, there is still a need for new techniques to detect recent botnets and improvements are needed.

In the literature [6, 8, 11, 13, 18], different methods have been proposed to detect the presence of a P2P botnet in a network by analyzing and classifying the network traffic. Among those methods, Artificial Neural Network (ANN) becomes the most widely used approach to detect P2P botnets [13]. These days, fuzzy logic approach has got much attention in different fields of study such as intrusion detection and prevention systems [19], antivirus detection [20] and P2P botnet detection [21]. However, the existing P2P botnet detection methods still have their own shortcomings such as low performance and relatively low detection rate.

To overcome these challenges, in recent years, researchers have proposed various P2P botnet detection methods. But there is still a need for further improvement in these methods [8, 13, 18] concerning performance and detection rate. Thus, this study proposes applying Fuzzy-Neuro Network (FNN) which is the hybrid of fuzzy logic and ANN into P2P botnet detection. The hybrid system will combine the capabilities of neural networks and fuzzy computations. Therefore, the proposed research work focuses on developing a P2P botnet detection model with improved performance and better detection rate.

1.4 Objectives

General Objective

The general objective of this research work is to design a model based on the anomaly approach to detect P2P botnets using Fuzzy-Neuro Network approach.

Specific Objectives

To achieve the general objective of the research work, the following specific objectives are devised:

- Explore latest P2P botnet datasets which will be used to train and test our model.
- Extract and select the best features from the datasets for model training and testing.
- Preprocess the features to prepare them for experimentation.
- Design and train the model using Fuzzy-Neuro Network approach.
- Conduct experiments to test and evaluate the performance of the model.
- Compare the performance with existing P2P botnet detection approaches.

1.5 Methods

In order to achieve the general and specific objectives of this study, we use the following methods.

- **Literature Review**

  We will perform a set of literature reviews from articles, journals, conferences and books to understand more about peer-to-peer botnets and the state-of-the-art detection approaches.

- **Data Collection**

  To build and implement the proposed model datasets will be collected. The datasets must have latest malicious and normal less redundant network traffic records.

- **Developmental Environments and Tools**

  We will use different free and open-source tools to develop our model. Python programming language with TensorFlow will be used to construct the system. The implementation will be done in Anaconda using Jupyter Notebook.

- **Experimental Evaluation**

  We will evaluate the performance of the proposed solution using different evaluation metrics such as accuracy, precision, recall and F-measure.

1.6 Scope and Limitations

- **Scope**

  The research mainly focuses on detecting the presence of P2P botnet attacks in the network. The aim is to be able to distinguish genuine network traffic from malicious one by analyzing the network flow data of the P2P botnets presence in the network.

- **Limitations**

  In this research, we will not study the technique to prevent or remove P2P botnet attacks in the network.

1.7 Application of Results

The possible applications of this research are:

- To identify networks of illegally controlled computers or P2P botnets that are employed for malicious activities.
- To detect cyber based P2P botnet attacks that the attackers seek to gain access to and then reside on a targeted network for an extended period of time.

1.8 Organization of the Rest of the Thesis

The rest of this thesis is organized as follows. Chapter Two presents and describes the review of literatures related to P2P botnet. Chapter Three presents related works which have been done on P2P botnet detection. The Fourth Chapter deals with our proposed system which is the design of anomaly-based P2P botnet detection model. In Chapter Five, experimentation and evaluation result analysis is presented. Finally, Chapter Six addresses conclusion, contribution and direction for future work.

Chapter 2: Literature Review

This Chapter gives a brief description of the major issues of botnet lead attacks on the Internet. Nowadays, botnet attacks constitute a major portion of cyber crimes. In this Chapter, overview and lifecycle of botnets, and the architecture of botnets are presented. In addition to that, the type of botnet led attacks as well as currently applied botnet detection techniques are discussed. Furthermore, the common classification algorithms, feature selection techniques and evaluation metrics are presented. Extensive literatures have been reviewed to understand the problem associated to the realm of this thesis and also to identify appropriate solution.

2.1 Botnets

2.1.1 Overview of Botnets

As we discussed, botnets are a set of Internet based computers under a common controller. Although the term can include legitimate networks of computers, the overwhelming use of the term is for computers that have been hacked and under the control of criminal hackers [22]. Botnets are typically formed through a variety of illicit means. A bot herder may have systems randomly scanning the Internet for systems with unpatched vulnerabilities that allow for remote hacking. If a vulnerable system is found, it is hacked and the botnet software is installed. Phishing messages can also lure naive users into downloading malicious software that adds the system to a botnet [22].

The reason why botnet has become a preferred tool for cyber criminals is due to its multilayer architecture. Botmasters don't control the bots directly or physically. They do it through the command and control servers. Thus, the Botmaster remains hidden and does all the activities remotely. Moreover, all the bots in a botnet, as well as the Botmaster, may not be located in the same place. The bots as well as the Botmaster may be physically dispersed and may be located in various locations around the globe. As the network can span various countries, therefore differences in boundaries, time zones, languages, cyber laws, etc. makes it very

difficult to track bot activities across international boundaries [23]. These reasons make detection scenario of botnets very bleak and provide safe heavens for the hackers.

Given the pervasiveness of botnets, it can be expected that almost all companies, universities, and other organizations will have some of their systems herded into a botnet. If an organization does not monitor its systems and networks properly, it could be an unknowing complicit in attacking other organizations [22].

2.1.2 Lifecycle of Botnets

As shown in Figure 2.1 [24], typical botnet is designed and developed by using five different phases. These are initial injection, secondary injection, connection phase, malicious C&C and upgrading/maintenance phase [25]. Botmaster follows this cycle to create, infect and control the bots.

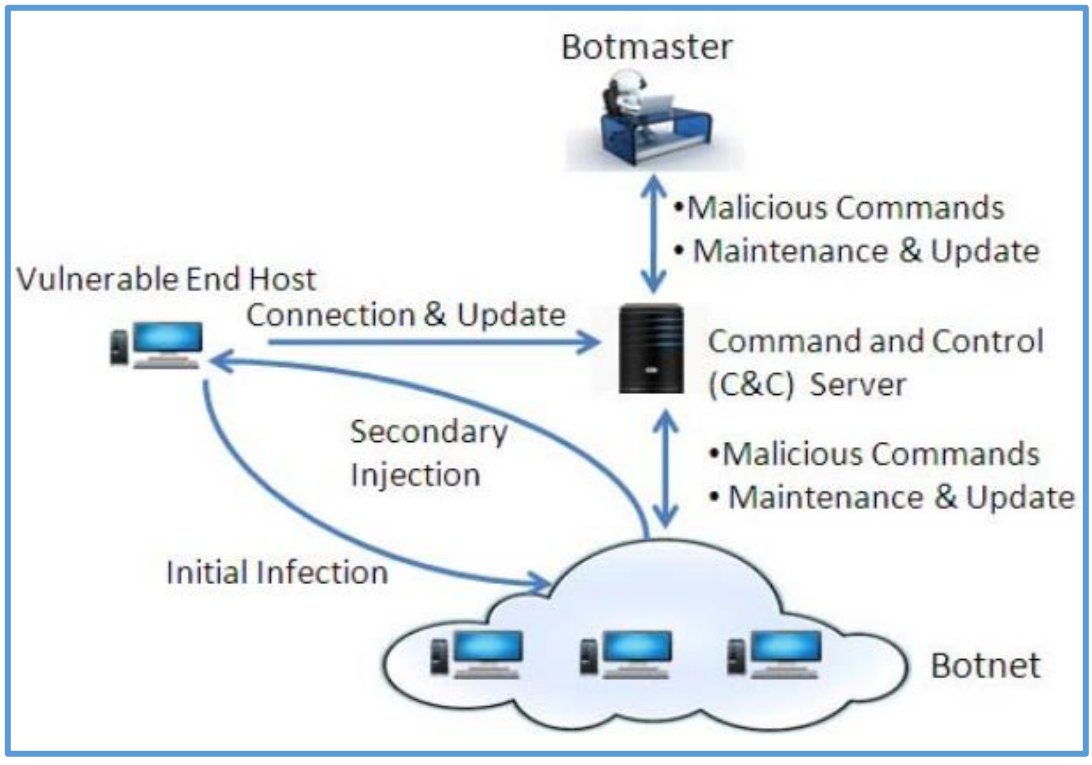

Figure 2.1: *Botnet Lifecycle*

1. Initial Infection

In the first step, an attacker searches for vulnerable hosts and infects them using various exploitation techniques like sending spam emails, phishing, creating backdoors, etc. The first

task of a Botmaster is to make the infected machines as a part of botnet and it must know the address of the command and control server [25].

2. Secondary Injection

After the successful initial infection, the next step is to download and run the botnet code to become a bot, under the control of a specific Botmaster. This method can be processed by using Trivial File Transfer Protocol (TFTP), File Transfer Protocol (FTP), and Hyper Text Transfer Protocol (HTTP) [24].

3. Connection

The main phase of Botnet Life cycle is establishing a connection where infected machines are connecting with command and control server to receive commands from Botmaster. In this phase, bots receive commands from Botmaster and send reply to command and control server. The Botmaster restarts all the bots to check activeness of the bots and to make sure that bots are able to receive commands [25].

4. Malicious Command and Control

When the fourth phase begins, the attacker is able to send commands to the bots through the C&C server to execute it in the victim's machine. Actual working of botnet starts from command and control server. Attacker controls zombie army or bots from common C&C server. All infected machines need to know the address of a C&C server while downloading the script, so that they can easily communicate with the server [25].

5. Maintenance and Update

The last phase in the botnet's lifecycle is the botnet maintenance and update phase. This phase is necessary to control all networks and is also a vulnerable phase to save the C&C server from detection by changing its location after some period of time. In this phase, Botmaster maintains an overall working of active bots and infects new host machines to increase the number of bots [25].

2.1.3 A Brief History of Botnets

Botnets were first time seen in late 1980s and early 1990s with the introduction of Internet Relay Chat (IRC) and were not used for any malicious purposes. The first malicious botnet was seen in 1998 named Global Threat Bot using IRC as its C&C channel [26]. This Botnet is primarily used for DDoS attack. Later in 1999, botnet programs like Sub7 (trojan) and Pretty Park (worm) entered into IRC network. These malwares, once installed wait for malicious commands from the server. Over time botnets were mitigated out of the IRC and started using HTTP, Secure Sockets Layer (SSL), Internet Control Message Protocol (ICMP) and P2P networks. In 2007, the biggest botnet of all time emerged called the "Storm", using P2P network as its C&C infrastructure [25]. It infected about 50 million computers and was used for many crimes such as spamming, password stealing, identity theft etc. Later in 2014, a botnet named "Stuxnet" affected the nuclear facility of Iran. It was very advanced and could control organizations. It is believed that foreign authorities were behind the attack, because of the advanced design of the malware [27].

2.2 Architecture of Botnets

Despite the countless variations and motivations behind individual botnets, the architectures governing these botnets generally fall into three categories: centralized (C&C), decentralized (P2P) and hybrid [28].

2.2.1 Centralized (C&C)

The first and simplest botnet architecture is the centralized approach [28]. In this architecture the bots receive commands from a centralized C&C server, often using protocols like HTTP or IRC to conduct communications. These botnets are usually efficient and easy to set up.

There are serious drawbacks to this approach from the perspective of the Botmaster. While a centralized approach means efficient communications from the C&C server to the infected machines, it also means that there is a single point of failure for the botnet. In other words, if authorities or other users can locate this C&C server, it can be shut down completely, or even taken over to strengthen a rival botnet [28]. This topology can be further sub-divided into star topologies, in which each bot is connected directly with the C&C server, and a hierarchical

topology, wherein multiple proxies are used between the Botmaster C&C server and the bots to add an extra layer of obfuscation [29].

2.2.2 Decentralized (P2P)

Although the central botnet has the advantages of being simple, efficient and cooperative, the control flow has a central single point of failure. Once the central command and control server is closed, the zombie host will not be able to continue to obtain the command, and the attacker will lose control of the zombie host [29].

To further improve the resilience against take down or network failure a P2P topology can be used. This topology consists only of bots where every bot can be a potential C&C server or acts as both the client and the server [30]. Every bot is connected to at least one other bot and commands can only reach the whole botnet if every bot has the ability to relay the commands to directly connected bots. Unlike its centralized predecessors, a P2P botnet does not rely on a single point of failure [28].

Although the P2P botnet command delay is higher than the central structure, it does not rely on the characteristics of vulnerable central nodes, making it difficult to be hijacked, measured and closed. The advantage of using this method is that peer to peer communications are harder to detect than the centralized communications. In addition, if one bot is discovered, this will not have implications on the remaining bots [29].

2.2.3 Hybrid Botnets

Finally, some botnets take an approach with elements of both centralized and P2P architectures as shown in Figure 2.2 [28]. While P2P topologies certainly improved botnet resilience and anonymity, applying such protocols across large networks risked being disrupted if parts of the network were disabled or suffered poor command transmission and peer locating [28]. These downsides could be mitigated using a hybrid botnet. Such a botnet was proposed for which multiple servant bots take on the roles of C&C servers to communicate commands to the client bots, making communicating these commands across the network quicker and more efficient while preserving resilience and anonymity if a bot is

seized [30]. Hybrid approaches are the newest and likely the most efficient topology, so it is likely that modern botnets will tend towards this structure in the near future.

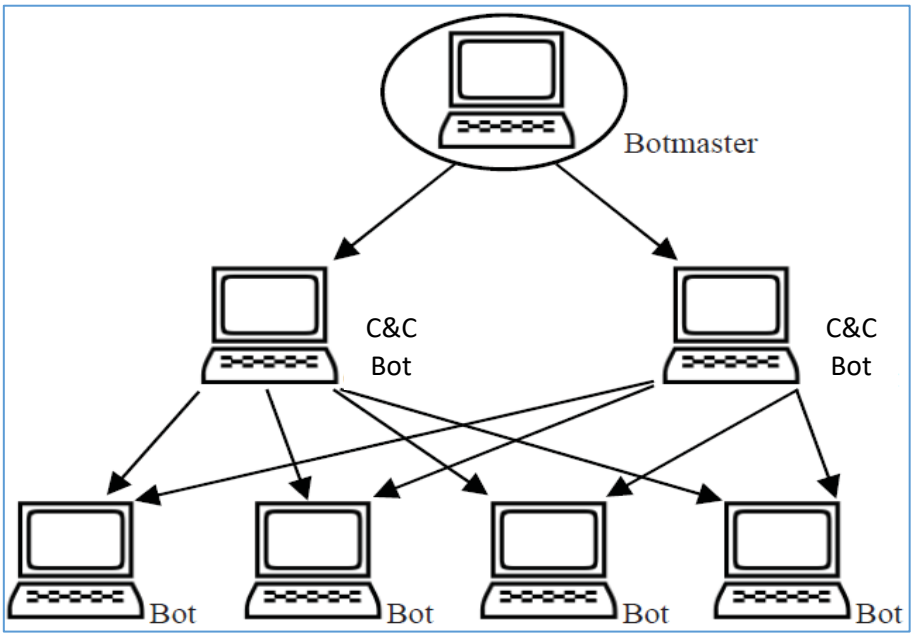

Figure 2.2: *Hybrid Botnet Architecture*

2.3 Types of Botnet Attacks

The cybercriminal uses the botnets for the purposes of carrying out malicious activities. The botnets are used for many online crimes such as distributed denial of service attacks, spamming, keylogging, identity theft, pay-per-click fraud, traffic sniffing and botnet spread [5].

2.3.1 Distributed Denial of Service

One of the greatest dangers posed by botnets is their ability to perform massive Distributed Denial of Service attacks against practically any target accessible from the Internet. In a DDoS attack, an attacker sends massive numbers of packets to a victim through the controller as shown in Figure 2.3 [31]. These packets will overwhelm the victim's ability to properly process these communications, thus blocking anyone from using the victim's networks. One technique to counter this is to simply block incoming packets coming from the attacker's IP address and consequently nullify the attack. However, although the IP addresses of a few attacking devices could simply be blocked by the victim in a DDoS attack, the sheer number

of devices with unique addresses in a botnet DDoS attack can make it infeasible for a victim to manually block the malicious traffic based on source address alone [28].

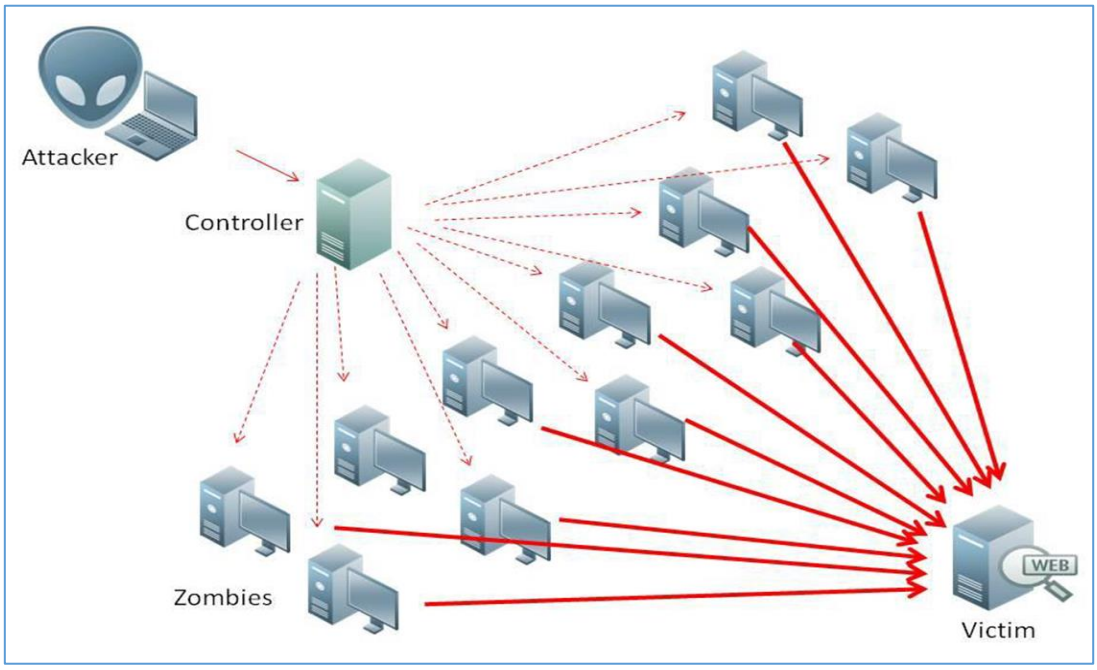

Figure 2.3: *Distributed Denial of Service Attack Overview*

Moreover, there are variations of DDoS attacks which can further complicate the task of defending against them. The DDoS attacks can be further divided into direct attacks and reflected attacks [32]. In a direct attack, the victim is overwhelmed by large number of packets sent directly by the attacker. Types of attack packets may be TCP, ICMP, User Datagram Protocol (UDP), or a mixture of them. A reflected attack is performed by using mediators like routers, called *reflectors*, that are purposely implemented as attack launchers. In this attack, each bot spoofs the source IP address of the victim machine and requests packets from other sources using that address, which subsequently overwhelm it, making tracking down the attacking machines even more difficult. The protocol which is used to conduct these attacks varies, although in one relatively recent study it was found that HTTP is overwhelmingly the most common protocol used, followed by TCP and UDP [33].

The effects of a DDoS attack can be severe; even a short attack can cause a victim to lose significant amounts of capital. In one study measuring the impact of DDoS attacks stemming from botnets, researchers followed two botnets and quantified the impacts upon their victims

[34]. The study showed that over 65% of victims were severely affected. Although the attacks themselves were relatively short, they often were accompanied with extortionary threats. If a business is victimized by a DDoS attack, its customers may be impatient for the web resources to return, or they may lose confidence in the victim's competence in cybersecurity and subsequently go elsewhere for those resources. Especially in the case of larger businesses, a successful DDoS attack can cost millions of dollars [35].

2.3.2 Spamming

A bot can be used as a sniffer to identify the presence of sensitive data in the infected machines or zombies. It can also locate competitor botnets if installed in the same machine and can be hijacked by the commander. Some bots may offer to open a SOCKS which is an Internet proxy protocol for TCP/IP that exchanges network packets between a client and server through a proxy server. When the SOCKS proxy is enabled on a compromised machine, it can be used for various purposes like spamming [36]. Bots use a packet sniffer to watch for the information or data passed by the compromised machine. The sniffer can retrieve sensitive information such as a username and password. With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of bulk email (spam). Some bots also implement a special function to harvest email-addresses [37].

One example of a notorious recent spam botnet is the Necurs botnet, which accounted for an astounding 60% of spam in the last quarter of 2017 [38]. Necurs first appeared in 2012 and continued to grow to encompass between five and six million infected machines, making it the largest spam botnet in history [39]. It has continued to evolve new capabilities including the ability to launch DDoS attacks and has demonstrated extreme resilience and anonymity.

2.3.3 Keylogging

Keystroke logging, or keylogging, is the practice of recording the keys a person types on a keyboard. With the help of keylogger, it becomes easy for a Botmaster to retrieve sensitive information and steal data [40]. If the compromised machine uses encrypted communication channels (e.g., Hyper Text Transfer Protocol Secure (HTTPS) or Post Office Protocol Version 3 Secured (POP3S)), then just sniffing the network packets on the victim's computer is useless since the appropriate key to decrypt the packets is missing. But most bots also offer features

to help in this situation. Using a keylogger program, an attacker can gather only the keys typed that come in the sequence of interesting words like PayPal, Yahoo, etc. If we suppose that this keylogger runs on thousands of compromised machines in parallel, we can imagine how quickly accounts can be harvested [41].

2.3.4 Identity Fraud

Identity fraud is a fast-growing crime on the Internet. This technique is often called 'identity theft', because it enables Botmasters to impersonate the victim, making further actions, like fraud, possible [24]. Different kinds of bots can be mixed to perform large-scale identity theft. A major use of identity fraud botnet operation is for the automatic extraction of user data and credentials from infected hosts. Bots can be used to appear as a legitimate company and ask the user to submit personal details like e-mail accounts, web shop credentials, social network accounts, bank account password, credit card details, taxation details, etc. [42]. Mass identity theft can be performed using phishing emails that trick victims into entering login credentials on websites like eBay, Amazon, or even their banks. These fake emails are generated and sent by bots via their spamming mechanism. Just as quickly as one of these fake sites is shut down, another one can pop up. In addition, keylogging and sniffing of traffic can also be used for identity theft [43].

2.3.5 Pay-Per-Click Abuse

Yet another illicit activity in which botnets excel is pay-per-click abuse which is also called click fraud [28]. Due to the large number of users who spend considerable time on the Internet, it is no surprise that advertising products on websites is a lucrative method of gaining new customers. Ad brokers pay substantial amounts of money to advertisers to display their ads based on the number of users who click or view that advertisement. Botnets are perfect for this task, as they can trigger ad clicks from distributed locations, making this action extremely difficult to catch [44]. Google's AdSense program allows websites to display Google advertisements and thereby earn money from them. Google pays money to the website owners on the basis of the number of clicks their advertisements gather. Compromised machines are used to automatically click on a site, inflating the number of clicks sent to the company with the advertisement [45].

2.3.6 Traffic Sniffing

Bots can also use a packet sniffer to watch for interesting clear-text data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords [43]. But the sniffed data can also contain other interesting information. If a machine is compromised more than once and also a member of more than one botnet, the packet sniffing allows to gather the key information of the other botnet. Thus, it is possible to "steal" another botnet [46].

2.3.7 Botnet Spread

Botnets are like any other malware, they can be spread as either a payload of a virus/trojan, by social engineering the victim to run the botnet attached to an email/malicious website [43]. In most cases, botnets are used to spread new bots. This is very easy since all bots implement mechanisms to download and execute a file via HTTP or FTP. The Witty worm, which attacked the I Seek You (ICQ) protocol parsing implementation in Internet Security Systems (ISS) products is suspected to have been initially launched by a botnet due to the fact that the attacking hosts were not running any ISS services [36].

2.4 Botnet Detection Techniques

Botnet detection has been a major research topic in recent years. Different techniques and approaches have been proposed for detection and tracking of botnet. The most widely used techniques are classified as signature-based, DNS-based, mining-based and anomaly-based [47]. The anomaly-based detection approach is further classified as host-based and network-based [17].

2.4.1 Signature-based Botnet Detection

A signature-based botnet detection technique uses the signatures of current botnets for its detection. Then by using pattern matching method it compares the signature of network traffic with the existing bots [48]. This method has several advantages, such as very low false alarm rate, immediate detection, easier to implement and there is better information about the type of detected attack. The limitation of signature-based technique is that it can only find the

botnets which are known and already traced [49]. Thus, unknown botnets cannot be detected by this method. Anomaly-based detection techniques are introduced to overcome this drawback.

2.4.2 DNS-based Botnet Detection

DNS-based botnet detection is a kind of botnet detection method which utilizes DNS-related network traffics to determine whether the machines involved are infected with a botnet or not [14]. DNS based techniques work with information collected from DNS queries. Botnet needs every bot to connect and communicate with the command and control server, thus it must use DNS queries that are typically hosts by dynamic DNS providers. Unlike the signature-based approach, the DNS based approach can be a more effective way of detecting botnets without prior knowledge of the botnet [50]. They are commonly based on detection of anomalous DNS network traffic generated by bot computers.

2.4.3 Mining-based Botnet Detection

Data mining botnet detection technique aims to recognize useful patterns to discover regularities and irregularities in large datasets [15]. Packet flow provides full information of flow data but in large file type. Data mining technique can be applied for optimization purpose. It enables to extract sufficient data for analysis from network log file. Most useful data mining techniques include correlation, classification, clustering, and aggregation for efficient knowledge discovery about network flows [15].

Flow correlation algorithms are useful to compare flow objects based on some characteristic other than packet content. This technique is very effective when content of packet is not available or encrypted, e.g., might compare arrival time. These kinds of algorithms utilize the characteristic values as inputs into one or more functions to create a metric used to decide if the flows are correlated [51]. Whereas, classification algorithms assume that incoming packets will match one of the previous patterns in order to detect new attacks [18].

Clustering is a well-known data mining technique where data points are clustered together based on their feature values and a similarity metric. Clustering differs from classification in that there is no target variable for clustering. Clustering algorithms divide the entire dataset

into subgroups or clusters containing relatively identical features. Thus, clustering provides some significant advantages over the classification techniques, since it does not require a labeled dataset for training [52]. To find particular pattern from large dataset is known as aggregation method, collecting and analyzing several types of records from different channels simultaneously.

2.4.4 Anomaly-based Botnet Detection

The idea behind anomaly-based detection approach is to perform botnet detection by considering several different network traffic anomalies. These includes high network latency, high traffic volume, traffic on unusual ports, and unusual system behavior that could indicate the presence of malicious bots in the network [12, 53]. This technique solves the problem of detecting unknown botnets. Anomaly-based detection techniques are further divided into host-based detection and network-based detection [17].

➢ Host-based Approaches

A host-based technique is a detection strategy which monitors and analyzes the internal parts of a computer system instead of network traffics by identifying abnormal computer usage (e.g., increased CPU usage and memory usage) [7]. In this approach the individual machine is monitored to find any suspicious behavior, including its processing overhead, and access to suspicious files. If suspicious activity is detected the botnet detection system will alert the user or administrator. It takes a snapshot of existing system files and matches it to the previous snapshot. If the critical system files were modified or deleted, an alert is sent to the administrator to investigate [54].

➢ Network-based Approaches

A network-based technique is a detection strategy which tries to detect botnets by monitoring network traffics. Network-based techniques can be classified into two categories [55]:

a. **Active monitoring** – based on the ability to inject test packets into the network, servers or application for measuring the reactions of network. Hence, it can produce extra traffics on the network. The injected packets can determine whether a human or a bot is managing that session. It works in a cause-effect correlation because for a large

portion of botnet command-and-control channels, a command and control interaction has a deterministic command response pattern. This technique shows effectiveness on real-world IRC based botnet detection [17].

b. **Passive monitoring** – observe data traffic in the network and look for suspicious communications that may be provided by bots or command and control servers. The passive monitoring uses some devices to inspect the traffics as they pass by. It does not increase the traffic on the network for inspection. This strategy usually requires a long time to inspect multiple stages or rounds of botnet communication and activities to detect botnets [17].

2.4.5 Comparison of Botnet Detection Techniques

As all techniques have some advantages and some limitations, Table 2.1 [25] shows a brief comparison of the botnet detection techniques. The comparison is based on key features including: ability to detect known/unknown bots, botnets with encrypted C&C channels, capability of botnet detection regardless of botnet protocol and structure, and real-time detection.

Table 2.1: *Comparison of Botnet Detection Techniques*

Botnet Detection Technique	Known Botnet Detection	Unknown Botnet Detection	Encrypted Traffic Detection	Structure and Protocol Independent	Real Time Detection
Signature based	Yes	No	No	No	Yes
DNS based	Yes	Yes	Yes	No	No
Mining based	Yes	Yes	No	Yes	No
Anomaly based	Yes	Yes	Yes	Yes	No

2.5 Artificial Intelligence

Artificial Intelligence (AI) enables machines to extract, integrate, exchange, and analyze large heterogeneous datasets to answer complex problems in a timely manner [56]. It is an emerging

component of Computer Science, which tries to make computers more intelligent. Machine Learning is one of the most rapidly emerging parts of AI [57].

2.5.1 Machine Learning

Machine Learning (ML) is a field of Computer Science, involving the study and construction of techniques that enable computers to self-study based on the input data to solve specific problems [58]. Machine Learning plays an important role in AI. It is widely used in many fields (such as pattern recognition, data mining, medical diagnosis, botnet detection and so on) because of its excellent performance [59]. ML algorithms extract the internal relationship of the data which can be presented by some rules or models for prediction and classification. As depicted in Figure 2.4 [60], ML is broadly classified as supervised, unsupervised and semi-supervised learning [61].

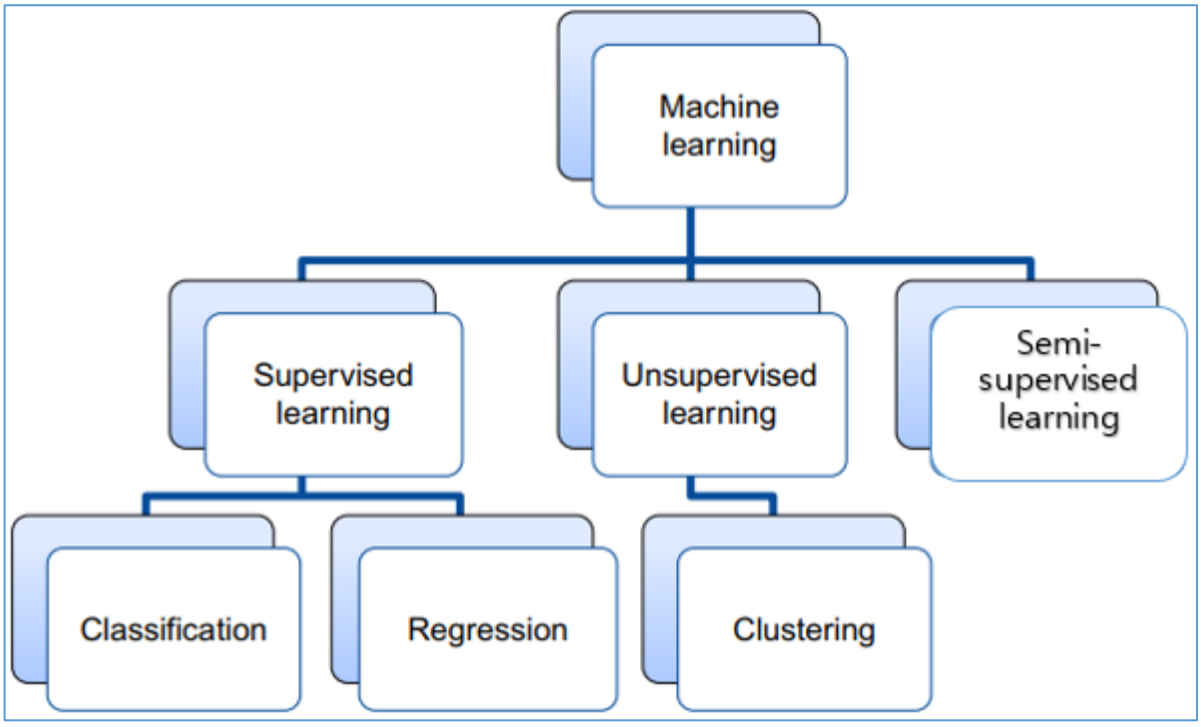

Figure 2.4: *Types of Machine Learning*

- **Supervised Learning**

Supervised learning is a learning model built to make prediction, given an unexpected input instance. A supervised learning model has two major tasks: classification and regression.

Classification is about predicting a nominal class label and used to predict discrete responses, whereas regression is about predicting the numeric value for the class label and used to predict continuous responses. Classification is recommended if the data can be categorized, tagged, or separated into specific groups or classes [60].

A supervised learning algorithm takes a known set of input dataset and its known responses to the data (output) to learn the classification/regression model [60]. A learning algorithm then trains a model to generate a prediction for the response to new data or the test dataset.

- **Unsupervised Learning**

Unsupervised learning is the opposite of supervised learning, where unlabeled data is used because a training set does not exist. Unsupervised learning finds structures in the data and none of the data can be presorted or classified beforehand. No labels are given to the learning algorithm, leaving it on its own to find structure in its input [62]. Unsupervised learning is often used in clustering problems. Clustering is used to draw inferences from datasets consisting of input data without class label or target values, or for exploratory data analysis to find hidden patterns. It groups data instances that are similar to each other in a cluster and data instances that are very dissimilar from each other into different clusters. The common unsupervised Machine Learning algorithms are K-Means algorithm, K-Nearest Neighbor, Hierarchical Clustering algorithm, and Graph Clustering algorithm [29].

- **Semi-supervised Learning**

Semi-supervised Machine Learning is an approach which incorporates both supervised and unsupervised Machine Learning [63]. When the labels of target class are not available in data, the points can be grouped together under different clusters. For classifications of future data instances at later stage, supervised learning methods are taken into account to obtain the trained models. The clustering technique of unsupervised Machine Learning is used for segmentation of data into unlabeled but separable points [29]. The similar instances of data are grouped into same cluster whereas the clusters are separated by an applied metric of dissimilarity. In semi-supervised mode, clusters are obtained to label the data in subsequent phase so that detections are possible for future instances [63].

2.5.2 Classification Algorithms

The most popular supervised Machine Learning classification algorithms are Artificial Neural Network, Decision Tree (DT), and Support Vector Machine (SVM) [29].

A. Artificial Neural Network

Artificial Neural Networks are computational techniques that belong to the field of Machine Learning. The aim of ANN is to realize a very simplified model of the human brain. In this way, ANN tries to learn tasks (to solve problems) mimicking the behavior of brain. The brain is composed by a large set of elements, specialized cells called *neurons*. Each single neuron is a very simple entity, but the power of the brain is given by the fact that neurons are numerous and strongly interconnected between them. The brain learns because neurons are able to communicate between each other [64].

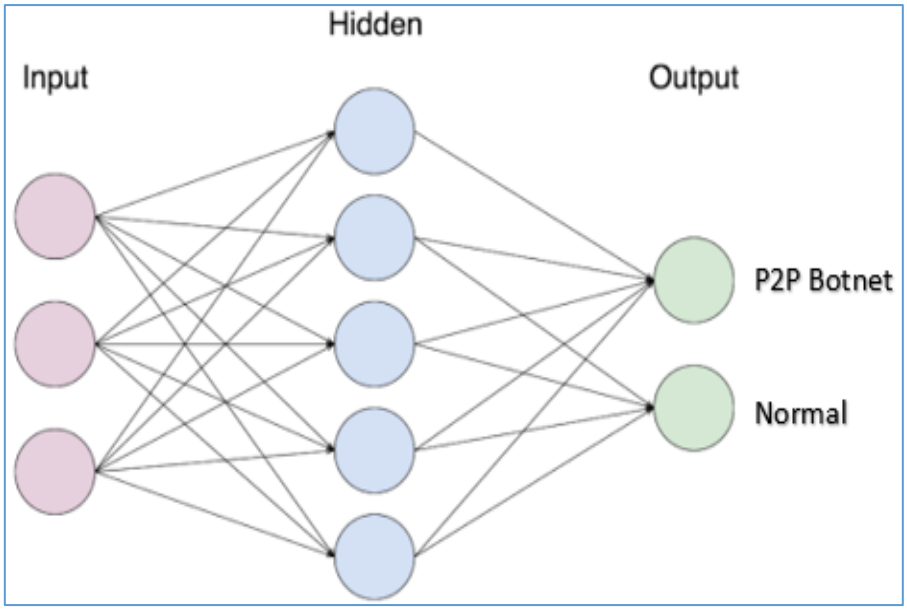

Figure 2.5: *Artificial Neural Network*

In analogy with the human brain, ANN uses a large set of elementary computational units, called themselves (artificial) *neurons*. Each neuron is able to only perform very simple tasks, and ANNs are able to perform complex calculations because they are typically composed by many artificial neurons, strongly interconnected between each other and communicating with each other. Each connection or edge, like the synapses in a biological brain, can transmit a

signal from one neuron to another as shown in Figure 2.5 [65]. A neuron that receives a signal processes it and subsequently conducts it outwards to other conjoined neurons [64, 66].

- **Fuzzy Logic**

To enable a system to deal with cognitive uncertainties in a manner more like humans, we can incorporate the concept of fuzzy logic into the neural networks [67]. Fuzzy Logic belongs to the family of many-valued logic. It focuses on fixed and approximate reasoning opposed to fixed and exact reasoning. A variable in fuzzy logic can take a truth value range between 0 and 1, as opposed to taking true or false in traditional binary sets [64]. The resulting hybrid system of ANN and fuzzy logic is called fuzzy neural, neural fuzzy, neuro-fuzzy or Fuzzy-Neuro Network [67]. Among the hybridized names, we will use FNN in this thesis. The hybrid system combines the capabilities of neural networks and fuzzy computations.

B. Decision Tree

Decision Tree learning (also referred to as a classification tree or a reduction tree) is a supervised Machine Learning technique for inducing a decision tree from training data. It is a predictive model which is a mapping from observations about an item to conclusions about its target value. DT generates a tree which can classify every record into different classes.

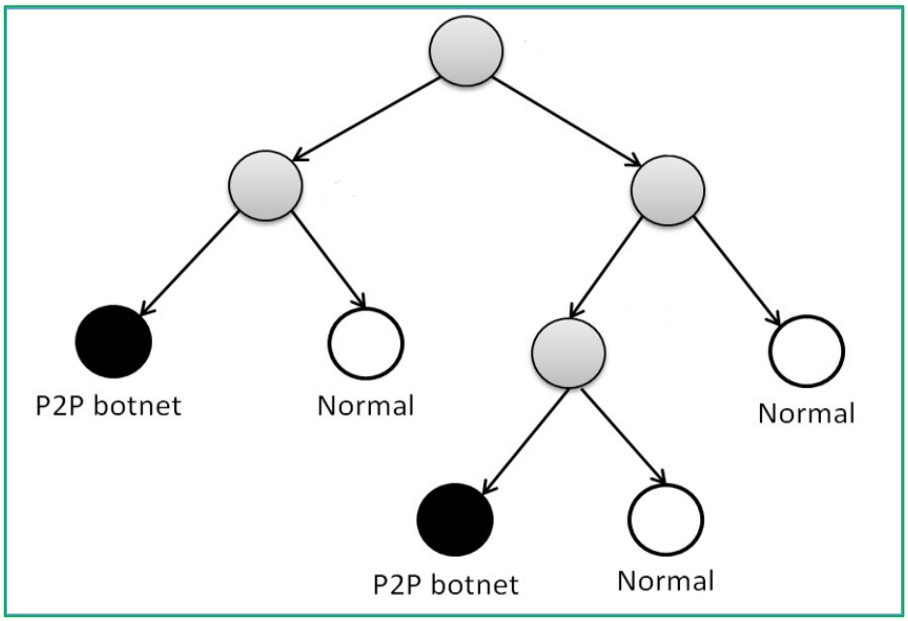

Figure 2.6: *Decision Tree*

As illustrated in the tree structure in Figure 2.6 [60], leaves represent classifications (also referred to as labels), non-leaf nodes are features, and branches represent conjunctions of features that lead to the classifications [68].

C. Support Vector Machine

Support Vector Machine is one of the best-known algorithms that would separate two classes using a hyperplane as shown in Figure 2.7 [60]. The idea is to find an optimal hyperplane that separates the feature points of the two different classes by the largest possible margin in the feature space. This can be achieved either within the same feature space, or by projecting the features into a higher dimensional plane where classification is easy [69]. Initially, SVM was designed to classify two classes, today it is also used to classify data which is composed of two or more classes that cannot be separated linearly [60].

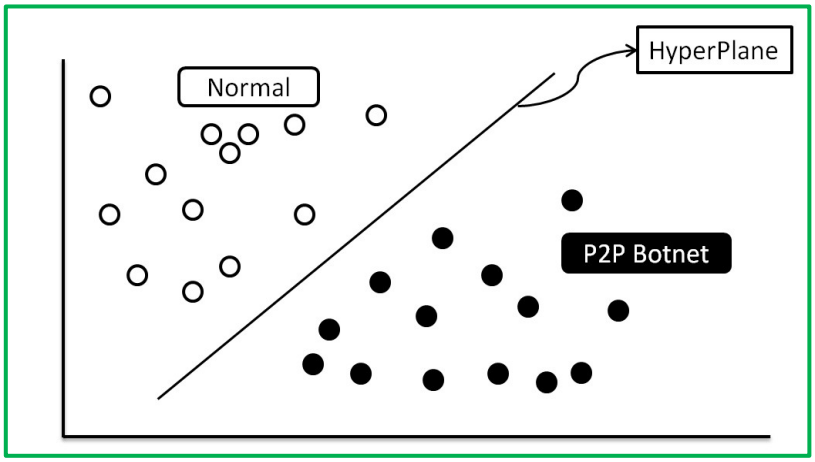

Figure 2.7: *Support Vector Machine*

2.6 Feature Selection

Feature selection is a process by which we search for the best subset of attributes or features in the dataset [70]. The aim of feature selection is to choose a suitable feature subset for improving classification performance or reducing the complexity of a classification model without significantly reducing the performance [71].

There are two types of well-known feature selection techniques: *Wrapper* and *Filter* [9].

2.6.1 Wrapper Method

Wrapper feature selection approach uses classification algorithms to select the best subset of features based on predetermined evaluation criteria [72]. According to [73], Figure 2.8 explains the process applied by the wrapper feature selection approach in selecting the best subset of features.

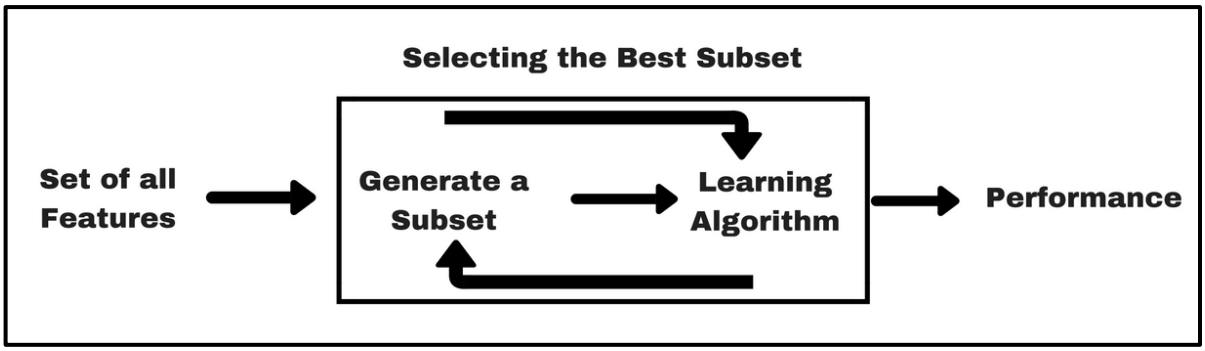

Figure 2.8: *Wrapper Feature Selection Method*

The three well-known wrapper feature selection algorithms are Artificial Bee Colony, Genetic Algorithm and Particle Swarm Optimization [70]. Wrapper approaches are computationally very expensive for high dimensional datasets as compared to filter approaches [72, 74]. When there are large number of features, filter feature selection methods are preferred [9].

2.6.2 Filter Method

Filter feature selection methods are independent of any classification algorithm in filtering out the irrelevant features [72]. Instead, features are selected based on various statistical tests to assign a scoring to each feature. The features are ranked by the score and the scores are used to remove low-scoring features and retain high-scoring features from the dataset [70]. According to [73], Figure 2.9 illustrates the process that is used by the filter feature selection approach in selecting the best subset of features.

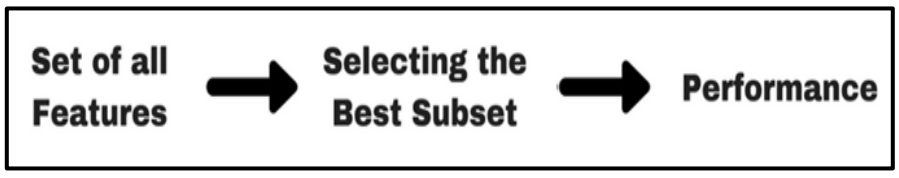

Figure 2.9: *Filter Feature Selection Method*

The three well-known filter feature selection approaches are Information Gain, Principal Component Analysis and Correlation Feature Selection [70]. Among the three approaches, we will further discuss Information Gain which is one of the best-known filter feature selection techniques [74].

- **Filter Feature Selection:** *Information Gain (IG)*

IG evaluates the importance of a given feature by analyzing its entropy value (how much information it provides) with respect to the class [70]. The features with higher ranks have better ability to represent the dataset, since they contain more information than other features. The features at the bottom of the ranked feature list are not good at representing the dataset, since they are less informative.

For a particular feature F, the higher the IG is, the more information F holds. The formula to compute IG on knowing the value of feature F is presented in Equation 1 [75].

$$\mathbf{IG\ (F) = H(S) - H(S|F)} \quad \mathbf{(1)}$$

where *H(S)* is the target element of a set of features, it can be $S_1, S_2, \ldots, S_m$. *H(S/F)* in the equation represents the classification model entropy without feature F. It can be calculated with Equation 2:

$$\mathbf{H(S|F)} = P_1 * H\ (S|F=F_1) + P_2 * H\ (S|F=F_2) + \ldots + P_n * H\ (S|F=F_n) \quad \mathbf{(2)}$$

where *n* is the number of values that can be assigned to feature F.

$$\mathbf{P_i} = \frac{\text{number of the instances with the value of F is } F_i}{\text{total number of the instances}} \quad \mathbf{(3)}$$

whereas

$$\mathbf{H(S|F=Fi)} = \sum_{j=1}^{m} (-p(S=S_j|F=F_i) * \log_2 (p(S=S_j|F=F_i))) \quad \mathbf{(4)}$$

and

$$\mathbf{p(S=Sj|F=Fi)} = \frac{\text{number of instances with } F_i \text{ and } S_j}{\text{number of instances with } F_i} \quad \mathbf{(5)}$$

2.7 Metrics

Highly accurate classifier puts as many instances as possible into the correct class and avoids classifying instances into the incorrect class. The metrics used for evaluating the performance of a classifier are discussed below.

2.7.1 Confusion Matrix

A confusion matrix is a 2D array that is often used to visualize the performance of a classification model (or "classifier") on a set of test data for which the correct labels are already known [72]. The confusion matrix as shown in Table 2.2 can help us to understand the evaluation metrics better. It has a total of 4 blocks: two rows and two columns which tell the values of true positives, true negatives, false positives, and false negatives [47, 61]. The malicious P2P botnet traffic are considered as positive instances and benign traffic as negative instances. The four metrics involved in a confusion matrix are discussed as follows:

- **True positive (TP):** represents the number of P2P botnet samples correctly classified as P2P botnet instances.
- **True negatives (TN):** indicates the number of normal samples correctly labeled as normal instances.
- **False positives (FP):** represents the number of normal samples misclassified as P2P botnet instances.
- **False negatives (FN):** indicates the number of P2P botnet samples misclassified as normal instances.

Table 2.2: *Confusion Matrix*

		Predicted Class	
		Normal	*P2P botnet*
Actual Class	*Normal*	True Negative **(TN)**	False Positive **(FP)**
	P2P botnet	False Negative **(FN)**	True Positive **(TP)**

2.7.2 Evaluation Metrics

To evaluate the performance of a classifier the most common and well-known evaluation metrics such as accuracy, precision, recall and F-measure are used [72, 76]. The four performance evaluation metrics are discussed as follows:

- **Accuracy:** It gives the percentage of correctly classified instances by using Equation 6:

$$\textbf{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad \textbf{(6)}$$

- **Precision:** It indicates that out of all the instances the classifier labeled as positive, what fraction were correct. It is calculated using Equation 7:

$$\textbf{Precision} = \frac{TP}{TP+FP} \quad \textbf{(7)}$$

- **Recall** (also called sensitivity or true positive rate): It indicates that out of all real positive instances, what fraction did the classifier labeled as positive. It is calculated using Equation 8:

$$\textbf{Recall} = \frac{TP}{TP+FN} \quad \textbf{(8)}$$

- **F-measure:** It is a harmonic mean of precision and recall. It is calculated according to Equation 9:

$$\textbf{F-measure} = 2 * \left[\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \right] \quad \textbf{(9)}$$

2.8 Summary

In this Chapter, we have introduced some basic information about botnets, their lifecycle and their brief history. We have also discussed architectures of botnets which are classified into three types: centralized botnet, P2P botnet and hybrid botnet. As discussed, the botnets can be

used to launch different kinds of attacks such as DDoS attacks, sending spams, record keylogging's, perform identity frauds, etc. We examined a set of techniques used for botnet detection and also attempted to compare each one based on their key features. In addition, we discussed three categories of Machine Learning algorithms namely supervised, unsupervised and semi-supervised learning along with the best-known classification algorithms. Furthermore, feature selection techniques such as wrapper and filter are presented. Finally, classification algorithms evaluation metrics such as confusion matrix, accuracy, precision, recall and F-measure are discussed.

Chapter 3: Related Work

In this Chapter, existing related works conducted on P2P botnet detection are discussed. The increasing popularity of P2P botnets has led to a vast amount of research that attempt to track and remove them. We briefly touch upon existing approaches for P2P botnet detection, and their relative advantages and disadvantages. We categorize the detection techniques into three sections in this thesis. The first section reviews the available botnet detection techniques that are based on DNS features, the second section focuses on signature-based detection techniques and the third section discusses the anomaly-based botnet detection research works. Finally, the gaps of the reviewed researches as well as the way the newly proposed solution fills those gaps is described briefly.

3.1 DNS based Botnet Detection

Khehra and Sofat [77] proposed DNS based botnet detection system named BotScoop. The system primarily requires network traffic (i.e., DNS traffic) from which an initial feed of malicious domains can be extracted. BotScoop performs the detection based on the analysis of the non-existent domains (i.e., unsuccessful domain name resolutions), and it analyses the query behavior and domain attributes in order to distinguish bot infected hosts from legitimate ones. The system consists of two main modules: filtering and feature extraction module and bot detection module. In filtering and feature extraction module, the captured network traffic is given as input to the system. This module examines the various stream of unresolved DNS queries in order to separate the suspicious and unsuspicious hosts. Whereas in the bot detection module, it separates the bot-infected hosts and group them together by applying a clustering algorithm which results in creating clusters of similar suspicious hosts. Then, the final step focuses on identifying the bot among the cluster candidates by considering a certain threshold. They designed the P2P botnet detection system using two ML techniques namely Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN). Their experimental results showed that the overall accuracy was 98.7% for RNN and 98.9% for CNN. However,

BotScoop has specific practical limitation. The system has a predefined time window so increasing the time window will cause false positive results and more processing cost.

In [8], the authors developed a system named BotDAD for detecting bot-infected machine in a network using DNS fingerprinting. The proposed BotDAD architecture consists of two main subsystems: DNS fingerprinting unit and anomaly detection engine. The DNS fingerprint unit generates hourly DNS fingerprint of each host in the network by inspecting network DNS traffic. The DNS fingerprint unit consists of three subunits: Feature Extractor, Host Profiler, and Fingerprint Generator. The feature extractor module processes the network capture file to extract various DNS request and response parameters that are useful for fingerprint generation. Then, the output of the DNS feature extraction module (i.e., request and response parameters) are parsed by the host profiler and a DNS profile for all hosts in the network will be generated by grouping the queries from each host. Finally, the DNS fingerprint generator module parses the host profile and creates a DNS fingerprint for each host in the network. The fingerprint is then examined by anomaly detection engine which groups it into two categories (i.e., bot or clean) based on presence or absence of an anomality. The anomaly detection module attempts to find anomalous DNS query behavior from various hosts in a network. They designed the P2P botnet detection system using a supervised ML classification algorithm named Decision Tree. Their experimental result showed that the overall accuracy of the classifier was 99.78%. The drawback is that the proposed system considers hourly traffic sample and generates a fingerprint for that period only. Since bots are not active throughout the day, it will be reported clean by the system if the bot is inactive during the investigation time period. The investigation time has to be extended to longer time intervals for detecting sophisticated bots in a network.

Wang *et al.* [12] proposed a botnet detection scheme based on an analysis of the query behavior of the DNS traffic designated as DBod. The authors stated that DBod distinguishes bot-compromised hosts from normal hosts on the basis of the difference in their query behaviors of bots and normal users, respectively. The proposed system consists of the following three modules namely: Filtering Module, Clustering Module and Group Identification Module. The filtering module distinguishes bot-compromised hosts from normal users on the basis of differences in the query behaviors of the hosts. Accordingly, the

module will identify and remove the “normal” hosts to reduce the volume of data to be processed. The filtered module results are input to the clustering module for further processing. The aim of the Clustering module is to group the hosts compromised by the same botnets. To achieve this goal, the query behaviors of the botnets are first translated to a correlation topology in accordance with their relationship strengths. Finally, the group identification module distinguishes the bot-compromised hosts from normal hosts using a supervised statistical algorithm. They designed the P2P botnet detection system using Chinese Whispers algorithm. Their experimental result showed that the overall accuracy of the classifier was 99.6%. However, the proposed system has practical limitation. The system is unable to detect bot-compromised hosts if the hosts have never attempted to connect to the botnet server during the detection time window. Since some bot-compromised hosts remain dormant for hours or even days after connecting to the botnet server, DBod is unable to identify the dormant bot-compromised hosts.

3.2 Signature based Botnet Detection

In [6], the authors implemented a peer-to-peer botnet detection method called BotShark. The method is designed with Classification and Regression Tree (CART). The proposed system consists of the following phases: Network Traffic Capturing, Packet Clustering, Attribute Classification and Decision Making. The architecture of their proposed system consists of three components namely Web Server, Client and BotShark server. The web server hosts the website application and the client is an end user who wishes to use the website for various purposes. The BotShark (Defense Server) is a middleware defense server application for the purpose of making the web server protected from Botnet attacks. All the access requests to the web server are handled by the BotShark and every packet on the network is captured by the defense server. Their main aim is to detect the botnet attack through the BotShark server since direct communication of end users and web server does not occur. The authors do not state the detection accuracy rate of their proposed system. The main limitation is that the model is not capable of detecting new types of attacks available in the test set but not found in the training set.

Chen *et al.* [13] proposed a Machine Learning based P2P botnet detection system. The proposed system consists of three stages: preprocessing, training, and confidence testing. In the preprocessing stage, it transforms the network packets into flows, labels each of them as malicious (bot) or benign (non-bot), and compresses the flows into a desired size. The compression is necessary since the size (packet number) of flows may range on different scales. The flow is defined on four IP packet attributes: IP source address, IP destination address, source port and destination port. In the training stage, it builds a classification model based on a CNN. The CNN consists of a feature extraction phase that contains alternating convolutional and pooling layers, and a recognition phase that is an Artificial Neural Network with fully-connected layers. In the confidence testing stage, the proposed system enhances detection accuracy by further classifying the flows of insufficient confidence ANN with a decision tree. The authors stated their system achieved 98.6% of detection accuracy and 0.5% of false positive rate on the known P2P botnet datasets. The main limitation is that the proposed solution can only identify botnets from predefined well-known P2P botnet datasets. It is only able to detect botnets from inside the local area network the bot resides.

In [47], the authors proposed reinforcement learning-based botnet detection system, designed to detect and identify infected hosts in a P2P botnet. The proposed detection approach comprises the following three phases: network traffic capture and packet reduction, feature extraction, and malicious activity detection. In the network traffic capture and packet reduction phase, the network traffic is sniffed based on the sliding time-window size, and later on utilized for traffic reduction. Then in feature extraction phase they analyze the reduced traffic to identify attributes that can be used to effectively characterize the botnet, and these attributes collectively form a feature. The network traffic feature extraction is implemented in two levels: packet-level (data packets that are transferred over the network) and flow-level (traffic stream between a source and a destination). Finally, the malicious activity detection phase includes three stages namely: offline stage (training), online detection stage, and reinforcement-learning stage. In the first stage, the classifier is provided with a group of legitimate and bot feature vectors for the purpose of training. When the training ends, the newly extracted features are uploaded to the online detection unit in order to classify the hosts activities within the network as legitimate or malicious. Later, the reinforcement-learning stage handles problems that involve difficulties in determining features as bots or normal

hosts. An ANN is utilized to classify the network traffic into malicious and normal. The authors stated that their system achieved 98.3% of detection accuracy and 0.012% of false positive rate. Generally, in the proposed solution, the use of network traffic reduction can minimize the training time required, but it also reduces the detection rate of P2P botnet features. The key limitation of this approach is high processing runtime.

Lu *et al.* [21] designed a P2P botnet detection system based on fuzzy association rules (fuzzy logic). The proposed detection system comprises the following components: features extraction, fuzzy clustering, fuzzy recognition and fuzzy association rules. First, they collect a P2P botnet network traffic records and extract UDP and TCP network traffic features. Then, the features of the datasets are clustered into common groups using the fuzzy clustering component. After that, the fuzzy recognizer calculates the level of membership for each feature through the proposed membership function (i.e., Sigmoid). Finally, through the association rules the proposed system recognize/detect the P2P botnet attacks. Their experimental result showed that the proposed system achieves 90% true negative rate. The authors do not state the detection accuracy rate of the system. The main limitation is that the model is not capable of detecting new types of attacks. Because, it can only identify botnets from predefined well-known P2P botnet datasets.

3.3 Anomaly based Botnet Detection

Wang *et al.* [11] developed a system named PeerGrep, to detect botnets in a P2P network. Their system starts with identifying long-living P2P botnets potentially engaged in P2P communications and distinguishes P2P bots from P2P hosts. PeerGrep first identifies all hosts that relate to P2P activities based on P2P-relevant groups. Then PeerGrep identifies P2P bots among the P2P relevant hosts by analyzing their active ratio, packet size and periodicity of connection to destination IP addresses. Particularly, their system aims to identify P2P bots even if the P2P botnet traffic coexists with traffic generated by legitimate P2P applications (e.g., Skype). The system collects flow records at the border of a monitored network and tries to identify internal hosts that are compromised by P2P bots. PeerGrep works in a two-stage scheme. In the first stage, all hosts that engage in P2P communications (either benign or malicious) are identified. In the second stage, PeerGrep differentiates P2P bots from other

benign P2P hosts by analyzing their network behavior patterns. The method is able to identify P2P bots even if the malicious P2P application and benign P2P application coexist within the same host in the monitored network. They designed the P2P botnet detection system using ANN. Their experimental result showed that the proposed system achieves 0.58% false positive rate. The authors do not state the detection accuracy rate of the system. The limitation is detecting botnets based on packet size is not feasible. Since, P2P botnet keeps hidden and undetectable in the network and does not share or download large amount of files.

In [78], the authors designed a P2P botnet detection system called BotCluster. It is a generic botnet-detection framework designed to identify P2P botnets based on behavioral similarities in the network traffic trace. BotCluster comprises of three stages, namely Session Extraction, Filtering and Grouping. In the Session Extraction Stage, BotCluster extracts the sessions and associated characteristics from the network flow (version 5) traffic log using a timeout threshold. Each record has attributes consisting of the timestamp, the source and destination addresses and ports, the number of packets, the number of incoming and outgoing bytes, the duration, and so on. To reduce the input size and lower the computing overhead, the Filtering Stage of BotCluster separates out legitimate activities and removes hosts with a lower likelihood of being P2P nodes. The non-P2P and benign traffic are then filtered out by reference to a whitelist. Finally, in the Grouping Stage, BotCluster aggregates the remaining sessions into groups in accordance with their feature similarities and produces a suspicious IP list for further investigation purposes. They designed the P2P botnet detection system using ANN. The experimental results showed that the proposed system had a detection precision rate of 97.58%. The main limitation is that the proposed BotCluster scheme is not capable of real-time detection of P2P botnets since it requires waiting time to accumulate the network flow log.

Zhuang and Chang [79] proposed a network-flow based P2P botnet detection system named Enhanced PeerHunter. They proposed a novel ML algorithm named Community Behavior Analysis to detect P2P botnet attacks. The system uses network-flow level behaviors to detect P2P botnet even though the P2P bot and legitimate P2P application is running on the same host. Enhanced PeerHunter has three components: P2P Network Flow Detector, Network-Flow Level Mutual Contacts Graph Extractor and P2P Botnet Detector. The P2P network flow

detector component aims to detect network flow data of hosts that engage in P2P communications. It collects a set of 5-tuple network flow data which consists: Source IP Address, Destination IP address, Protocol (TCP or UDP), and Outgoing and Incoming bytes-per-packets statistics. The network-flow level mutual contacts graph extractor component aims to extract mutual contacts graph using the network-flow level data. Mutual contacts are P2P bots that share the same natural characteristic ("botnet behavior") of P2P botnets. The basic idea of using mutual contacts is to build a mutual contact graph by clustering bots into groups. The proposed system clusters the bots and the other hosts into their own groups using the mutual contact graph. Finally, the P2P botnet detector component aims to detect P2P bots from the mutual contact graph based on their characteristics. The experimental results showed that the proposed system had a detection accuracy rate of 99.03%. However, detecting botnets using their natural characteristics is not that much accurate. Since bots from different botnets are controlled by different Botmasters, they will not have the same characteristics.

Kirubavathi and Anitha [18] proposed a botnet detection technique through mining the network traffic flow characteristics of the bot. Their main objective is to identify the bots irrespective of their structural properties. At initial stage, they extract important features which are used to model the behavior of the botnets. Then, they used significant features like initial packet length, small packets, bot response packet and packet ratio to identify botnet traffic. Finally, the significant features from the network traffic flows are passed to the classification algorithms to differentiate the normal and botnet traffic flows. To classify the network traffic as botnet or normal, they utilize three different ML algorithm such as DT, Naïve Bayesian and SVM. The authors stated that their proposed system has the ability of detecting botnets even if the communications is encrypted. Their experimental results showed that the overall accuracy was 95.86%, 99.14% and 92.02% for DT, Naïve Bayesian (NB) and SVM respectively. The main limitation of their proposed system is that they assume all botnets will have the same communication structure which will be reflected in the network traffic flow. But in reality, different botnets will have different communication structure.

3.4 Summary

This Chapter presented related works on P2P botnet detection techniques along with their respective advantages and limitations. We categorized the researches into three classes: DNS-

based, Signature-based and Anomaly-based. There have been several attempts to detect P2P botnets but there remain certain limitations that are uncovered by these research works. The P2P botnet detection techniques we reviewed have their own major shortcoming such as low performance and relatively low detection rate. In addition, the signature-based P2P botnet detection approaches can only detect known attacks. As we can observe on the limitations, lots of works should be done on providing more efficient P2P botnet detection technique. So, this study is intended to fill the gaps observed on previous works and design improved anomaly-based technique by combining fuzzy logic and Artificial Neural Network to detect P2P botnet attacks.

Chapter 4: The Proposed System Architecture

In this Chapter, the architecture of the proposed P2P botnet detection model is presented and individual components are described. The primary goal of this research work is to design a model that detects P2P botnet attacks using Fuzzy-Neuro Network.

4.1 Overview of the Architecture

The proposed system architecture is composed of six major components namely: Feature Extractor, Feature Selector, Dataset Constructor, Preprocessor, Classifier and P2P Botnet Detector.

In the feature extraction and selection stage: first, we extract all available features and then select the most relevant features. The dataset constructor is used to convert the comma separated value (CSV) file into sets and split the dataset as training and testing sets.

The major activities of the data preprocessor are data cleaning, data transformation and data reduction. Then, the classifier is responsible in creating a model using Fuzzy-Neuro Network. Finally, the P2P botnet detector is used to classify the network traffic into two classes: P2P botnet or normal.

Figure 4.1 shows the proposed architecture. In Section 4.2 all the major components of the architecture are described in detail.

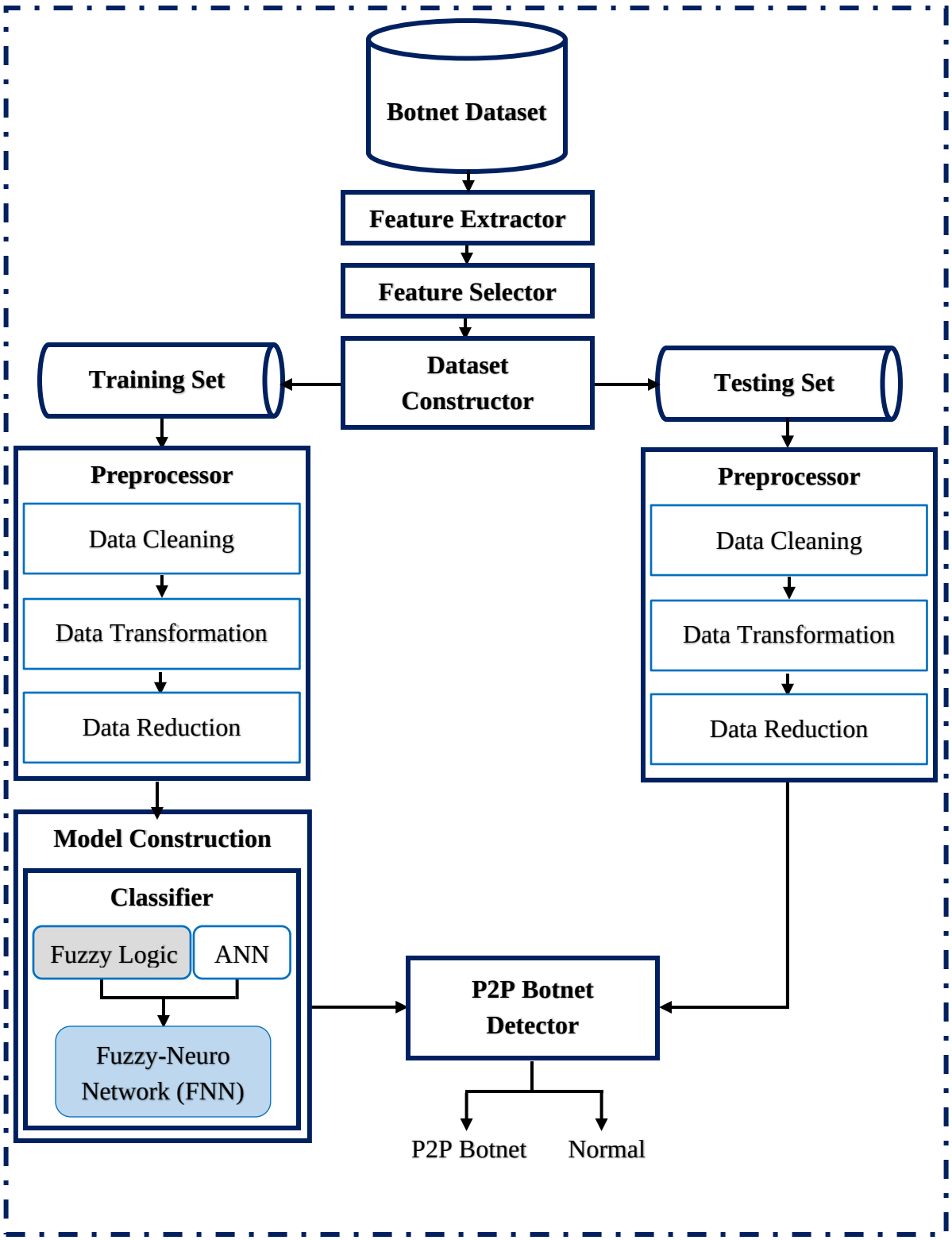

Figure 4.1: *The Proposed P2P Botnet Detection Architecture*

4.2 Description of Major Components of the Architecture

4.2.1 Botnet Dataset

To make sure that our system will be able to detect any types of P2P botnet attacks, we must train and test it using network traffic data from a diverse variety of dataset sources. For that reason, we have obtained our dataset /network Packet Capture (PCAP) files/ from two publicly available P2P botnet datasets, Bot-IoT dataset [80] and UNSW-NB15 dataset [81, 82]. The datasets include both genuine and malicious network traffics.

- **Bot-IoT Dataset**

Bot-IoT dataset incorporates genuine and simulated network traffic, along with various types of P2P botnet attack records. The dataset is a mixture of benign and attack network traffic, resulting in a 69.3 GB dataset in the form of PCAP files. The dataset consists of more than seventy-three million (73,360,900) botnet attack records and over nine thousand (9,543) legitimate network traffic records.

The dataset consists of three types of P2P botnet attacks: probing attacks (operating system fingerprint and service scanning), denial of service attack (DoS and DDoS) and information theft (keylogging and data theft). It also includes normal P2P network traffic generated using Ostinato software. The tool was utilized to generate a large amount of normal network traffic between virtual machines.

- **UNSW-NB15 Dataset**

The UNSW-NB15 dataset was created using an IXIA Perfect Storm software in the cyber range lab of the Australian Centre for Cyber Security. The dataset has a mixture of real network traffic and synthesized attack activities. The dataset consists of more than three-hundred thousand (321,283) botnet attack records and over two-million (2,218,761) legitimate network traffic.

The dataset includes realistic activities of normal traffic that were captured from virtual servers and has nine categories of P2P botnet attack records namely: fuzzers, analysis,

backdoors, DoS, exploits, generic, reconnaissance, shellcode and worms. A tcpdump tool was used to capture 100 GB of raw network traffic in the form of PCAP files.

4.2.2 Feature Extractor

Feature extraction is quite a complex concept concerning the translation of raw network traffic data into the inputs that our particular P2P botnet detection model requires. Network traffic can be described as a summary of connection between two hosts [83]. It provides helpful information about the network behavior. Network traffic is defined based on a combination of 5-tuple, source and destination IP addresses, source and destination port numbers, and protocol [76]. A single 5-tuple network traffic information usually does not provide sufficient evidence to decide if a particular device is infected or if a particular request has malicious symptoms [84]. Therefore, it is quite common to extract more features from network traffic records. There are tools that are useful to extract relevant flow information from a network traffic. In our work, we have used a feature extraction application named *CICFlowMeter* (Canadian Institute for Cybersecurity Flow Meter) [85, 86] to extract features from the datasets discussed in Subsection 4.2.1. The tool is used to extract more vital features rather than the common 5-tuple network traffic information and then calculate some statistical features. The features extracted from Bot-IoT and UNSW-NB15 datasets are presented in Annex B [80] and Annex C [81, 82] respectively.

4.2.3 Feature Selector

The quality of the features has a significant impact on the performance of our proposed P2P botnet detection model. There may be some features that will not affect the classification result at all, or perhaps might make the result worse. Therefore, since some attributes might not equally contribute to the classification result, through feature selection the most relevant subset of features in the dataset are selected for analysis. The well-known feature selection techniques are discussed earlier in Chapter 2 under Section 2.6. In our model, we apply one of the best-known filter feature selection technique named Information Gain to choose the best feature subset, which is a rather straightforward approach that measures which subset of the features has the highest predictive power. It's not dependent on any algorithm and can be

used in any domain. To apply the IG filter feature selection approach, a series of tests was conducted using the best data mining tool Weka [87], which is an open source software written in Java.

4.2.4 Dataset Constructor

In this phase, the dataset constructor has two main tasks. First, the constructor will help us to divide the network traffic data as training and testing sets. Second, it will enable us to construct a dataset using the partitioned CSV data. Since the network traffic records are stored in a CSV file, we cannot use them directly in our model.

At initial stage, the constructor will read the CSV file using Pandas, which is a data manipulation and analysis tool built on top of the Python programming language, and then passes it to NumPy arrays. NumPy is a Python library used as an array processor for numbers, strings and records. Finally, after obtaining the array values from NumPy, the dataset constructor performs the following two tasks:

- **Task 1: Splitting the Dataset**

The first important task of the constructor is splitting the CSV dataset into two sets: *Training* and *Testing* sets. In Machine Learning, we design a model to predict the test data. Therefore, we use the training set to fit the model and testing set to evaluate it.

A. Training Set

The model will be fit using the training dataset, which contains a set of legitimate and malicious network traffic records to train the model. Our training subset consists of 70% of the total network traffic data.

B. Testing Set

The testing dataset is used to assess the classification performance of our model. It provides an unbiased evaluation of the final model that is fit on the training dataset. Our testing subset consists of 30% of the total network traffic data.

To split the CSV dataset as training and testing sets, the dataset constructor uses the Sklearn "*train_test_split*" function [88]. Sklearn (or Scikit-learn) is a Python library that offers various features for data processing that can be used to split a dataset into random train and test subsets. The Sklearn "*train_test_split*" function has four parameters. The basic syntax of the function is shown in Figure 4.2.

```
import sklearn.model_selection as model_selection

from sklearn.model_selection import train_test_split

train_test_split(X, train_size=0.7, test_size=0.3, random_state=0)
```

Where,

- **X**: The first parameter is the dataset we are going to use.
- **train_size**: This parameter sets the size of the training dataset.
- **test_size**: This parameter specifies the size of the testing dataset.
- **random_state**: Controls the shuffling applied to the data before applying the random split.

Figure 4.2: *Syntax of the Sklearn Function*

- **Task 2: CSV Data Conversion**

The CSV file cannot be used directly in our model and needs to be converted to the appropriate format using TensorFlow, which is a Machine Learning platform that operates at large scale and in heterogeneous environments. The dataset constructor will create a set using the "*tf.data.experimental.make_csv_dataset*" TensorFlow built-in function [89] as illustrated in Figure 4.3.

```
def get_dataset(file_path, **kwargs):
  dataset = tf.data.experimental.make_csv_dataset
    (
     file_path,
     // List of file paths containing CSV records.

     batch_size=10,
     // An integer representing the number of records to combine
     in a single batch.

     select_columns= column_names,
     // Specifies a subset of columns of CSV data to select which
     are provided in column_names variable.

     num_epochs=1,
     // An integer specifying the number of times this dataset
     is repeated. If None, cycles through the dataset forever.

     ignore_errors=False,
     // If True, ignores errors with CSV file parsing, such as
     malformed data or empty lines, and moves on to the next
     valid CSV record. Otherwise, the dataset raises an error
     and stops processing when encountering any invalid records.

      **kwargs
       //The ** allows us to pass any number of keyword arguments.
     )
  return dataset

raw_train_data = get_dataset(train_set)   //70% of the dataset
raw_test_data = get_dataset(test_set)    //30% of the dataset
//Now we can pass our CSV data to the function and get a dataset.
```

Figure 4.3: *TensorFlow Built-in Function to Construct a Dataset from CSV File*

4.2.5 Preprocessor

Data preprocessing stage is very important to prepare the dataset for experimentation before building our model. It aims to facilitate the training/testing process by appropriately scaling and transforming the entire dataset [90]. A dataset can be viewed as a collection of data objects, which are also called as records, vectors or patterns. Data objects are described by a number of features or attributes. In this stage, the dataset gets transformed to bring it to such a state that our model can easily analyze and interpret it. We perform three major activities during the data preprocessing stage namely: data cleaning, data transformation and data reduction.

A. Data Cleaning

Data cleaning is the process of preparing data for analysis by detecting and modifying (or removing) data that is duplicated, empty or inaccurate record from a record set. For our purpose, we used the data cleaning technique to fill null values in our dataset. First, we inspect our dataset and identify the rows or columns that had empty or missing values. Next, we checked the values before and after the records containing empty values and calculated their mean. Finally, we replaced the empty values with the mean. This removed outliers shaped our dataset to make it more consistent. The basic syntax of the data cleaning process is shown in Figure 4.4.

```
//Checking 'Null' values in the training/testing set.
Training_NULL.isnull().sum()

//Checking the 'Mean' value of the null feature.
Training_NULL['dur'].mean()

//Looking the 'Header' of the feature.
Training_NULL['dur'].head()

//Filling 'Missing (Null) Values' of the feature using Mean.
Training_NULL['dur'].replace(np.NaN,Training_NULL['dur'].mean())
.head()
```

Figure 4.4: *Syntax of the Data Cleaning Process*

B. Data Transformation

Data transformation is the process of transforming data from one structure into another appropriate structure suitable for our model. The data transformation process includes data normalization task. Data normalization involves transforming all numerical features in the dataset to a specific range. The normalization is done in order to scale or unify the data values in a specified range (0.0 to 1.0 or -1.0 to 1.0) since numeric data should always be normalized.

For our purpose, we used the Z-score normalization technique to scale and unify the data values. Z-score (also called a *standard score*) converts all indicators to a common scale with an average of zero and standard deviation of one.

The basic Z-score normalization formula is shown in Equation 10 [91].

$$\textbf{Z-score} = \frac{\textbf{(value} - \boldsymbol{\mu}\textbf{)}}{\boldsymbol{\sigma}} \qquad \textbf{(10)}$$

where, μ is the mean value of the feature and σ is the standard deviation of the feature. If a value is exactly equal to the mean of all the values of the feature, it will be normalized to 0. If it is below the mean, it will be a negative number, and if it is above the mean it will be a positive number. The size of those negative and positive numbers is determined by the standard deviation of the original feature. If the unnormalized data had a large standard deviation, the normalized values will be closer to 0.

We perform normalization for the following two reasons. First, to eliminate the influence of one feature over another (i.e., to give features equal chances in model building). Second, to enhance the performance of the model (it will be faster with normalized data than with un-normalized one).

C. Data Reduction

Data reduction is the conversion of numerical or categorical data into ordered and simplified form. The basic idea is to reduce countless amounts of data down to meaningful parts. Our network traffic dataset includes two kinds of statistical data types numerical and categorical. The basic syntax of the data reduction process is shown in Figure 4.5.

```
//A function that will pack together all the numerical columns.
def pack(features, label):
  return tf.stack(list(features.values()), axis=-1), label


//We apply the previous 'pack' function to each element of the dataset.
packed_dataset = temp_dataset.map(pack)


//Creating a vocabulary list of the categorical features.
categorical_columns = []
for feature, vocab in CATEGORIES.items():
cat_col=
       tf.feature_column.categorical_column_with_vocabulary_list
       (key=feature, vocabulary_list=vocab)


//Display the pack function results (features and labels).
for features, labels in packed_dataset.take(1):
  print(features.numpy())
  print()
  print(labels.numpy())


//Creating a categorical layer of the vocabulary list.
categorical_layer =
tf.keras.layers.DenseFeatures(categorical_columns)
print(categorical_layer(store_batch).numpy()[0])


//We add the two feature column collections and pass them to a tf.keras.layers.DenseFeatures function.
preprocessing_result = tf.keras.layers.DenseFeatures
 (numeric_columns + categorical_columns)
```

Figure 4.5: *Syntax of the Data Reduction Process*

In general, the features can be:

- **Numerical**: Features whose values are integer-valued or continuous. They are represented by numbers. For instance, feature "*label"* whose value is a binary set: {0 or 1}.

- **Categorical** : Features whose values are taken from a defined set of values. For instance, feature "*attack category"*: {Normal, Reconnaissance, Backdoor, DoS, Exploits, Analysis, Fuzzers, Worms, Shellcode, Generic} is a category because its value is always taken from this set.

Generally, we want to convert from those mixed types to a simplified and fixed length vector before feeding the data into our model. To do that, we create a list of numerical and categorical features, respectively. Next, we pack the numerical features into a single vector and the categorical features into vocabulary list. Finally, we combine the two feature column collections into one and pass it to our model.

4.2.6 Classifier

In Machine Learning, classification is a supervised learning concept which basically categorizes data according to a predefined set of classes and assign each item the label of the proper class [76]. We discussed different supervised based Machine Learning classification techniques earlier in Chapter 2 under Subsection 2.5.2, which can be used to classify legitimate and P2P botnet traffics. Among the techniques, we are going to use the classification technique named Fuzzy-Neuro Network to build our model.

- **Fuzzy-Neuro Network**

A Fuzzy-Neuro Network (or *neuro-fuzzy system*) is a hybrid system of fuzzy logic and Artificial Neural Network. As we discussed it earlier in Chapter 2 under Subsection 2.5.2, FNN integrates the human-like reasoning style of fuzzy systems (i.e., fuzzy logic) with the learning and connectionist structure of neural networks [67].

In general, FNN takes an input in the form of a vector through fuzzy logic layer. Then, it forwards the input across a number of hidden layers in the neural network. Each hidden layer

consists of several neurons that are connected to all the prior neurons. Finally, the result of the hidden layers is displayed in the output layer of the neural network. The FNN used in this research comprises 1 input layer (consists of 4 neurons), 1 hidden layer (consists of 6 neurons) and 1 output layer (consists of 2 neurons). The general structure of FNN classifier which includes fuzzy logic and neural network layers is illustrated in Figure 4.6.

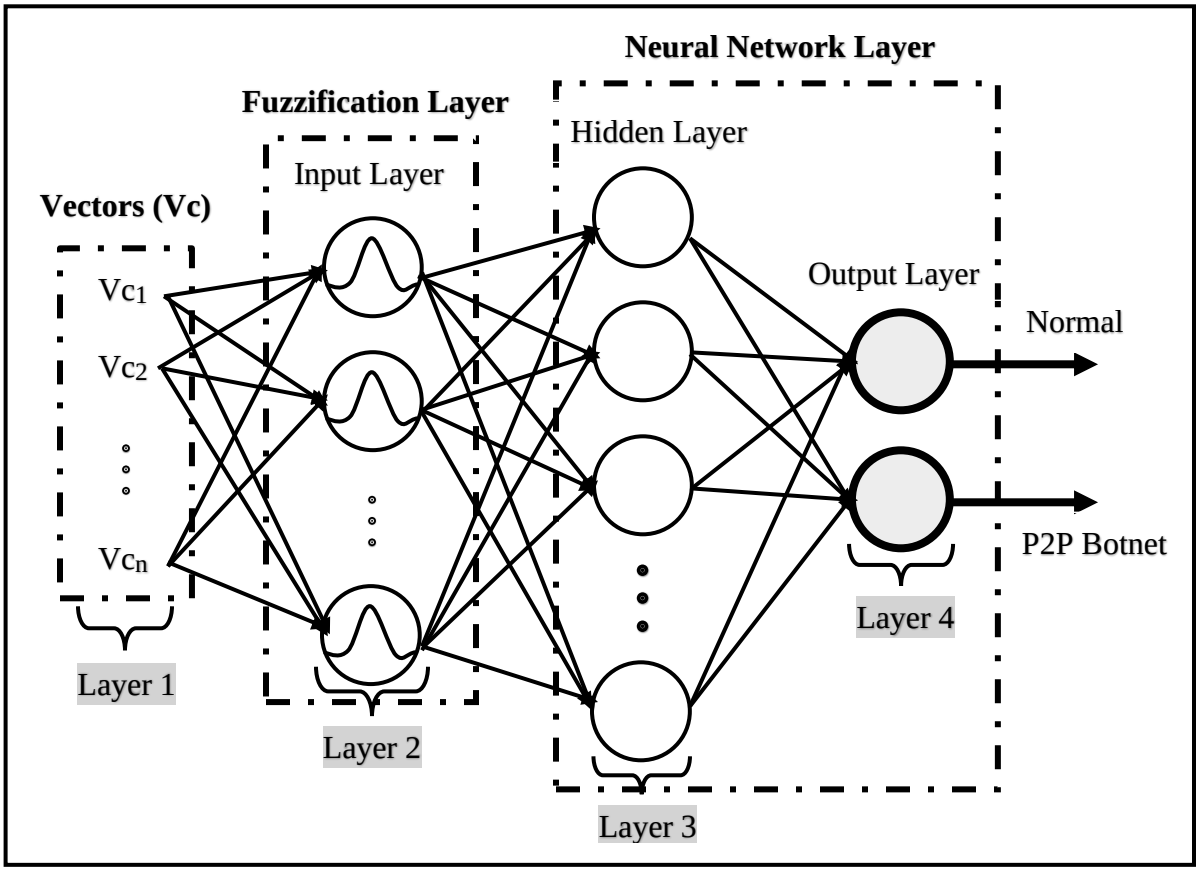

Figure 4.6: *The Structure of Fuzzy-Neuro Network Classifier*

The Fuzzy-Neuro Network classifier is composed of four layers:

A. Layer 1: *Vectors*

In the first stage, the FNN classifier is provided with a number of malicious and legitimate input data vectors (Vc). The vectors are denoted by $Vc = \{Vc_1, Vc_2, Vc_3, \ldots, Vc_n\}$, where n is the number of vectors or records available in the dataset. The vectors are given as an input for the fuzzification layer and no computations are performed at this stage.

B. Layer 2: *Input Layer (Fuzzification Layer)*

Fuzzification is a process of determining the degree to which an input vector belongs to each of the appropriate fuzzy sets via the membership function [92]. The fuzzification process provides the degree of membership of each vector with respect to the different class labels. The fuzzification layer uses the membership function to calculate the input vectors and obtain the membership value μ. The membership function $\mu_{ij}(Vc_i)$ is usually taken as a Gaussian function, which can be calculated using Equation 11 [93].

$$\mu_{ij}(Vc_i) = \exp\left[- \frac{(Vc_i - C_{ij})^2}{X_{ij}} \right] \quad \textbf{(11)}$$

where Vc_i is the input vectors; $Vc = \{Vc_1, Vc_2, Vc_3, \ldots, Vc_i\}$, C_{ij} and X_{ij} are the center (mean) and width (standard deviation) of the Gaussian membership function $\mu_{ij}(Vc_i)$, respectively, of the j^{th} term (fuzzy set) of the i^{th} input vector of Vc_i.

To apply the Gaussian membership function, we use fuzzy c-means (FCM) algorithm. FCM algorithm [94] is a clustering algorithm which allows each piece of data to belong to a particular cluster or category. The FCM is defined in Equation 12 as follows:

$$\sum_{j=1}^{k} \sum_{Vc_i \in C_j} u_{ij}^{m} (Vc_i - \mu_j)^2 \quad \textbf{(12)}$$

where u_{ij} is the fuzzy membership degree to which a vector Vc_i belongs to a cluster c_j, μ_j is the center of the cluster j, m is the fuzzy weight, and k is the number of vectors in the dataset. The variable u_{ij}^{m} is defined in Equation 13 as follows:

$$u_{ij}^{m} = \frac{1}{\sum_{l=1}^{k} \left(\frac{|Vc_i - c_j|}{|Vc_i - c_k|} \right)^{\frac{2}{m-1}}} \quad \textbf{(13)}$$

The outputs of the fuzzification layer neurons are used as input for the hidden layers of the neural network.

C. Layer 3: *Hidden Layer (Reasoning Layer)*

The hidden layer in a neural network layer that resides between input and output layers, where artificial neurons take in a set of fuzzified weighted inputs and produce an output through an activation function. In our case, the hidden neural network layer is fine-tuned and calibrated through the most popularly used training algorithm in FNN named backpropagation [95]. Backpropagation is a supervised learning classification algorithm, for training FNN by computing the gradient descent of the loss function with respect to each weight. It is an efficient method in the supervised training of FNN models where its main goal is to optimize the weights so that the intelligent model can learn to accurately map the inputs to the outputs. In other words, we want to find a set of weights that reduces the output of the loss function. The general scheme of the backpropagation training algorithm is shown in Figure 4.7 [96].

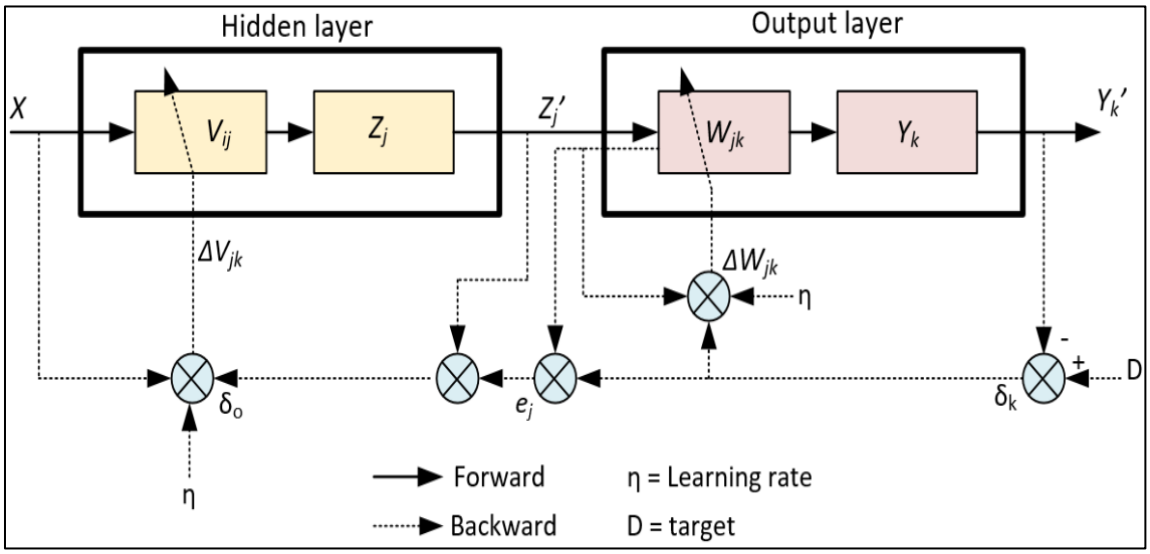

Figure 4.7: *The Scheme of Backpropagation Training Algorithm*

The backpropagation algorithm works in three stages as discussed in [96]:

- **Stage 1**: *Forward Propagation*, the input signal (X*i*) is propagated to the hidden screen using the specified activation function (i.e., Sigmoid**)**. The output of each unit

is a hidden screen (Z*j*) that are then propagated forward again to the output screen using the activation function determined. And so on to produce the neural network output (Y*k*). Next, the neural network output (Y*k*) is compared with the target (D). The difference D-Y*k* is the error that occurred. If this error is smaller than the specified tolerance limits, then the iteration is stopped. However, if the error is still greater than the tolerance limit, then the weight of each line in the network will be modified to reduce the errors that occurred.

- **Stage 2**: *Backward Propagation*, based on the error D-Y*k*, calculated factor δ*k* (k= 1, 2, ... , m) that is used to distribute the error in the unit Y*k* to all hidden units that are connected directly to the Y*k*. δk is also used to change the weightings that relate directly to the output unit. In the same way, count δ*o* factor in each unit on the screen is hidden by a basic change in the weight of all the lines, coming from the unit hidden in the screen below. It is until all the factors δ*o* in hidden units that are directly related to the input unit are calculated.
- **Stage 3**: *Change in weight*. After all, factors δ is calculated, the weight of all the lines simultaneously modified changes in weight of a line based on the factor δ neurons in the hidden screen. For example, changes in the weight of the lines leading to the output screen based on the existing δ*k* output unit. The third phase is repeated continuously until the termination conditions are met. Generally, the termination condition that is often used is the number of iterations or error. Iteration will be terminated if the number of iterations performed has exceeded the maximum number of iterations specified, or if the error that occurs is smaller than the allowable tolerance limits.

The general steps involved in backpropagation is described in Algorithm 4.1.

```
Assign all network inputs and output

Initialize all weights with small random numbers, typically
between -1 and 1

repeat

 for every pattern in the training set
   Present the pattern to the network
```

```
//Propagated the input forward through the network
    for each layer in the network
      for every node in the layer
         1. Calculate the weight sum of the inputs to the node
         2. Add the threshold to the sum
         3. Calculate the activation for the node
      end
    end

//Propagate the errors backward through the network
    for every node in the output layer
      calculate the error signal
    end

    for all hidden layers
      for every node in the layer
         1. Calculate the node's signal error
         2. Update each node's weight in the network
      end
    end

//Calculate error
    Calculate the Error Function

 end

while ((maximum  number of iterations < than specified) AND
       (Error Function is > than specified))
```

Algorithm 4.1: *Backpropagation Algorithm*

D. Layer 4: *Output Layer*

The output layer in a Fuzzy-Neuro Network is the last layer of neurons that produces a given output. This layer produces the classification result based on the backpropagation neural network algorithm. The number of neurons in the output layer is equal to the number of class labels available in the dataset. In our case, the two class labels are the possibilities of the input vector to be P2P botnet or Normal, respectively.

4.2.7 P2P Botnet Detector

In this phase, the P2P botnet detector examines and produces the classification result of the network traffic. The anomaly-based detector that consists of a set of FNN rules will classify the network traffic into one of the two categories as P2P botnet or normal, respectively. The

P2P botnet detector will classify the network traffic using the "*tf.keras.Model.predict*" TensorFlow built-in function as illustrated in Figure 4.8.

```
//We use tf.keras.Model.predict to infer labels on a batch or a
dataset of batches.
predictions = FNN_model.predict(test_data)


//Classify the network traffic into "P2P botnet" or "Normal".
for prediction, Label in zip(predictions[:10],
list(test_data)[0][1][:10]):
  print("Predicted P2P Botnet: {:.2%}".format(prediction[0]),
        " | Actual Outcome: ",
        ("P2P Botnet" if bool(Label) else "NORMAL"))
```

Figure 4.8: *TensorFlow Built-in Function to Classify Network Traffic*

To assess the performance of the P2P botnet detector in detecting malicious P2P network traffic, the testing set was used. The testing set consists of legitimate and different kinds of P2P botnet attack records such as fuzzers, information theft, DoS, exploits, generic, reconnaissance, shellcode and worms. At this stage, the anomaly-based P2P botnet detector is capable of classifying the network traffic as malicious or genuine.

4.3 Summary

In this chapter, we explained all components of our proposed P2P botnet detection model. Every component in our model is described in detail including the processes. As discussed, our proposed model comprises the following components: feature extractor and selector, dataset constructor, preprocessor, classifier, and P2P botnet detector. The feature extractor component extracts feature from the network traffic whereas the feature selector component selects vital features based on their information gain value. After that, the constructor splits the network traffic data into training and testing sets and constructs a dataset. Next, the preprocessor organizes the dataset by converting those mixed types to a fixed length vector before feeding the network traffic data into our classifier. Finally, the Fuzzy-Neuro Network

classifier classifies the network traffic as P2P botnet or normal using the botnet detector. The sample source code of the model is attached in Annex A. The next Chapter will discuss the experiment results and evaluation of our proposed P2P botnet detection model.

Chapter 5: Experimentation and Evaluation

5.1 Overview

In this Chapter, experimentation results and evaluation of our proposed model in detecting P2P botnets from a network traffic is presented. The set of experiments conducted to validate the proposed model is evaluated by using various evaluation metrics such as accuracy, precision, recall and F-measure.

Section 5.2 presents the experimental procedures which include the datasets we used for experimentation, feature extraction tools, feature selection results, libraries and tools used in the experiment, and the experimental setup. Following that, the evaluation results are discussed in Section 5.3. Finally, the discussion part in Section 5.4 briefs the P2P botnet detection results.

5.2 Experimental Procedure

In order to evaluate the performance of the proposed model, a series of experiments is conducted. In this section, we start by discussing the description of the dataset records used in the experiment followed by the feature selection results and experimentation tools.

5.2.1 Dataset Used

In our experiment, we used two publicly available P2P botnet datasets, Bot-IoT [80] and UNSW-NB15 [81, 82] to evaluate the proposed system. The datasets include both legitimate and malicious network traffics. The total network traffic records of the datasets were split into 70:30 ratio where 70% was used for training the FNN classifier while 30% was used for testing. The descriptions of the datasets were discussed earlier in Chapter 4 under Subsection 4.2.1. In this section, the network traffic records of the datasets used in the experimentation process are presented.

A. Bot-IoT Dataset

Bot-IoT dataset has a mixture of benign and attack network traffic records. The dataset consists of three types of P2P botnet attacks: probing attacks, denial of service attack and information theft. As shown in Table 5.1, we used a total of 320,202 network traffic records for the experimentation purpose. Among the records, 310,659 are botnet attacks and 9,543 are legitimate network traffics.

Table 5.1: *Statistics of Bot-IoT Records used for Experimentation*

	Category	Subcategory	Training Set	Testing Set
Normal Records (9,543)	Normal	-	6,679	2,864
Malicious Records (310,659)	Information gathering /Probing attacks/	Service scanning	50,998	22,170
		Operating system fingerprint	12,513	5,401
	Denial of Service	DDoS	82,942	35,347
		DoS	70,951	30,258
	Information theft	Keylogging	55	18
		Data theft	3	3
Total Records (320,202)		**Total**	**224,141**	**96,061**

Among the total 320,202 records, we used 70% (224,141 records) for training and 30% (96,061 records) for testing.

B. UNSW-NB15 Dataset

The UNSW-NB15 dataset has a mixture of real network traffic and synthesized attack activities. The dataset has nine categories of P2P botnet attack records namely: fuzzers,

analysis, backdoors, DoS, exploits, generic, reconnaissance, shellcode and worms. As shown in Table 5.2, we use a total of 259,673 network traffic records for the experimentation purpose. Among the records, 164,673 are botnet attacks and 95,000 are genuine network traffics.

Table 5.2: *Statistics of UNSW-NB15 Records used for Experimentation*

	Category	Training Set	Testing Set
Normal Records (95,000)	Normal	66,568	28,432
Malicious Records (164,673)	Backdoors	1,617	712
	Analysis	1,834	843
	Reconnaissance	9,822	4,165
	Shellcode	1,043	468
	Fuzzers	16,937	7,309
	Worms	123	51
	Generic	41,221	17,650
	DoS	11,442	4,911
	Exploits	31,164	13,361
Total Records (259,673)	**Total**	**181,771**	**77,902**

Among the total 259,673 records, we used 70% (181,771 records) for training and 30% (77,902 records) for testing.

5.2.2 Feature Extraction Tools

In the experimentation, we have used a feature extraction application named *CICFlowMeter* [85, 86] to extract features from the datasets discussed in Chapter 4 under Subsection 4.2.1. The tool extract useful network traffic features from static PCAP file and deliver the result in CSV format.

➢ CICFlowMeter

CICFlowMeter is used to extract features from the botnet datasets static PCAP files. The application is written in Java and gives us more flexibility in terms of choosing the features we want to extract. The working environment of CICFlowMeter feature extractor that we used is shown in Figure 5.1.

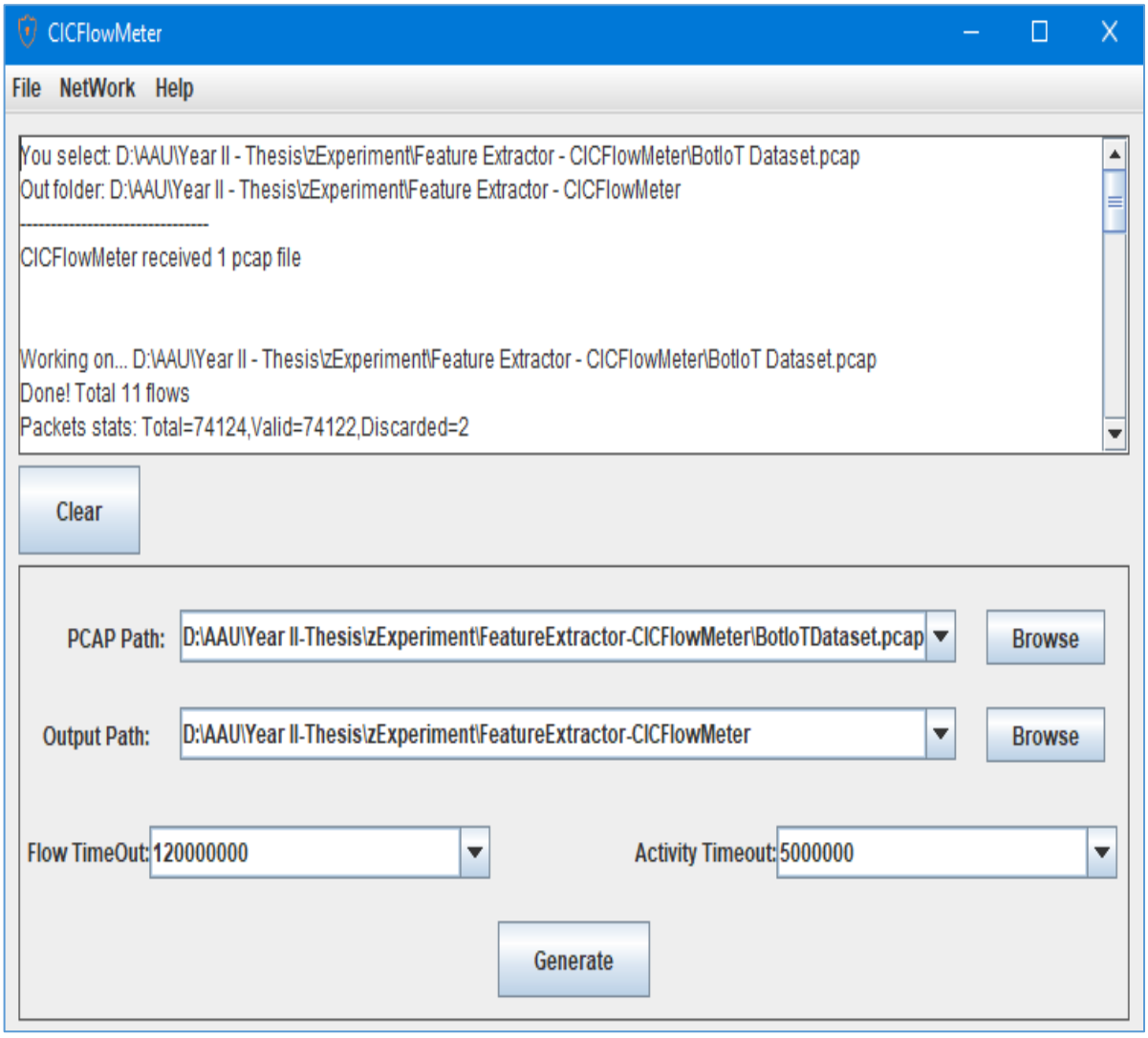

Figure 5.1: *CICFlowMeter Feature Extractor*

The features extracted from Bot-IoT (48 features) and UNSW-NB15 (53 features) are presented in Annex B [80] and Annex C [81, 82] respectively.

5.2.3 Feature Selection Results

As discussed earlier in Chapter 4 under Subsection 4.2.3, we apply one of the best-known filter feature selection technique named IG to choose the best features. We used Weka data mining tool to apply the IG feature selection approach.

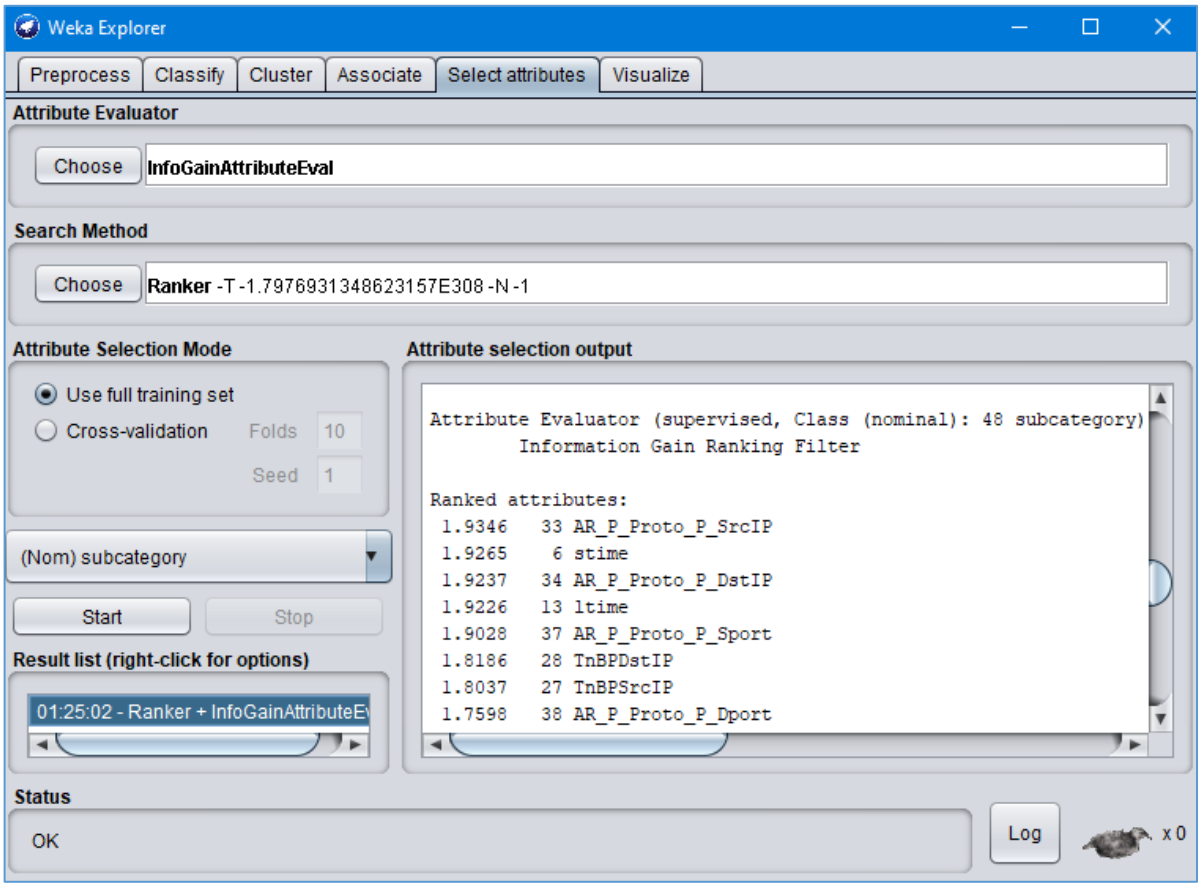

Figure 5.2: *Information Gain Feature Ranking in Weka*

After examining the IG value of the features in Weka as shown in Figure 5.2, we selected the top ranked features into our final feature list and discarded the low ranked features. The feature ranking results are presented as follows:

- **Information Gain Result:** *Bot-IoT dataset*

Based on IG, 35 (including F1 to F4) of the original 48 features (see Annex B) with gains greater than 0.9 were selected for the classification purpose. We set the cutting point to be 0.9 because IG evaluates the importance of a given feature by analyzing its entropy (how much information it provides). The higher the entropy the more information it contains. Table 5.3

shows the significance of features ranked by the IG approach, where features F33, F6, F34, F13, F37, F28, F27, F38, F10, F24, F14, F22, F32, F25, F31, F30, F39, F19, F17, F15, F29, F20, F40, F48, F9, F16, F11, F12, F47, F46 and F18 can be used for distinguishing between infected and genuine connections, and the remaining features are discarded. However, features F1 to F4 are automatically selected because they are basic features.

Table 5.3: *Feature Ranking of Bot-IoT Dataset*

Feature	Rank	Feature	Rank	Feature	Rank	Feature	Rank	Feature	Rank
F33	1.935	**F14**	1.735	**F29**	1.405	**F18**	0.948	F44	0.020
F6	1.927	**F22**	1.729	**F20**	1.395	F5	0.790	F41	0.020
F34	1.924	**F32**	1.690	**F40**	1.354	F8	0.623	F45	0.020
F13	1.923	**F25**	1.654	**F48**	1.354	F7	0.623	F43	0.020
F37	1.903	**F31**	1.635	**F9**	1.176	F23	0.545		
F28	1.819	**F30**	1.538	**F16**	1.106	F21	0.444		
F27	1.804	**F39**	1.449	**F11**	1.058	F35	0.408		
F38	1.760	**F19**	1.442	**F12**	1.058	F36	0.364		
F10	1.748	**F17**	1.441	**F47**	1.044	F26	0.247		
F24	1.744	**F15**	1.409	**F46**	1.044	F42	0.020		

- **Information Gain Result:** *UNSW-NB15 dataset*

Based on IG, 33 (including F1 to F4) of the original 53 features (see Annex C) with gains greater than 0.9 were selected for the classification purpose. We set the cutting point to be 0.9 because IG evaluates the importance of a given feature by analyzing its entropy (how much information it provides). The higher the entropy the more information it contains. Table 5.4 shows the significance of features ranked by the IG approach, where features F18, F9, F8, F25, F31, F7, F36, F15, F19, F16, F24, F11, F17, F30, F6, F5, F29, F12, F32, F33, F34, F23, F21, F22, F53, F13, F20, F28 and F52 can be used for distinguishing between infected and genuine connections, and the remaining features are discarded. However, features F1 to F4 are automatically selected because they are basic features.

Table 5.4: *Feature Ranking of UNSW-NB15 Dataset*

Feature	Rank	Feature	Rank	Feature	Rank	Feature	Rank	Feature	Rank
F18	1.300	**F24**	1.065	**F34**	0.956	F10	0.453	F27	0.078
F9	1.297	**F11**	1.049	**F23**	0.947	F14	0.448	F35	0.016
F8	1.282	**F17**	1.027	**F21**	0.946	F44	0.387	F38	0.014
F25	1.242	**F30**	1.006	**F22**	0.946	F46	0.359	F39	0.014
F31	1.241	**F6**	0.988	**F53**	0.943	F41	0.306	F51	0.010
F7	1.217	**F5**	0.988	**F13**	0.941	F40	0.291	F50	0.010
F36	1.206	**F29**	0.981	**F20**	0.935	F42	0.242	F49	0.010
F15	1.193	**F12**	0.970	**F28**	0.934	F43	0.226	F48	0.010
F19	1.178	**F32**	0.966	**F52**	0.925	F37	0.120	F47	0.010
F16	1.164	**F33**	0.963	F45	0.463	F26	0.119		

5.2.4 Development and Experimentation Tools

In this section, experimentation tools and development languages that we used for implementation and evaluation of the proposed model are described. For the development of different components, we have used various tools and libraries. The tools, libraries and languages with their respective description are discussed in Table 5.5.

Table 5.5: *List of Development and Experimentation Tools*

No.	Tool	Description	Version
1.	Anaconda	Anaconda is an open source distribution of the Python programming language for scientific computing that aims to simplify package management and deployment. In our experiment, we used it to install and execute all the libraries.	3.0.0
2.	Python	Python is a high-level general-purpose programming language. In our experiment, we used it to write our program (code).	3.7.6

No.	Tool	Description	Version
3.	TensorFlow	TensorFlow is a library used for programming across a range of tasks. It is a symbolic math library which is used for Machine Learning applications, in our case, neural networks.	2.1.0
4.	Keras	Keras is a neural network library written in Python. It is capable of running on top of TensorFlow.	2.3.1
5.	Matplotlib	Matplotlib is a figure plotting library for the Python programming language. In our case, it is used to plot the experimentation results.	3.2.0
6.	NumPy	It's an array processor for numbers, strings and records.	1.18.1
7.	Pandas	Pandas is a Python library used for data structure and analysis.	1.0.1
8.	Sklearn	Sklearn is a Python library used to split the dataset into random train and test subsets.	0.22.1
9.	Jupyter Lab	Web-based interactive computing environments that are used to create live code, visualizations and narrative texts.	2.0.1
10.	Jupyter Notebook		6.0.3
11.	CICFlowMeter	It's a feature extraction tool written in java.	3.0.0
12.	IntelliJ IDEA	Editing software used to run the CICFlowMeter.	2020.1.2
13.	Wireshark	Network traffic packets analyzer which is used to read the PCAP files.	3.2.2
14.	Weka	A popular Machine Learning and data mining tool. We used it to select the best features using IG.	3.8.4

5.2.5 Experimental Setup

The experimental setting of the development environment that is used in applying our proposed model is carried out on a laptop computer. The experiments are conducted on Intel Core (TM) i7-7500U CPU with 2.90 GHz speed processor, 8 GB RAM, 1 TB HDD capacity and 64-bit Windows 10 operating system. All necessary packages listed in Table 5.5 were installed and configured for the development and testing of the proposed model.

5.3 Performance Evaluation

This section covers the overall experimentation results of our proposed P2P botnet detection model. First, we will discuss the performance evaluation result of our model in Subsection 5.3.1. Then, in Subsection 5.3.2, we will discuss the binary classification (confusion matrix) result of the classifier. Finally, in Subsection 5.3.3, we will compare the evaluation result of our model with existing researches conducted on P2P botnet detection. In order to evaluate the performance of our model, we have used the evaluation metrics discussed earlier in Chapter 2 under Subsection 2.7.2.

5.3.1 Evaluation Result

In this subsection, the evaluation results are presented and the performance of our model is assessed using the evaluation metrics such as accuracy, precision, recall and F-measure in order to validate its feasibility. Getting very high accuracy somehow can be achieved by thoroughly selecting the sample size of the dataset. So, if we use accuracy alone to evaluate the performance of our model, the result can be subjective and it cannot give us the desired performance result. For this reason, precision and recall performance evaluation measures which are not dependent on the sample size of the testing and training sets are also considered. Precision and recall are two very vital model evaluation metrics. They are to some extent inversely proportional to each other. Precision refers to the percentage of results which are relevant, whereas recall refers to the percentage of total relevant results correctly classified by our model. Hence, we also use one more evaluation measure named F-measure which is a harmonic mean of precision and recall.

To improve the performance of the ANN, a fuzzy logic is added onto it. Table 5.6 shows the accuracy, precision, recall, and F-measure results of the ANN classifier before the fuzzy logic technique is applied onto it. As can be seen in Tables 5.6 and 5.7, the performance of the ANN classifier trained without fuzzy logic is lower than the performance gained by incorporating fuzzy logic technique onto it.

Table 5.6: *Performance Evaluation Result of ANN Classifier without Fuzzy Logic*

	Classifier - ANN			
Dataset	**Accuracy** ***(Average)***	**Precision** ***(Average)***	**Recall** ***(Average)***	**F-measure** ***(Average)***
Bot-IoT	96.4%	96.7%	96.6%	96.2%
UNSW-NB15	96%	96.3%	96.2%	96%

Table 5.7 shows the average accuracy, precision, recall and F-measure evaluation results of the FNN classifier.

Table 5.7: *Performance Evaluation Result of ANN Classifier with Fuzzy Logic*

	Classifier - FNN			
Dataset	**Accuracy** ***(Average)***	**Precision** ***(Average)***	**Recall** ***(Average)***	**F-measure** ***(Average)***
Bot-IoT	100%	100%	100%	100%
UNSW-NB15	99.9%	99.9%	100%	99.9%

As we can see in Table 5.7, using the Bot-IoT dataset, the average detection accuracy, precision, recall and F-measure rate of our model was 100% for all evaluation metrics. Whereas, using the UNSW-NB15 dataset, the proposed model has an average highest detection accuracy, precision, recall and F-measure rates respectively at 99.9%, 99.9%, 100 % and 99.9%. It is clear from Table 5.7 that the proposed model has almost the same performance result using both P2P botnet datasets. Overall, after considering the results obtained using the two datasets, Bot-IoT and UNSW-NB15, it can be concluded that the proposed system has a superior performance result.

5.3.2 Binary Classification

We have evaluated our classification technique using two-dimensional (2x2) confusion matrix. Figure 5.3 illustrates the confusion matrix result of our FNN classifier for the two-class classification. It shows the TP, TN, FP and FN results of our classifier after tested using Bot-IoT and UNSW-NB15 test sets. In the confusion matrix, class P2P botnet represent positive instances whereas normal represents negative instances.

Bot-IoT Test Set

		Predicted Class	
		Normal	**P2P Botnet**
Actual Class	**Normal**	**TN** 2,864	**FP** 0
	P2P Botnet	**FN** 0	**TP** 93,197

UNSW-NB15 Test Set

		Predicted Class	
		Normal	**P2P Botnet**
Actual Class	**Normal**	**TN** 28,427	**FP** 5
	P2P Botnet	**FN** 0	**TP** 49,470

Figure 5.3: *Confusion Matrix for our Classifier using the Test Sets*

The confusion matrix in Figure 5.3 shows the actual and predicted classes of the test sets. In Bot-IoT test set, among the total (96,061) positive and negative instances in the test set, the model correctly classified 93,197 instances as positive (TP) and 2,864 samples as negative (TN) which is 100%. Accordingly, the model gained zero FP and FN results. Whereas using UNSW-NB15 test set, among the total (49,470) positive instances in the test set, all are correctly classified as positive (TP) by the model which is 100% and from the total (28,432) negative instances in the test set, the model correctly classified 28,427 samples as negative (TN) which is 99.9%. The model gained a high number of TP and TN results with small FP (5) and zero FN results.

5.3.3 Comparison with Existing Work

The overall performance comparison of our proposed botnet detection model with that of some methods is shown in Table 5.8. The comparison result shows that our proposed model attains better botnet detection accuracy than the existing methods in [8, 13, 18]. It has high detection accuracy rate as compared to existing methods. As shown in Table 5.8, the classification accuracy rates of DT [8], CNN [13], NB [18] and SVM [18] were 99.78%, 98.60%, 99.14% and 92.02%, respectively. The comparison result shows that the proposed FNN classifier has better detection accuracy rate than other classification methods. As shown in Table 5.8, the detection accuracy rate of the proposed system is 99.9%, which, on average, indicates a 0.12% improvement in comparison to the best value obtained in [8].

Table 5.8: *Performance Comparison with Existing Techniques*

Technique	Year Published	Classifier	Accuracy (%)
Singh *et al.* [8]	2019	DT	99.78
Chen *et al.* [13]	2018	CNN	98.60
Kirubavathi *et al.* [18]	2017	NB	99.14
		SVM	92.02
Proposed model	**-**	**FNN**	**99.90**

5.4 Discussion

The proposed solution has been analyzed through experimentation to evaluate its performance in detecting P2P botnets. The experimental results show that our proposed model can successfully distinguish P2P botnet traffic from normal ones with high detection accuracy rate. We compared our model with researches proposed in [8, 13, 18]. The performance comparison result of our model to those published approaches was shown earlier in Table 5.8. From our analysis, it has been demonstrated that based on the average detection accuracy rate, the proposed FNN classification technique performs better in comparison with other classification algorithms such as DT [8], CNN [13], NB [18] and SVM [18]. Table 5.8 shows the P2P botnet detection accuracy rate of the classifiers, where FNN has the highest accuracy rate (99.9%) using UNSW-NB15 dataset, followed by DT (99.78%), CNN (98.6%) and NB (99.14%). Among the classifiers, SVM had the least accuracy rate of 92.02%. Based on these comparison results, we can clearly notice that our model has better detection accuracy than the studies in references [8, 13, 18] in P2P botnet detection.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

Peer-to-Peer botnets have recently been identified as one of the serious threats against network security due to their distributed and evolving nature. They are utilized by attackers to perform numerous criminal operations such as DDoS, keylogging, spamming, traffic sniffing, phishing and so forth. In recent years, lots of researchers have proposed a number of P2P botnet detection methods, but due to the evolving nature of botnets, there is still a need for new techniques to identify recent botnets. In this thesis, we presented a model that is able to detect malicious P2P botnets through network traffic analysis. The main aim of this research work is to build an efficient model that is able to classify network traffic records into P2P botnet and normal using FNN.

The proposed system architecture is composed of six major components namely: Feature Extractor, Feature Selector, Dataset Constructor, Preprocessor, Classifier and P2P Botnet Detector. The feasibility of our proposed model has been validated through experiments using network traffic records acquired from two publicly available P2P botnet datasets: Bot-IoT and UNSW-NB15. Our model is implemented in Python programming language using these datasets. The implementation started with extracting features from the datasets and based on their IG values the best features are selected. Then comes splitting the dataset and preprocessing the training and testing sets. During classification, we have created and trained our model. Finally, the FNN classifier is utilized to classify the network traffic into P2P botnet and normal.

The evaluation result shows the proposed model is effective in detecting P2P botnets. Based on the evaluation results of our classifier, using Bot-IoT dataset, the model scored 100% for all evaluation metrics. Whereas, using the UNSW-NB15 dataset, the model scored highest classification accuracy of 99.9%, precision of 99.9% and recall of 100% with F-measure rate of 99.9%. The performance comparison of the proposed model with existing P2P botnet

detection techniques shows that the proposed FNN classification technique performs better in comparison with other classification methods such as DT [8], CNN [13], NB [18] and SVM [18]. From the comparison result, we can conclude that our proposed model has better detection rate and performance than other researches in P2P botnet detection.

6.2 Contribution

In general, this research has the following contributions:

- Incorporate the concept of fuzzy logic into the Artificial Neural Network for P2P botnet detection.
- Improvement in the detection accuracy of previous researches related to P2P botnet detection.

6.3 Future Work

This research work explores different points that can be further improved in future researches. The following are promising research directions beyond the scope of this study.

- Working on different techniques to prevent P2P botnet attacks.
- Exploring various feature selection approaches other than IG for choosing the best subset of features.
- Implementing real time detection of P2P botnet attacks.
- Furthermore, applying the system on different P2P botnet datasets can be considered in future studies.

References

[1] H. Dhayal and J. Kumar, “Botnet and P2P Botnet Detection Strategies: A Review,” in *2018 7th IEEE International Conference on Communication and Signal Processing (ICCSP)*, pp. 1077-1082, Chennai, 2018.

[2] V. Kant, E. M. Singh and N. Ojha, "An efficient flow based botnet classification using convolution neural network," *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 941-946, Madurai, 2017.

[3] Böck, L., Vasilomanolakis, E., Wolf, J., and Mühlhäuser, M. “Autonomously detecting sensors in fully distributed botnets,” in *Science Direct Computers and Security*, Vol. 83, pp.1-13, 2019.

[4] A. Kapre and B. Padmavathi, "Adaptive behavior pattern-based botnet detection using traffic analysis and flow intervals," *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, pp. 410-414, Coimbatore, 2017.

[5] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "BotViz: A memory forensic-based botnet detection and visualization approach," *2017 International Carnahan Conference on Security Technology (ICCST)*, Madrid, pp. 1-8, 2017.

[6] P. M. Pondkule and B. Padmavathi, “BotShark — Detection and prevention of peer-to-peer botnets by tracking conversation using CART,” *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, pp. 291-295, Coimbatore, 2017.

[7] S. Maeda, A. Kanai, S. Tanimoto, T. Hatashima, and K. Ohkubo, "A Botnet Detection Method on SDN using Deep Learning," *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1-6, Las Vegas, NV, USA, 2019.

[8] M. Singh, M. Singh, and S. Kaur, “Detecting bot-infected machines using DNS fingerprinting”, in *Science Direct Digital Investigation*, Vol. 28, pp. 14-33, 2019.

[9] L. Mathur, M. Raheja, and P. Ahlawat, “Botnet Detection via mining of network traffic flow”, in *Science Direct Procedia Computer Science*, Vol. 132, pp. 1668-1677, 2018.

[10] H. Xia, “Research on Bot-Net Prevention and Control Technology Based on P2P,” *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pp. 1986-1990, Xi'an, 2018.

[11] P. Wang, F. Wang, F. Lin, and Z. Cao, “Identifying Peer-to-Peer Botnets Through Periodicity Behavior Analysis,” *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pp. 283-288, New York, NY, 2018.

[12] T. Wang, H. Lin, W. Cheng and C. Chen, “DBod: Clustering and detecting DGA-based botnets using DNS traffic analysis”, *Computers and Security*, Vol. 64, pp. 1-15, 2017.

[13] S. Chen, Y. Chen, and W. Tzeng, “Effective Botnet Detection Through Neural Networks on Convolutional Features,” *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, pp. 372-378, New York, NY, 2018.

[14] X. Li, J. Wang and X. Zhang, "Botnet Detection Technology Based on DNS", *Future Internet*, Vol. 9, No. 4, pp. 55, 2017.

[15] H. A. Madni, Z. Anwar, and M. A. Shah, "Data mining techniques and applications — A decade review," *2017 23rd International Conference on Automation and Computing (ICAC)*, Huddersfield, pp. 1-7, 2017.

[16] J. Álvarez Cid-Fuentes, C. Szabo, and K. Falkner, "An adaptive framework for the detection of novel botnets", *Computers and Security*, Vol. 79, pp. 148-161, 2018.

[17] M. Singh, M. Singh, and S. Kaur, "Issues and challenges in DNS based botnet detection: A survey", *Computers and Security*, Vol. 86, pp. 28-52, 2019.

[18] G. Kirubavathi and R. Anitha, "Botnet detection via mining of traffic flow characteristics", *Computers and Electrical Engineering*, Vol. 50, pp. 91-101, 2017.

[19] S. Jin, Y. Jiang, and J. Peng, “Intrusion Detection System Enhanced by Hierarchical Bidirectional Fuzzy Rule Interpolation,” *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 6-10, Miyazaki, Japan, 2018.

[20] M. Komar, A. Sachenko, V. Kochan, and T. Skumin, “Increasing the resistance of computer systems towards virus attacks,” *2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO)*, pp. 388-390, Kiev, 2017.

[21] J. Lu, F. Lv, Q. Liu, M. Zhang, and X. Zhang, "Botnet Detection based on Fuzzy Association Rules," *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 578-584, Beijing, 2018.

[22] I. Winkler, A. Gomes, and D. Shackleford, Advanced Persistent Security. Amsterdam: Elsevier Syngress Publisher, pp. 72-73, 2017.

[23] B. Sangita, “Botnet Detection: Analysis of Various Techniques”, *International Journal of Computational Intelligence and IoT*, Vol. 2, pp. 461-467, 2019.

[24] P. Panimalar and K. Rameshkumar, "A review on taxonomy of botnet detection," *2014 International Conference on Advances in Engineering and Technology (ICAET)*, Nagapattinam, pp. 1-4, 2014.

[25] N. Kaur and M. Singh, "Botnet and botnet detection techniques in cyber realm," *2016 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, pp. 1-7, 2016.

[26] M. Osagie, O. Enagbonma, and A. Inyang, "The Historical Perspective of Botnet Tools", *Current Journal of Applied Science and Technology*, pp. 1-8, 2019.

[27] A. Nourian and S. Madnick, "A Systems Theoretic Approach to the Security Threats in Cyber Physical Systems Applied to Stuxnet," in *IEEE Transactions on Dependable and Secure Computing*, Vol. 15, No. 1, pp. 2-13, Jan.-Feb. 2018.

[28] T. Lange and H. Kettani, "On Security Threats of Botnets to Cyber Systems," *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, India, pp. 176-183, 2019.

[29] X. Dong, J. Hu, and Y. Cui, "Overview of Botnet Detection Based on Machine Learning," *2018 3rd International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, Huhhot, pp. 476-479, 2018.

[30] G. Vormayr, T. Zseby, and J. Fabini, "Botnet Communication Patterns," in *IEEE Communications Surveys and Tutorials*, Vol. 19, No. 4, pp. 2768-2796, 2017.

[31] “What Is DDOS Attack? What Is Botnets?”, *Crackitdown- Learn Ethical Hacking and more, 2020.* [Online]. Available: https://www.crackitdown.com/2017/11/what-is-ddos-attack.html. [Accessed: 05- Jan- 2020].

[32] B. Nagpal, P. Sharma, N. Chauhan, and A. Panesar, "DDoS tools: Classification, analysis and comparison," *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, pp. 342-346, 2015.

[33] A. Wang, W. Chang, S. Chen, and A. Mohaisen, "Delving Into Internet DDoS Attacks by Botnets: Characterization and Analysis," in *IEEE/ACM Transactions on Networking*, Vol. 26, No. 6, pp. 2843-2855, Dec. 2018.

[34] S. Akbar, Endroyono, and A. D. Wibawa, "The impact analysis and mitigation of DDoS attack on local government electronic procurement service (LPSE)," *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Lombok, pp. 405-410, 2016.

[35] M. Guri, Y. Mirsky, and Y. Elovici, "9-1-1 DDoS: Attacks, Analysis and Mitigation," *2017 IEEE European Symposium on Security and Privacy (EuroSandP)*, Paris, pp. 218-232, 2017.

[36] "Botnets and Their Types | EC-Council Official Blog", EC-Council Official Blog, 2020. [Online]. Available: https://blog.eccouncil.org/botnets-and-their-types/. [Accessed: 05- Jan-2020].

[37] C. Meda, E. Ragusa, C. Gianoglio, R. Zunino, A. Ottaviano, and E. Scillia, "Spam detection of Twitter traffic: A framework based on random forests and non-uniform feature sampling," 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), San Francisco, CA, pp. 811-817, 2016.

[38] A. Basset, C. Beek, and N. Minihane, McAfee Labs Threats Report, 2020. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-mar-2018.pdf. [Accessed: 05- Jan- 2020].

[39] "The Necurs Botnet: A Pandora's Box of Malicious Spam", Security Intelligence, 2020. [Online]. Available: https://securityintelligence.com/the-necurs-botnet-a-pandoras-box-of-malicious-spam/. [Accessed: 05- Jan- 2020].

[40] J. V. Monaco, "SoK: Keylogging Side Channels," *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, pp. 211-228, 2018.

[41] S. Samtani and H. Chen, "Using social network analysis to identify key hackers for keylogging tools in hacker forums," *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, Tucson, AZ, pp. 319-321, 2016.

[42] S. Dutta, A. K. Gupta, and N. Narayan, "Identity Crime Detection Using Data Mining," *2017 3rd International Conference on Computational Intelligence and Networks (CINE)*, Odisha, pp. 1-5, 2017.

[43] "Types of Botnet Attacks", Jpdias.me, 2020. [Online]. Available: https://jpdias.me/botnet-lab//anatomy/types-of-attacks.html. [Accessed: 05- Jan- 2020].

[44] X. Li, Y. Liu, and D. Zeng, "Publisher click fraud in the pay-per-click advertising market: Incentives and consequences," *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, Beijing, pp. 207-209, 2011.

[45] B. Kitts, J. Y. Zhang, G. Wu, and R. Mahato, "Click fraud botnet detection by calculating mix adjusted traffic value: A method for de-cloaking click fraud attacks that is resistant to spoofing," *2013 IEEE International Conference on Intelligence and Security Informatics*, Seattle, WA, pp. 151-153, 2013.

[46] J. Yang, Y. Zhang, R. King, and T. Tolbert, "Sniffing and Chaffing Network Traffic in Stepping-Stone Intrusion Detection," *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Krakow, pp. 515-520, 2018.

[47] M. Alauthman, N. Aslam, M. Al-kasassbeh, S. Khan, A. Al-Qerem, and K. Raymond Choo, "An efficient reinforcement learning-based Botnet detection approach", *Journal of Network and Computer Applications*, Vol. 150, pp. 1-16, 2020.

[48] N. S. Raghava, D. Sahgal, and S. Chandna, "Classification of Botnet Detection Based on Botnet Architecture," *2012 International Conference on Communication Systems and Network Technologies*, Rajkot, pp. 569-572, 2012.

[49] J. Vania, A. Meniya, and H. Jethva, "A Review on Botnet and Detection Technique," *International Journal of Computer Trends and Technology*, Vol. 4, No. 1, pp. 23-29, 2013.

[50] K. Alieyan, A. Almomani, M. Anbar, M. Alauthman, R. Abdullah, and B. Gupta, "DNS rule-based schema to botnet detection", *Enterprise Information Systems*, pp. 1-20, 2019.

[51] Z. Chen, X. Yu, Y. Ling, B. Song, W. Quan, X. Hu, and E. Yan, "Correlated Anomaly Detection from Large Streaming Data," *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, pp. 982-992, 2018.

[52] L. Rasyid and S. Andayani, "Review on Clustering Algorithms Based on Data Type: Towards the Method for Data Combined of Numeric-Fuzzy Linguistics", Journal of Physics: Conference Series, Vol. 1097, pp. 1-8, 2018.

[53] R. Sharma and A. Thakral, "Identifying Botnets: Classification and Detection", *International Journal of Innovative Technology and Exploring Engineering*, Vol. 8, No. 9, pp. 131-137, 2019.

[54] "Botnet detection", Jpdias.me, 2020. [Online]. Available: https://jpdias.me/botnet-lab//countermeasures/detection.html. [Accessed: 10- Jan- 2020].

[55] S. Asha, T. Harsha, and B. Soniya, "Analysis on botnet detection techniques," *2016 International Conference on Research Advances in Integrated Navigation Systems (RAINS)*, Bangalore, pp. 1-4, 2016.

[56] D. Feng, *Biomedical information technology*, 2nd ed. Amsterdam, Netherlands: Elsevier Inc., p. 26, 2020.

[57] N. Kulkarni and V. Bairagi, *EEG-based diagnosis of Alzheimer disease*, 1st ed. Amsterdam, Netherlands: Elsevier Inc., pp. 45-47, 2018.

[58] X. Hoang and Q. Nguyen, "Botnet Detection Based On Machine Learning Techniques Using DNS Query Data", *Future Internet*, Vol. 10, No. 5, pp. 43-45, 2018.

[59] D. Wu, B. Fang, J. Wang, Q. Liu, and X. Cui, "Evading Machine Learning Botnet Detection Models via Deep Reinforcement Learning," *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, pp. 1-6, 2019.

[60] G. Shobha and S. Rangaswamy, "Machine Learning", *Handbook of Statistics*, pp. 197-228, 2018.

[61] M. Asadi, M. Jabraeil Jamali, S. Parsa, and V. Majidnezhad, "Detecting botnet by using particle swarm optimization algorithm based on voting system", *Future Generation Computer Systems*, Vol. 107, pp. 95-111, 2020.

[62] M. Talabis, R. McPherson, and J. Martin, *Information security analytics*. Amsterdam: Elsevier, 2015.

[63] M. Aamir and S. Zaidi, "Clustering based semi-supervised Machine Learning for DDoS attack classification", *Journal of King Saud University - Computer and Information Sciences*, 2019.

[64] S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, *Encyclopedia of bioinformatics and computational biology*. Amsterdam: Elsevier, 2019.

[65] J. Amrutha and A. S. Remya Ajai, "Performance analysis of Backpropagation Algorithm of Artificial Neural Networks in Verilog," *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT)*, Bangalore, India, pp. 1547-1550, 2018.

[66] J. Pagel and P. Kirshtein, *Machine dreaming and consciousness*. Amsterdam: Elsevier, pp. 83-92, 2017.

[67] K. P. Korshunova, "A Convolutional Fuzzy Neural Network for Image Classification," *2018 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC)*, Vladivostok, pp. 1-4, 2018.

[68] C. Bird, T. Menzies, and T. Zimmermann, *The art and science of analyzing software data*. Waltham, MA: Morgan Kaufmann, 2015.

[69] V. Gandhi, *Brain-computer interfacing for assistive robotics*. Amsterdam: Elsevier, 2016.

[70] M. M. Sakr, M. A. Tawfeeq, and A. B. El-Sisi, "Filter Versus Wrapper Feature Selection for Network Intrusion Detection System," *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, Cairo, Egypt, pp. 209-214, 2019.

[71] M. S. S. Sumi and A. Narayanan, "Improving Classification Accuracy Using Combined Filter + Wrapper Feature Selection Technique," *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore, India, pp. 1-6, 2019.

[72] S. Maza and M. Touahria, "Feature Selection Algorithms in Intrusion Detection System: A Survey," *KSII Transactions on Internet and Information Systems*, Vol. 12, No. 10, pp. 5079-5099, 2018.

[73] S. Kaushik, "Feature Selection methods with example", *Analytics Vidhya*, 2020. [Online]. Available: https://www.analyticsvidhya.com/introduction-to-feature-selection-methods. [Accessed: 25- Jun- 2020].

[74] K. R. Pushpalatha and A. G. Karegowda, "CFS Based Feature Subset Selection for Enhancing Classification of Similar Looking Food Grains- A Filter Approach," *2017*

2nd International Conference On Emerging Computation and Information Technologies (ICECIT), Tumakuru, pp. 1-6, 2017.

[75] L. Gao, M. Ye, X. Lu, and D. Huang, "Hybrid Method Based on Information Gain and Support Vector Machine for Gene Selection in Cancer Classification", *Genomics, Proteomics and Bioinformatics*, Vol. 15, No. 6, pp. 389-395, 2017.

[76] A. Bakhshandeh and Z. Eskandari, "An efficient user identification approach based on Netflow analysis," *2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, Tehran, pp. 1-5, 2018.

[77] G. Khehra and S. Sofat, "BotScoop: Scalable Detection of DGA Based Botnets Using DNS Traffic," *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bangalore, pp. 1-6, 2018.

[78] C. Wang, C. Ou, Y. Zhang, F. Cho, P. Chen, J. Chang, and C. Shieh, "BotCluster: A session-based P2P botnet clustering system on NetFlow", in *Science Direct Computer Networks*, Vol. 145, pp. 175-189, 2018.

[79] D. Zhuang and J. M. Chang, "Enhanced PeerHunter: Detecting Peer-to-Peer Botnets Through Network-Flow Level Community Behavior Analysis," in *IEEE Transactions on Information Forensics and Security*, Vol. 14, No. 6, pp. 1485-1500, 2019.

[80] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset", *Future Generation Computer Systems*, Vol. 100, pp. 779-796, 2019.

[81] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, pp. 1-6, 2015.

[82] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set", *Information Security Journal: A Global Perspective*, Vol. 25, No. 1-3, pp. 1-14, 2016.

[83] "NetFlow Basics: An Introduction to Monitoring Network Traffic", *Auvik Networks Inc.*, 2020. [Online]. Available: https://www.auvik.com/franklymsp/blog/netflow-basics/. [Accessed: 15- Jun- 2020].

[84] R. Kozik and M. Choraś, "Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection", *Security and Communication Networks*, Vol. 2017, pp. 1-10, 2017.

[85] G. Drapper-Gil, A. Habibi, M. Saiful, and A. Ghorbani, "Characterization of Encrypted and VPN Traffic Using Time-Related Features", *In the proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, Italy, pp. 407-414, 2016.

[86] A. Habibi, G. Draper-Gil, M. Saiful, and A. Ghorbani, "Characterization of Tor Traffic Using Time Based Features", *In the proceedings of the 3rd International Conference on Information System Security and Privacy*, Portugal, pp. 253-262, 2017.

[87] I. Witten, E. Frank, and M. Hall, *The WEKA Workbench. Online Appendix for Data Mining : Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, Calif.: Morgan Kaufmann, 2016.

[88] "Sklearn: Split data into random train and test subsets", 2020. [Online]. Available: https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed: 22- Jul- 2020].

[89] "TensorFlow Core Make CSV Dataset v2.2.0", 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/data/experimental/make_csv_dataset. [Accessed: 21- Jul- 2020].

[90] S. Misra, H. Li, and J. He, *Machine Learning for subsurface characterization*. Amsterdam, Netherlands: Elsevier Inc., pp. 129-130, 2020.

[91] "Normalization | Codecademy", *Codecademy*, 2020. [Online]. Available: https://www.codecademy.com/articles/normalization. [Accessed: 26- Jul- 2020].

[92] E. Kayacan, M. Khanesar, and J. Mendel, *Fuzzy neural networks for real-time control applications.* Amsterdam, Netherlands: Elsevier Inc., pp. 13-14, 2017.

[93] L. Kaiju, L. Xuefeng, M. Chaoxu, and W. Dan, "Short-term photovoltaic power prediction based on T-S fuzzy neural network," *2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Nanjing, pp. 620-624, 2018.

[94] C. Ming-Xia, Z. Han, and L. Shun-Yan, "An Intrusion Detection Scheme Combining FCM and Kohonen Network," *2019 11th International Conference on Measuring*

Technology and Mechatronics Automation (ICMTMA), Qiqihar, China, pp. 239-243, 2019.

[95] P. de Campos Souza, "Fuzzy neural networks and neuro-fuzzy networks: A review the main techniques and applications used in the literature", *Applied Soft Computing*, Vol. 92, pp. 1-18, 2020.

[96] K. Kuspijani, R. Watiasih, and P. Prihastono, "Faults Identification of Induction Motor Based on Vibration Using Backpropagation Neural Network," *2020 International Conference on Smart Technology and Applications (ICoSTA)*, Surabaya, Indonesia, pp. 1-5, 2020.

Annex A: Sample Source Code of the Model

Setup the Environment

In [2]:
```
from__future_____import absolute_import, division, print_function, unicode_literals
import functools

import numpy as np
import tensorflow as tf
import pandas as pd
```

Importing Dataset - 'Total Set' -

In [3]:
```
# --- Import the dataset. --- #
UNSW_NB15_DATASET_CSV = pd.read_csv ('D:\\AAU\\Year II -Thesis\\zExperiment\\.UNSW-

# --- Read the records of the dataset. --- #
UNSW_NB15_DATASET_CSV
```

Out[3]:

	Label	srcip	sport	dstip	dsport	proto	state	dur	dbytes	...
0	1	175.45.176.1	0	149.171.126.12	0	gre	INT	0.000009	0	...
1	1	175.45.176.1	0	149.171.126.12	0	ggp	INT	0.000005	0	...
2	1	175.45.176.1	0	149.171.126.12	0	prm	INT	0.000008	0	...
3	1	175.45.176.1	0	149.171.126.12	0	netblt	INT	0.000009	0	...
4	1	175.45.176.1	0	149.171.126.12	0	sdrp	INT	0.000009	0	...
...	...	...	...	...	...	...	...	...	...	...
259668	1	175.45.176.1	29092	149.171.126.15	80	tcp	FIN	1.518915	354	...
259669	1	175.45.176.0	4618	149.171.126.17	80	tcp	FIN	0.694885	268	...
259670	1	175.45.176.2	63237	149.171.126.12	80	tcp	FIN	8.734500	914758	...
259671	1	175.45.176.0	10010	149.171.126.10	80	tcp	FIN	0.605943	268	...
259672	1	175.45.176.0	54306	149.171.126.12	80	tcp	FIN	0.258671	268	...

259673 rows × 33 columns

1. Dataset Constructor

Task 1 - 'Splitting Dataset'

In [4]:
```
import sklearn.model_selection as model_selection
from sklearn.model_selection import train_test_split

# --- Splitting training set (70 %) and testing set (30 %). --- #
train_set, test_set = model_selection.train_test_split(UNSW_NB15_DATASET_CSV,train_
```

---------- A. Training Set ----------

In [5]:
```
# --- Read all values of the training set. --- #
 #print ("Train Set: ", train_set)
train_set
```

Out[5]:

	Label	srcip	sport	dstip	dsport	proto	state	sbytes	dbytes	...
129575	1	175.45.176.1	1043	149.171.126.18	53	udp	INT	114	0	...
242196	0	59.166.0.5	39594	149.171.126.2	45675	tcp	REQ	450	0	...
211695	0	59.166.0.3	46696	149.171.126.9	33422	tcp	FIN	320	1910	...
145091	1	175.45.176.1	1043	149.171.126.18	53	udp	INT	114	0	...
227872	0	59.166.0.9	61399	149.171.126.0	23292	tcp	FIN	2196	556	...
...	...	...	...	...	...	...	...	...	...	...
176963	0	59.166.0.7	26753	149.171.126.4	62258	tcp	FIN	1648	728	...
117952	1	175.45.176.1	47439	149.171.126.18	53	udp	INT	114	0	...
173685	0	59.166.0.4	1630	149.171.126.3	5190	tcp	FIN	68199	612	...
43567	1	175.45.176.1	32280	149.171.126.12	1530	unas	INT	200	0	...
199340	0	59.166.0.0	33341	149.171.126.0	31287	tcp	FIN	320	1826	...

181771 rows × 33 columns

---------- B. Testing Set ----------

In [6]:
```
# --- Read all values of the testing set. --- #
 #print ("Test Set: ", test_set)
test_set
```

Out[6]:

	Label	srcip	sport	dstip	dsport	proto	state	sbytes	dbytes	...
135442	1	175.45.176.2	47019	149.171.126.16	80	udp	INT	114	0	...
52588	1	175.45.176.2	1819	149.171.126.10	80	iso-ip	INT	200	0	...
120505	1	175.45.176.0	500	149.171.126.13	500	udp	INT	114	0	...
158480	0	59.166.0.1	42445	149.171.126.0	143	tcp	FIN	1540	1644	...
241382	0	59.166.0.2	3221	149.171.126.9	5190	arp	INT	46	0	...
...	...	...	...	...	...	...	...	...	...	...
157547	0	59.166.0.0	8599	149.171.126.7	42664	tcp	FIN	3598	46036	...
220050	0	59.166.0.0	21608	149.171.126.0	15994	tcp	FIN	2958	33044	...
146869	1	175.45.176.1	47439	149.171.126.18	53	udp	INT	114	0	...
193050	0	59.166.0.5	5642	149.171.126.5	25	udp	CON	146	178	...
202325	0	59.166.0.0	39232	149.171.126.9	53	tcp	FIN	4014	59460	...

77902 rows × 33 columns

Task 2 - 'CSV Data Conversion'

In [10]:
```
# --- Read the CSV data from the file and create a dataset. --- #

def get_dataset(file_path, **kwargs):
    dataset = tf.data.experimental.make_csv_dataset(
          file_path,
          #Defining the batch size as 10. It stores 10 records in a single batch.
          batch_size=10,
          label_name=LABEL_COLUMN,
          na_value="?",
          num_epochs=1,
          ignore_errors=True,
          **kwargs)
     return dataset

# --- Store the CSV data conversion results of the training and testing set invaria
raw_train_data = get_dataset(train_file_path)
raw_test_data = get_dataset(test_file_path)
```

2. Preprocessing

A. Data Cleaning

In [14]:
```
# --- Checking 'Null' values in the training set. --- #
Training_NULL.isnull()
#Training_NULL.isnull().sum()
```

Out[14]:

	Label	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	...	dmean
0	False	False	False	False	False	False	False	False	False	False	...	False
1	False	False	False	False	False	False	False	False	False	False	...	False
2	False	False	False	False	False	False	False	False	False	False	...	False
3	False	False	False	False	False	False	False	False	False	False	...	False
4	False	False	False	False	False	False	False	False	False	False	...	False
...	...	...	...	...	...	...	...	...	...	...	...	...
181766	False	False	False	False	False	False	False	False	False	False	...	False
181767	False	False	False	False	False	False	False	False	False	False	...	False
181768	False	False	False	False	False	False	False	False	False	False	...	False
181769	False	False	False	False	False	False	False	False	False	False	...	False
181770	False	False	False	False	False	False	False	False	False	False	...	False

181771 rows × 33 columns

In [15]:
```
# --- Checking the 'Mean' value of the null feature. --- #
Training_NULL['dur'].mean()
```

Out[15]: 1.2437627565288192

In [16]:
```
# --- Looking the 'header' of the feature. --- #
Training_NULL['dur'].head()
```

Out[16]:
```
0       0.000009
1      21.025787
2       0.002967
3       0.000009
4       0.783632
Name:   dur, dtype: float64
```

In [17]:
```
# ---       Filling 'Missing (Null) Values' of the feature using Mean. --- #
Training_NULL['dur'].replace(np.NaN,Training_NULL['dur'].mean()).head()
```

Out[17]:
```
0       0.000009
1      21.025787
2       0.002967
3       0.000009
4       0.783632
Name:   dur, dtype: float64
```

B. Data Transformation

---------- Z-Score Normalization ----------

In [23]:
```
# --- Select and store the numeric features in a variable. --- #
NUMERIC_FEATURES = ['dur', 'spkts', 'dpkts','sbytes','dbytes','rate','dttl','sload'
                    'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb', 'dwin','tcp 'dmean',
                    'ct_state_ttl']

# --- Pack the numerical features of the 'training dataset'. --- #
packed_train_data = raw_train_data.map(
      PackNumericFeatures(NUMERIC_FEATURES))

# --- Pack the numerical features of the 'testing dataset'. --- #
packed_test_data = raw_test_data.map(
      PackNumericFeatures(NUMERIC_FEATURES))
```

In [27]:
```
# --- Calculate the 'Mean' --- #
MEAN = np.array(Read_Num_Features.T['mean'])

# --- Calculate the 'Standard Deviation ' --- #
STD = np.array(Read_Num_Features.T['std'])
```

In [28]:
```
# --- Perform 'z-score' normalization --- #

def normalize_numeric_data(data, mean,std): # -
   -- Z-Score  Normalization  Formula.  --- # return
   (data-mean)/std
```

In [31]:
```
# --- Result of the 'Z-Score normalizer' which unify the data values in a specified

numeric_layer = tf.keras.layers.DenseFeatures(numeric_columns)
numeric_layer(store_batch).numpy()
```

```
Out[31]:  array([[-0.087,  -0.073,  -0.119, -0.047, -0.103,  -0.565,   0.488,  -0.376,
                  -0.275,  -0.045,  -0.114, -0.121,  0.017, -0.03 ,  -0.107,   0.041,
                  -0.571,   0.506,   0.062,  0.42 ,  0.746,   0.826,  -0.412,  -0.306,
                  -0.318],
                 [-0.209,  -0.131,  -0.175, -0.049, -0.105,   0.682,  -0.751,   0.48 ,
                  -0.276,  -0.075,  -0.134, -0.132, -0.09 ,  -0.111,  -0.151,  -0.961,
                  -0.738,  -0.738,  -0.941, -0.497, -0.441,  -0.487,  -0.185,  -0.482,
                   0.685],
```

C. Data Reduction

---------- Numerical Features ----------

In [32]:
```
# --- Select and store the numeric features in a variable. --- #

SELECT_COLUMNS = ['Label', 'dur', 'spkts', 'dpkts','sbytes','dbytes','rate','dttl',
                  'sinpkt', 'dinpkt', 'sjit', 'djit', 'swin', 'stcpb', 'dtcpb','dwi 'smean', 'dmean',
                  'ct_state_ttl']
DEFAULTS = [0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0,
            0.0, 0.0, 0.0]

temp_dataset = get_dataset(train_file_path,
                           select_columns=SELECT_COLUMNS,
                           column_defaults = DEFAULTS)

# --- Display the result of the batch. --- #
show_batch(temp_dataset)
```

In [35]:
```
# --- We apply the previous 'pack' function to each element of the dataset. --- #
packed_dataset = temp_dataset.map(pack)

for features, labels in packed_dataset.take(1):
# --- Display the pack function results (features and labels). --- #
    print(features.numpy())
    print()
    print(labels.numpy())
```

---------- Categorical Features ----------

In [37]:
```
# --- A list of categorical features (attributes) in the datasets (Training and Testing set)

CATEGORIES = {

'proto':
          ['udp', 'arp', 'tcp','igmp','ospf'],

'state' :
          ['INT','FIN','REQ','ACC','CON','RST','CLO'],

'attack_cat' :
          ['Normal','Reconnaissance','Backdoor','DoS','Exploits','Analysis']
}
```

In [38]:
```
# --- Creating a vocabulary list of the categorical features (attribiutes). --- #

categorical_columns = []
for feature, vocab in CATEGORIES.items():
    cat_col = tf.feature_column.categorical_column_with_vocabulary_list( key=feature,
            vocabulary_list=vocab)
    categorical_columns.append(tf.feature_column.indicator_column(cat_col))
```

In [39]:
```
# --- See what we have just created. --- #
categorical_columns
```

Out[39]:
```
[IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='pr
oto', vo cabulary_list=('udp', 'arp', 'tcp', 'igmp', 'ospf'
), dtyp e=tf.string, default_value=-1, num_oov_buckets=0)),

 IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='
state', vo cabulary_list=('INT', 'FIN', 'REQ', 'ACC', 'CON', 'RST', 'CLO'),
dtype=tf.string, d efault_value=-1, num_oov_buckets=0)),

 IndicatorColumn(categorical_column=VocabularyListCategoricalColumn(key='a
ttack_ca t', vocabulary_list=('Normal', 'Reconnaissance', 'Backdoor',
'DoS', 'Exploits', 'An alysis', 'Fuzzers', 'Worms', 'Shellcode',
'Generic'), dtype=tf.string, default_valu e=-1, num_oov_buckets=0))]
```

3. Model Construction

'Model' : Fuzzy Neuro-Network (FNN)

A. ------- Fuzzification Layer : Fuzzy C-means (FCM) Algorithm -------

In [45]:
```
# --- Number of Attributes. --- #
num_attr = len(Vc.columns) - 1

# --- Number of Clusters. --- #
k = 4

# --- Maximum number of iterations. --- #
MAX_ITER = 100

# --- Number of data points. --- #
n = len(Vc)

# --- Fuzzy parameter (Weight). --- #
m = 2.00
```

In [46]:
```
# --- Initialize membership matrix. --- #

def initializeMembershipMatrix():
    membership_mat = list()
    for i in range(n):
        random_num_list = [random.random() for i inrange(k)]
        summation = sum(random_num_list)
        temp_list = [x/summation for x inrandom_num_list]
        membership_mat.append(temp_list)
    return membership_mat
```

In [48]:
```
# --- Updating Membership Value. --- #

def updateMembershipValue(membership_mat,cluster_centers): p =
    float(2/(m-1))
    for i in range(n):
        x = list(Vc.iloc[i])
        distances = [np.linalg.norm(map(operator.sub, x, cluster_centers[j])) for j
        for j in range(k):
            den = sum([math.pow(float(distances[j]/distances[c]), p) for c inrange(
            membership_mat[i][j] = float(1/den)
    return membership_mat
```

In [50]:
```
# --- Fuzzy C-Means Clustering Results --- #

def fuzzyCMeansClustering():
    # Membership Matrix
    membership_mat = initializeMembershipMatrix()
    curr = 0
    while curr <= MAX_ITER:
        cluster_centers = calculateClusterCenter(membership_mat)
        membership_mat = updateMembershipValue(membership_mat,cluster_centers)
        cluster_labels = getClusters(membership_mat)
        curr += 1
    print(membership_mat)
    return cluster_labels, cluster_centers
```

B. ------- Neural Network Layer: Backpropagation Algorithm -------

In [51]:
```
from random import seed
from random import random
from math import exp

# --- Initialize the Fuzzy-Neuro Network (FNN). --- #
def initialize_network(n_inputs, n_hidden, n_outputs): network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i inran
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i inran
    network.append(output_layer)
    return network

# --- Printing the weight and bias of the layers --- #
seed(1)
network = initialize_network(4, 6, 2)
for layer in network:
    print(layer)
```

Stage 1: Forward Propagation

In [52]:
```
# --- Calculate neuron activation for an input. --- #
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation
```

In [53]:
```
# --- Activation function: 'Sigmoid' --- #
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))
```

In [54]:
```
# --- Forward propagate input to a network output. --- #
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
```

Stage 2: Backward Propagation

In [56]:
```
# --- Backpropagate error and store in neurons. --- #
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))): layer =
        network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta']) errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])
        for j in range(len(layer)): neuron
            = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

C. ------- Compiling Our 'FNN' Model -------

In [58]:
```
# --- Building our model using 'tf.keras.Sequential', starting with the'''preproces
FNN_model = tf.keras.Sequential([
preprocessing_result,
   tf.keras.layers.Dense(4,activation='relu'),
   tf.keras.layers.Dense(6,activation='relu'),
   tf.keras.layers.Dense(2),
])

# --- Compiling our model and add 'Accuracy', 'Precision', 'Recall' and 'Fmeasure'e
FNN_model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True), optimizer='adam',
    metrics=['accuracy', precision, recall, fmeasure])
```

4. 'Training' and 'Evaluating'

A. Training Our 'FNN' Model

In [60]:
```
# --- We set the number of epochs the model trains. --- #
epochs = 10

# --- Now the training begins. --- #
TRAINING = FNN_model.fit(train_data, epochs=epochs)

# --- Displays the model summary. --- #
FNN_model.summary()
```

```
Epoch 1/10
18173/18173 [==============================] - 93s 5ms/step - loss: 0.0046 - accu
racy: 0.9990 - precision: 0.9992 - recall: 0.9987 - fmeasure: 0.9988
Epoch 2/10
18173/18173 [==============================] - 59s 3ms/step - loss: 0.6457e-04 -
accuracy: 0.9999 - precision: 0.9999 - recall: 0.9999 - fmeasure: 0.9999
Epoch 3/10
18173/18173 [==============================] - 60s 3ms/step - loss: 0.0011 - accu
racy: 0.9999 - precision: 0.9999 - recall: 0.9999 - fmeasure: 0.9999
Epoch 4/10
18173/18173 [==============================] - 58s 3ms/step - loss: 0.0010 - accu
racy: 1.0000 - precision: 0.9999 - recall: 0.9999 - fmeasure: 0.9999
Epoch 5/10
18173/18173 [==============================] - 54s 3ms/step - loss: 0.1899e-04 -
accuracy: 1.0000 - precision: 0.9999 - recall: 0.9999 - fmeasure: 0.9999
Epoch 6/10
18173/18173 [==============================] - 55s 3ms/step - loss: 0.2039e-05 -
accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000 - fmeasure: 1.0000
```

B. Testing (Evaluating) Our 'FNN' Model

In [61]:
```
# --- Once our FNN model is trained, we can check its accuracy on the `test_data`se
test_accuracy, test_precision, test_recall, test_fmeasure =FNN_model.evaluate

print('\n\nTest Accuracy: {}, \nTest Precision: {}, \nTest Recall:
        .format(test_accuracy, test_precision, test_recall,test_fmeasure))
```

Out[61]:
```
Test Accuracy: 0.9999871850013733,
Test Precision: 0.9999743103981018,
Test Recall: 1.0,
Test F-Measure: 0.9999856948852539
```

Annex B: The 48 Features of Bot-IoT Dataset

Feature	Feature Name	Description
Flow Features		
F1	saddr	Source IP address
F2	sport	Source port number
F3	daddr	Destination IP address
F4	dport	Destination port number
F5	proto	Textual representation of transaction protocols present in network flow
Basic Features		
F6	stime	Record start time
F7	flgs	Flow state flags seen in transactions
F8	flgs_number	Numerical representation of feature flags
F9	pkts	Total count of packets in transaction
F10	bytes	Total number of bytes in transaction
F11	state	Transaction state
F12	state_number	Numerical representation of feature state
F13	ltime	Record last time
F14	dur	Record total duration
F15	mean	Average duration of aggregated records
F16	stddev	Standard deviation of aggregated records
F17	sum	Total duration of aggregated records
F18	min	Minimum duration of aggregated records
F19	max	Maximum duration of aggregated records
F20	spkts	Source-to-destination packet count
F21	dpkts	Destination-to-source packet count
F22	sbytes	Source-to-destination byte count
F23	dbytes	Destination-to-source byte count
F24	rate	Total packets per second in transaction
F25	srate	Source-to-destination packets per second
F26	Drate	Destination-to-source packets per second
Additional Generated Features		
F27	TnBPSrcIP	Total Number of bytes per source IP
F28	TnBPDstIP	Total Number of bytes per Destination IP.
F29	TnP_PSrcIP	Total Number of packets per source IP.
F30	TnP_PDstIP	Total Number of packets per Destination IP.
F31	TnP_PerProto	Total Number of packets per protocol.
F32	TnP_Per_Dport	Total Number of packets per dport

Feature	Feature Name	Description
F33	AR_P_Proto_P_SrcIP	Average rate per protocol per Source IP. (calculated by pkts/dur)
F34	AR_P_Proto_P_DstIP	Average rate per protocol per Destination IP.
F35	N_IN_Conn_P_SrcIP	Number of inbound connections per source IP.
F36	N_IN_Conn_P_DstIP	Number of inbound connections per destination IP.
F37	AR_P_Proto_P_Sport	Average rate per protocol per sport
F38	AR_P_Proto_P_Dport	Average rate per protocol per dport
F39	Pkts_P_State_P Protocol_P_DestIP	Number of packets grouped by state of flows and protocols per destination IP.
F40	Pkts_P_State_P Protocol_P_SrcIP	Number of packets grouped by state of flows and protocols per source IP.
F41	Fw_blk_rate_avg	Average number of bulk rate in the forward direction.
F42	Fw_seg_min	Min segment size observed in the forward direction.
F43	Bw_byt_blk_avg	Average number of bytes bulk rate in the backward direction.
F44	Bw_pkt_blk_avg	Average number of packets bulk rate in the backward direction.
F45	Bw_blk_rate_avg	Average number of bulk rate in the backward direction
Labeled Features		
F46	Attack	Class label: 0 for Normal traffic, 1 for Attack Traffic
F47	Category	Traffic category
F48	Subcategory	Traffic subcategory

Annex C: The 53 Features of UNSW-NB15 Dataset

Feature	Feature Name	Description
Flow Features		
F1	srcip	Source IP address
F2	sport	Source port number
F3	dstip	Destination IP address
F4	dsport	Destination port number
F5	proto	Transaction protocol
Basic Features		
F6	state	Indicates to the state and its dependent protocol
F7	dur	Record total duration
F8	sbytes	Source to destination transaction bytes
F9	dbytes	Destination to source transaction bytes
F10	sttl	Source to destination time to live value
F11	dttl	Destination to source time to live value
F12	sloss	Source packets retransmitted or dropped
F13	dloss	Destination packets retransmitted or dropped
F14	service	http, ftp, smtp, ssh, dns, ftp-data ,irc
F15	sload	Source bits per second
F16	dload	Destination bits per second
F17	spkts	Source to destination packet count
F18	dpkts	Destination to source packet count
F19	rate	Total packets per second in transaction
Content Features		
F20	swin	Source TCP window advertisement value
F21	dwin	Destination TCP window advertisement value
F22	stcpb	Source TCP base sequence number
F23	dtcpb	Destination TCP base sequence number
F24	smeansz	Mean of the row packet size transmitted by the src
F25	dmeansz	Mean of the row packet size transmitted by the dst
F26	trans_depth	Depth of http request/response transaction
F27	res_bdy_len	Size of the data transferred from server’s http service.
Time Features		
F28	Sjit	Source jitter (mSec)
F29	djit	Destination jitter (mSec)
F30	sintpkt	Source interpacket arrival time (mSec)
F31	dintpkt	Destination interpacket arrival time (mSec)

Feature	Feature Name	Description
F32	tcprtt	TCP connection setup round-trip time, the sum of 'synack' and 'ackdat'.
F33	synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
F34	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
Additional Generated Features		
F35	is_sm_ips_ports	If source (F1) and destination (F3) IP addresses equal and port numbers (F2) (F4) equal then, this variable takes value 1 else 0
F36	ct_state_ttl	No. for each state (F6) according to specific range of values for source/destination time to live (F10) (F11).
F37	ct_flw_http_mthd	No. of flows that has methods such as Get and Post in http service.
F38	is_ftp_login	If the ftp session is accessed by user and password then 1 else 0.
F39	ct_ftp_cmd	No of flows that has a command in ftp session.
F40	ct_srv_src	No. of connections that contain the same service (F14) and source address (F1) in 100 connections according to the last time (F26).
F41	ct_srv_dst	No. of connections that contain the same service (F14) and destination address (F3) in 100 connections according to the last time (F26).
F42	ct_dst_ltm	No. of connections of the same destination address (F3) in 100 connections according to the last time (F26).
F43	ct_src_ ltm	No. of connections of the same source address (F1) in 100 connections according to the last time (F26).
F44	ct_src_dport_ltm	No of connections of the same source address (F1) and the destination port (F4) in 100 connections according to the last time (F26).
F45	ct_dst_sport_ltm	No of connections of the same destination address (F3) and the source port (F2) in 100 connections according to the last time (F26).
F46	ct_dst_src_ltm	No of connections of the same source (F1) and the destination (F3) address in in 100 connections according to the last time (F26).
F47	subfl_bw_byt	The average number of bytes in a sub flow in the backward direction
F48	idl_avg	Mean time a flow was idle before becoming active
F49	idl_std	Standard deviation time a flow was idle before it's active
F50	idl_max	Maximum time a flow was idle before becoming active

Feature	Feature Name	Description
F51	idl_min	Minimum time a flow was idle before becoming active
Labeled Features		
F52	attack_cat	The name of each attack category. In this data set, nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
F53	Label	0 for normal and 1 for attack records

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Tewodros Worku Osman

Signature: _____

Date: _____

Confirmed by advisor:

Name: Solomon Gizaw (PhD)

Signature: _____

Date: _____

Place and date of submission: Addis Ababa University, October 2020.