AN EFFICIENT APPROACH TO ONLINE BOT DETECTION BASED ON A REINFORCEMENT LEARNING TECHNIQUE

M M ALAUTHMAN

PHD

2016

AN EFFICIENT APPROACH TO ONLINE BOT DETECTION BASED ON A REINFORCEMENT LEARNING TECHNIQUE

MOHAMMAD MANSOUR ALAUTHMAN

A thesis submitted in partial fulfilment of the requirements of the University of Northumbria at Newcastle for the degree of Doctor of Philosophy

Research undertaken in the

Faculty of Engineering and Environment

June 2016

ABSTRACT

In recent years, Botnets have been adopted as a popular method used to carry and spread many malicious codes on the Internet. These codes pave the way to conducting many fraudulent activities, including spam mail, distributed denial of service attacks (DDoS) and click fraud. While many Botnets are set up using a centralized communication architecture such as Internet Relay Chat (IRC) and Hypertext Transfer Protocol (HTTP), peer-to-peer (P2P) Botnets can adopt a decentralized architecture using an overlay network for exchanging command and control (C&C) messages, which is a more resilient and robust communication channel infrastructure. Without a centralized point for C&C servers, P2P Botnets are more flexible to defeat countermeasures and detection procedures than traditional centralized Botnets.

Several Botnet detection techniques have been proposed, but Botnet detection is still a very challenging task for the Internet security community because Botnets execute attacks stealthily in the dramatically growing volumes of network traffic. However, current Botnet detection schemes face with significant problem of efficiency and adaptability.

The present study combined a traffic reduction approach with reinforcement learning (RL) method in order to create an online Bot detection system. The proposed framework adopts the idea of RL to improve the system dynamically over time. In addition, the traffic reduction method is used to set up a lightweight and fast online detection method. Moreover, a host feature based on traffic at the connection-level was designed, which can identify Bot host behaviour. Therefore, the proposed technique can potentially be applied to any encrypted network traffic since it depends only on the information obtained from packets header. Therefore, it does not require Deep Packet Inspection (DPI) and cannot be confused with payload encryption techniques.

The network traffic reduction technique reduces packets input to the detection system, but the proposed solution achieves good a detection rate of 98.3% as well as a low false positive rate (FPR) of 0.012% in the online evaluation. Comparison with other techniques on the same dataset shows that our strategy outperforms existing methods. The proposed solution was evaluated and tested using real network traffic datasets to increase the validity of the solution.

DECLARATION

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work. I also confirm that this work fully acknowledges opinions, ideas and contributions from the work of others.

Name: Mohammad Mansour Alauthman.

Signature: Mohammad

Date:27/05/2016

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my supervisor Dr. Nauman Aslam for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I also want to thank my supervision committee members Dr. Rafe Alasem and Dr. Li Zhang for reviewing my thesis and providing constructive comments that help me improve my work. As well, I owe special thanks to Professor Alamgir Hossain for his excellent guidance throughout the preliminary stages of my research.

My work has benefited from and has been influenced by discussions with a number of people over the years. For these discussions, I express my gratitude to my friends Dr Majed Alsanea, Dr Loai Kayed Bani Melhim and Dr Sami Smadi. They offered a lot of help and advice on my research and wonderful friendship.

Likewise, I would also like to express my special appreciation for Ahmad Alshorman and Amer Hatamleh for spending countless hours with me in proofreading advice to help the presentation of this research.

In addition, I thank my fellow PhD researchers of LAB F7 and LAB F6 for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last four years. Also, I acknowledge all my colleges in college of science Al-Zulfi, KSA.

On a personal note, I cannot say enough to thank my dearest parents brothers and sisters for their prayers and continued support. Last but not the least, I am also thankful to my wife for always being there for me in the ups and downs of my life and tremendous support.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

ACC	Accuracy
ACK	TCP flag indicates the value in acknowledgement is valid
AUC	Area Under the ROC curve
C&C	Command And Control
CART	Classification And Regression Tree
DDoS	Distributed Denial Of Service
DNS	Domain Name System
DPI	Deep Packet Inspection
DR	Detection Rate
FIN	TCP flag indicates that No more data from sender
FNR	False Negative Rate
FPR	False Positive Rate
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IRC	Internet Relay Chat
ISCX	Information security centre of excellence
ISOT	Information Security And Object Technology
JPCAP	A Network Packet Capture Library
LBNL	Lawrence Berkeley National Lab
MCC	Matthews Correlation Coefficient
MDP	Markov Decision Process
NDEI	Non-Dimensional Error Index
OSI	Open Systems Interconnection
P2P	Peer-To-Peer
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning

RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic Curves
RST	TCP flag indicates that Reset The Connection
SYN	TCP flag initiate a TCP connection
TCP	Transmission Control Protocol
TNR	True Negative Rate
TPR	True Positive Rate

1 INTRODUCTION

1.1 Introduction

Internet services are increasing in popularity and many new online services appear every day. The use of online services leads to a massive volume of online financial transactions, where sensitive information is exchanged via the Internet. The attacker's interest may thus be converted from curiosity to economic benefit. Attackers utilise different types of malware to accomplish their goals. Among the diverse types of malware, the Botnet is considered to be the most dangerous means of performing online crimes (Rgio S. C. Silva, Rodrigo M. P. Silva, Raquel C. G. Pinto, & Ronaldo M. Salles, 2013).

A Botnet network contains Bots, which are computers infected by malware such as Trojan horses, backdoors or worms without the user’s permission. The Botmaster remotely manages a Botnet through a C&C channel (Gu, Perdisci, Zhang, & Lee, 2008). Recently, Botnets have been sold and rented in an underground market by Botmasters for commercial profit. They can begin many cyber-crimes: creating phishing web pages, carrying out massive amounts of spam emails, stealing sensitive users information and generating DDoS attacks (Ullah, Khan, & Aboalsamh, 2013).

According to a recent Symantec Internet Security Threat Report in April 2014 (Symantec Corporation, 2014), Botnets accounted for 76% of all spam sent out in 2013, which was about 10 billion per day on average. Botnet infections are a global pandemic. Recently Microsoft alone estimated that, as of April 2015, more than one million machines are currently infected by the Ramnit worldwide Botnet (Batchelder et al., 2014).

The scale of Botnet contaminations worldwide makes the detection of Botnet activity an important task. Botnet detection has been a significant subject in the cyber security domain for the last decade. Despite concerted efforts reported in the literature degrade the malicious activities of Botnets, the diversity of Botnet structures and protocols creates from the Botnet detection a demanding task for the cyber-security society (Demarest, 2014; IBM, 2013; Plohmann, Gerhards-Padilla, & Leder, 2011)

1.2 Research Motivation

Analysing network traffic and identifying host malicious activity inside a network is a significant requirement for network admin in order to manage their networks and detect infected computers. Therefore, network administrators require an efficient strategy to keep the network free from any suspicious activity. Additionally, Botnets grow rapidly in terms of both volume and variety, and they have begun to infect infrastructures such as industrial control systems (Falliere, Murchu, & Chien, 2011) and smartphones (Mullaney, 2012).

As a result, Botnets have been realized to be one of the most dangerous threats to Internet safety. It is therefore crucial to detect, prevent and mitigate Botnet activities. Developing Botnet detection systems is a primary concern since it serves as an essential step in further prevention and mitigation strategies. In this regard, network-based Botnet detection systems are particularly desired due to the visibility of the network behaviour of all hosts in monitored networks.

The number of networked computers and devices is enormous and keeps grow, and volumes of network traffic are high and rapidly increasing. This means that detection systems require the efficiently processing of a massive volume of network traffic. However, most existing Botnet detection systems (Chen & Lin, 2015; Goebel & Holz, 2007; Gu, Perdisci, et al., 2008; Gu, Zhang, & Lee, 2008; Lu, Rammidi, & Ghorbani, 2011; Rafique & Caballero, 2013; Seewald & Gansterer, 2010; R. Tyagi, Paul, Manoj, & Thanudas, 2015; Yen & Reiter, 2008) rely on DPI to analyse packet content, which is computationally expensive and inefficient in recognizing unknown payload signatures.

Consequently, when these detection systems are deployed in high-speed or high-volume networks, they may not be able to perform a comprehensive analysis of all network traffic and thus lead to the failure to immediately detect Bot hosts.

However, identifying infected computers before the Bot exploits a host machine in a serious way is a challenging task in cyber-security. In the few last years, several methods have been proposed to identify Botnets threats that represent a risk for cyber-security systems. The majority of these studies have focused on ways of improving offline Botnet detection systems. Generally, the exact detection results obtained using these approaches reflect only the past situation of the network traffic.

Therefore, the results of these approaches may become worthless later the when status of the network environment changes. In this case, all of these offline methods may become invalid since they do not use online detection approaches. Therefore, a Bot detection system needs to be developed that is able to monitor Bot host activities in an online manner and to repeat Bot host activities to network admin as soon as possible.

1.3 Research Aims and Objectives

The aim of this research is to develop an online P2P Bot host detection system based on RL. The approach proposed in this research has the following characteristics. It detects Bots during the propagation phase before any malicious action has been taken. Furthermore, it does not require DPI analysis for signature matching, and does not need to analyse the entire network traffic. It detects Bots independent of port numbers, IP addresses and host characteristics. Therefore, the main objectives of this research are:

1. To investigate the characteristics of network traffic that can be used to discriminate the behaviour of P2P Bots from normal traffic.
2. To develop a RL system has the ability to recognize zero-day attacks caused by a P2P Botnet.
3. To generate a host traffic representation based on traffic reduction, that is used to detect the hosts of Bots.

1.4 Thesis Contributions

The objective of this thesis is to develop an efficient network-based Bot host detection system. The thesis introduces a network-based solution, which achieves the following requirements. Firstly, efficiency is enhanced by using a traffic reduction method to build a lightweight detection system able to deal with massive network volumes of traffic. Secondly, Bot detection is accomplished earlier by detecting the Bot in the propagation phase before it starts malicious activities. Finally, it is adaptable due to the use of a RL approach to a detection system able to learn online new Bot behaviour from the network environment**.** This thesis makes three specific contributions which as following:

1. A network traffic reduction approach has been designed which will be able to increase the performance of the proposed framework.
2. The 'connection-based' detection mechanism is payload-independent, and depends on only information obtained from the headers of TCP control packets.
3. A new model-based RL algorithm computes the reward from the dynamic environment.

Firstly, in Chapter 3, a new traffic reduction technique is introduced to facilitate the deployment of Bot host detection systems on a high-speed network. As discussed above, the majority of Botnet detection schemes rely on DPI and examine the entire network traffic. The use of DPI assumes access to the payload of each packet. This method can be accurate in classifying network traffic if the packet payloads are not encrypted. However, the majority of new malware applies evasion methods such as the encryption of payloads or protocol encapsulation and obfuscation which mean that the payload is covered (P. Wang, Wu, Aslam, & Zou, 2015). Furthermore, examining all packets on a high-speed network is an expensive task because of the speed of the networks and the amounts of packets transferred via a network is increasing daily. However, a detection system which applies DPI may suffer from efficiency limits when processing a large volume of traffic from high-volume or high-speed networks (Jun et al., 2008). The goal of the present study is to increase the effectiveness of detection systems by decreasing the volume of traffic which needs to be analysed without affecting the accuracy of the detection process in an ideal solution. To achieve this goal, a novel traffic reduction method is proposed for a Bot host detection framework which selects only TCP control packets. This framework can efficiently and effectively reduce the amount of traffic that will be entered into the detection system.

Reducing network traffic can be accomplished by generating a representation of all of the entire network packets. Moreover, the behaviour of the representative traffic should reflect the behaviour of all network traffic. Using a representative traffic approach will reduce the volume of the traffic needed to be analysed. Therefore, it means faster analysis and lower computation time. To the best of our knowledge, this is the first P2P Bot host detection approach applying such a reduction technique to achieve efficiency in Bot detection host.

Secondly, in Chapter 4, host traffic features have been designed based on the connection-level that can differentiate between a Bot and a legitimate network host. More specifically, the proposed features contribute to the identification of the Bot host by using a minimum set of packets that need to be utilized in developing an efficient Bot detection system. The goal of the proposed feature set is designed to boost the effectiveness of P2P Bot detection in three challenging scenarios: i) the Bot performs malicious activities in a stealthy way by using an evasion approach such as encryption techniques; ii) the earlier detection of P2P Bot at the primary stage of its life cycle, the propagation stage; iii) the feature set helps the detection method to detect an infected machine if it is the only one in the network. The framework solves the above challenges by working on the headers of TCP control packets to bypass encrypted network traffic. Moreover, focusing on the connection behaviour will help the detection system to recognize Bot behaviour at an earlier stage when the Bot propagates and tries to contact other peers to find new updates. Furthermore, the proposed feature sets are estimated for every host in the network in order to detect any single infected machine. To the best of our knowledge, this is the first time that connection-based features have been used in P2P Bot host detection. As the features are extracted from the headers of the network packets, they do not rely on packet payloads. With this characteristic, our detection approach will not be affected by traffic encryption. Moreover, the proposed approach can also be used to detect unknown P2P Bots. Furthermore, the feature set helps the detection system to identify P2P Bot infects even if it just one.

Finally, in Chapter 5, a new model-based reinforcement learning method is built to solve a Bot host detection problem. More specifically, an online RL system is designed to detect a P2P Bot in the connection (propagation) stage. The goals of the RL model are to satisfy the requirements of adaptability, novelty and early detection. To accomplish these goals, a new algorithm for RL is designed to boost the adaptability of the detection system, evaluate any new Bot host pattern and adapt the detection system according to the new Bot pattern.

The neural network has been adopted with a resilient back-propagation learning algorithm as a classification technique. This has robust capabilities for dealing with a nonlinear problem due to its ability of approximation. In addition, utilizing a feature set based a traffic reduction technique with the RL algorithm improves the capability of the detection system to detect timely Bot host behaviour and enhances of the online system so as to learn new kinds of attack patterns (zero-day). To the best of our knowledge, this work is the first to provide an online Bot detection method that is based on a new RL algorithm. Also, RL techniques require some set of action-selection procedures, which guarantee that there is an balance between exploration and exploitation. The difficulty is to obtain good action-selection tactics which apply a good balance of exploration and exploitation. The proposed approach introduces an adaptive threshold factor to manage the adaption of a new Bot pattern and to make a balance between exploration and exploitation. The proposed online Bot host detection is timely because detection is achieved for each host, and when the required features accumulated from the host are adequate then the judgment can be made instantly. Hence, an infected host can be identified within a short time. Also, to ensure the generalization of the proposed detection approach, we use a testing dataset from a different network traffic source in order to ensure the generalization of the classifier.

However, Bots and the users of computers exploit the Internet network in the same way, but with different objectives. The proposed framework should be able to differentiate between malicious traffic generated by Bot activity and legitimate user or application activities. Therefore, the main expected contribution of this research is to design an online detection of a P2P Bot host which focuses on both traffic reduction and RL in order to achieve efficient Bot detection able to complete detection in a short time.

1.5 Research Methodology

The general research methodology used in the research is the positivist approach. According to (Iivari, Hirschheim, & Klein, 1998) this method contains hypothesis and testing experiments, therefore, it’s suitable for the research. Besides, the general experimental procedures used in statistics approaches such as neural network, machine learning and fuzzy to obtain conclusions from the data comprise the following steps (Wechsler & Harry, 2000):

1. State the problem
2. Formulate the hypothesis
3. Design the experiment/generate the data
4. Collect the data and perform pre-processing
5. Estimate the model
6. Interpret the model/draw the conclusions

The main stages of the adopted research methodology in this research as show on in Figure 1.1 include the literature review, the literature analysis, design and modelling and performance evaluation.

1.6 Thesis Scope

The scope of this thesis is limited to developing a P2P Bot detection approach based only on TCP network traffic. The TCP network traffic is captured from a local area network. Moreover, the information of the control packets header is used.

The UDP packets are excluded in this research because UDP is a connectionless protocol, the information in a UDP packets is inadequate to decide if it as control or payload packets unless we have information about packet's application. Thus, it is impossible to classify UDP packets into control and payload packets immediately as in the state of TCP.

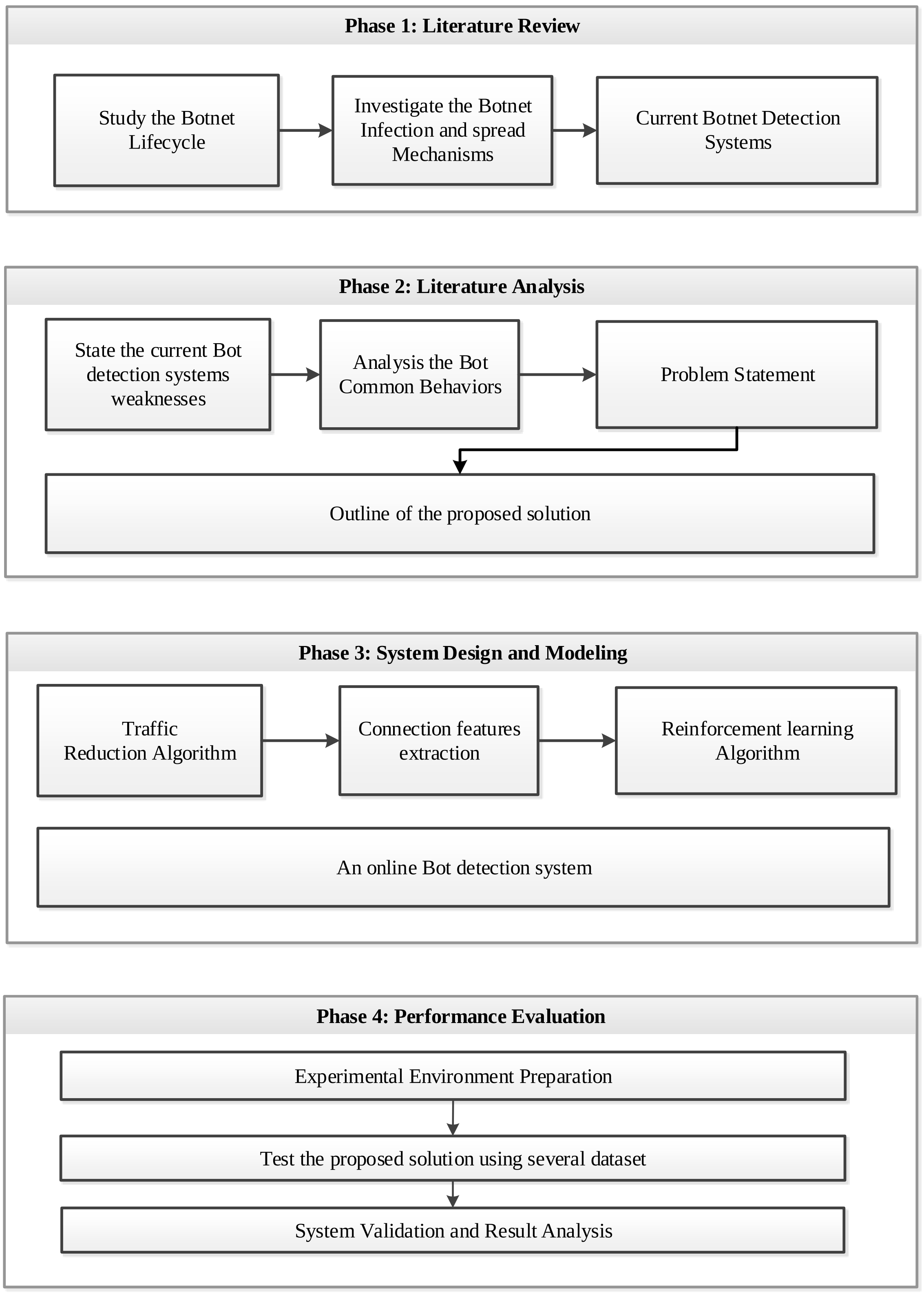
Phase 1: Literature Review
Study the Botnet Lifecycle
Investigate the Botnet Infection and spread Mechanisms
Current Botnet Detection Systems
Phase 2: Literature Analysis
State the current Bot detection systems weaknesses
Analysis the Bot Common Behaviors
Problem Statement
Outline of the proposed solution
Phase 3: System Design and Modeling
Traffic Reduction Algorithm
Connection features extraction
Reinforcement learning Algorithm
An online Bot detection system
Phase 4: Performance Evaluation
Experimental Environment Preparation
Test the proposed solution using several dataset
System Validation and Result Analysis

Figure 1.1 Research Methodology

1.7 Thesis Outline

This thesis is organized into six chapters. This chapter presents the objectives of this thesis. It starts by presenting a background discussion of the Bot problem along with the research goals and contributions.

Chapter 2 gives an overview of Botnets concepts. The Botnet life cycle is described, and the risks of Botnets are listed. Previous Botnet detection approaches and relevant research using machine learning are reviewed as well. Taxonomy of Botnet detection techniques is provided, and the advantages and disadvantages of each type are discussed.

Chapter 3 discusses the design of the proposed traffic reduction algorithm which aim to increase the efficiency of the Bot detection system. Besides, a briefly detailed for each component of the Botnet detection system is presented. Also, a detailed description of the network datasets used in this study are introduced in this chapter.

Chapter 4 explains the connection-based feature extraction process. In addition, the chapter presents the offline Bot detection system based on connection-level feature set. Also, the procedures followed in the experiments are discussed.

Chapter 5 provides an introduction to RL, Markov decision processes and the partially observable Markov decision process. In this chapter, a formulation is given of Botnet problems based on RL. Besides, a new model-based RL algorithm for Bot host detection is introduced in a dynamic partially observable environment. Finally, assessment based on a real-world dataset is presented in the chapter.

Finally, Chapter 6 draws the conclusion of the thesis and discusses potential future research directions to improve or extend the present work.

2 BACKGROUND INFORMATION AND LITERATURE REVIEW

2.1 Introduction

In recent years, there has been increasing interest in Botnet problems compared to others threats to computing. Malware has infected every area of the Internet, and this shows no signs of stopping. Despite the relative novelty of Botnets, a significant number of studies have attempted to find a solution to the problems they create.

Botnet hazards have increased in the internet environment subsequent to the first known Botnets found at the beginning of the 1990s based on the Internet Relay Chat (IRC). The IRC was set up in the late 1980s to allow the computer user to connect to the Internet anywhere and to join live chats. Botnets exploited the benefits of this channel so as to set up communication between the Botmaster and the Bot in the victim’s computer.

The reason for investigating the Botnet threat in depth is that electronic crime has increased, and in the past few years, Botnet targets have changed so that secret information found on the victim’s machines is taken. The difficulty of detection has given the Botnet the leading position in cyber-crime. Furthermore, Botnets are improving methods of evasion along with the development of spreading techniques, and this also increases the difficulty of Botnet detection. Although, a considerable number of studies have been published on Botnet detection, new types of Botnet continually come up with new techniques to avoid detection by existing methods.

This thesis focuses on establishing a P2P Bot detection strategy that utilizes neural networks combined with a RL approach to detect hosts on the network that generate malicious traffic behaviours. Furthermore, this approach should work online, and at the same time achieve good accuracy and high detection rates.

This chapter began by introducing the history of Botnets. The following section gives definitions of terms related to Botnets and explains the life cycle. Section 2.3 introduces the threats from Botnets and Section 2.3 illustrates Botnet evolution. Sections 2.4 and 2.5 classify Botnets and Botnet detection approaches respectively. A summary of this chapter and its relevance to the present study are given in Section 2.6.

2.2 Background of Botnets

This section describes Bots, Botnets, C&C and victim hosts, and then elaborates on the role of Botnet in cyber-crime.

2.2.1 Definitions related to the Botnet

- Bot: the word Bot derives from the word robot, which means "worker." In the world of computers, a Bot is a general term adopted to describe an automated operation (Schiller & Binkley, 2011). In other words, a Bot refers to a malicious code on victim computer that allows the attacker to control the computer remotely and perform specific operations (Rgio S. C. Silva et al., 2013).
- A Botnet: is a collection of compromised computers (zombies) connected through the network, and it is under the control of a Botmaster via a C&C channel (Huy, Xuetao, Faloutsos, & Eliassi-Rad, 2013; Lashkari, Ghalebandi, & Reza Moradhaseli, 2011). The Bot is commonly installed on the victim's computer in several ways, such as when an untrusted website is surfed or a malicious email attachment is open. Generally the Bot is configured to be launched when it infects the victim's machine, and then the Bot will be ready to receive a command from the Botmaster through the C&C server (Rgio S. C. Silva et al., 2013).
- Command and Control (C&C): is a communication channel used to transfer orders between the Botmaster and Bots to achieve various distributed attacks remotely (Feily, Shahrestani, & Ramadass, 2009; Nagaraja et al., 2011). Furthermore, the interaction between the Botmaster and Bots through the C&C communication channel can be classified into three groups: message types, message directions and communication protocols (Rodríguez-Gómez, Maciá-Fernández, & García-Teodoro, 2013). C&C message types can be classified as command or control messages. A command message is used by the Botmaster to send an order to the

Bots to execute an action on the victim's computer. The other type of C&C message is a control message that gives the Botmaster information about the status of Botnets, such as the number of active Bots. What is more, C&C message directions may be divided into two categories: pulls and pushes (Gu, Zhang, et al., 2008). In a pull style, the C&C server sends a command to Bots and waits for them to respond before sending the second order. On the other hand, if the C&C server sends a command and it does not wait for a response, from this is a push case message. These approaches are used by centralized Botnet structures such as IRC and HTTP Botnet-based. The communication protocols perform a significant part of the communication between the C&C server and the Bots. IRC, HTTP and P2P protocols are the most common types of the protocol used in C&C server communications (Rodríguez-Gómez et al., 2013).

- The Botmaster (attacker): is the person who creates the Bots and coordinates all the operations going on between the Bots and the C&C server. In addition, the Botmaster builds and develops the ability of the Bots to infect a victim’s machine as well as coordinating the communication between Botnet system components (Boshmaf, Muslukhov, Beznosov, & Ripeanu, 2013). However, on the internet there are many toolkits that can be used to build and manage Botnet systems (Boshmaf et al., 2013).
- The victim, the main aim of the Botmaster is to spread the Bot code to infect any connected computer and then control these computers via the C&C server. For instance, system, person or network could be a Botnet targets. The victims vary depending on the objective of the attacks or the Botnet type, for example, receiving spam email or stealing confidential information from the victim’s machine. In another example, DDoS attacks have played a key role in companies losing millions of dollars (Rodríguez-Gómez et al., 2013).

2.2.2 Generic Botnet Life Cycle

This section reviews the main stages of the Botnet life cycle. Botnet behaviour is addressed in terms of the set of operations used by a Botnet during its life cycle phase. The majority of Botnet detection approaches focus on the specific stage of a Botnet life cycle via studying its behaviour during these phases. As a result, the analysis of the Botnet life cycle is also important in understanding previous work on Botnet detection and Botnet behaviour. (Zhaosheng et al., 2008), (Feily et al., 2009) and (Rgio S. C. Silva et al., 2013) addressed Botnet life cycles in similar ways with slight differences, dividing it into three stages: infection, communication and attack. However, the Botnet life cycle can be described in details in five phases: initial infection, secondary injection, connection, command and control, and updating and maintenance (Feily et al., 2009) as illustrated in Figure 2.1.

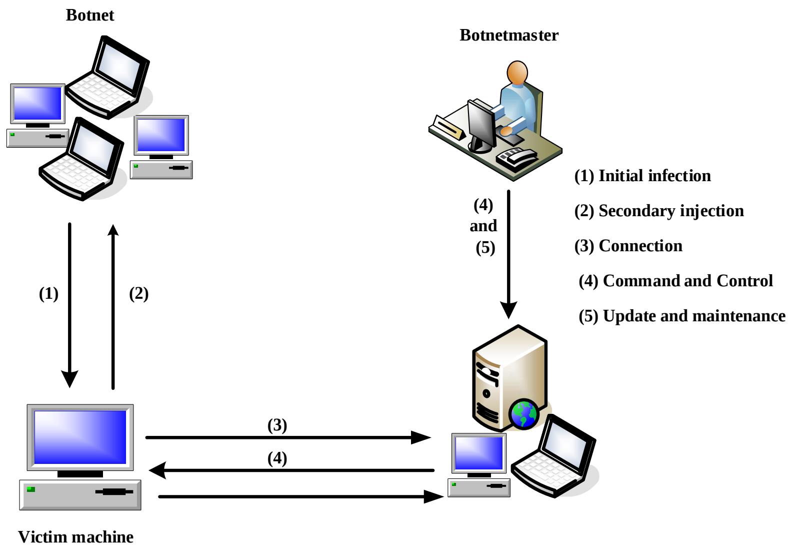

Figure 2.1 Generic Botnet life cycle (Feily et al., 2009).

The first phase of creating a Botnet is a critical phase; the Botmaster tries to exploit a known computer operating system's vulnerability to infect the user's machine. Moreover, scanning techniques are used by an attacker to insert the Bot inside the target's machine (Feily et al., 2009). There are several methods for installing Bots in end-user computers, such as opening malicious spam email attachments or browsing malicious webpages (Lu et al., 2011).

When the initial infection is accomplished, then the secondary injection phase starts by executing the dropper script code in the infected machine. The execution of the dropper script code downloads the Bot binary from specific internet server using a File Transfer Protocol (FTP), HTTP or Peer-to-Peer (P2P), and then setup a newer Bot code on the victim machine. At the end of this phase, the infected machine turns into a zombie (Bot).

After that, the third phase begins by launching the C&C server to issue the communication channel with an army of recruited Bots which gives the Botmaster control ever the Botnet network (Feily et al., 2009). The fourth phase starts when the Botmaster has the ability to use the C&C server to send commands to the Bot in order to execute it on the target's machine. The final phase of the Botnet’s life cycle is updating and maintenance, where the Botmaster updates the Bot software for several reasons. For instance, the Botmaster may need to add a new function to enhance the Botnets future attacks or to improve the evasion methods. In addition, update the IP address of a new C&C server can be updated to keep it working and thus then prevent it from being blocked due to the evolution of Botnet detection techniques.

2.2.3 P2P Botnet Life Cycle

The lifecycle of the P2P Botnet consists of four primary phases, namely: initial infection, peer propagation, secondary injection and attack. These phases are shown in Figure 2.2 (Felix, Joseph, & Ghorbani, 2012). Firstly, the Bot code is created for insertion into an end-user computer using different techniques such as vulnerability exploitation, web downloads, automatic scanning and email attachments (Chao, Wei, & Xin, 2009)

Secondly, the Bot tries to connect with other Bots on infected hosts based on its own hard-coded peer list. Thirdly, the Bot downloads the latest update of the Bot code through the C&C channel, which will update it for future tasks. In this phase, a host is considered a Bot in the Botnet network. Finally, the Bot initiates malicious activities such as spam or phishing emails, DDoS, stealing information, and scanning activities.

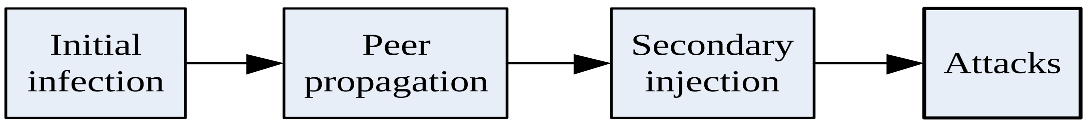

Figure 2.2 P2P Botnet life cycle.

2.3 Botnet Threats

A Botnet is more dangerous than previous more traditional threats such as worms and viruses. The Honeynet project listed many kinds of Botnet attacks, including such as DDoS, Spam, Stealing information and Exploiting resources (Bacher, Holz, Kotter, &

Wicherski, 2005). Moreover, Lanelli et al. reported that Botnets can be exploited in several kinds of cybercrimes (Ianelli & Hackworth, 2005).

2.3.1 Distributed Denial of Service

The DDoS is one of the most potent threats produced by Botnets. In the 2014 Information Security Breaches Survey report in the UK, 38% of big organisations were attacked by DDoS in the previous year (Mille, Horne, & Potter, 2014). The massive number of members in a Botnet network gives the DDoS considerable destructive power. The Botmaster uses the Botnet network to take down the victim system of control as the Bots members send huge numbers of requests to this system. In addition, some massive Botnets can even be harmful to Internet Service Providers (ISPs).

2.3.2 Spam

Spam is an operation where an overwhelming quantity email messages containing advertisements or malicious links are sent to a large number of users. A Botnet is the best choice for an attacker use as a tool to send spam emails. The spam attacks start by sending commands to the Bots from Botmaster before they begin sending spam email to the victim's address. In this case, the detection approaches that used a blacklist technique become useless and hereby hard to detect a real attacker.

Ramachandram et al. identified Botnets as the major cause of email spam problems (Ramachandran & Feamster, 2006). In a study which set out to determine the source of email spam, John et al (2009) found that the Botnets were responsible for 79% of spam email received at the University of Washington (John, Moshchuk, Gribble, & Krishnamurthy, 2009).

2.3.3 Stealing Information

A Botmaster employs Bots to collect secret information from victim hosts by using techniques such key logging, reading log files and screen capture. For example, the SDBot is a type of Botnet which employs a keylogging technique to gather users' sensitive information. This can then be sold to others in order to perform illegitimate actions (Bailey, Cooke, Jahanian, Yunjing, & Karir, 2009). In addition, the Zeus Bot's main tools use keylogging methods to steal credit card information and private bank accounts, which allows the Botmaster to extract passwords and usernames from a bank's web page, emails, and social network accounts (Selvaraj, 2014). Moreover, this Bot exploits the Windows application program interface (API) to extract private user information before the web browser can encrypt it (Hannah & Gianvecchio, 2015).

2.3.4 Exploiting resources

Bot hosts are controlled to perform illegal activities. For example, the Bot uses the victim's computer to visit a website periodically to increase the number of website visitors without the user's permissions. In addition, they can be used to cast fake votes or to grow the number of followers on Twitter and Facebook.

2.4 Botnet Classification

As can be seen from the Botnet life cycle, the C&C server mechanism is the most important component of a Botnet system. Based on the C&C mechanism, the Botmaster able to communicate with Bots, and infrastructure of the C&C communicational channel is the main difference between a Botnet and other malware (Zeidanloo, Bt Manaf, Vahdani, Tabatabaei, & Zamani, 2010). In contrast to other malware which is used to perform malicious behaviour individually, a Botnet works as a group of infected hosts based on the C&C communication channel. Therefore, the Botmaster can use this channel to deliver a command to thousands of Bots in order to launch an attack or receive information from victim computers. In 2005, Cooke and co-workers classified Botnets depending on their C&C mechanism into three different groups: centralized, distributed, and random. This paper also contained the first academic analysis of the P2P Botnet (Cooke, Jahanian, & McPherson, 2005). Dittrich and Dietrich grouped Botnets into four classes in terms of their development environments as IRC, HTTP, P2P and hybrid Botnets (David Dittrich & Sven Dietrich, 2008). However, in this thesis, the Botnet network is described based on the structure of its C&C channels and the type of protocol used in Botnet communications as follows.

2.4.1 Botnet Classification According to Control and Command Structure

The C&C server is what makes Botnets more powerful than other types of malicious malware. Botnet structure based on the C&C server can be classified into centralized, decentralized and unstructured C&C architectures (Chao et al., 2009).

- Centralized architecture: Here, all Bots member are connected to one or many C&C servers as shown in Figure 2.3, such as in HTTP and IRC Botnets. The C&C server plays a significant role in delivering commands from the Botmaster to Bots, and there are no direct connections between Bots. In addition, the centralized architecture is considered to be the easiest type of Botnet to construct, but it does suffer from the fact that it has a single point of failure in the C&C server. A shutdown of the C&C server would result in the loss of communication between the Bots and Botmaster (Ludl, McAllister, Kirda, & Kruegel, 2007). In spite of this weakness, it is widely used in cyber-crimes, because the commands are sent more quickly with low latency. However, it is not so difficult to detect the C&C server, and thus to crush the whole Botnet network.
- Decentralized (P2P) Botnet: In this architecture, there is no centralized point for the C&C, so mitigating or detecting these Botnets is very challenging. Due to the distributed network structure of P2P systems, all peers in the network work as a Bot (client) and C&C (server) at the same time. In this case, the Botmaster plays the main role by sending commands to any infected peers to execute any order or requesting information at any time as shown in Figure 2.4. However, in order to avoid the weakness of a single point of failure, Botnet attackers have recently started to build Botnets based on decentralized C&C infrastructures such as the P2P Botnet (Felix et al., 2012), the P2P model was adopted by many types of Botnet, for example Storm Bot, Conficker Bot and Waledac Bot (Davis, Fernandez, & Neville, 2009).

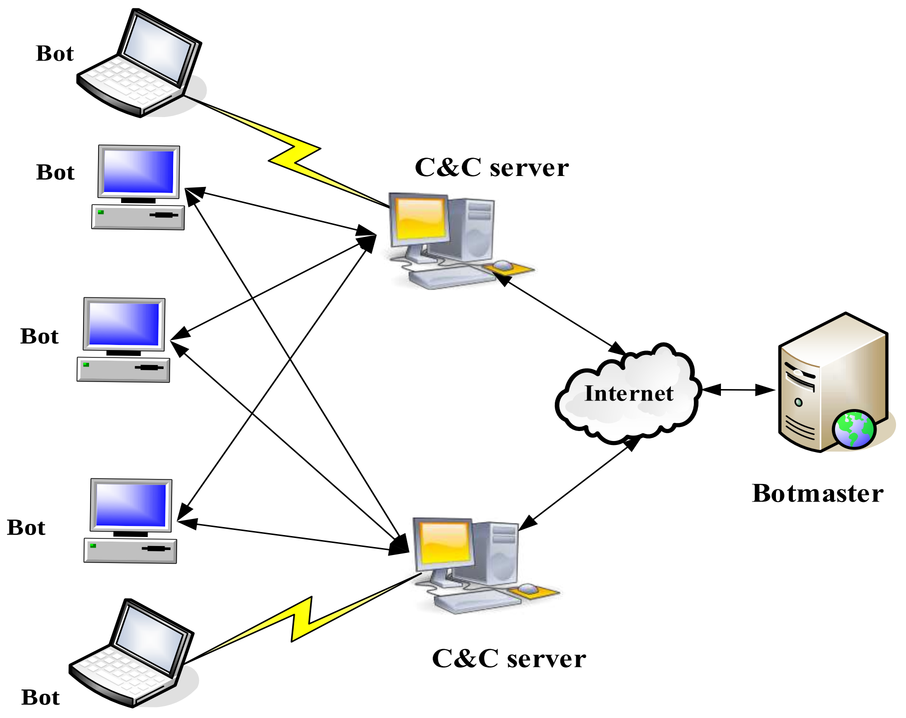

Figure 2.3 A typical centralized Botnet structure.

In 2007, the Storm Botnet showed that the power of decentralizing C&C structure to protect the viability of a Botnet. Decentralizing the C&C introduces a serious challenge to defenders who cannot remove an individual set of points to destroy a Botnet (Grizzard, Sharma, Nunnery, Kang, & Dagon, 2007; Stover, Dittrich, Hernandez, & Dietrich, 2007). A decentralized Botnet architecture is hard to detect as a result of the anonymity involved and the dispersed nature of the P2P network's design (Han & Im, 2012).

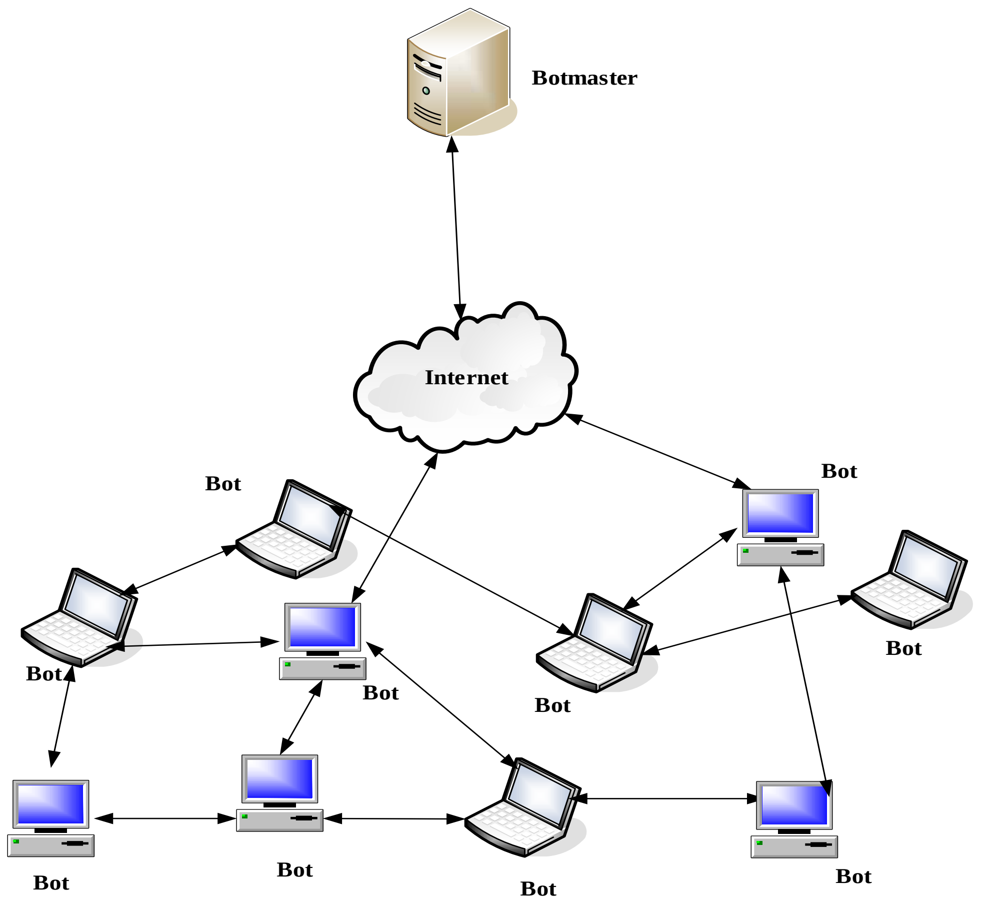

Figure 2.4 A typical decentralized (P2P) Botnet architecture.

- Unstructured C&C (Hybrid) architecture, as demonstrated in Figure 2.5, this model is considered an extreme form of P2P Botnet; where every Bot has a connection with one peer and it does not own information about other peers in the Botnet network. Furthermore, the Bots are organized randomly in this architecture (Rgio S. C. Silva et al., 2013). In this type, there cannot be a direct communication between the Botmaster and the Bot where has to search randomly on the Internet to find a Bot in ordered to submit a new task. What is more, it is not affected by a single point of failure, as is centralized architecture. In addition, Wang et al.(2010) introduced a hybrid Botnet model as a new idea that combined the fundamental characteristics of centralized and decentralized C&C mechanisms in order to gain the benefits of both a low latency of communication and P2P flexibility (P. Wang, Sparks, & Zou, 2010). However, this architecture does not have a warranty for the message delivery, and it suffers from a high rate of C&C message latencies (Bailey et al., 2009).

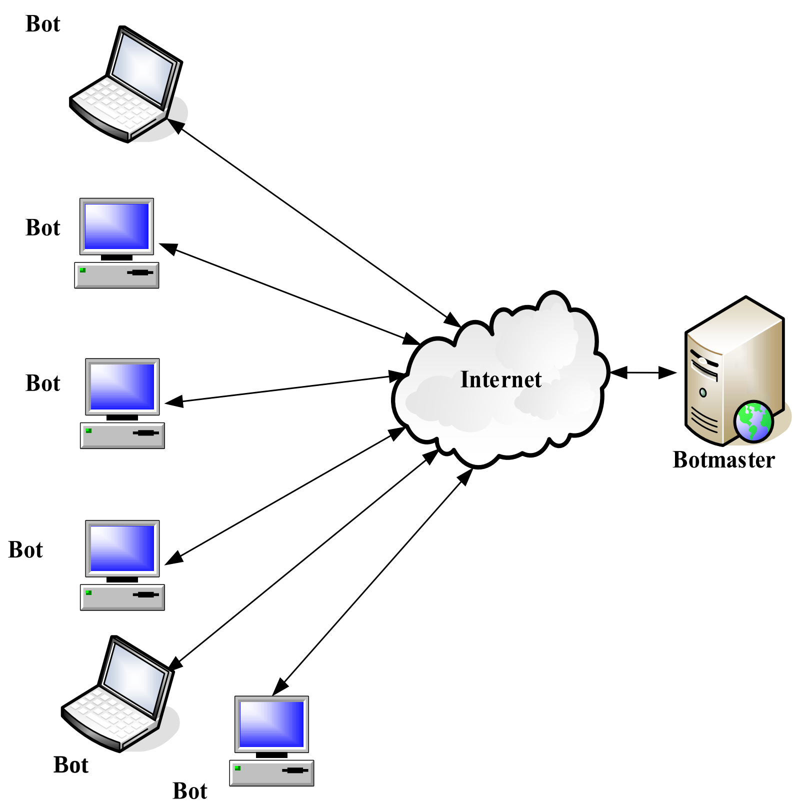

Figure 2.5 Unstructured C&C architecture.

The general properties of the different Botnet structures are summarized in Table 2.1 (Bailey et al., 2009).

Table 2.1 C&C structures and basic properties (Bailey et al., 2009).

Topology	Complexity	Detectability	Message Latency	Survivability
Centralized	Low	Medium	Low	Low
Decentralized (P2P)	Medium	Low	Medium	Medium
Unstructured	Low	High	High	High

2.4.2 Botnet Classification Based on the Communication Protocols

It is necessary to own a communication channel linking the Botmaster with their Bots inside victim machines in order to facilitate the flow of send/receive information between them. Based on existing network communication protocols, Botnets can be categorized according to protocols as IRC-based, Web-based, P2P-based and Custom protocols (A. K. Tyagi & Aghila, 2011). Table 2.2 shows a comparison of Botnet communication protocols.

- IRC-based: In this type of Botnet, an Internet Relay Chat (IRC) channel plays a key role in Botnets development. Initially, the idea of IRC Bots was developed to support chatting services, not taking into account the fact that this idea was utilized by malicious developers, and then the first IRC-based Botnet appeared. The Botmaster used the IRC Botnet to bring the victim machine under control and to exploit it to execute malicious activities. According to Trend Micro report, examples of IRC Botnets are Rbot, Phatbot, GTBot and Sdbot (Trend-Micro, 2006). Nevertheless, the IRC can be efficiently identified by configuring the devices of network security in order to hinder IRC traffic.
- Web-based: HTTP protocol is used by this type of Botnet as the main communication channel, as the basis of the widespread HTTP protocol. The Botmaster uses this protocol to spread malicious activities, which is difficult to detect and capable of bypassing network security devices. Through the World Wide Web, the Botmaster uses HTTP to manage his Bots. The Botmaster identifies a web server, and then the Bots periodically connect to the specific web server in order to receive commands or send information. Unlike IRC Botnets, HTTP Botnet communication can be hidden in legitimate HTTP traffic in order to evade detection systems. There are many examples of this Botnet such as the Rustock Bot (Chiang & Lloyd, 2007) and blackEnergy Bot (Daswani & Stoppelman, 2007). However, HTTP Botnets and IRC Botnets suffer from the disadvantage of a single point of failure in the C&C server (K. Wang, Huang, Lin, & Lin, 2011).
- P2P-based: Napster was the one of the first peer-to-peer networks; P2P protocols then became popular. The main concept of the P2P network is that every node works as a server and client at the same time. Several protocols may be followed such as Gnutella, eDonkey, BitTorrent and Kademlia. The core of these protocols is totally decentralized and that attracted the attention of Botmasters (Mukamurenzi, 2008). P2P Botnets adopt a decentralized architecture using an

overlay network to exchange command and control data making their detection even more difficult. So, the P2P Botnet is named based on its use of P2P mechanisms or protocols. Many Botnets utilize the P2P network, such as the Conficker (R. Weaver, 2010), Storm (Holz, Steiner, Dahl, Biersack, & Freiling, 2008), Nugache (Stover et al., 2007) and Waledac (Stock et al., 2009).

- Custom protocols: In addition to the previously listed types, there are kinds of Botnets that use their own protocols based on the TCP/IP stack, and they only use transport-layer protocols such as UDP, TCP and ICMP.

Table 2.2 Comparison of Botnet communication protocols.

Communication protocols	**Example**	**Topology**	**Weakness**	**Advantages**
IRC-base	Rbot, Phatbot, GTBot and Sdbot.	Centralized	Single point of failure in the C&C server.	It is widely used in cybercrimes, because the commands are sent quickly with low latency.
Web-based (HTTP)	Rustock	Centralized	Single point of failure in the C&C server.	HTTP Botnet communication can be hidden in legitimate HTTP traffic to evade detection systems.
P2P-based	BlackEnerg, Storm and Zeus	Decentralized	-	Avoid the weakness of a single point of failure.

2.5 Taxonomy of Botnet Detection

Recent years have witnessed several Botnet detection techniques which can be classified as signature-based, anomaly-based, DNS-based and data mining-based (Feily et al., 2009). Other researchers such as Han et al. have classified P2P Botnet detection systems into three general types: data mining, machine learning and network behaviour and traffic analysis (Han & Im, 2012). What is more, Zeidanloo and colleagues classify the Botnet detection system as Honeynets or intrusion detection systems (IDS), and they also divide the IDS system into three sub-groups of anomaly-based, specification-based and signature-based. In addition, the Botnet detection system can be classified based on its installation point as host-based, network-based and hybrid systems (Zeidanloo, Shooshtari, Amoli, Safari, & Zamani, 2010). Lu et al (2011) have classified Botnet detection techniques on the basis of machine learning type as supervised or unsupervised Botnet detection (Lu et al., 2011).

2.5.1 Honeynet-based Detection

Honeynets are one of the most common detection methods used by many researchers recently. This technique that imitates an infected machine so as to convince the Botmaster that it is a Bot in his Botnet in order to record all communication and actions between them. This mechanism is commonly used in the initial phase of Botnet detection. A Honeynet method contains two components: the Honeypot and Honeywell (Bacher et al., 2005). The Honeypot points out a vulnerable host. What is more, the Honeywell refers the group of tools used to capture and analyse the send and receive traffic from the honeypot. By utilizing the information gathered by a Honeynet, it is possible to perform a comprehensive analysis and to extract the main features of a Bot to understand its technology and therefore uses the extracted features in improving Botnet detection. GenIII (Balas & Viecco, 2005) and Honeyd (Provos, 2003) are two popular Honeynets in the field of malware detection.

In 2006, Baecher et al. introduced a Nepenthes platform as a framework for collecting information from self-replicating malware based on the honeypot. The Nepenthes framework is one of the most practical ways to provide the developer of an antivirus system with information about unknown malware (Baecher, Koetter, Holz, Dornseif, & Freiling, 2006). Rajab and co-workers proposed distributed multifaceted Honeynets, effectively capturing the activities of IRC Bots (Rajab, Zarfoss, Monrose, & Terzis, 2006). Moreover, the honeypot mechanism was used in the Botminer method to understand the behaviour of two Botnets, Nugache and Storm. However, there are many Botnet detection techniques which utilize Honeynets such as (Barford & Yegneswaran, 2007; Cooke et al., 2005; Freiling, Holz, & Wicherski, 2005; Kang et al., 2009; Pham & Dacier, 2011).

Despite the success of the Honeynet in reducing the effects of Internet malware, it has some shortcomings. It takes time to analyse information about the malware binaries. Moreover, if an attacker has knowledge of the existence of the Honeypot, therefore, it will not send anything to it, or it may send fake commands in order to give the honeypot the wrong information. Table 2.3 summarize Botnet detection methods that utilize the Honeynet.

Table 2.3 Summary of Honeynet detection methods.

Method	Technique	Shortcoming
(Baecher et al., 2006)	Collecting information from self-replicating malware based on the honeypot.	- Malware binaries analysing time. - Providing false attack information by the attacker.
(Rajab et al., 2006)	Distributed multifaceted Honeynets.	

2.5.2 Signature-based Detection

Signature-based detection includes exploring the traffic in the network to find a set of traits such as a series of bytes or sequences of packets and a matching set of pre-specified signature lists. Whenever there is a match in particular network traffic, the administrators are alerted or there a predefined action will be taken by the system. Some IDS, applying the signature approach use a repository to store signatures. The repository is frequently explored to match predefined patterns such as the content of payload packets or system activities to determine whether it contains known signatures. So, the quality of signatures plays a significant role in the performance of signature-based detection. Despite an attempt to generate automatic signatures for malware (Kreibich & Crowcroft, 2004), it is still a restricted to human expertise and knowledge.

Unfortunately, signature-based detection does not have the ability to detect an unknown Botnet. For example (Lu et al., 2011) proposed an approach for detecting the Botnet's malicious traffic by using an n-gram feature selection algorithm to analyse payload content. Then they clustered P2P applications into groups based on payload content using a decision tree model to distinguish between known applications and malicious Botnet traffic. In the clustering stage, three clustering algorithms used in the approach are K-means (Jain, Murty, & Flynn, 1999), merged X-means and un-merged X-means (Pelleg

& Moore, 2000). Moreover, the approach is based on the hypotheses that the diversity of Botnet packet content is less than that of legitimate traffic. Although the approach is able to detect Botnets independently of protocol and network structure, it is vulnerable to methods of encryption of payload content and authorization to read the actual content of the packets. Clearly, this method will no longer work because today's Botnets are much more sophisticated.

SNORT is one of the popular network intrusion detection schemes. It examines the network traffic and applies certain rules/patterns to identify well-known signatures of Bots (Alder et al., 2007). SNORT is suitable for detecting Bots that have information about it, with low FPR and instant detection. However, it fails to classify similar Bots with hardly changed signatures or new types of Bots till their signatures have been determined and attached to the rule set database.

In 2007, Goebel and Holz presented another technique of using a signature-based approach called Rishi. The method works by comparing the IRC communication traffic with known IRC Bot nickname patterns, or using unusual channels for communication (Goebel & Holz, 2007). But the Rishi approach fails to detect non-IRC Bots or new (unknown) nicknames, or if the Bot applies an encryption algorithm in communication.

In 2007, Gu and colleagues suggested a BotHunter that utilizes the correlation analysis of malicious behaviour. It correlates SNORT (Roesch, 1999) alarms in the bidirectional communication between external and internal hosts to detect the C&C communication and malicious activities such as scanning and exploit usage. Then this evidence is used in a rule-based system to detect the host infected by the Botnet (Gu, Porras, Yegneswaran, Fong, & Lee, 2007). The BotHunter also has its weaknesses. This method will be avoided if Botnets update their predefined infection procedures or if the C&C interactions frequency is very low (Gu, Zhang, et al., 2008). In general, the main advantage of signature-based approaches is to achieve a high detection rate since it uses the signature found in the database. However, a major drawback is its incapability to detect new Bot attacks, or so-called zero-day attacks (N. Weaver, Paxson, Staniford, & Cunningham, 2003). Another drawback of signature-based detection is that it needs the involvement of human expertise to create the signatures.

Table 2.4 Summary of signature-based methods.

Method	**Technique**	**Shortcoming**
(Lu et al., 2011)	K-means, Un-merged X-means, Merged X-means clustering	- Payload encryption content. - Privacy issue.
(Goebel & Holz, 2007)	N-gram analysis	- Fail to detect non-IRC Bots. - Payload encryption. - Detect zero-day attack.
BotHunter (Gu et al., 2007).	Correlation analysis	- Detect zero-day attack. - Need human expertise to create the signatures.

2.5.3 Anomaly-based Detection

Anomaly-based detection techniques have been explored a lot in the last decade, and they are the most general detection technologies. They try to determine the "normal" behaviour of the system to be protected and then look for any considerable changes in network behaviours (García, Zunino, & Campo, 2014). This includes any behaviour that is considered an unusual activity such as traffic at uncommon ports, network traffic with high volumes, latency with high network traffic and abnormal system behaviour based on a predefined pattern of normal system behaviour. These approaches attempt to build a model of abnormal system behaviour in order to find any similarities with previously expected malicious behaviour located in the range of a given threshold. According to Zeidanloo and co-workers, anomaly-based methods are classified on the basis of data collection location into host-based and network-based. Network-based techniques can be broken down into active and passive (Zeidanloo, Shooshtari, et al., 2010). The main advantage of anomaly-based methods is their ability to detect new types of attacks, known as zero-day attacks. These attacks are malicious activities that are not already known by the detection system, and cannot be detected by signature-based approaches. However, the quality of the features selected for use in the detection system and high false alarm rates are the most common limitations of such detection approaches which apply anomaly-based techniques (N. Weaver et al., 2003).

Host-based

A host-based Botnet technique attempts to detect a Bot binary as a virus and so, it treats the infected machine as a way of anti-virus software. This approach is based on the hypothesis that a Bot programme executes a series of calls to system libraries which are dissimilar to those performed using normal processes (Trend-Micro, 2006). Host-based techniques monitor the machines activities and record system events such as remote control activities, register updates, file deletion and traffic sent to or received from a host. An alert is activated when it detects Botnet activities on the host.

In 2007, Stinson and Mitchell proposed a BotSwat as a host-based detection technique based on the above premise. BotSwat has tools to monitor and track the interactions of computer program calls with system libraries that receive data from the untrusted network in order to discriminate between Botnet command responses from normal host activities. Moreover, this method was created with the aim to detect Botnets independent of C&C architectures or communication protocols (Stinson & Mitchell, 2007).

EFFORT (Seungwon, Zhaoyan, & Guofei, 2012) a host-based detection approach that collects Bot characteristics at client and network levels, and correlates Bot-related information by monitoring local computer activity such as keystrokes and monitoring connections with other computers. This approach applies one class of supervised support vector machine algorithms to model legitimate user behaviour (Witten & Frank, 2005). Furthermore, fifteen Bot samples were used to evaluate the method and a 100% true positive rate was achieved with less than 1% FPR. The main advantage of this method is that it does not depend on the protocol and communications topology used. In addition, it is able to detect Bots that use encryption techniques to hide malicious behaviour. The major limitations of this method are critical to evasion techniques, such as fast-flux, and it also cannot be proven as a real-time detection approach.

In 2008, Liu and colleagues introduced a BotTracer as a Botnet detection tool based on a virtual machine. This method is based on the idea that a Bot has three features. Firstly, the Bot has automatic start-up activities without involving any user actions. Secondly, the Bot must establish C&C a communication channel with its Botmaster. Finally, a Bot must launch an attack remotely or locally. These features represent the three basic stages of a

Bot attack: injection, update, and attack. Besides this, a Bot should communicate with a rendezvous point in order to launch a C&C channel with its Botmaster, and BotTracer catches these channels and analysed them to identify the Bot C&C channels. The BotTracer runs a virtual machine on the host that contains a copy of the host system file that automatically starts without human interaction. Then, it will monitor all auto start communication processes to find a Bot C&C channel fingerprint (Liu, Chen, Yan, & Zhang, 2008). Therefore, it will detect the Bot when it begins a malicious activity. This is a real-time technique that is capable of detecting unknown Bots without considering the communication protocol. Moreover, it achieves a low FPR regardless of the encryption of Botnet communication traffic. However, in BotTracer high levels of computation are required due to the virtual machine's degradation of host performance. What is more, many Bots have the ability to check for the presence of a virtual machine, so, in this case, the BotTracer will not work.

Al-Hammadi and Aickelin proposed a P2P Botnet detection approach by correlating behavioural features. The approach developed a program to monitor and extract suspicious API function calls in order to use these features as input to the correlation algorithm. Moreover, the Storm P2P Bot was used as a case study (Al-Hammadi & Aickelin, 2010). However, the main shortcomings of this technique are that the detection threshold is undefined, and it is evaluated using only one type of Bot. Another host-based study in Botnet detection introduced by Nummipuro presented some of the P2P Botnet's behavioural characteristics such as using the System Service Table (SST) Hooking (Nummipuro, 2007). Although this host-based approach achieved satisfactory results in reducing the spread of malware, it works an individual host and so the monitoring and analysis operation is costly, complex and non-scalable.

Table 2.5 summarize a host anomaly-based Botnet detection methods.

Network-based

Nowadays, network-based approaches are widely used for Botnet detection by analysing the entire network traffic (Barsamian, 2009). Furthermore, this technique is installed at the end of the network such as in the firewall or router unlikely host-based methods that analyse individual host activities. Network-based approaches have been further divided into active and passive monitoring.

Table 2.5 Summary of host anomaly-based methods.

Method	Technique	Shortcoming
EFFORT (Seungwon et al., 2012)	SVM and one Class SVM.	- Critical to evasion techniques, such as fast-flux. - Not proven as a real-time detection approach.
BotTracer (Liu et al., 2008).	Virtual machine.	- Virtual machine's degradation of host performance. - Providing false attack information by the attacker.
(Al-Hammadi & Aickelin, 2010)	Correlation algorithm.	- The detection threshold is undefined. - Evaluated using only one type of Bot.
(Nummipuro, 2007)	Using the System Service Table (SST) Hooking.	- The leak of scalability.

In passive monitoring techniques, information about traffic on the network is gathered to find suspicious communications in order to detect Botnets. A key idea behind passive monitoring is that Bots create communication behaviour different from that of a normal host and Bots belongs to a Botnet network that presents similar communication patterns (Trend-Micro, 2006). The Botmaster has to make connections with its Bots to issue an attack or update command. Moreover, because the Bot is pre-programmed, they react with the Botmaster using a similar pattern. Furthermore, the Botnet uses the same protocol in each phase of the Botnet life cycle (Trend-Micro, 2006). Many researchers have investigated such similarities in network traffic to identify Bot behaviour.

For example, Gu colleagues (2008) use the fact that Bot is pre-programmed software and has a similar pattern to the C&C server to develop a BotSniffer detection method based on the spatial-temporal correlation. It depends on the hypothesis that Botnets favour to contact in an extremely synchronized way, unlike human activities. BotSniffer can identify C&C servers and a compromised host based on the similarity of spatial-temporal data. Additionally, it can recognize C&C channels for IRC-based and HTTP-based Botnets. What is more, this technique is network-based, so it can identify a host with comparable suspicious network behaviour such as spamming and scanning (Gu, Zhang, et al., 2008). However, Botsniffer was developed to detect Botnets with a centralized architecture. Consequently, it cannot identify Bots that use a different architecture for the C&C server and it is not able to recognize an individual infected host. Moreover, it was developed to identify a Botnet in a local area network, so it is not applicable at the Internet level. Also despite having a low FPR, the Botsniffer can be avoided by utilizing encoded channels or using a decentralized architecture for the C&C server as in P2P Botnets.

In 2007, Karasaridis et al. presented an anomaly-based algorithm for detecting IRC Botnet controllers using the transport layer data in the backbone of the network, such as Tier-1 ISP networks. The statistical characteristics of the C&C server traffic are used to find considerable quantities of the data of the network traffic. This data is gathered by utilising the sceptical host activity findings (ports scan, email spam and generating distributed denial of service attack traffic) collected from chosen network connections by matching a well-known IRC traffic signature, such as the low amount of network traffic, chat-like or a network traffic which has a PING-PONG pattern. After collecting network data methods are applied to detect the connections of candidate controllers that use unusual IRC ports. Firstly, it finds the suspected Bot flow with a remote machine that acts as a server. Secondly, it identifies flows whose behaviour is within the range of normal IRC traffic. Finally, they analyse the conversation of a candidate control to recognize suspicious controllers and their ports (Karasaridis, Rexroad, & Hoeflin, 2007). However, although the Karasaridis technique is able to work passively with large-scale networks and achieve less than 2% FPR, and so it is suitable to detect IRC Bots, but it may not be able to detect modern kinds of Botnet such P2P and HTTP.

As opposite to passive monitoring that interacts with Botnet behaviour, active monitoring techniques interact with a Botnet directly by probing the network host with active communication and analysing its responses. Moreover, it actively confuses Botnet activity by meddling with the Bots’ communication with the C&C server. The majority of detection techniques are passive, while only a few, such as BotProbe (Guofei, Yegneswaran, Porras, Stoll, & Wenke, 2009), are active.

BotProbe was introduced by Gu and colleagues as an active detection mechanism. The main target of BotProbe is to determine whether or not a Bot or user is using the host at that side by injecting packets dynamically in a communication session. The authors noted that a Bot is a pre-programmed reply to any contact based on a set of predefined rules. So, they discriminated the human client from a Botnet with regards to the frequency and pattern of responses. This technique was tested on a number of IRC Bots and around 100 real users (Guofei et al., 2009).

However, active techniques have the serious shortcoming of greatly increasing network traffic by sending extra packets to suspicious clients. Furthermore, and most essentially, injecting packets to facilitate detection may be lead to legal issues. In addition, the passive detection approach has the advantage of detecting a Botnet without any direct interaction with the Bot, but only using the Bots behaviour within a network. Table 2.6 summarize network anomaly-based Botnet detection methods.

Table 2.6 Summary of network anomaly-based methods.

Method	Technique	Shortcoming
BotSniffer (Gu, Zhang, et al., 2008)	Spatial-temporal correlation	- Payload encryption content. - Privacy issue. - Detect single Bot infection.
(Karasaridis et al., 2007)	Correlation algorithm	- Fails to detect non-IRC Bots. - Detect Zero-day attack.
BotProbe (Guofei et al., 2009)	Injecting packets in a communication session.	- Increasing network traffic - Injecting packets to facilitate detection may be lead to legal issues.

2.5.4 Machine learning based Detection

Machine learning plays a significant role in the domain of artificial intelligence because it has excellent performance, and so it is widely used in many fields such as date mining, pattern recognition, and medical diagnosis. Machine learning algorithms extract hidden relationships and rules within data, which can be used to create models for prediction and classification, and thus its goal is to construct systems which have the ability to learn from data (Mitchell, 1997; Witten & Frank, 2005). Learning in this context indicates the ability to identify complicated patterns and utilize labelled data to make qualified decisions. One of the main challenges in machine learning is how to make a generalization of knowledge extracted from a previous dataset or derived from a limited set of previous experiences, in order to construct a prediction system for new and unseen datasets. To deal with this problem, algorithms are developed based on statistical, artificial intelligence, information theory, biology, philosophy, cognitive science, control theory and computational principles (Mitchell, 1997). Machine learning algorithms are classified in terms of the type of learning involved, which are: supervised learning, unsupervised learning and RL.

Supervised learning algorithms are trained using a labelled dataset to generate a model that is able to classify an unlabeled dataset in the future. It is as if a supervisor is helping you out, to be able to classify in the future, which is why it is called supervised. The principle of supervised learning is used by popular machine learning algorithms, for example, in Classification and Regression Trees (CART) (Breiman, Friedman, Olshen, & Stone, 1984), neural networks (Gurney, 1997) and Support Vector Machines (SVM) (Cristianini & Shawe-Taylor, 2000). The field of supervised learning may be divided into classification and regression problems. In Botnet detection problems, supervised machine learning mechanisms are employed to train with both Botnet traffic datasets and normal traffic datasets in order to construct classifiers.

Compared to supervised learning, unsupervised learning algorithms do not require a labelled dataset for training. The goal of unsupervised learning methods is to divide an unlabeled dataset into different sub-groups depending on specific metrics. Furthermore, the dataset is learned from in order to understand its structure and to find patterns, instead of creating a generalization model from an available labelled dataset as in supervised learning approaches. Nilsson defined unsupervised learning as the use of “procedures that attempt to find natural partitions” (Nilsson, 1996). The most common unsupervised learning algorithms used to detect Botnets are hierarchical clustering, X-means and K-means algorithms.

In RL approaches, an agent learns what to do via some experiences including trial and error (Barto & Andrew, 1998). RL agents modify themselves according to the states of the environment to increase the number of rewards gained in the long run. To maximize the gains, RL agents estimate action-value function, which are specified as the relationships between state-action pairs and the measures of returns that the agents will obtain in the future. More details of RL are given in Chapter 5.

A recent study in the field of P2P Botnet detection by Babak et al.(2014) proposed a PeerRush, which uses a one-class classification approach to classifying various types of normal and abnormal P2P traffic. One-class classifies including the k-Nearest Neighbors algorithm (KNN), Parzen, and Gaussian data description classifiers (TAX, 2001) are used. An application profile is initially created by learning traffic samples of known P2P applications. Moreover, features such as interval delays between packets and flow duration are used to classify P2P applications (Babak, Roberto, Andrea, & Kang, 2014). This approach achieves high accuracy rates in classifying P2P applications depending on the features selected. On the other hand, this method does not show clearly how to detect P2P Botnets, and also detection can be easily avoided by changing the delay between packets.

Garg et al. (2013) presented several machine learning algorithms, such as KNearest Neighbour, Naive Bayes, and J48. These were analysed for the detection of P2P Botnets using various network traffic features. The results show that the accuracy of the classifiers trained using the Nearest Neighbour and J48 is good (Garg, Singh, Sarje, & Peddoju, 2013). However, the detection of legitimate traffic is very weak.

Jiang and Shao (2012) presented a method that focuses on the C&C traffic of P2P Bots regardless of how they perform their malicious activity. This method developed a detection mechanism based on Bots that exhibit connection flow dependency with other Bots in the same Botnet network. According to the flow dependency behaviour, this approach uses a single-linkage hierarchical clustering mechanism to differentiate between P2P Bots and normal hosts (Jiang & Shao, 2012). This method was built based on the similarity of Botnet traffic, and so it will fail to detect Botnets that use the irregularity of traffic flow, such as Storm Bot (Li, Hu, & Yang, 2012). Also, it has a limitation in identifying individual Bot behaviour.

One study by Junjie et al.(2011) introduced a P2P Botnet detection system that can identify stealthy P2P Botnets. The proposed approach focuses on identifying Bots based on the monitoring of C&C traffic. They extracted four features for each traffic flow, including the numbers of bytes received and sent and numbers of packets received and sent. The hierarchical clustering (Jain et al., 1999) and BIRCH algorithms (T. Zhang, Ramakrishnan, & Livny, 1997) were used to cluster network flow (Junjie, Perdisci, Wenke, Sarfraz, & Xiapu, 2011). Furthermore, the approach is independent of payload signatures and has also achieved high detection rates of both malicious and legitimate hosts, with an FPR of 0.2% and TPR of 100%. Although this system can detect Botnets regardless of how they perform their malicious activities, it focuses only on P2P Botnets and cannot detect other types such as IRC or HTTP Bots. However, the proposed technique is vulnerable to some evasion methods such as flow disturbance packets and using the DGA and Fast-flux algorithms as a communication facility to provide a high level of C&C privacy.

Wen-Hwa and Chia-Ching (2010) used a methodology based on packet size to distinguish between P2P Botnet traffic and legitimate P2P traffic. They presented the following observations. Firstly, P2P Bots try to update information for other Bots rather than staying idle. Secondly, the Bot mainly transmits data with a minimum rate of connections. Bayesian networks, Naïve Bayes and J48 are used to classify network traffic (Wen-Hwa & Chia-Ching, 2010). Furthermore, the accuracy rates for these three algorithms are 87%, 89% and 98% respectively. However, it was found that the size of P2P Botnet packets is smaller than that of any other P2P applications.

Zhao and co-workers (2010) introduced a P2P Botnet detection system using machine learning techniques based on the flow intervals of network traffic. In addition, they applied a Bayesian network and decision tree (REPTree) as a classification method to investigate online P2P Bot detection (Zhao et al., 2013). The main drawback of this technique was its sensitivity to evasion methods such as the random connection interval. For example, the connection interval of the Srizbi Bot is random in the interval from 60 to 1200 seconds (Dae-il, Kang-yu, Minsoo, Hyun-chul, & Bong-Nam, 2010).

Nogueira et al.(2010) introduced a Botnet detection approach based on the identification of traffic using artificial neural networks to classify legal and illegal patterns (Nogueira, Salvador, & Blessa, 2010). This technique has several advantages, such as being independent of protocol and network structure and having the ability to detect encrypted Bot traffic. Nevertheless, the trained neural network was able to classify only 87% of network traffic. The main drawback is the need for external judgment in order to provide adaptive operation.

In 2012, F. Tegeler et al. introduced a network Botnet detection approach called BotFinder, which detects separate hosts infected by Bots focusing on the statistical features of network flow based on frequent Bot C&C communications constructed in a controlled environment. Additionally, they used clustering based on a local Shrinking algorithm (X. Wang, Qiu, & Zamar, 2007) as the machine learning method used to separate the captured network flow into legitimate and malicious classes where the final model will decide whether the flow generated by hosts is malicious or not (Tegeler, Fu, Vigna, & Kruegel, 2012). On the other hand, BotFinder has detection rates varying from 49% for the Bifrose Bot to 100% for the Banbra Bot. This technique has several advantages such as IP address blacklisting or DPI of contents being unnecessity.

The detection system introduced by Fedynshyn et al (2011). uses a host-based approach to detect Bots using the property of temporal persistence. They utilized a J48 classifier and a Random Forest algorithm to sort various kinds of Botnet infection categorized according to C&C model (HTTP, IRC and P2P). Moreover, they found similarities in C&C structures for different categories of Bots that are different from those of legitimate network traffic (Fedynyshyn, Chuah, & Tan, 2011).

A recent study in the Botnet detection field by Saad et al.(2011) addresses the P2P Botnets detection problem by using several machine learning techniques, including an artificial neural network (ANN), linear SVM, a Gaussian based classifier, Nearest Neighbour classifier, and a Naive Bayes classifier (Witten & Frank, 2005). The study evaluated the ability of these machine learning techniques in terms of on-line Botnet detection requirements such as adaptability, novelty detection and early detection (Saad, 2011). They showed that all of the machine learning algorithms had great potential for detecting patterns of Botnet traffic, achieving detection rates greater than 89%. However, SVM and ANN took the most time in the training phase. Furthermore, the performance of these techniques is highly dependent on the features selected for classification or cluster analysis and they often have high computational requirements.

Strayer et al.(2006) introduced one of the first techniques that utilize machine learning for the purpose of Botnet detection in network traffic. This approach is an extension of Strayer's previous work (Strayer, Walsh, Livadas, & Lapsley, 2006) and works conducted by Livadas et al. (Livadas, Walsh, Lapsley, & Strayer, 2006). Bayesian Network, C4.5 Tree and Naive-Bayes classifiers as machine learning approaches were evaluated in classifying IRC traffic as legitimate or malicious flows (Timothy, David, Robert, & Carl, 2008). Although these methods were effective in detecting Botnets, the techniques are still restricted to particular types of Botnets such as IRC Botnets or specific architectures such as centralized hierarchies. Furthermore, they need human experts to make the final decision.

Masud et al. (2008) introduced an approach to Botnet detection based on the observation that a Bot has many reaction patterns that are different from those of humans. This approach can detect Bots by correlating incoming packets with outgoing packets, new outgoing connections, and application startup in hosts. Several machine learning algorithms such as the C4.5 decision tree, support vector machine, Naive Bayes, Bayes network classifier and Boosted decision tree (Witten & Frank, 2005) were compared and evaluated in the detection of IRC Botnets. The result of the evaluation showed that all machine learning algorithms achieved over 95% detection rates, less than 3% FPR and under 5% false negative rates (FNR). The greatest overall performance was reached by a Boosted decision tree (Masud, Al-khateeb, Khan, Thuraisingham, & Hamlen, 2008). However, one major drawback of this approach is that it cannot detect Botnets that use encrypted communication due to the need to access the contents of payload packets. On the other hand, the method has been tested on IRC Bots, so it is unable to deal with modern types of malware such as P2P Botnets.

Gu et al. (2008) introduced Botminer as a network-based detection method which detects Botnet by correlating machines with comparable malicious activities and comparable C&C communications. Botminer utilizes X-means and hierarchical clustering methods to identify a Botnet using the observation that a Botnet is a collection of malware instances that are administered through the C&C channel and it has a similarity of the temporal behaviour. The detection process operates by detecting hosts with activities of similar communications in the C-plane where hosts are communicating with different hosts, in other words, hosts which its traffic flows are related in respect of flows per hour (fph), bytes per second (bps), bytes per packet (bpp) and packets per flow (ppf). Besides this, hosts are defined with traffic of similar attack in the A-plane showing who hosts is doing what, such as hosts performing ports scan, downloading the same files and spamming. The detection results are obtained by creating a cross-correlation between the A-Planes and C-Planes in order to classify machines that share similar malicious activity patterns and similar communications (Gu, Perdisci, et al., 2008). The main advantages of Botminer it can identify several Botnet kinds such as IRC-based, P2P-based and HTTP-based Botnets with 99% true positive rate and low FPR around 1%. Nevertheless, correlating activities that generated by various hosts needs at least two machines on the network be infected by the similarly Bot type. Consequently, Botminer fails in the situation of an individual machine is infected by Bot or when several machines inside the network are infected with diverse Bots types. Furthermore, Botminer fails to detect Bots that exchange C&C messages without any suspicious activity.

In Wei et al. (2009) study, they suggested BotCop as an online Botnet traffic detection system. In this method, network traffic is categorized into various applications using a decision tree technique. The network's payload characteristics are utilized and then, based on each application community obtained, the temporal frequency properties of their flows are examined to classify a communication as malicious or legitimate traffic (Wei, Tavallaee, Rammidi, & Ghorbani, 2009). Table 2.7 summarize machine learning based Botnet detection methods.

2.5.5 DNS-based Detection

At the same time as efforts to detect Botnet passively based on network traffic, other researchers started to look for suspicious Botnet behaviour in DNS traffic. The Domain Name System (DNS) is a distributed naming system for devices that are connected to the Internet; the DNS is responsible for converting domain names to IP addresses (Goerzen, 2004). Bots exploit the DNS to find the Botmaster IP address, and the DNS responds by giving IP addresses that connect the compromised computers with the C&C server.

Accordingly, Kristoff (2005) introduced a technique that can identify a Botnet by monitoring the DNS traffic, and the technique blacklists any connected servers that spread malicious malware (Kristoff, 2005). In 2005, Dagon detected the activity of Botnets using a comparison of the rate of malicious DNS to legitimate DNS traffic (Dagon, 2005). However, both approach can easily be avoided, whenever the Botmaster generates a fake DNS query or applies DDNS queries.

Table 2.7 Summary of machine learning based detection methods.

Method	Technique	Shortcoming
PeerRush (Babak et al., 2014)	One-class classifies including the k-Nearest Neighbors algorithm (KNN), Parzen, and Gaussian data description classifiers	- Evaded by changing the delay between packets.
(Garg et al., 2013)	KNearest Neighbour, Naive Bayes, and J48	- Detection of legitimate traffic is very weak.
(Jiang & Shao, 2012).	single-linkage hierarchical clustering mechanism	- Detection single Bot infection.
(Junjie et al., 2011).	The hierarchical clustering and BIRCH algorithms	- Fail to detect non-P2P Bots. - Evaded by DGA and Fast-flux algorithms.
(Wen-Hwa & Chia-Ching, 2010)	Bayesian networks, Naïve Bayes and J48	- NAT technology makes it difficult to detect P2P flows.
(Zhao et al., 2013).	Bayesian network and REPTree decision tree	- Sensitivity to evasion methods such as the random connection interval.
(Nogueira et al., 2010)	Artificial neural networks	- Need an external judgment to provide adaptive operation.
(Timothy et al., 2008)	Bayesian Network, C4.5 Tree and Naive-Bayes classifiers	- Fail to detect non-IRC Bots. - Need human experts to make the final decision.
(Masud et al., 2008)	C4.5 decision tree, SVM, Naive Bayes, Bayes network and Boosted decision tree	- Payload encryption content. - Privacy issue.
Botminer (Gu, Perdisci, et al., 2008).	Spatial-temporal correlation	- Detection single Bot infection. - detect Bots that exchange C&C messages without any suspicious activity.

Choi et al.(2009) introduced BotGAD as an anomaly detection approach based on group activities on Botnet DNS traffic (Choi, Lee, & Kim, 2009). The authors indicated that the group activities of DNS were key features of traffic used to differentiate a Botnet DNS from a normal DNS request. BotGAD is capable of detecting novel Botnet attacks on networks of huge scale in real time. The main weakness of the method is that it requires a long time for processing to monitor large volumes of network traffic (Han & Im, 2012). What is more, it is able to detect Botnets that execute group DNS traffic activities. Thus, it cannot detect Bots, which use the DNS once and never return to it.

Some of the common DNS-based techniques try to detect Botnets by detecting anomalies in DNS traffic (Villamarin-Salomon & Brustoloni, 2008), or detecting Bots based on DNS group behaviour (Choi & Lee, 2012), using DNSBL (DNS Black List) (Ramachandran, Feamster, & Dagon, 2006), or constructing a reputation system for DNS queries (Antonakakis, Perdisci, Dagon, Lee, & Feamster, 2010). But many new models of Botnets as P2P and hybrid P2P do not involve DNS services in their operation, and so these approaches are significantly limited in detecting such Bots (Stevanovic, Revsbech, Pedersen, Sharp, & Jensen, 2012). Summarize of DNS-based Botnet detection methods. Summarize of DNS-based Botnet detection methods.

Table 2.8 Summary of DNS-based detection methods.

Method	**Technique**	**Shortcoming**
(Kristoff, 2005)	White and black lists	- Avoided by generating a fake DNS query - Avoided by using DDNS queries.
(Dagon, 2005).	Correlation algorithm	- Avoided by generating a fake DNS query - Avoided by using DDNS queries.
BotGAD (Choi et al., 2009)	Monitoring group behavior through DNS traffic.	- Detection single Bot infection - Processing time.

2.5.6 Hybrid Botnet Detection Approaches

In parallel with standard network-based and client-based detection techniques a new class of hybrid detection methods has appeared. This type of method detects Botnets by collecting the features of Bots at both client and network levels. The main reason behind hybrid strategies is that it is likely to afford increases in performance in Botnet detection by connecting findings from client-based and network-based detection systems.

For example, Yuanyuan et al. (2010) proposed a hybrid detection approach that detects Botnets by combining the host and network level behaviour. The approach is based on the hypothesis that two sources of Bot observations will complement each other in making detection decisions. The structure of the approach consists of three parts: network analysis host analysis and a correlation engine (Yuanyuan, Xin, & Shin, 2010). Another study by Wang et al. combined three detection approaches (Szymczyk, 2009). In Honeypot-based Botnet detection, were host-based Botnet detection and network-based Botnet detection methods are all utilized.

2.6 Summary

The existing Bot and Botnet detection systems described above have advantages and shortcomings compared to others. Different Botnet detection methods can be categorized based on various measures, such as being host-based or network-based, detecting individual Bots or Botnet networks, machine learning based, anomaly-based or signature-based Bot detection. They may be limited to one class of C&C topology or can detect Bots that apply multiple C&C structures.

3 TRAFFIC REDUCTION APPROACH FOR BOT DETECTION

3.1 Introduction

Previous chapters have presented the Botnet phenomena and demonstrated why former work has not been adequate to counter the Botnet menace. Furthermore, various existing detection methods are still confined to detecting Botnet in an off-line way because it was unable to analysis the whole network traffic immediately.

One challenge of a network-based Botnet detection system is the inability to monitor and analyse the network traffic in high-speed networks in the real time. Packets that are not checked on time can be ignored and these packets may comprise the attack signature (Yang, Fang, Liu, & Zhang, 2004).

On the one hand, the benefit of the network-based detection is that it has a more extensive scope than the host-based detection schemes (Egele, Scholte, Kirda, & Kruegel, 2008; Marpaung, Sain, & Hoon-Jae, 2012). On the other hand, the difficulty of network-based systems is that the volume of traffic will grow, which indicates that extra traffic will be added to the network traffic (Rgio S. C. Silva et al., 2013). Therefore, traffic reduction technique becomes essential to help the detection system to work online. Consequently, the main goal of this chapter is to introduce traffic reduction method gives the detection approach the ability to work online at the network level despite the size of network traffic. Furthermore, it not influenced by packets encryption algorithm.

The next section provides an overview of the proposed detection system. A briefly detailed for each component of the Botnet detection framework is presented. Section 3.3 presents the network traffic capture approach. Section 3.4 presents the traffic reduction technique. Section 3.5 introduces the traffic reduction algorithm and traffic reduction evaluation. Discussion and the chapter summary are given in Section 3.6. and 3.7 respectively.

3.2 Overview of Bot Detection Approach

The architecture of the proposed P2P Bot detection system consists of four main components: network traffic capturing, network traffic reduction, network traffic feature extraction and malicious activity detection using a RL agent as shown in Figure 3.1.

The traffic capture module is an interface between packets of the network and the proposed system; packets detected in the monitored network will be forwarded in their raw form to the subsequent phase in the framework. The phase of the reduction network traffic is responsible for filtering that traffic and is achieved by selecting TCP control packets as a representative of the connection conversations of the captured network traffic.

In the feature extraction phase there are two processing phases: parsing connection conversations and host feature extraction. The parsing connection conversations phase is responsible for constructing connection features between nodes inside the monitored network, according to packets captured and the size of the selected slide window. The connection is defined as a 5-tuple: source IP, destination IP, source port, and destination port and protocol type. In the node feature extraction phase, the vectors of node features are constructed that represent the status of the node based on connection features during the current sliding time-window. More details of connection-level features are given in Chapter 4.

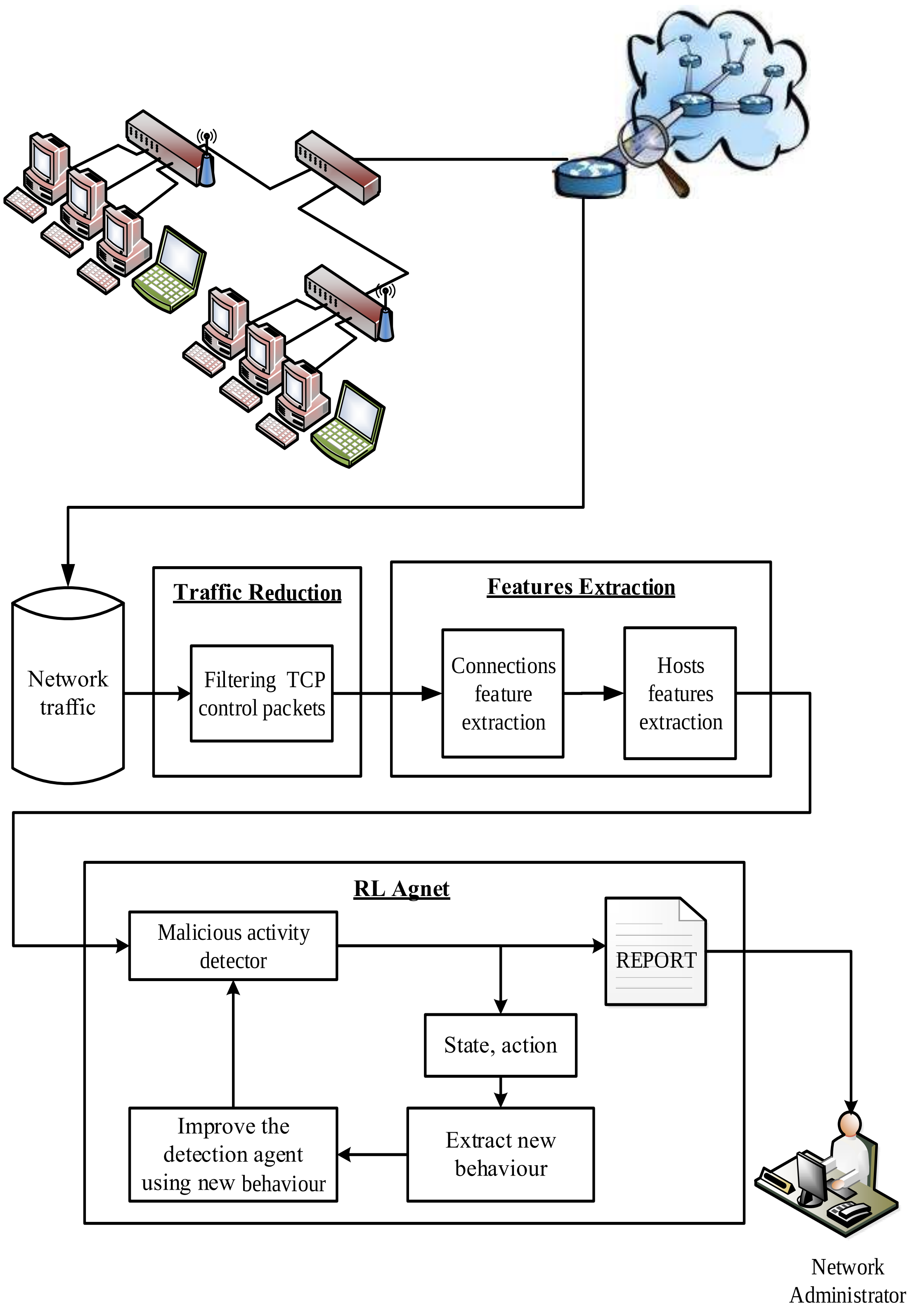
Network traffic
Traffic Reduction
Filtering TCP control packets
Features Extraction
Connections feature extraction
Hosts features extraction
RL Agnet
Malicious activity detector
REPORT
State, action
Extract new behaviour
Improve the detection agent using new behaviour
Network Administrator

Figure 3.1 Bot detection framework

The malicious activity detector takes a node feature vector as input and classifies nodes according to their features into two categories: malicious nodes infected with a Bot or normal legitimate nodes. So, a multi-layer feedforward neural network with resilient learning is selected as a classification algorithm due to its high adaptive and the accuracy rates. The RL agent is responsible for evaluating neural network decisions in order to extract new features which help to improve the detection system in recognizing a zero-day attack. More details of RL are given in Chapter 5.

3.3 Network Traffic Capture

The network traffic capture tool is utilized as an interface between the monitored network and the second stage of the proposed detection system. The primary objective of this phase is to sniff network traffic according to the specific size of the sliding time-window mechanism in order to forward it to the traffic reduction phase

The result of this stage consists of raw captured packets. In our experiments, network packets are passively captured using a Java library used for capturing network traffic, namely Jpcap (Shen & Wang, 2009). One of the primary reasons for adopting a passive strategy is that it does not raise the amount of the traffic inside the network. In addition, the passive detection approach has the advantage of detecting Botnets without any direct interaction with them, but only the Bot behaviour within the network is used. This means that the detection technique operates stealthily and cannot be detected by the Botmaster. However, in this research several time-window sizes are evaluated in order to determine a suitable size of time-window that can improve the accuracy.

The packets traffic flows appear continuously at the network edge, which is difficult to analyse all packets streams immediately. In the proposed method, it only deals with the new packets in the traffic flow in order to detect malicious behaviour instantly. Consequently, we adopt the sliding time window mechanism to capture the recent packets. As shown in Figure 3.2, the time window slides are going to hold the newest packets continuously as time passing, and then send these packets to the next step of the proposed approach.

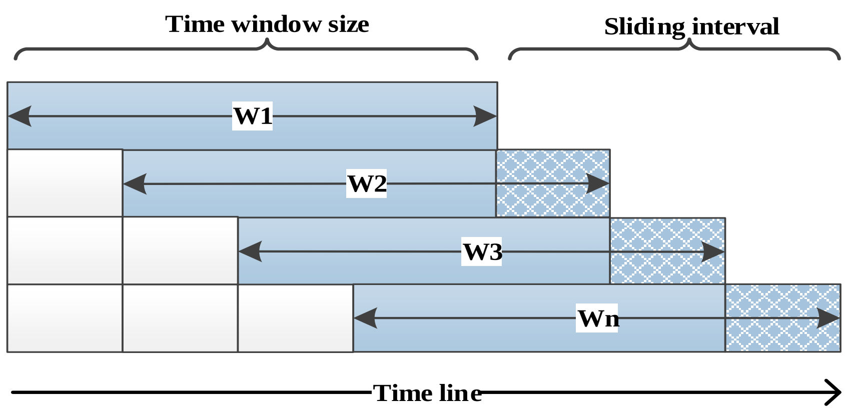

Figure 3.2 The sliding time-window mechanism.

3.4 Network Traffic Reduction

Nowadays, the number of packets passing through a high-speed network is massive and is affected by the capacity of the links and the number of Internet users. Thus, the reduction of network traffic for the detection of malicious activities is essential in order to manage enormous amounts of network traffic when resources such as memory and hard disk space are restricted. The most difficult part of the reduction of network traffic is to identify it is behaviour by inspecting a small number of packets per flow.

Therefore, this study introduces a new traffic reduction technique to facilitate the deployment of Bot host detection systems on high-speed networks. As discussed above, the majority of Botnet detection systems rely on DPI and examine the entire network traffic. DPI assumes that the payload of every packet will be examined. This technique can be accurate when the payload is not encrypted. However, the majority of new types of malware generation apply evasion methods such as the encryption of payloads or protocol encapsulation and obfuscation, which mean that the payload is concealed (P. Wang et al., 2015). Furthermore, examining all packets in a high-speed network is an expensive task because of the speed of networks and the amounts of packets transferred, which is continually increasing. However, a detection system which applies DPI may suffer from efficiency limits due to on processing a large volume of traffic from high-volume or high-speed networks (Jun et al., 2008). The goal of the present work is to increase the effectiveness of detection systems by decreasing the volume of traffic to be analysed, without affecting the accuracy of the detection process. To achieve this goal, a novel traffic reduction method is proposed in a Bot host detection framework which selects only TCP control packets.

In this work, the filtration of TCP control traffic packets is used in order to reduce the volume of network traffic as well as to increase the performance of the proposed approach. The filtering phase splits the operation into two steps: filtering all traffic related to the TCP protocol; then extracting the TCP control packet SYN, ACK, FIN and RST. Algorithm 3.1 shows the process of redacting network traffic. In Line 2 an array of TCP_Control_Packets_list is initialized. By iterating over the packets, new packets are added to the array of the (TCP_Control_Packets_List) from Line 3 to 15 until the last packet in the file is reached. Line 4 examines the TCP packet headers and Line 5 selects packets with no payload data. Line 6 then gets the packet header. From Lines 7 to 10, the code reads the packet, which is TCP, and extracts the packets which have SYN, ACK, FIN and RST flags.

The framework can efficiently decrease the volume of traffic that will enter the detection system. Network traffic reduction is achieved by generating a representative traffic of the entire network. The characteristic of this representative traffic has to reflect the behaviour of network traffic as a whole. Using the proposed traffic reduction approach decreases the quantity of network traffic to be examined. In summary, the network traffic reduction Algorithm 3.1 includes six rules to pick the desired packets:

- Rule 1 (R1): Packet contents Syn flag.
- Rule 2 (R2): Packet contents Syn-Ack flag.
- Rule 3 (R3): Packet contents Ack flag.
- Rule 4 (R4): Packet contents Fin-Ack flag.
- Rule 5 (R5): Packet contents Rest-Ack flag.
- Rule 6 (R6): Packet contents Rest flag.

Algorithm 3.1 Network Traffic Reduction.

```
1: Procedure Traffic Reduction (packets)
2: ArrayList <Packet> TCP_Control_Packets_List ;
3: For i=1 to size(Packets)
4:     IF Packets(i) has (TCP header)  then
5:         IF Packets (i) has (TCP. payloadSize==0) then
6:             pktheader= packet.getHeader(Packets(i));
7:                 IF ((pktheader.flags.syn=1OR pktheader.flags.ack=1 OR
                       pktheader.flags.rest=1 OR pktheader.flags.fin=1)
                       AND NOT (pktheader.flags.cwr=1 OR
                       pktheader.flags.ecn=1 OR pktheader.flags.push=1
                       OR pktheader.flags.urg=1))
8:                     TCP_Control_Packets_List.Add(packets(i));
9:                 ELSE
10:                    Discard the Packet;
11:                End IF
12:        End IF
13:  End IF
14: End For
15: Return TCP_Control_Packets_list;
16: End Procedure
```

3.5 Traffic Reduction Evaluation

3.5.1 Description of Experimental Datasets

In this research, three main datasets, which contain malicious and non-malicious traffic, were used in evaluating the proposed system. The first is the Information Security And Object Technology (ISOT) dataset (Saad, 2011) that contains malicious traffic from the French Chapter of the Honeynet Project involving the Waledac and Storm Bots. It also contains non-malicious traffic collected from the Traffic Lab at Ericsson Research in Hungary and the Lawrence Berkeley National Laboratory (LBNL). Whereas the second dataset is the Information Security Centre of Excellence (ISCX) dataset (Shiravi, Shiravi, Tavallaee, & Ghorbani, 2012) which includes legitimate activity traffic. The third dataset contains four legitimate volumes of traffic from P2P applications, namely Vuze, Frostwire, eMule and uTorrent, and the traffic of three P2P Botnets, namely Zeus, Storm and Waledac. These volumes were acquired from the University of Georgia, UAS (Babak et al., 2014), whose authors generated the P2P application traffic using AutoIt scripts in order to simulate human- activity on P2P hosts, as shown in Table 3.1

Table 3.1 Dataset distribution.

Traffic Source	**Purpose**	**Duration**
Storm Bot traffic - ISOT dataset (D1)	Train	3115 seconds
Waledac Bot traffic - ISOT dataset (D2)	Train	605 seconds
Normal traffic - ISOT dataset (D3)	Train	126273 seconds
eMule - University of Georgia dataset (D4)	Train/Test	24 hours
uTorrent - University of Georgia dataset (D5)	Train/Test	24 hours
Vuze - University of Georgia dataset (D6)	Train/Test	24 hours
FrostWire - University of Georgia dataset (D7)	Train/Test	24 hours
Normal traffic – ISCX dataset (D8)	Testing	24 hours
Storm Bot traffic - University of Georgia (D9)	Testing (Zero-day attack)	24 hours
Waledac Bot traffic - University of Georgia (D10)	Testing (Zero-day attack)	24 hours
Zeus Bot traffic - University of Georgia (D11)	Testing (Zero-day attack)	24 hours

The total size of the dataset used in our experiment about 216 GB, which is distributed in 10.09 GB from ISOT dataset, 16.1 GB from ISCX dataset and 189 GB from Georgia university dataset.

3.5.2 Traffic Reduction Approach Evaluation

The goal of the traffic reduction technique is to reduce the size of the captured traffic by using TCP control packets to represent the whole of network traffic. To evaluate this goal, a set of an experiment on dataset was performed, and show that the reduction traffic algorithm reduces the traffic within average 50% of network traffic as shown in Table 3.2.

Table 3.2 Traffic reduction rates.

Traffic Source	**Number of packets**	**Number of control packets**	**Rate of traffic reduction**
D1	128241	64551	50.3%
D2	118379	69936	59.1%
D3	564999	226308	40.1%
D4	6736353	2780725	41.3%
D5	6278385	4237135	67.5%
D6	11732688	741677	6.3%
D7	4429535	2406066	54.3%
D8	3776931	1686962	44.7%
D9	4251875	1169900	27.5%
D10	12915757	9395310	72.7%
D11	114548	59255	51.7%

Table 3.3 shows the rate of control packets that obtained by each traffic reduction rule. Meanwhile, Figure 3.3 compares the efficacy of the proposed traffic reduction rules between legitimate traffic and Botnet traffic. As shown in Figure 3.3 the rule (R1) obtained the highest reduction rates for both Bot and legitimate traffics rates around 30.7% and 24.1% respectively. Furthermore, the rule (R6) has the highest percentage of traffic about 19.6% on Botnet dataset evaluation comparing with the legitimate dataset traffic about 7.9% rate.

Table 3.3 Network traffic reduction rules rates

Traffic Source	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
D1	34.7%	7.0%	13.0%	8.7%	16.7%	20.0%
D2	30.0%	11.2%	8.8%	6.7%	21.3%	22.0%
D3	22.5%	25.8%	20.5%	17.8%	8.3%	5.0%
D4	20.0%	26.7%	15.8%	17.5%	13.3%	6.7%
D5	26.7%	22.2%	22.8%	10.7%	7.7%	10.0%
D6	25.0%	21.3%	18.3%	18.2%	5.2%	12.0%
D7	26.7%	19.3%	24.0%	15.0%	8.5%	6.5%
D8	23.8%	20.0%	19.0%	22.7%	6.7%	7.8%
D9	29.8%	8.2%	8.5%	8.8%	22.8%	21.8%
D10	29.0%	15.0%	6.0%	8.0%	17.0%	25.0%
D11	30.2%	7.7%	4.0%	9.8%	20.5%	27.8%

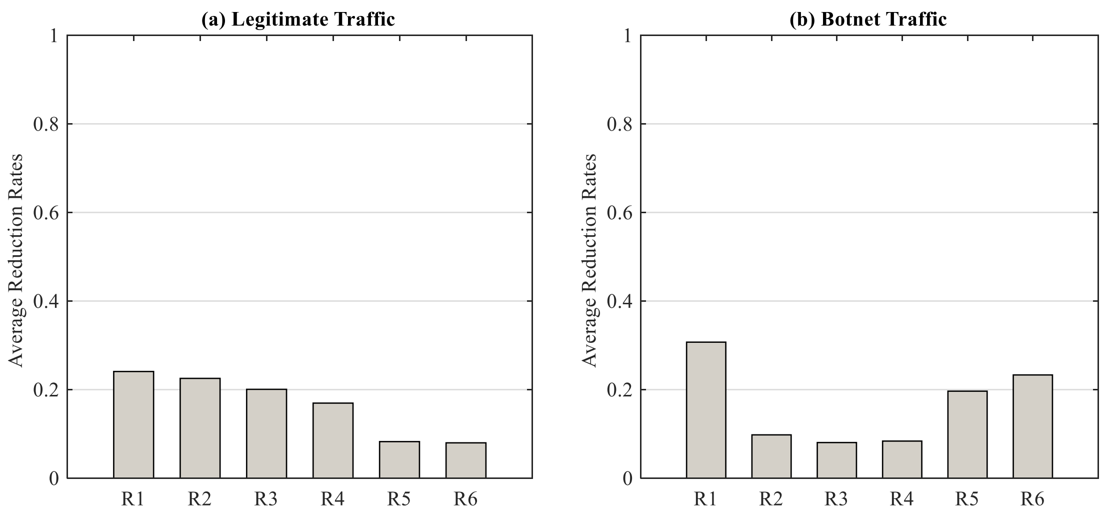

Figure 3.3 Average rates of network traffic reduction rule :(a) Legitimate traffic and (b) Botnet traffic.

3.6 Discussion

Recent years have witnessed of many Botnet detection techniques due to the severity of Botnet threats. Botnet detection has attracted intense research effort. For example, in flow-based detection schemes such as (Babak et al., 2014; Gu, Perdisci, et al., 2008; Timothy et al., 2008; Zhao et al., 2013), every packet is analysed one by one. Thus, that situation is not suitable for the online detection in a high-speed network.

In addition, several detection systems have been suggested (Goebel & Holz, 2007; Gu, Perdisci, et al., 2008; Gu et al., 2007; Gu, Zhang, et al., 2008; Yen & Reiter, 2008), but despite excellent detection results, these methods may suffer from restrictions of the scalability when they analyse huge volumes of network traffic. Their shortcomings in scalability mainly derive from their reliance on DPI techniques. For example, BotHunter (Gu et al., 2007) utilizes a payload-based anomaly detector and a signature-based detection engine, BotSniffer (Gu, Zhang, et al., 2008) and Rishi (Goebel & Holz, 2007) require the parsing of the IRC communications contents and TAMD (Yen & Reiter, 2008) inspects the payloads of packets in order to estimate similarities in contents. Therefore, the proposed a network traffic reduction mechanism gives the detection approach the ability to work online at the network level despite the size network traffic, because we only focus on a small part of the TCP packets, which are used to initialize the connections.

The bottleneck of the machine learning approaches for Bot detection relates to the dimensionality and size of the dataset considered, because the amount of the packets, needing to be scanned is enormous. Therefore, this study proposed a network traffic reduction approach to reduce the amount of network traffic to be analysed. The approach reduces the packets to around 50%, which will lead to enhancing the features extraction stage (Chapter 4), and online RL stage (Chapter 5).

Table 3.4 compares the performance of several detection approaches based on the use of traffic reduction. It is noticed that the proposed approach has a high detection rate with a relatively low FPR as shown in Section 4.5.5 in Chapter 4. Moreover, this approach proves the possibility of finding the Botnet activities without analysing all the network flow. Based on the above discussion, the proposed reduction method presented in this study showed the rate of traffic reduction is high comparing to the other methods.

The bottleneck of the machine learning approaches for Bot detection relates to the dimensionality and size of the dataset considered, because the amount of the packets,

needing to be scanned is enormous. Therefore, this study proposed a network traffic reduction approach to reduce the amount of network traffic to be analysed. The approach reduces the packets to around 50%, which will lead to enhancing the features extraction stage (Chapter 4), and online RL stage (Chapter 5).

Table 3.4 Comparison of traffic reduction with other Bot detection techniques.

Approach	**Rate of traffic reduction**	**True positive rate**	**False positive rate**
Babak et al. (2014)	0%	99.09%	0.1%
Zhao et al. (2013)	0%	98.1%	2.1%
Timothy et al. (2008)	N/A	92%	11-15%
Gu et al. (2008)	N/A	100%	0-6%
K. Wang et al. (2011)	> 70%	95%	0-3%
The proposed approach	**40% -70%**	**99.1%**	**0.01%**

3.7 Summary

In this chapter, we have introduced a solution to this problem, which includes a network packet reduction algorithm. The evaluation results of the proposed reduction approach show that our solution achieves suitable reduction rate using real-world network traces. The next chapter will present the proposed connection-level features and the offline Bot detection method.

4 CONNECTION-LEVEL FEATURES FOR BOT HOST DETECTION

4.1 Introduction

It is a concern of a network administrator to recognize hosts in the network that are infected by a Botnet. Infected computers can be used by adversaries to extract valuable information or they may wait for an order from a Botmaster to reroute spam. However, the detection of Botnets is currently a special challenge for the following reasons. Botnets employ hidden tactics such as using a random port rather than the usual port or utilizing an encryption technique to hide malicious behaviour on network traffic (D. Dittrich & S. Dietrich, 2008; Holz et al., 2008). Besides that, Botnets utilize regular protocols such as IRC, HTTP, and P2P. Therefore, it is very difficult to differentiate Botnet behaviour from legitimate network traffic (Grizzard et al., 2007; Jiang & Shao, 2012). Therefore, to discriminate the Botnet behaviour from legitimate network traffic, the feature extraction and selection set is a critical point for the efficiency of any Botnet detection system.

The quality of the feature set is one of the most important factors that affect the performance of the machine learning algorithms. Therefore, the main goal of this chapter is to introduce the host traffic feature based on the connection-level that can differentiate between a Bot and a legitimate network host. More specifically, the feature able to identify Bot host by using a minimum set of packets after network traffic reduction stage (Chapter 3). In addition, to achieve earlier Bot detection and bypass the encrypted network traffic, the feature set utilizes the information in the header of TCP control packets that helps the detection system to efficiently analysis massive volume of network traffic without suffering from encrypted traffic.

The previous chapter introduced the traffic reduction approach that helps to reduce the amount of the network traffic to be analysed. In this chapter, we will utilise the traffic reduction to present our contribution on feature extraction at a connection-level. Furthermore, the chapter presents the evaluation of the offline Bot detection system.

4.2 Overview of Offline Bot Detection Approach

The architecture of the proposed offline Bot detection system consists of four main components: network traffic reduction, network traffic feature extraction and malicious activity detection as illustrated in Figure 4.1. The phase of the reduction network traffic is responsible for filtering that traffic to increase the scalability and avoiding DPI problem as we addressed in the previous chapter.

In the feature extraction phase, there are two processing stages: extracting connection conversations and host feature extraction. The extracting connection conversations phase is subject to constructing connection features between nodes. In the node feature extraction phase, the vectors of node features are assembled that represent the status of the node based on connection features.

Finally, the malicious activity detector takes a node feature vector as input and classifies nodes according to their features into two categories: malicious nodes infected with a Bot or normal legitimate nodes. Therefore, a multi-layer feedforward neural network with resilient learning is selected as a malicious activity detector due to its high adaptive and the accuracy rates.

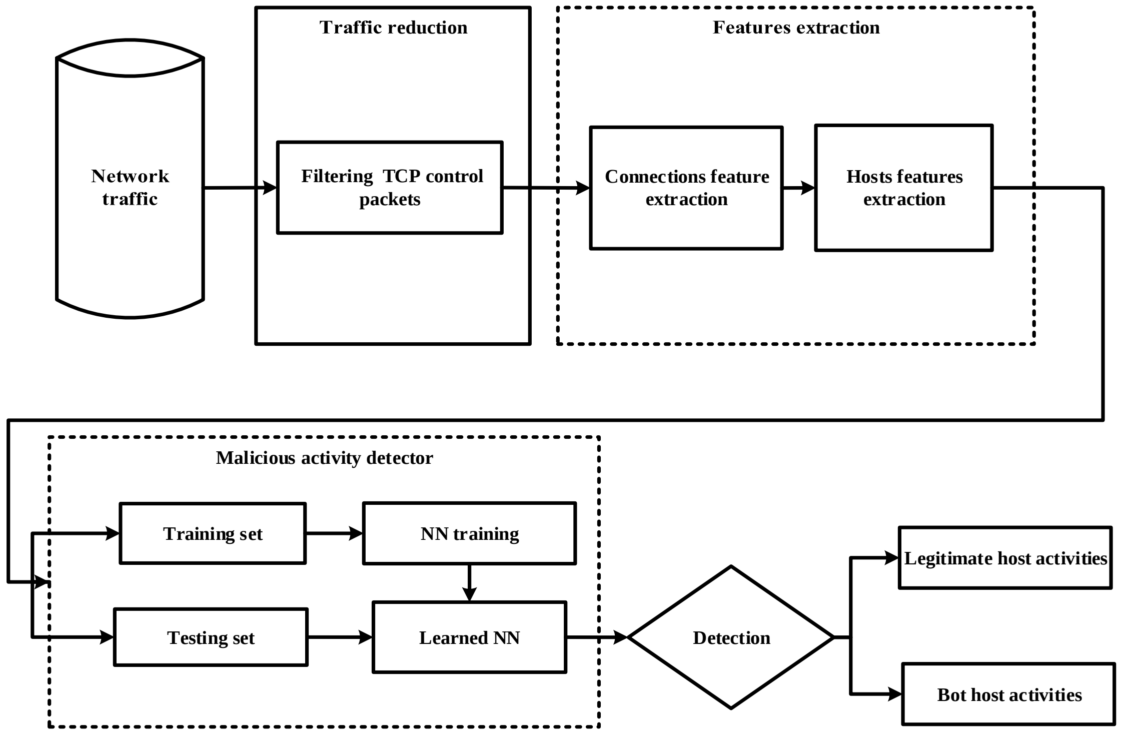
Traffic reduction
Features extraction
Network traffic
Filtering TCP control packets
Connections feature extraction
Hosts features extraction
Malicious activity detector
Training set
NN training
Testing set
Learned NN
Detection
Legitimate host activities
Bot host activities

Figure 4.1 Overview of Offline Bot detection phases.

4.3 Features Extraction

The proposed connection features relies on two fundamental concepts. Firstly, it passively monitors network traffic (Zeidanloo, Shooshtari, et al., 2010). Secondly, it utilizes the fact that Bots during the propagation phase will frequently communicate with their C&C servers/peers in order to discover other peers and receive the latest updates of tasks due to their pre-programmed nature (Han, Lim, & Im, 2009; Sang-Kyun, Joo-Hyung, Jae-Seo, Bong-Nam, & Hyun-Cheol, 2009). Bots are different from other types of malware; they work as a group and primarily need a communication channel in order to coordinate malicious activities. These connections are described as the way by which the Botmaster communicates with his Bots (Chao et al., 2009).

Features are composed of a small set of attributes that are needed to characterize a dataset. In particular, a vector of an attribute that represents each instance of data is used as input to a machine learning algorithm. The quality of features is significant in order for the machine learning algorithm to detect the behaviour of the network traffic class. The network traffic features could be extracted at three levels as follows: packet-level, flow-level, and connection-level (Roughan, Sen, Spatscheck, & Duffield, 2004). Moreover, classification can be based on the level of packet inspection, as in shallow or DPI.

The features in data packets are extracted from the packet payload or header. For example, features such as packet length, average packet length, the variance of packet length, TCP flag and the packet direction are simple to calculate and can be gathered directly from the packet. It is very beneficial for the detection approach to using packet-level data to represent the states of the network.

Flow-level features introduce a statistic summary of the network flows. Flow is described as packets with the same 5–tuple (source IP, destination IP, source port, destination port and protocol type). For instance, flow-level features are the cumulative number of packets in a flow, the duration of the flow and the average packet size. This statistic can be collected at the edge of the network using tools such as Cisco NetFlow (Cisco, 2012).

At the flow-level, the difference between the outbound and inbound traffic can be recognized. The main advantage of flow-features is that they do not require the extra resources and exhaustive processes of packet-level features. However, a shortcoming is that the flow may sometimes accumulate packets that relate to several applications in a single flow, which would distort the features of the flow. On the other hand, connection-level features are required when it is necessary to track some behaviour which correlates with a connection-oriented protocol such as TCP. The contrast between connection and flow is that the start and termination of a TCP connection are well-defined handshakes between a client and a server. The connection-level data affords more high-quality information than flow-level features (Roughan et al., 2004) but requires further overhead to follow the state of the connection.

Packet inspection approaches that have been used in practice in networking environments can be divided into two categories. These are shallow and DPI. Techniques that use DPI approaches are designed to permit network administrators to recognize specifically the header and the payload content of each packet of data that crosses over the network. According to AbuHmed et al. when applying DPI approaches on the network traffic there exists various challenges, such as the complexity of the search algorithm, a growing number of intruder signatures, signatures overlapping, unknown signature locations and encrypted packet payload contents (AbuHmed, Mohaisen, & Nyang, 2007). On the other hand, techniques that apply shallow packet inspection to read information from the network and transport layers of the OSI model. Thus, they cannot examine the session, presentation and application layers of a packet (Petersen, 2014).

In the proposed framework, a combination of connection and packet levels are used. The features are captured at both packet-level and connection-level. For example, the feature of inter-arrival time between packets in each connection requires data to be gathered at packet-level and then aggregated into connections to collect statistical information about connection states. Thus, the proposed Bot detection system can capture the local characteristics at packet-level and collect connection-level characteristics as the global features. Based on this approach, the features will have the benefits of both packet and connection levels. Thereby, in contrast to other relevant work, the proposed system can potentially be applied to any encrypted application since it applies a shallow packet inspection approach in order to extract connection features without using IP addresses, port numbers and payload content.

The features used in the proposed framework are extracted in two phases. Firstly, the connection feature are extracted. Secondly, the connection features as host features are aggregated to represent the state of the host during the sliding time-window. Therefore, the final detection decision is based on the host features extracted according to the headers of control packets (packet-level) and connection statistical features (connection-level). Connection and host levels features are discussed in more detail in the next sections.

4.3.1 Connection-level Features

In this phase, features that are important in detecting the P2P Botnet's malicious behaviour are extracted. Forty-three features are collected based on the size of the sliding window. These features are extracted based on the definition of a connection as a group of control packets exchanged between two different hosts, which are identified by the 5-tuple (source IP address, destination IP address, source port and destination port, protocol). In proposed method, all features are extracted directly from the control packet header, which is different from previous approaches that use a deep inspection of the payload (Dan, Yichao, Yue, & Zongwen, 2010; Lu et al., 2011; Perdisci, Guofei, & Wenke, 2006; Xiaomei, Fei, Xiaohua, & Xiaocong, 2010). Consequently, performance is increased while the use of system resources such as memory and computation in the processor is reduced. Table 4.1 shows 43 features extracted in the proposed connections-based P2P Botnet detection approach. These features are generated from a sliding time-window and are composed of a feature vector to represent the connection status through the duration of the sliding window.

Table 4.1 Extracted features of network traffic connections.

Features	**Description**
$\hat{F}1$	Number of control packets per connection in a given time interval.
$\hat{F}2$	Number of control packets transmitted per connection in a given time interval.
$\hat{F}3$	Number of control packets received per connection in a given time interval.
$\hat{F}4$	Number of transmitted bytes per connection in a given time interval.
$\hat{F}5$	Number of received bytes per connection in a given time interval.
$\hat{F}6$	Number of transmitted SYN packets per connection in a given time interval.
$\hat{F}7$	Number of received SYN packets per connection in a given time interval.
$\hat{F}8$	Number of transmitted ACK packets in a sequence of one per connection in a given time interval.
$\hat{F}9$	Number of received ACK packets in a sequence of one per connection in a given time interval.
$\hat{F}10$	Number of transmitted duplicate ACK packets per connection in a given time interval.
$\hat{F}11$	Number of received duplicate ACK packets per connection in a given time interval.
$\hat{F}12$	Average length of transmitted control packets per connection in a given time interval.
$\hat{F}13$	Average length of received control packets per connection in a given time interval.
$\hat{F}14$	Average length of control packets per connection in a given time interval.
$\hat{F}15$	Number of transmitted failed connection per connections in a given time interval.
$\hat{F}16$	Number of received failed connection per connections in a given time interval.
$\hat{F}17$	Number of transmitted SYN-ACK packets per connection in a given time interval.
$\hat{F}18$	Number of received SYN-ACK packets per connection in a given time interval.
$\hat{F}19$	Number of transmitted SYN-ACK packets in a sequence of one per connection in a given time interval.
$\hat{F}20$	Number of received SYN-ACK packets in a sequence of one per connection in a given time interval.

$\hat{F}21$	Total number of bytes per connection in a given time interval.
$\hat{F}22$	Ratio of incoming control packets per connection in a given time interval.
$\hat{F}23$	Ratio of average length of outgoing control packets over the average length of control packets per connection in a given time interval.
$\hat{F}24$	Ratio of the difference between the number of transmitted SYN packets and the number of received ACK packets in a sequence of one over the number of transmitted SYN packets.
$\hat{F}25$	Difference between the number of transmitted SYN packets and the number of received SYN-ACK packets per connection in a given time interval.
$\hat{F}26$	Number of transmitted FIN-ACK packets per connection in a given time interval.
F27	Number of received FIN-ACK packets per connection in a given time interval.
$\hat{F}28$	Number of transmitted RST-ACK packets per connection in a given time interval.
$\hat{F}29$	Number of received RST-ACK packets per connection in a given time interval.
$\hat{F}30$	Average time between attempts to create connections in a given time interval.
$\hat{F}31$	Number of received RST packets per connection in a given time interval.
$\hat{F}32$	Number of transmitted RST-ACK packets in a sequence one of per connection in a given time interval.
$\hat{F}33$	Number of transmitted RST packets in a sequence of one per connection in a given time interval.
$\hat{F}34$	Number of received RST-ACK packets in a sequence of one per connection in a given time interval.
$\hat{F}35$	Inter-arrival time of packets between SYN and ACK packets that generated by the host per connection in a given time interval.
$\hat{F}36$	Inter-arrival time of packets between SYN and RST packets that generated by the host per connection in a given time interval.
$\hat{F}37$	Inter-arrival time of packets between SYN and RST-ACK packets that generated by the host per connection in a given time interval.
$\hat{F}38$	Inter-arrival time of packets between SYN packet from host side and RST packet from another side per connection in a given time interval.
$\hat{F}39$	Inter-arrival time of packets between SYN packet from host side and RST-ACK packet from another side per connection in a given time interval.

$\hat{F}40$	Inter-arrival time of packets between FIN-ACK packet from host side and RST packet from another side per connection in a given time interval.
$\hat{F}41$	Inter-arrival time of packets between ACK packet from host side and RST packet from another side per connection in a given time interval.
$\hat{F}42$	Inter-arrival time of packets between SYN packet from host side and SYN-ACK packet from another side per connection in a given time interval.
$\hat{F}43$	Connection duration.

4.3.2 Connection Features Reduction

Feature reduction is a technique of reducing the number of attributes, with the purpose of eliminating those features from the learning algorithm that have only a small influence on the classification problem (Nguyen, Petrović, & Franke, 2010). Feature reduction is used to decrease the 'over-fitting' problem (Livadas et al., 2006) and is important in overcoming the imbalanced dataset problem (Van der Putten & Van Someren, 2004). Therefore, the quality of the feature reduction mechanism is one of the most important factors that affect the accuracy of the classification algorithm.

In this study, the aim of feature reduction is to choose a suitable subset of features which will improve neural network performance and decrease the complexity of a classification model without significantly decreasing accuracy rates. A classification and regression tree (CART) (Breiman et al., 1984) is employed as the feature reduction approach used to eliminate worthless features, with the aim of reducing the quantity of data needed to obtain better rates of neural network learning and classification accuracy.

The decision tree produced by the CART algorithm consists of two types of nodes: internal nodes with two children, and leaf nodes without children. Any internal node is associated with a decision function to indicate which node to visit next. To begin the construction of the tree, the training samples that contain a set of features and their class labels are required. Recursively the training dataset is partitioned into smaller subgroup during the construction of the tree. Depending on the confusion matrix of the classes distribution in the training set, all resulting node is assigned a predicted class.

The test at internal nodes is determined based on a measure of impurity to select which feature and threshold values are selected. The best-known measure of impurity for CART is entropy impurity, which is given by:

$$E(t) = -\sum_{j}^{C} p\left(\frac{j}{t}\right) log_2\, p\left(\frac{j}{t}\right) \tag{4.1}$$

where $E(t)$ is the entropy impurity at node t, $p\left(\frac{j}{t}\right)$ is the relative frequency of class j at node t, and C is the number of classes.

The best value of the split node (t) is chosen from a set of all values splitting(x), so that the maximum drop in impurity is a difference between the impurity at the root node and the impurity at the children nodes:

$$\Delta E(x,t) = E(t) - (P_l E(t_l) + P_r E(t_r)) \tag{4.2}$$

where $\Delta E(x,t)$ is the drop of impurity, $E(t_l)$ and $E(t_r)$ are the impurities of the left and right branch nodes, P_l and P_r are the percentage of objects go to the left (t_l) or right (t_r) child nodes.

Table 4.2 provides a ranking of the importance of features selected by the entropy algorithm using training dataset. The features $\hat{F}1$, $\hat{F}3$, $\hat{F}6$, $\hat{F}7$, $\hat{F}8$, $\hat{F}9$, $\hat{F}15$, $\hat{F}19$, $\hat{F}20$, $\hat{F}25$, $\hat{F}26$, $\hat{F}27$, $\hat{F}31$, $\hat{F}32$, $\hat{F}33$, $\hat{F}34$, $\hat{F}35$, $\hat{F}36$, $\hat{F}37$ and $\hat{F}43$ show the best discrimination of connection behaviour, whereas the features $\hat{F}4$, $\hat{F}5$, $\hat{F}10$, $\hat{F}11$, $\hat{F}12$, $\hat{F}13$, $\hat{F}14$, $\hat{F}16$, $\hat{F}17$, $\hat{F}18$, $\hat{F}21$, $\hat{F}22$, $\hat{F}23$, $\hat{F}24$, $\hat{F}28$, $\hat{F}29$, $\hat{F}30$, $\hat{F}38$, $\hat{F}39$, $\hat{F}40$, $\hat{F}41$ and $\hat{F}42$ do no discrimination between legitimate and malicious connections.

Feature selection is performed depending on the contribution of the input samples that made the creation of the decision tree. Feature importance is decided by the role of each input sample either as a main splitter or as a surrogate. Surrogate splitters are represented as backup rules that approximately simulate the action of the primary splitting rules. The features that give the best discrimination of connection behaviour it will be used to generate host features in the next step of features extraction.

Table 4.2 Features importance ranking by entropy algorithm.

Feature	Importance	Feature	Importance	Feature	Importance
$\hat{F}1$	0.8130	$\hat{F}33$	0.5319	$\hat{F}18$	0
$\hat{F}2$	0.8100	$\hat{F}34$	0.5092	$\hat{F}21$	0
$\hat{F}3$	0.7876	$\hat{F}35$	0.4493	$\hat{F}22$	0
$\hat{F}6$	0.7741	$\hat{F}36$	0.3712	$\hat{F}23$	0
$\hat{F}7$	0.7634	$\hat{F}37$	0.2870	$\hat{F}24$	0
$\hat{F}8$	0.7548	$\hat{F}43$	0.1944	$\hat{F}28$	0
$\hat{F}9$	0.7438	$\hat{F}4$	0.082941	$\hat{F}29$	0
$\hat{F}15$	0.7181	$\hat{F}5$	0.069167	$\hat{F}30$	0
$\hat{F}19$	0.7031	$\hat{F}10$	0.012049	$\hat{F}38$	0
$\hat{F}20$	0.6604	$\hat{F}11$	0.01191	$\hat{F}39$	0
$\hat{F}25$	0.6198	$\hat{F}12$	0.01153	$\hat{F}40$	0
$\hat{F}26$	0.6010	$\hat{F}13$	0.000515	$\hat{F}41$	0
$\hat{F}27$	0.5734	$\hat{F}14$	3.81E-06	$\hat{F}42$	0
$\hat{F}31$	0.5670	$\hat{F}16$	6.12E-09		
$\hat{F}32$	0.5512	$\hat{F}17$	0		

4.3.3 Host Feature Extraction

Table 4.3 shows the 16 host features created in the proposed approach. However, our P2P Bot detection framework is based on the following three observations. Firstly, the Bot hosts share certain malicious characteristics in their network behaviours that are distinct from those of normal hosts (Yen, 2011). Secondly, the behaviour of Bot in the propagation phase repeats itself frequently whenever it infects the hosts during the propagation stage (Felix et al., 2012; Han et al., 2009; Sang-Kyun et al., 2009). Thirdly, the Bot connections are generated by a software program (Scanlon & Kechadi, 2012).

The feature extraction phase can start immediately if packets are transferred between hosts. In order to extract the properties of a node more accurately, the collection of adequate network traffic is required before the feature extraction operation starts. Therefore, in the proposed approach the behaviour of hosts is observed by analysing their traffic packets within the time of the sliding window in order to gain adequate packets. As a result of the feature extraction stage, each host is represented by its individual feature vector. This host feature vector set is then utilized to differentiate between malicious Botnet traffic and legitimate network traffic by employing online machine learning methods with reinforcement techniques.

Table 4.3 Host features of network traffic.

Feature	**Description**
F1	Total number of transmitting connection per host in a given time interval
F2	Total number of transmitting unique connections per host in a given time interval.
F3	Total number of connection tries per host in a given time interval.
F4	Rate of high severity destination port numbers in a given time interval.
F5	Rate of using unique destination ports per host in a given time interval.
F6	Rate of using unique source ports per host in a given time interval.
F7	Rate of transmitting unique connections per host in a given time interval.
F8	Rate of high severity source port numbers in a given time interval.
F9	Rate of failures in connection per host in a given time interval.
F10	Entropy rate of total control packets in a connection per host in a given time interval.
F11	Entropy rate of transmitting control packets in a connection per host in a given time interval.
F12	Entropy rate of receiving control packets in a connection per host in a given time interval.
F13	Average time between connections per host.
F14	Average client Inter-arrival time between control packets.
F15	Average connection duration.
F16	Index of dispersion.

Port scanning is one of the most famous malicious activities. Port scanning is used by Bots in many aspects of the Botnet life cycle, such as propagation and attack behaviours. For example, in the propagation phase, a Bot tries to discover and contact other Bots in the same network in order to receive copies of the latest updates. Therefore, monitoring and analysing the rate of newly established connections is important in measuring and detecting malicious Bots behaviour. Port scanning may be divided into three classes: vertical scanning, horizontal scanning, and block scanning (Staniford, Hoagland, & McAlerney, 2002). A horizontal distribution scan is achieved by specific port access for many numbers of destination IP addresses. On the other hand, a vertical scan is performed on a specific destination IP address over a range of ports. Finally, a block scan is a mixture of horizontal and vertical scanning for different ports and destination IPs. The diversity of port number and destination IPs often indicates the capability of a Bot to exploit the victim host. In addition, computer ports are divided into two categories: high-severity and low-severity ports. According to the Dshield organization (Dshield.org, 2013) high-severity ports contain those most likely to be scanned; all other ports are marked as low-severity ports. Thereby, this research utilizes the port scanning for malicious activities, and so, features F1-F8 represent scanning behaviour.

Based on our understanding and observation of Botnet traffic behaviour, it is natural for Bots to produce network connection failures. When a Bot joins a Botnet, it needs to find an entrance point, which could be either a C&C server or a peer host, to notify its current situation and receive new instructions. Consequently, any peers attempting to establish a connection with these hosts could lead to failures in connection. A failed connection feature F9 based on the TCP is represented as failed if the 3-step handshake is incomplete (Limmer & Dressler, 2009).

The control packet count of legitimate Internet traffic shows more diversity than that of Bot connection traffic. Computer-human users can use many applications, which each one has a different behaviour for the number of control packets in concoctions. Therefore, we do not expect to notice any trend in the control packet frequency. On the other hand, in the propagation phase Bots try to contact other peers on the Botnet network to inquire for an update. Thus, they repeat the same connection behaviour, and this shows a trend in the style of connection. Therefore, an entropy algorithm (Cover & Thomas, 2012) is used in this study to measure the amount of entropy or randomness in control packet variation per host, and an entropy algorithm is utilized by modelling the number of control packets in connections per host as a discrete symbol. A legitimate connection is expected to have high entropy while a Botnet connection is expected to have low entropy. The entropy of control packet frequency per host is calculated from a set $C_h = [c_1, c_2, \ldots. c_n]$, where each c_i denotes the number of control packets per host connections. This is estimated as

$$E(t) = -\sum_{i}^{n} c_i \; log_2 \, c_i \tag{4.3}$$

However, Jian et al. (2009) introduced approaches to detecting Storm Bot by utilizing entropy theory (Jian & Jun-Yao, 2009). A significant difference between their method and the present strategy is that Jian et al. applied the entropy theory over the packet payload content with DPI, which is not suitable for the detection of Botnets that use encryption techniques.

Features F13-F15 are related to the client's inter-arrival control packets. The inter-arrival packet time is the required time for the application to create and transfer data to the transport layer (Jaber, Cascella, & Barakat, 2011), and is estimated by extracting the time between any two consecutive packets in the same connection. We assume that additional time added because of the changes in network conditions is negligible. The proposed framework focuses on host features based on the network level and the target is to detect an infected machine, so it focuses on the time between of packets outgoing from the host.

Finally, for feature F16, the index of dispersion for counts (IDC) is adopted to discriminate arrival processes consisting of packets sent by the host on the network. R. Gusella highlighted the importance of using the indices of dispersion in identifying packet variability (Gusella, 1991), where the index of dispersion is a measure used to quantify whether a set of observed events is clustered or scattered in correlation with a standard statistical model. The IDC is represented as the ratio of the variance to the mean. The following equation gives the definition of the IDC:

$$IDC = \frac{\sigma^2}{\mu} \tag{4.4}$$

where μ, σ^2 denote the mean and variance respectively.

4.4 The Malicious Activity Detector

The operation of the malicious activity detector consists of three stages: an off-line stage (training), the online detection stage, and a reinforcement learning stage. In the training phase, the classifier is provided with a set of labelled Bot feature vectors and legitimate feature vectors for the training mission. Once the training stage is finished, the detection phase starts by entering the extracted features to the classifier in order to classify the activities of the hosts inside the network as malicious or legitimate.

A neural network is used as a malicious activity detector because it has robust capabilities for nonlinear system identification and control due to an inherent ability to approximate an arbitrary nonlinear problem (Nigrin, 1994; Razi & Athappilly, 2005; Tsai, Hsu, Lin, & Lin, 2009). The neural network is trained with a resilient back-propagation learning algorithm, where the use of this algorithm is to minimize the damaging effects of the volumes of fractional derivatives. The sign of the derivative is only used to locate the trend of the weight update, whereas the volume of the derivative has no negative role overweight update. The size of the weight change is solely determined by the following formula (M. Riedmiller & Braun, 1993):

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)}\,, & if\ \dfrac{\partial E(t)}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)}\,, & if\ \dfrac{\partial E(t)}{\partial w_{ij}} < 0 \\ 0, & else \end{cases} \tag{4.5}$$

where $\Delta w_{ij}^{(t)}$ is the change in weight between the input layer and hidden layers in the current iteration (t), and $\frac{\partial \mathrm{E(t)}}{\partial \mathrm{w_{ij}}}$ denotes the partial derivative with respect to each weight. Once the weights are calculated, the new updated weight value is determined. This is accomplished with the following formula:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+}.\Delta_{ij}(t)\,, & if\ \dfrac{\partial E(t-1)}{\partial w_{ij}}.\dfrac{\partial E(t)}{\partial w_{ij}} > 0 \\ \eta^{-}.\Delta_{ij}(t)\,, & if\ \dfrac{\partial E(t-1)}{\partial w_{ij}}.\dfrac{\partial E(t)}{\partial w_{ij}} < 0 \\ \Delta_{ij}(t-1)\,, & else \end{cases} \tag{4.6}$$

where $\Delta_{ij}^{(t)}$ denotes the updated value for the current iteration t, and η^{+} is the positive step value which is typically 1.2 and η^{-} is the negative step value which is typically 0.5

(M. Riedmiller & Braun, 1993). The neural network classifier proposed in this study contains 16 input and two output parameters. To avoid over fitting by using too many hidden layers, the method proposed in a previous study (Boger & Guterman, 1997) is used to determine the number of neurons in the hidden layers.

4.5 Experimental Results and Evaluation

4.5.1 Experimental Tools

To estimate the performance of the proposed solution a series of experiments on the dataset were carried out. These experiments are carried on an Intel Xeon processor with a six-core monster clocked at 2.1GHz (with a 2.6GHz Turbo) and 64 GB RAM. Besides, the proposed approach implemented using Matlab 2014b. Table 4.4 shows the software tools and libraries used in the experiments.

Table 4.4 Experimental Tools.

Name	**Description**	**Version**
Wireshark (Wireshark, 2015).	Network protocol analyser.	1.12.4
Jpcap (Shen & Wang, 2009).	Java library for capturing and sending network packets.	0.7
Tcpreplay (TcpReplay, 2014).	Replays Pcap files onto the network	3.4.4

4.5.2 Experimental Procedure

To evaluate the proposed offline Bot detection approach, an experimental dataset

Table 3.1 was obtained to evaluate its capability on Bot detection. Additionally, for the purpose of simulating a real network traffic situation, a testbed was configured in order to replay malicious Botnet traffic, normal daily activity traffic and P2P application traffic using the TcpReplay tool (TcpReplay, 2014). The reply network traffics runs on the same network interface card for the purpose of homogenizing the network traffic behaviour presented by all datasets. TcpReplay is utilized to replay the traffic from the traffic files. The replayed network traffic is then captured by the JPCAP tool (Shen & Wang, 2009).

The experimental procedure was scheduled into five steps, listed as following:

1. Replay entire malicious trace and legitimate trace file, and capture packets using various time-window sizes.
2. Reduce network traffic using the proposed network traffic reduction technique.
3. Extract feature vectors to generate host feature set.
4. Get the classification results by using the prepared training and testing sets using the offline proposed Bot detection approach.
5. Identify the time window size that achieves a better detection performance and better stability in the offline and online stages.

The reason behind dividing the network traffic in time-windows is to analyse the massive traffic volume of packets. Moreover, time-window is required to submit a result to the network administrator in a timely fashion. The idea behind of not using time-window smaller than 10s is that the number of captured packets is too small to show the traffic behaviour characteristics. In the other hand, the reason for not using time-window larger than the 60s is that it cannot satisfy earlier of detection when using a large time-window size. In addition, Bots tend to generate a temporal behaviour following the infection phase (Hegna, 2010), and so this behaviour helps to capture the necessary Bot behaviours in the time-window. Therefore, in this research, we start with 10-seconds time-window and gradually increase the size of time-window in order to reach the acceptable performance rate. On the other hand, 10% of the time-window size is utilized as sliding interval between time windows to achieve rapid detection of any malicious activities instead of idling for the next entire time window to ending and then the network traffic to be collected.

4.5.3 Evaluation Metrics

The present assessment used P2P Bot host instances as positive instances and legitimate host instances as negative instances. Moreover, various metrics were used to evaluate the result of the experiments, namely detection rate (DR), FPR, precision, F-measure, accuracy (ACC), Root Mean Square Error (RMSE), Non-Dimensional Error Index (NDEI), Matthews Correlation Coefficient (MCC) and area under the ROC (AUC).

Despite accuracy metrics have been adopted to measure performance in some studies (Saad, 2011; Wen-Hwa & Chia-Ching, 2010; Zhao et al., 2013), its use could be doubtful.

With the use of imbalanced datasets, where the numbers of instances relating to each class are significantly diverse, using accuracy as a measure of classification performance can be inaccurate. For instance, if a classifier is applied to a dataset that contains 95% legitimate activity and 5% malicious activity and the classification model predicts that all activity is legitimate then an accuracy rate of 95% is obtained. Nevertheless, this result does not mean a successful classification, since no malicious activity was detected. Consequently, imbalance measurements such as MCC and area under the ROC were applied in the evaluation in order to comprehensively assess of the proposed approach in situations of imbalanced datasets. The evaluation metrics were calculated using Equations 4.7 to 4.14.

- True positive (TP): represents the number of Bot instances accurately identified as malicious activities.
- True negative (TN): indicates the number of normal instances accurately identified as legitimate activities.
- False positive (FP): shows the number of normal instances identified as malicious activities.
- False negative (FN): represents the number of Bot instances identified as legitimate activities.

The FPR shows the percentage of legitimate instances misclassified as Botnet instances:

$$FPR = \frac{FP}{(TN + FP)} \tag{4.7}$$

DR, also called recall, indicates the percentage of Botnet instances that were detected as Botnet instances.

$$DR = \frac{TP}{(TP + FN)} \tag{4.8}$$

ACC indicates the percentage of correct predictions of all instances.

$$ACC = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{4.9}$$

Precision indicates the percentage of instances correctly classified as positive instances.

$$Precision = \frac{TP}{(TP + FP)} \tag{4.10}$$

The F-measure is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score.

$$F - measure = \frac{(2 \times \ Precision \times \ Recall)}{(\ Precision + Recall)} \tag{4.11}$$

RMSE indicates the differences between the target value and the actual value estimated by the detection method.

$$RMSE \ = \sqrt{\sum_{i=1}^{N} \frac{(yi - ti)^2}{N}} \tag{4.12}$$

where N is the number of input samples, yi represents the actual output of the model, and ti is the targets of the samples. $RMSE \ = \ 0$ indicates that the output of the model exactly matches the targets. Root mean square Error ($RMSE$) is an important measure of variations between the values expected from a model or an estimator and the values actually observed.

NDEI is defined as the RMSE divided by the standard deviation of the target series, which is used to estimate the prediction quality (Espinosa & Vandewalle, 2000).

$$NDEI = \frac{RMSE}{std(ti)} \tag{4.13}$$

The MCC is used to evaluate the efficiency of the classifier in imbalanced classes (Matthews, 1975).

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{((TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.14}$$

The receiver operating characteristic (ROC) is a graphical plot that depicts a binary classifier's performance. ROC curves plot the TPR on the vertical axis against the FPR on the horizontal axis. The AUC denotes the classifier's performance (Swets, 2014). Moreover, the AUC is known to be a much more robust estimator of classifier performance (Fawcett, 2006).

To ensure the quality of the learned neural network agent, a cross-validation method is used to estimate the error rate of classifiers. In cross-validation, the dataset is partitioned randomly into N samples and evaluations are run for N iterations. At each iteration, $N -$

1 samples are selected for training and the final sample is used to evaluate the accuracy of the classifier.

4.5.4 Host Feature Set Evaluations

To assess the quality of the proposed detection approach and the selected of the hosts features (Table 4.3) towards a successful detection, the normalized average for each feature is estimated by using Min-Max Normalization (Al Shalabi & Shaaban, 2006).

$$X' = \frac{Xi - X\,min}{X\,max\ - X\,min} \tag{4.15}$$

Where X' is the normalized value of Xi. The $Xmin$ and $Xmax$ are the minimum and maximum values of features vector.

Figure 4.2 shows the average distribution of the normalization value for each feature. We found different distributions between Bot host traffic and normal host traffic. As shown in Figure 4.2 the features F5, F10, F12, F15 and F16 are a decrement features to help in Botnet detection. Whereby the features F14, F1 and F3 have a low differentiate between network traffic.

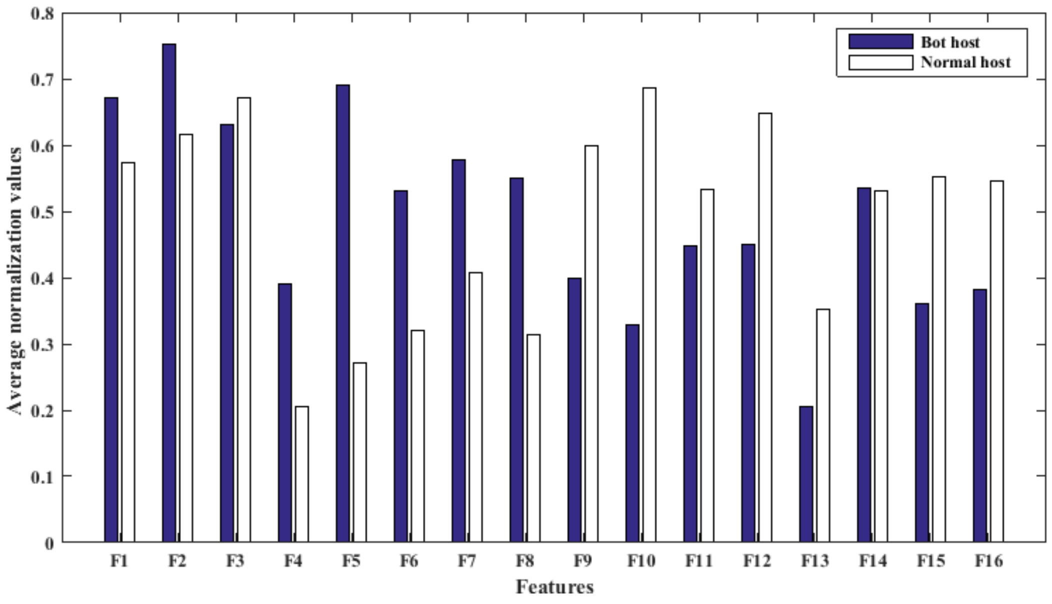

Figure 4.2 Normalized host features comparison.

Figure 4.3 shows the difference in entropy values for the total number of control packets between flows for normal and Bot traffic. The plots show the entropy values for normal host traffic is between 0 to 5 while it is under 0.5 for a Bot host traffic.

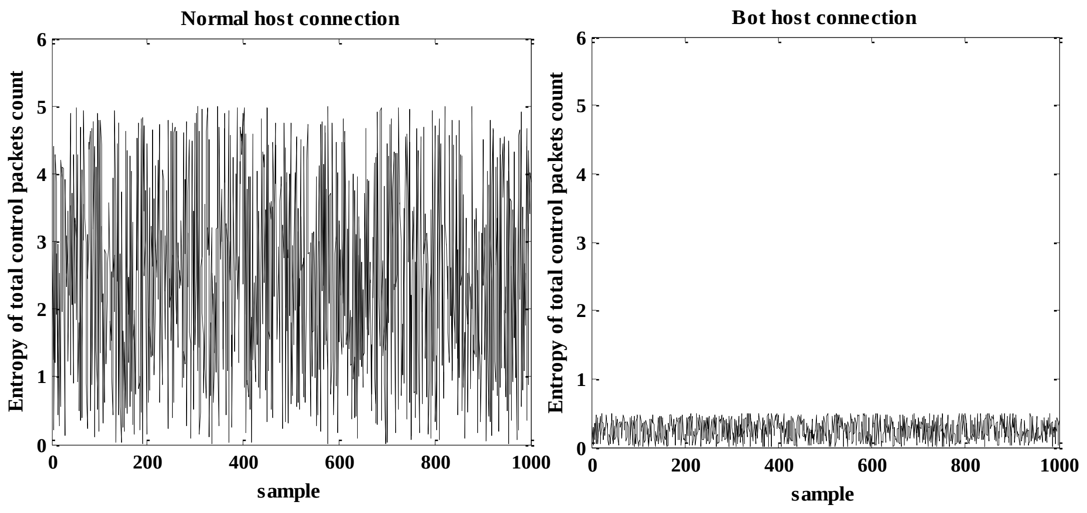

Figure 4.3 Entropy rates of total control packets per host.

Interestingly, there were also differences in the ratios of the entropy of transmitting and receiving numbers of control packets between Bot host traffic and normal host traffic as shown in Figure 4.4 and Figure 4.5. The contrast in entropy values between normal and Bot hosts due to the Bot is an automated computer programme and has regularity in control packets count. Whereby, the normal host traffic has a diversity and random value for the count of control packets. Consequently, normal hosts have a high entropy value, whereas the Bot has a low entropy value.

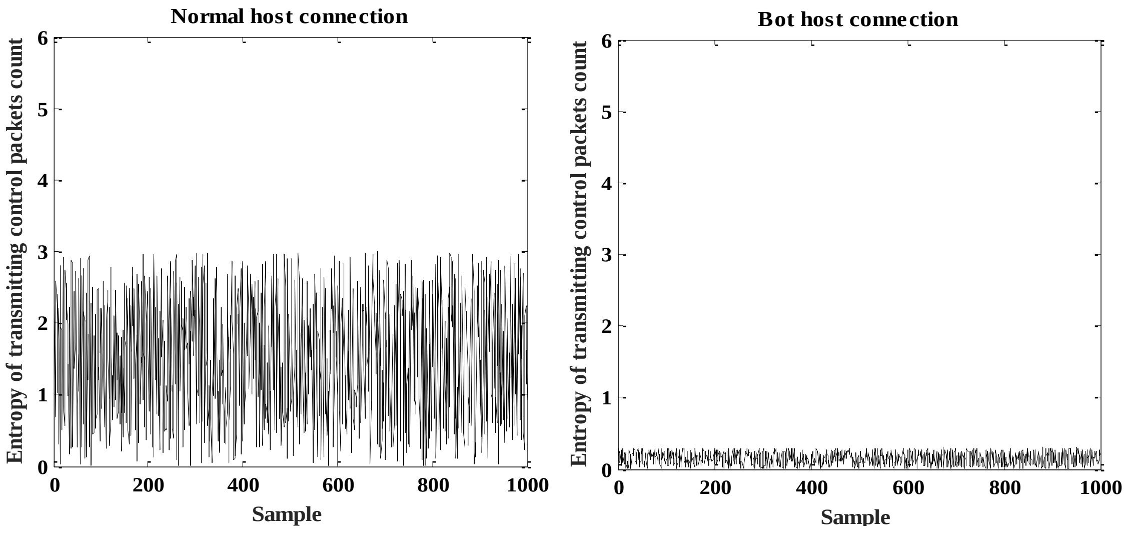

Figure 4.4 Entropy rate of transmitting control packets per host.

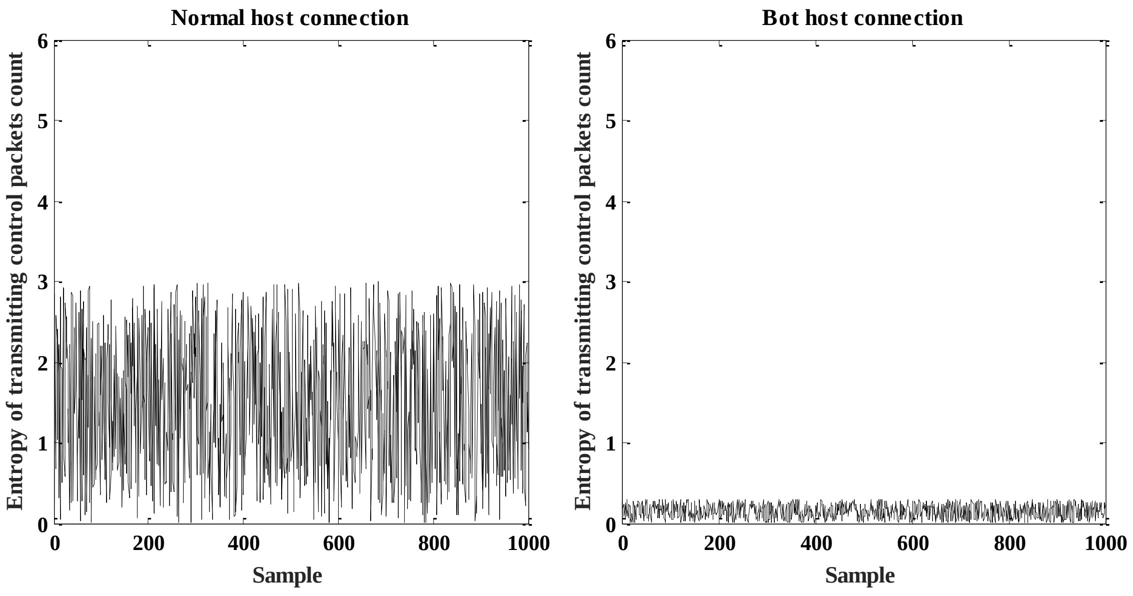

Figure 4.5 Entropy rates of receiving control packets per host.

4.5.5 Offline Bot Detection Approach Evaluation

The assessment results of the offline phase based on training dataset (Table 3.1) are demonstrated in Figures 4.6 - 4.9 the x-axis represents the size of the time window used for the feature extraction phase. It can be clearly seen that different performance measurements result from different time-window sizes. Based on these, the average values of cross-validation results for the time-windows are calculated. Therefore, the proposed approach with offline dataset gives the highest ACC, DR and F-measure rates of around 98.3%, 99% and 98.9 respectively based on a 60-seconds time-window; meanwhile, the worst performance achieved with a 10-seconds time window as shown in Figure 4.6

Figure 4.7 gives the AUC and MCC results of the Botnet detection system using the various time window sizes. The results show that the highest average AUC and MCC rates were 99.1% and 95.6% respectively with the training dataset and a 60-second time window; while the lowest AUC and MCC rates were 97.5% and 88.1% with the system with 10-second time-window. AUC and MCC are considered the most reliable performance measures for imbalanced datasets.

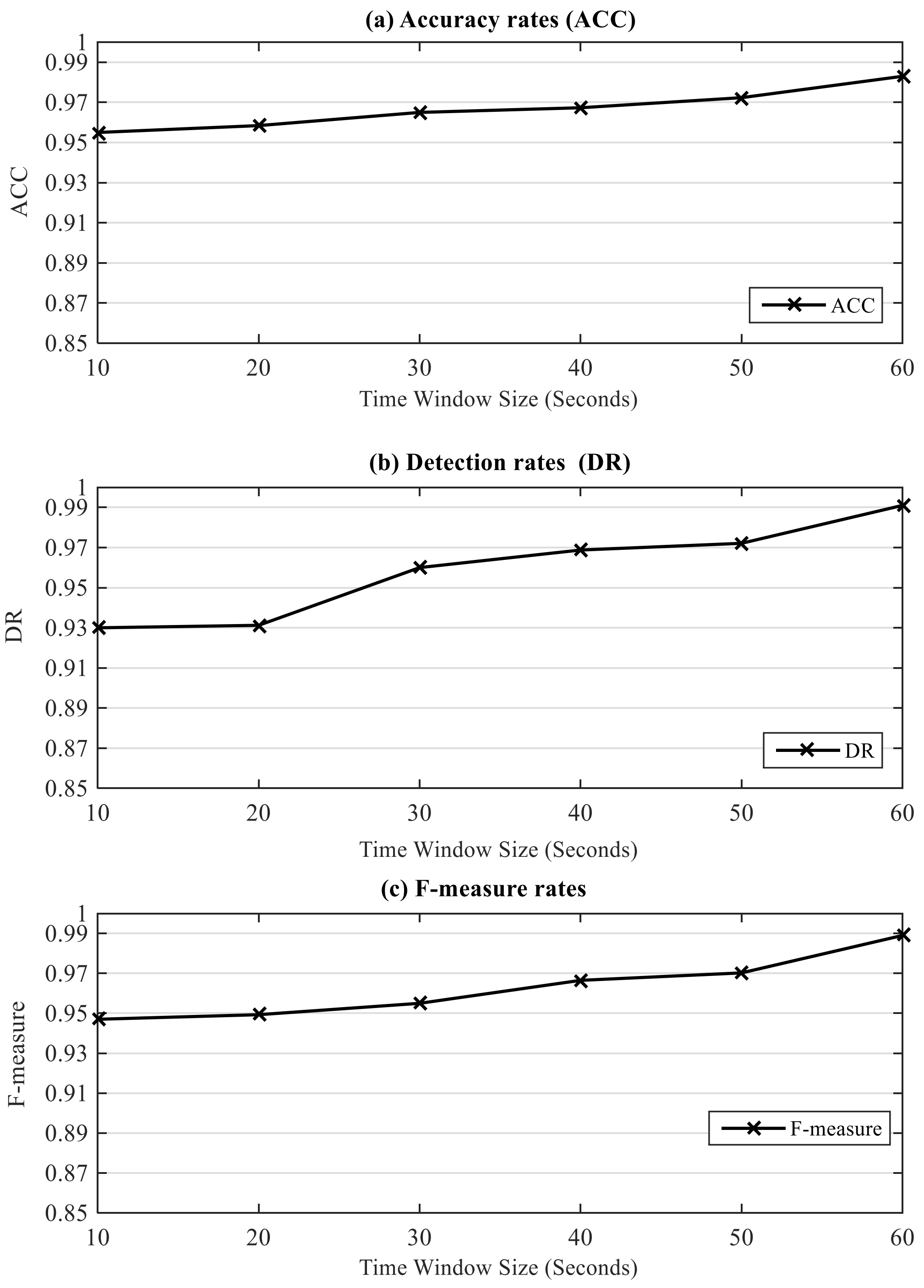

Figure 4.6 (a) ACC rates, (b) DR rates, (c) F-measure rates.

Subsequently, the performances of the proposed approach according to time window size was compared based on the average RMSE and NDEI, and the 60-second time-window achieved the best RMSE and NDEI rates at around 0.068 and 0.136 respectively as shown in Figure 4.8. In addition, the lowest FPR and FNR were given with 60-second time window size as shown in Figure 4.9

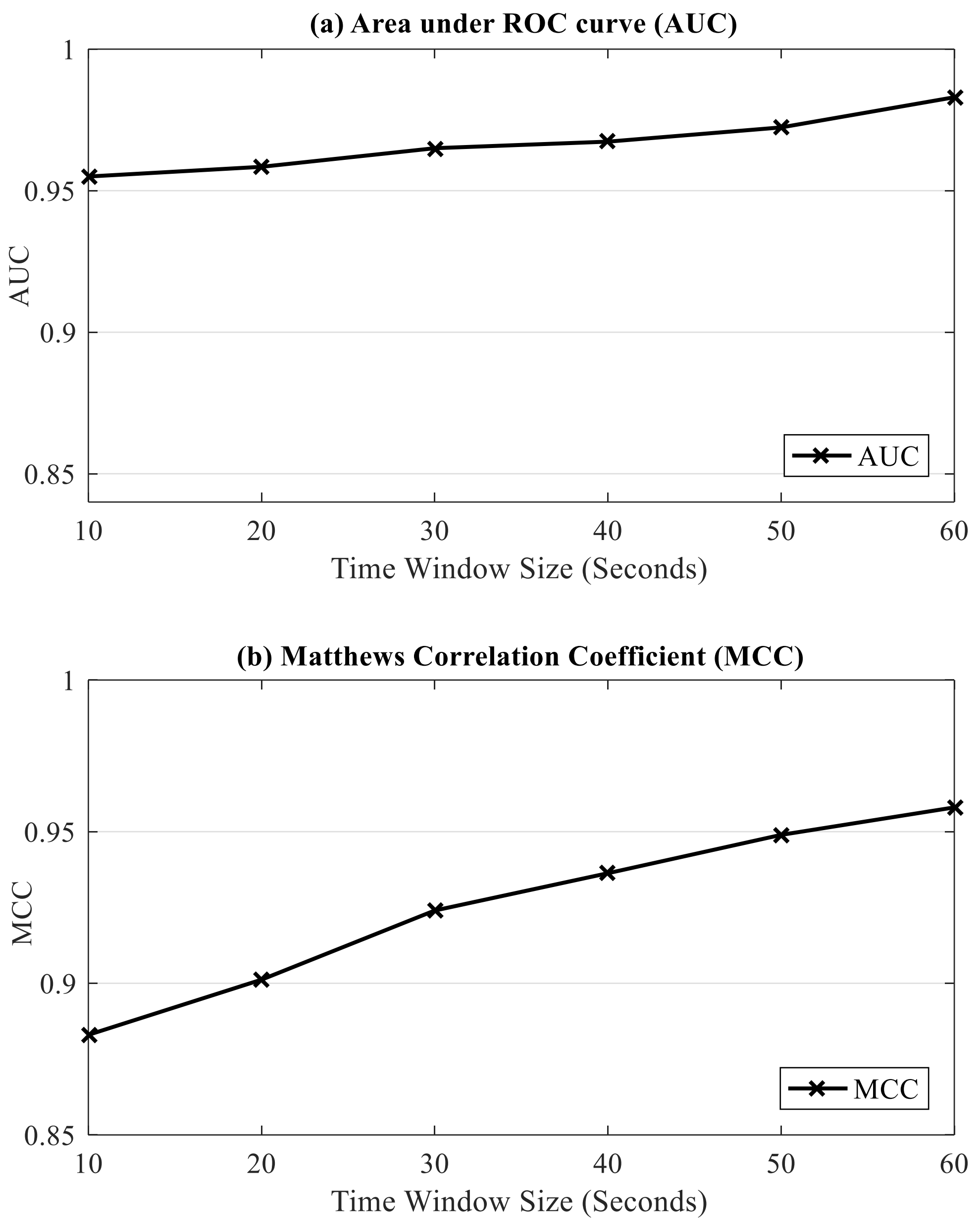

Figure 4.7 (a) AUC rates, (b) MCC rates.

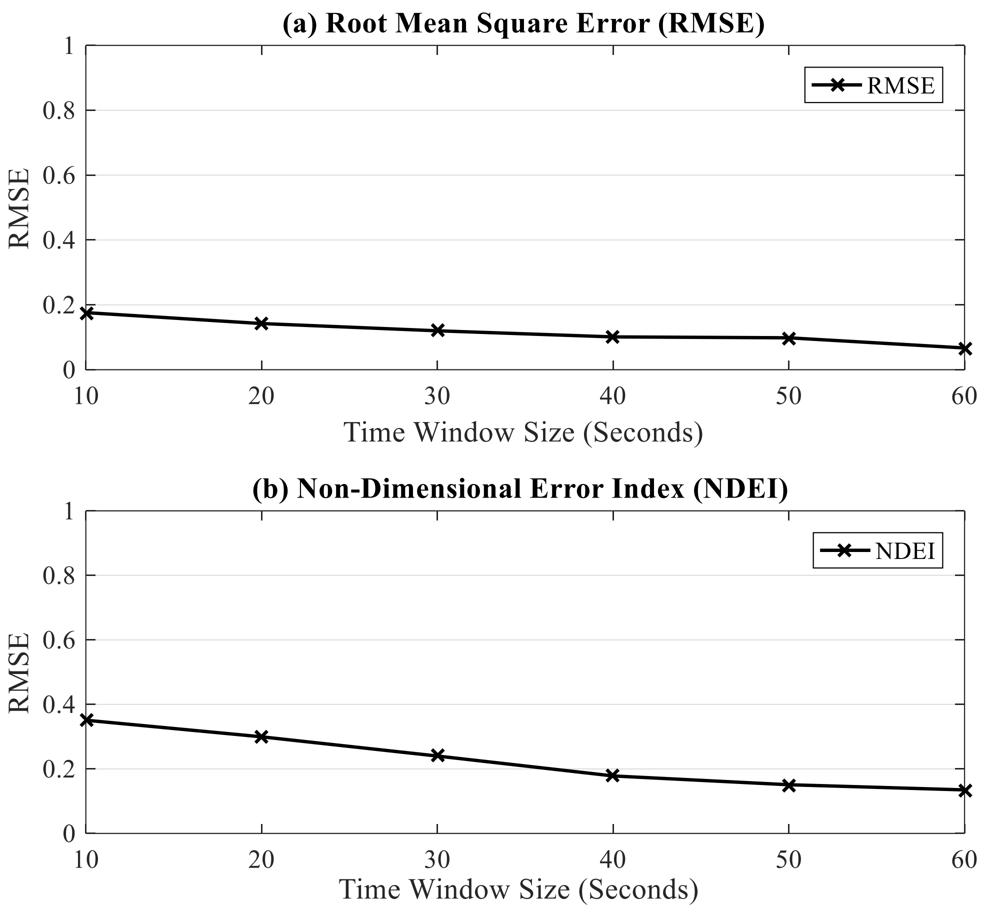

Figure 4.8 (a) RMSE rates, (b) NDEI rates.

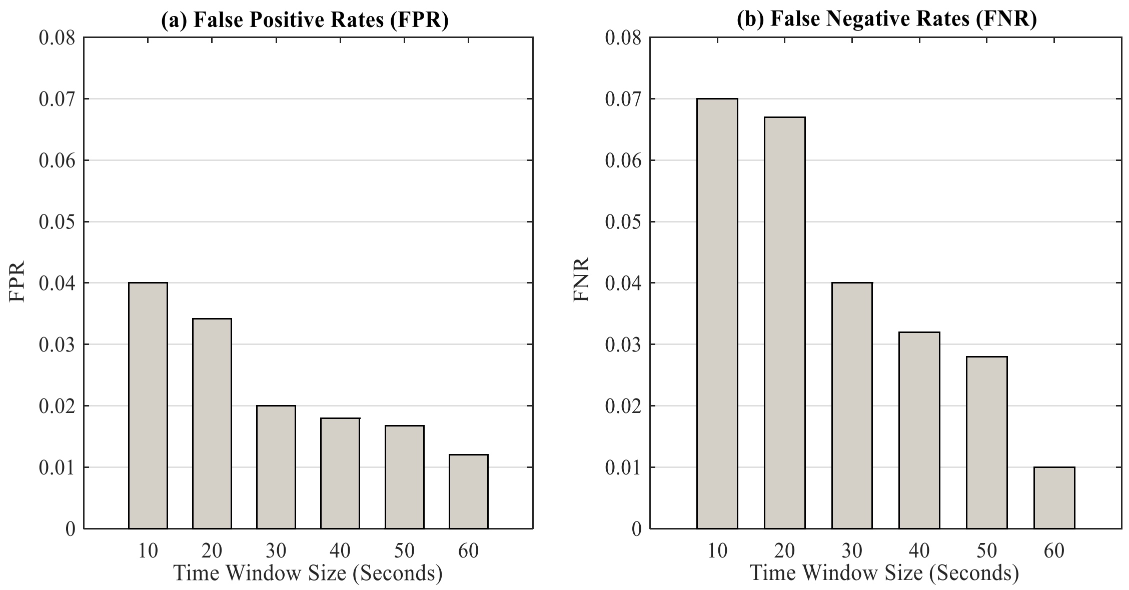

Figure 4.9 (a) FPR rates, (b) FNR rates.

From the results, the proposed approach in the offline phase is able to detect Bots with high identification accuracy along with low rates of false positives. Note that these outcomes are with the training dataset and not the testing dataset. The primary target of the offline phase is to prepare the classifier agent to start work in the online phase. More details for the experiment parameters are provided in Appendix C.

4.6 Discussions

The results of the offline phase were estimated using the average of cross-validation. It can clearly observe that the classifier showed various performances levels with different sizes of time-window. The time-window of 60-seconds achieved the best performance based on the ACC, DR, F-measure, FPR, AUC, MCC, NDEI and RMSE measures as shown in Figure 4.6 to Figure 4.9.

In addition, to measure the stability of the results in the offline phase, the standard deviation between cross-validation folds results was estimated. As shown in Table 4.5 the time-window size of 60-seconds achieved the lowest standard deviation for FPR, F-measure, NDEI, RMSE and AUC at 0.08%, 0.05%, 0.220%, 0.0123% and 0.002% respectively. Meanwhile, a time-window size of 10-seconds achieved the lowest standard deviation for DR, ACC, and MCC at 0.03%, 0.0045% and 0.03% respectively.

In summary, 10-seconds and 60-seconds time-windows showed the better stability of the results than the other time-window sizes as shown in Table 4.5. But the 60-second window based on imbalanced dataset measurements such as AUC and MCC achieved the best performance. Therefore, from the time-window size evaluation, 60-seconds were determined to be an appropriate window size in the proposed detection framework in the offline stage.

The 60-seconds time-window achieves high-performance along with acceptable stability results in our experiment. Meanwhile, a small time-window size, such as 10-seconds does not contain sufficient amount of network traffic which required classifying the network traffic as malicious or legitimate behaviours. Moreover, the Bots tend to generate a temporal behaviour following the infection phase (Hegna, 2010). Therefore, the 60-seconds time window size is suitable to capture adequately of the network traffic that helps in correct classification as discussed in the experimental procedure Section 4.5.2.

Table 4.5 Standard deviation of evaluation matrices.

Time window Size	**FPR S.D%**	**DR S.D%**	**ACC S.D%**	**F-Measure S.D%**	**RMSE S.D%**	**NDEI S.D%**	**MCC S.D%**	**AUC S.D%**
10-seconds	0.0974	0.03	0.0045	0.06	0.0124	0.238	0.03	0.0024
30-seconds	0.11	0.178	0.024	0.16	0.0339	0.246	0.035	0.0035
60-seconds	0.08	0.04	0.031	0.05	0.0123	0.220	0.041	0.002

However, due to its design, our solution is able to detect single Bot infections and it is not necessary to associate activity among multiple hosts during the detection phase, as in the case with TAMD (Yen & Reiter, 2008), BotMiner (Gu, Perdisci, et al., 2008) and BotSniffer (Gu, Zhang, et al., 2008). On the other hand, several existing schemes (Goebel & Holz, 2007; Gu et al., 2007) support the detection of individual Bot infections, but they use DPI. In contrast, our solution needs only information about network connections; it does not examine payload content. Therefore, it is immune to Botnets that use encryption methods.

4.7 Summary

In this chapter, we have introduced the connection-level feature set and the main component of the proposed offline Botnet detection mechanism. Besides, we have evaluated the introduced feature set using real network traffic. The output model of the offline stage it will be utilized in the next stage of the online Bot detection system. Therefore, the next chapter will present the proposed RL approach for online Bot detection scheme.

5 REINFORCEMENT LEARNING APPROACH FOR ONLINE BOT DETECTION

5.1 Introduction

Identifying infected computers before the Bot exploits the host machine is a challenging task in cyber-security. In the past few years, several methods have been proposed to identify Botnet threats, which represent a risk to cyber-security systems. The majority of these studies have focused on improving offline Botnet detection. However, the results obtained using these approaches reflect only the state of network traffic at the time. Therefore, these approaches may become useless once the network environment changes. In this situation, all offline techniques may become invalid since they do not include online strategies. Thus, the main goal of this chapter is to introduce an efficient online Bot detection approach using RL.

The previous chapters focused on the network traffic reduction, feature extraction and introduced the offline Bot detection approach. This chapter gives a brief introduction to RL including the components of an RL system, the Markov property, the partially observable Markov decision process and a classification of RL models. Furthermore, this chapter formulations the Botnet problem based on RL, followed by a model-based algorithm to achieve an online efficient Bot detection in a dynamic environment.

5.2 Reinforcement Learning

RL is a domain of machine learning inspired by behavioural psychology, based on how software agents take action in an environment in order to increase rewards. RL is learning by trial and error, where information about the state of an environment is received by the agent, who performs an action. Once the action is completed, the agent estimates the numerical reward resulting from the action. Increasing the rewards received is the goal of the agent at all times.

A wide range of algorithms has been suggested that use selective action in order to explore the environment, and to develop a strategy that leads to achieving the best reward (Barto & Andrew, 1998; Kaelbling, Littman, & Moore, 1996). These algorithms have been successfully utilized to solve complicated problems, for instance, elevator dispatch (Crites & Barto, 1998), board games (Tesauro, 1995), motor control tasks (Schaal & Atkeson, 1994) and job-shop scheduling (W. Zhang & Dietterich, 1996)

Along with the disciplines of supervised learning (Harmon & Harmon, 1996) and dynamic programming, reinforcement learning is used to generate robust machine learning algorithms (Bertsekas & Tsitsiklis, 1996). Beyond a technique for solving control problem, RL can be considered as “one of the only designs of value in understanding the human mind” (Werbos, 1992). It is a way of learning optimal policy in an undiscovered or partially observed environment. Thereby, RL is based on the idea of trial and error in interplay with a dynamic environment (Barto & Andrew, 1998).

5.2.1 Components of Reinforcement Learning System

The main components of the RL or control problem are briefly reviewed in this section. The relationships between these components are also depicted in Figure 5.1 (Barto & Andrew, 1998).

- Environment: The environment is matched to any system such as elevator dispatch, motor control tasks, board games or an intrusion detection system. The development of the environment depends on the history and actions executed by the agent. For each interaction, a reward R_t is transmitted to the agent, and this operates as an evaluation measure for the agent’s subsequent action in the new environment state. The environment states can be continuous or discrete.
- Agent: The agent refers to the controller of the system. It has completed an observation or at least partial observation of the environment while interacting with it to receive an observation about state X_t. Therefore, the agent receives the rewardR_{t+1} from performing action A_t based on receiving the environment status X_t. Furthermore, the reward R_{t+1} can apply to enhance the agent policy.

- Actions (A): Actions result from the evolution of the environment. They refer to changes in the agent's environment. Moreover, some actions cannot instantly change the system, but often need a specificied delay. However, actions of agent can be restricted depend on the problem solution setting. Corresponding to the setting of the problem, actions can be discrete or continuous.
- Policy (π): is the mapping between the environment's state and the action which can be considered in this state, and is a sequence of actions which reflects the agent's learning rate at a given time. In some situations, the policy can use a lookup table of rules or simple functions. According to Sutton and Barto "the policy is the core of a RL agent in the sense that it alone is sufficient to determine a behaviour" (Barto & Andrew, 1998).
- Reward (R): The term reward refers to the goal of RL. It represents the direct reward the agent receives for executing a particular action in a given system state. Therefore, it determines the utility of an action taken by an agent. Strictly speaking, in the long term, the primary target of a RL agent is to maximize the total rewards received. Thus, a reward function indicates which action is correct immediately, while the value function specifies what is good in the long run.

As shown in Figure 5.1, the RL model includes an environment and an agent. At a given time , the environment gives a state S_t to the agent and then the agent executes an action A_t according state S_t and policy π. After that, the environment changes to a new state S_{t+1} Additionally, at the same time, the environment also gives a numeric reward R_{t+1}, which is an immediate reward or penalty for choosing action A_t in state S_t (Barto & Andrew, 1998). However, the goal of any RL approach is to improve the policy in order to maximize the long-term reward.

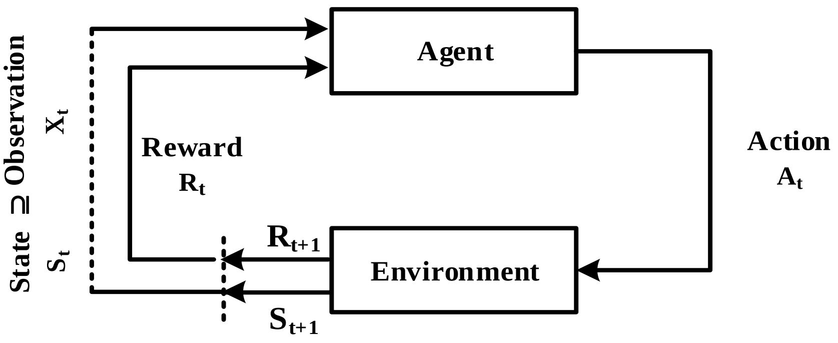

Figure 5.1 General RL system architecture (Barto & Andrew, 1998).

5.2.2 The Markov Property

5.2.2.1 Markov Decision Process

The mathematical foundation of general theoretical RL problems is a Markov decision process (MDP), which explains the evolution of a fully observable system. Therefore, if the state and action space are deterministic, the dynamics of the MDP can be described by the probability that the next state will be S_{t+1} based on the fact that the current state is S_t and the chosen action is A_t . An MDP is primarily represented by a tuple (S, A, T, R) with the following objects (Feinberg, Shwartz, & Altman, 2002; Kaelbling, Littman, & Cassandra, 1998; Puterman, 2014).

- $t \in N$ specifies the time step.
- S refers to a state space of the environment.
- A indicates an action, $A(S_t)$ represents the authorized actions in the state $S_t \in S$.
- $T(S_{t+1}|S_t, A_t)$ is a deterministic state transition function: $S \times A \times S \rightarrow [0, 1]$, which denotes the probability of achieving state S_{t+1} starting from state S_t and using action A_t with S_t, $S_{t+1} \in S$ and $A_t \in A(S_t)$.
- a reward function $Rt = R(S_t): S \rightarrow R$ indicated the one-step direct reward starting from state S_t.

The relationship between the various objects is explained by a one-step transition given an open (controllable) dynamic system for a state space S. Being in an arbitrary state $S_t \in S$ at time step t, the agent takes an action $A_t \in A(S_t)$. As a result, the system develops to the next state $S_{t+1} \in S$ based on the transition function $T(S_{t+1}|S_t, A_t)$. At the same time, the agent obtains the reward $R(S_{t+1})$ from state S_{t+1} (Feinberg et al., 2002). The sequence of actions and states produced are termed a trajectory.

Due to the definition of the Markov property (def. 5.1) the next state S_{t+1} is based on the action A_t and the current state S_t. In other words, the Markov property states that the evolution of the system is based on the last action taken and the system state (Feinberg et al., 2002; Gass & Fu, 2013). Therefore, it is independent of its history of previous states and actions. The Markov property in discrete time is thus stated as S_0 to refer to an arbitrary beginning state (Gass & Fu, 2013).

Definition 5.1. Markov property: a discrete stochastic process $S_t \in S$ with action $A_t \in A$ and a transition function $T(S_{t+1}|A_t, S_t)$ is called Markovian if for every $t \in N$ it is:

$$T(S_{t+1}|A_t, S_t, A_{t-1}, S_{t-1}, \ldots, A_0, S_0) = T(S_{t+1}|A_t, S_t) \tag{5.1}$$

Additionally, the Markov property can be used in a formalized RL problem. When an agent is at time step t, the agent receives information about the environment state S_t, and it must utilize the state S_t information to predict action A_t. If the agent takes an action depending on the current state S_t and not based on any of the previous states $S_{t-1}, S_{t-2}, \ldots., S_0$, or any of the previous actions $A_{t-1}, A_{t-2}, \ldots., A_0$ or any of the previous rewards $R_{t-1}, R_{t-2}, \ldots, R_0$, then the state possesses the Markov property and is a Markov state. If all of the states in the environment have this characteristic, we can say it is a Markov environment and has the Markov property.

The Markov property plays a vital role in any RL system because the agent makes an action based only on information about the current state. The majority of real system environments are not completely Markovian, but they approximate a Markov environment.

Figure 5.2 represents the basic RL problem using a Markov decision process (MDP). The system environment is fully observable. Therefore, the observation of the agent's X_t is equivalent to the deterministic environment state S_t. Therefore, based on the Markov property, the agent has all the information needed to select its next action A_t according to the sequential policy used for mapping between an observation state $X_t(= S_t)$and the next action. The environment then develops due to the transition function $T(S_{t+1}|A_t, S_t)$.

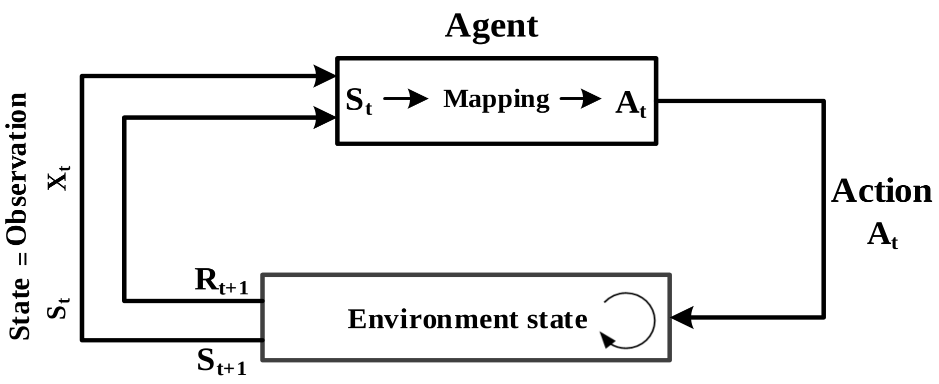

Figure 5.2 Markov decision process (Schäfer, 2008).

5.2.2.2 Partially Observable Markov Decision Process

The partially observable Markov decision process (POMDP) was first explicitly studied by Drake in 1962 (Drake, 1962). Smallwood and Sondik later introduced the first algorithmic work based on POMDP by combining the basic idea of the model of the Markov decision process with the concept of an agent which might be unable fully to know the environment state (Smallwood & Sondik, 1973).

The POMDP differs from the MDP in the way that the state space S is not completely detectable. This is regularly applied to a real system's operation in a dynamic environment, such as the Botnet detection problem. The agent only obtains an observation $X_t \in X$ as a sign of the immediate state of the system, $S_t \in S$. Formally, a POMDP can be represented by a tuple (S, X, A, T, R), where X represents the observation space, which is a space that is contained within the state space S and may also include extra irrelevant information.

Figure 5.3 provides a general graphical illustration of a partially observable Markov decision process. As opposed to the MDP (Figure 5.2), the state of an environment S_t is only partially observable by the agent, which is represented by the expression $X_t \subset S_t$. This indicates that the agent has the additional job of having to approximate to decide actions A_t. In particular, the agent has to construct a model of the environment, which it applies as the foundation for its decision-making. Moreover, it utilizes the system's past time state to improve future actions.

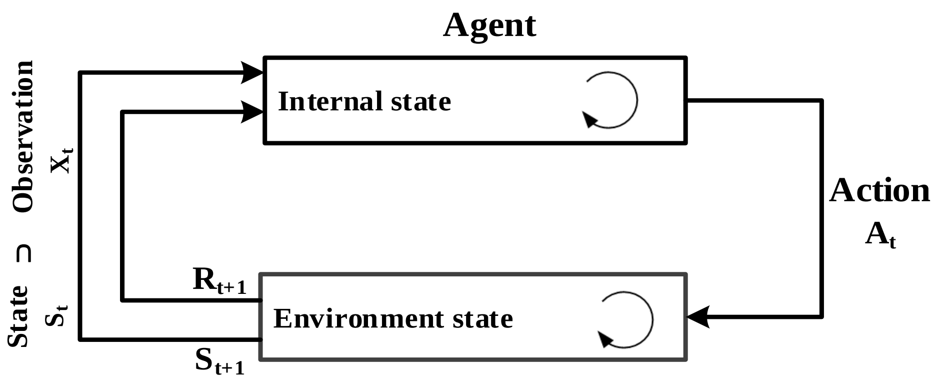

Figure 5.3 Partial Markov decision process (Schäfer, 2008).

5.2.3 Reinforcement Learning Models

There are several various ways to classify RL techniques. A primary difference can be created between table-based and function approximation methods. In the table-based method, the agent stores the environment state and action within a look-up table. This approach suffers from the limitation of the computational requirements due to table size. Thus, such techniques are essentially used in RL with low-dimensional discrete state space problems, for instance Q-learning (Watkins, 1989) and adaptive heuristic critic (A. G. Barto, Sutton, & Anderson, 1983). On the other hand, in function approximation techniques the state and action are represented by an approximation function in order to increase the performance of the system and can be used for high dimensions problems. Examples of these are temporal difference approaches with neural networks (Tesauro, 1994; Tsitsiklis & Van Roy, 1997) local basis functions (Barto & Andrew, 1998), as well as neural fitted Q-iteration (Martin Riedmiller, 2005).

Another significant contrast can be made between model-free and model-based algorithms. In brief, model-free methods teach a controller without learning model, and without using a labelled dataset to build transition function policy. So, they immediately learn from data without making any effort to create a model. This leads to an efficient and simple implementation. In a model-based indirect adaptive method, the system first learns a model and then utilizes it to find an optimum policy. Therefore, it needs further computation, but gives the model extensions to deal with real-world problems like Botnet detection, which cannot be easily controlled without an available dataset to learn the model in the initial phase.

In addition, function approximation machine learning approaches, such as artificial neural networks, determine optimal performance by looking at examples in the training dataset. This procedure is very beneficial when examples of optimal behaviour are readily available. However, in some problems a training dataset that represents the problem does not exist or it is incomplete, and so only limited information is available about the optimal solution. RL techniques discover the optimal behaviour by trial and error, which implies that no information is needed in advance about the optimal behaviour. This characteristic makes RL an important domain in artificial intelligence applications since it does not depend on a complete training dataset being supplied.

5.2.4 Exploration Versus Exploitation

RL is a technique of preparing an agent to learn. The agent learns by getting rewards following each action. It somehow keeps track of these rewards and then chooses actions that lead to maximizing the reward, not automatically for the next action but in long-term execution (Barto & Andrew, 1998). The agent normally goes through the same environment several times to learn how to decide upon optimal actions. Balancing exploration and exploitation is especially necessary here; the agent may have obtained a sound goal on one path, but there might be an even best one on the different path. Thereby without exploration, the agent will regularly return to the first goal, and the most beneficial goal will not be attained. Alternatively, the goal may lie after many steps of agent action. Therefore, it is significant to balance exploration and exploitation in order to guarantee that the agent is learning the optimal actions. However, any RL techniques require a strategy to guarantee that there is such a balance. Various methods that can be utilized to achieve a good balance between exploration and exploitation, such as greedy exploration, frequency and recency-based exploration (Barto & Andrew, 1998), R-Max (Brafman & Tennenholtz, 2003), decaying exploration, and persistent exploration (Singh, Jaakkola, Littman, & Szepesvári, 2000).

5.3 Formulation of Botnet Problem Using Reinforcement Learning

Sutton and Barto studied the RL algorithms for learning to control a system effectively by communicating with the environment and perceiving the rewards received (Barto & Andrew, 1998). RL methods are a common selection for problems where it is hard to specify precisely an explicit software solution, but where it is possible to produce a reward signal, which is exactly the situation in our Botnet problem. Here the RL obstacle is expressed in the context of partially observable Markov decision processes (POMDP). POMDPs are normally used to represent dynamic systems such as Botnet detection systems.

A POMDP is described by a set of states (S), depicting the probable states of the controller agent state (AG_{St}), neural network agent state (NN_{St}) and host state (H_{St}). The neural network agent action at time t is (NN_{At}), and the neural network agent chooses actions based on policy π, where $NN^{\pi}(H_{St}, A)$ is the probability of the agent choosing action A when it is the host in the state (H_{St}). A reward function R is estimated as $R(AG_{St})$. A transition function for the system control agent is (T_{St}).

The Markovian transition function defines the dynamics of the system and provides the possibility $T(AG_{St}, NN_{At}, AG_{St+1})$ of transitioning to state Agent AG_{St+1} after taking action NN_{At} in state AG_{St}. The reward function assigns the number of new hosts state H_{St} and the total number of host states in the system is processed as integer numbers to state agent AG_{St}.

At any time, POMDP represents the system state, and when an action is selected by the neural network agent NN_{At} the host state value and controller agent reward are estimated. Next, according to the size of the reward collected, the transition function of the controller agent T_{St} changes the neural network agent to a new state NN_{St+1}. In this research, P2P Bot detection is expressed as a RL problem. This primarily involves selecting the value state function, the reward function, the action space and the transition function.

Action Space: In defining the action space, the node on the network at every time window will be given a probability as a legitimate or Bot node. After that, the RL agent takes this probability into account in order to estimate the reward from these states.

Agent Reward Function: The reward signal is defined at any time step to be equal to the number of new states processed by any node in the network during the number of the time window. Note that this reward signal will estimate the important of the new state using the value state function in set time windows, and here the new state can be a legitimate node or a node infected by a Bot.

Value state function: Any (H) node inside the network has many states based on the mode of use**.** Moreover, the value state function represents the expected reward from each host state H_{St} under the policy NN^{π}**.** The neural network agent output for every host state in every time-window can be divided into two sub-state of probabilities as Bot $E(B)$ or legitimate$E(N)$, and so the outcome of every host state is represented as $(E.B\ (H_{St}))$ or $(E.N\ (H_{St}))$, as shown in Figure 5.4

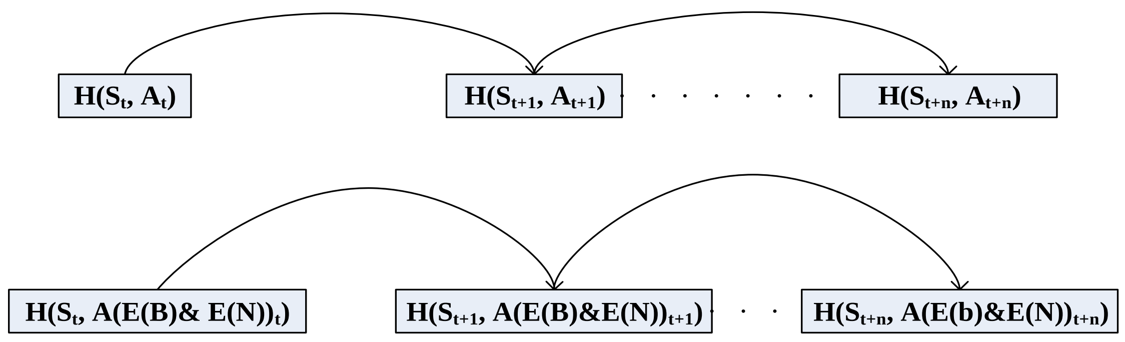

Figure 5.4 Host states.

- Value state function evaluation for Bot hosts:

$$EV^{\pi}(H(B)) = \frac{\sum_{i=0}^{n} E(B_{St(i)})}{n} \tag{5.2}$$

$EV^{\pi}(H(B))$ represents the average expected Bot status for the host in the n time window. Here, $E\big(B_{St(i)}\big)$ is the probability a malicious behaviour outcome from the computer under the current neural network agent policy.

- Value state function evaluation for legitimate hosts:

$$EV^{\pi}(H(N)) = 1 - EV^{\pi}(H(B)) \tag{5.3}$$

$EV^{\pi}(H(N))$ represents the average expected legitimate status for the host in the n time window. Here, $E(N_{St(i)})$ is the probability of a malicious behaviour outcome from the computer under the current neural network agent policy.

- Value state function evaluation for controller Agent :

$$V(s)=V(s) + \begin{cases} V(B_{St})=argmax\big(B(Actions)\big) & EV^{\pi}(H(B)_{St}>EV^{\pi}(H(N)_{St} \\ V(N_{St})=argmax\big(N(Actions)\big) & EV^{\pi}(H(B)_{St}<EV^{\pi}(H(N)_{St} \end{cases} \tag{5.4}$$

where $V(s)$ is the accumulated states which achieve maximum reward based on the current policy of the neural network agent NN^{π}.

Transition function. Next, the (controller agent) transition function must be defined. Any technique in the RL field requires some kind strategy which ensures that there is a balance between exploration and exploitation. The problem is how to find a good action-selection policy based on the appropriate amounts of exploration and exploitation.

One purpose of this study is to find a beneficial technique in order to make a balance between exploration and exploitation for Bot detection problems. Therefore, a directed exploration approach is used. The goal of the exploration approach is to explore as much of the state and action as possible before switching strategy and starting to exploit this knowledge.

The simplest directed exploration techniques are greedy methods that, in each state, select the state with the highest probability of experiencing value. Furthermore, the explorative strategy is followed by a series of steps in order to find hidden goals. If the goal is a new unique state for the system, then it is easy for the system to change from explorative to exploitive when this seems to be more beneficial.

The transition function is then

$$T_{St} = \frac{\sum new\ V(s)}{\sum V(s)} \geq \theta \tag{5.5}$$

where T_{St} indicates the rate of exploring new state $new\ V(s)$ over all the environment state $V(s)$. Thus, the value of T_{St} is variable depending on the amount of the analysed network traffic. In addition, θ is an adaptive threshold value that is determined by the network administrator depending on the desired security level of the network, for example, in an army network θ is very small in contrast to another type of networks such as, universities networks. Moreover, when θ has a low value this means that the learning rate is high.

5.3.1 Bot Detection Algorithm Using Reinforcement Learning

In terms of Bot detection, here is an explanation of the development of the Algorithm 5.1. The main components of the proposed algorithm, which its function is to detect Bot, are discussed in the previous section. In addition, Figure 5.5 illustrates the Algorithm 5.1 steps.

The proposed Bot detection system, the Algorithm 5.1, continually monitors the network environment. Firstly, it extracts an observation from the environment and decides an action based on the current neural network policy. Meanwhile, a vector V is used to accumulate the new state and action for each observation. Whenever the agent accumulates the beneficial amount of new states, it changes to the exploitation state to utilize these states. Finally, the main control agent evaluates the performance of the new neural network agent before using it.

Algorithm 5.1: Bot detection using an RL technique.

```
1: Initialize V(s) = 0;
2: Initialize $T_{st}$ =0;
3: All_Dataset=RefDataset;
4: Temp_Dataset=0;
5: Read current environment observation (state ($S_t$));
6: Perform action $NN\pi \; (A \mid St, St+1)$;
7: Execute action and extract rewards (R);
8: Estimate the probability of Bot node:
        $EV^{\pi}(H(B)) = \frac{\sum_{i=0}^{n} E(B_{St(i)})}{n}$
9: Estimate the probability of legitimate node:
        $EV^{\pi}(H(N)) = \mathbf{1} - EV^{\pi}(H(B))$
10: Extract the state with high expected reward:
        $V(s)=V(s)+\begin{cases} V(B_{St})=argmax(B(Actions)) & EV^{\pi}(H(B)_{St}>EV^{\pi}(H(N)_{St} \\ V(N_{St})=argmax(N(Actions)) & EV^{\pi}(H(B)_{St}<EV^{\pi}(H(N)_{St} \end{cases}$
11: Check the amount of extracted reward:
        $T_{St} = \frac{\sum new\ V(s)}{\sum V(s)} \geq \theta$
12: If $T_{st}$ >= θ
        ▪ Temp_Dataset = Temp_Dataset +V(s).
        ▪ Reset V(s).
13: ($NN2^{\pi}$): Creation and Evaluation:
        ▪ Create a new neural network (NNπ2) using Temp_Dataset.
        ▪ Evaluate the performance of ( NNπ2 ) using cross-validation
          techniques.
        ▪ Evaluate the performance of (NNπ2) using RefDataset.
14: IF ($NN^{\pi}2$) pass the evaluation phases
        ▪ $NN^{\pi}$ = $NN^{\pi}2$;
        ▪ All_dataset= All_dataset+ Temp_Dataset;
        ▪ Reset Temp_Datase;
        ▪ EndIF
Return to step:1.
EndIF
```

The key benefit of the approach introduced is that it will remain to a strategy for a period of time, and will not perform one-step in the exploratory direction and-one-step in the exploitative direction. However, managing the rate of learning (exploration) new behaviour (states) depend on the network traffic state. In the case of huge volumes of network traffic the controller agent will be found a high number of new state comparison with low network traffic. Once the system determines the most beneficial amount of reward it changes to the exploitative strategy by producing a new dataset from the old dataset and newly extracted states to use for retraining a new neural network agent. Once the new neural network agent is trained, three procedures are used to grantee the quality of the system outcome. Firstly, a cross-valuation approach is applied to evaluate the result of the new neural network agent and estimate performance evaluation matrices such as AUC, MCC, accuracy, and RMSE. Secondly, the new neural agent is evaluated using the old reference dataset (state and action) and estimating the performance evaluation of AUC, MCC, accuracy, and RMSE. Thirdly, if the system passes the evaluation test then the main controller of the system will replace the neural network agent with a new one. However, if the new neural agent fails to achieve good performance, the system retains the current neural network agent and reset the new state and action buffer.

In summary, there are three neural network agents in the system’s reference neural network. The first initial agent’s neural network is trained using a reference dataset (states and actions). The second the neural network is created using new environment observations (states). Finally, the neural network with the best configuration that passes the evaluation phase is used in the detection process.

The complex nature of the proposed approach is derived from the complex of the neural network with resilient backpropagation learning. Resilient back-propagation (RPROP) is considered the best an algorithm which combined robustness, speed and accuracy (M. Riedmiller & Braun, 1993). Furthermore, according to Christian Igel et al. (2005) the RPROP algorithm has linear time and space complexity (Igel, Toussaint, & Weishui, 2005)

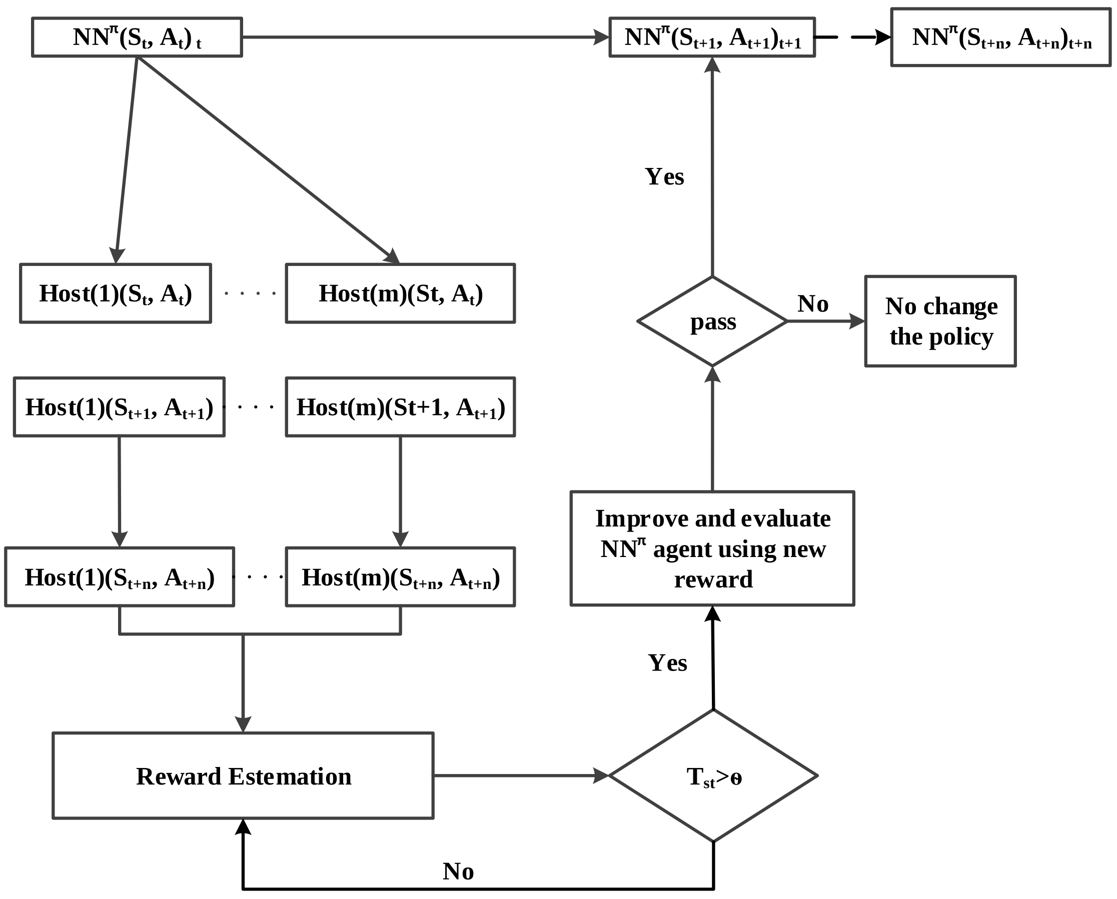

Figure 5.5 Neural Network Agent state.

A multilayer neural network with i inputs, h hidden units, and u outputs has $H\ (i\ +1)$ weights on the first layer and $u\ (h\ +1)$ weights in the second layer. Both space and time complexity of an MLP is $O\ (h\ \cdot\ (u\ +\ i))$. When e denotes the number of training epochs, training time complexity is $O(e\ \cdot h\ \cdot\ (u\ +\ i))$. In an application, i and u are predefined and h is the parameter that we play with to tune the complexity of the model (Alpaydin & Ethem, 2014).

The complexity of the proposed approach is based on the complexity of the neural network. So, in our proposed approach, the complexity of create a new neural network $NN^{\pi}2$ is:

$$O\ (e\ \cdot\ h\ \cdot\ (u\ +\ i))\ =\ O\ (N) \tag{5.6}$$

Where N denote to the number of weights.

In addition, the complexity of evaluation of new neural network based cross-validation approach is:

$$f * O\,(e \cdot h \cdot (u + i)) = O\,(N) \qquad (5.7)$$

Where f is the number of folds.

Finally, the complexity of proposed approach is $O(N)$.

5.4 Online Bot Host Detection Approach

At online detection stage, the learned neural network agent classifies the host inside the network continually and sends a report about the hosts' activities to the network administrator. Moreover, as shown in Figure 5.6, the RL agent at the same time works to extract any new features that will help to improve the performance of the detection agent in the future.

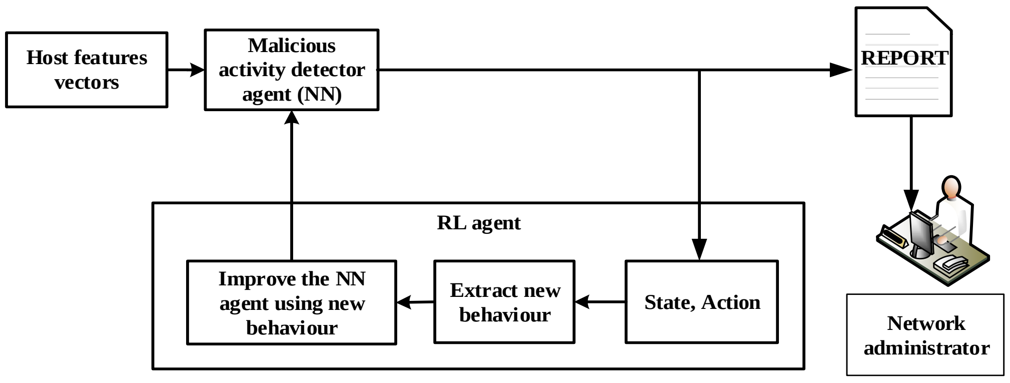

Figure 5.6 Overview of On-line Bot Detection Phases.

This research introduces a new technique in which the RL agent's activities are divided into two phases: a) new behaviour is extracted as shown in Figure 5.7, and then b) improving the neural network agent using the newly extracted behaviour as shown in Figure 5.8.

A. Extract new behaviour using a RL agent:

1. Extract the state of the environment and the action that it performs by use of the current neural network agent.
2. Check if the state is a new state according to the reference set of states (training dataset), go to step 4.
3. The agent is waiting for the next state and action, go to step 1.
4. The agent creates a vector for every host on the network to store the state and action that occurred.

5. Check if the total number of the new states for any host in the network is equal to a threshold number, and then estimate the average of the actions that was done by the host in order to find the best reward value based on the probability and frequency of the action. Otherwise, return to step 1.
6. Check that the overall amount of reward is the balanced between exploration and exploitation based on the adaptive threshold value. So, if a rate of new states satisfies the value of the adaptive threshold, then go to the next phase of the proposed framework in order to exploit the newly extracted features to improve the system. Otherwise, return to step 1 to increase the exploration rate.

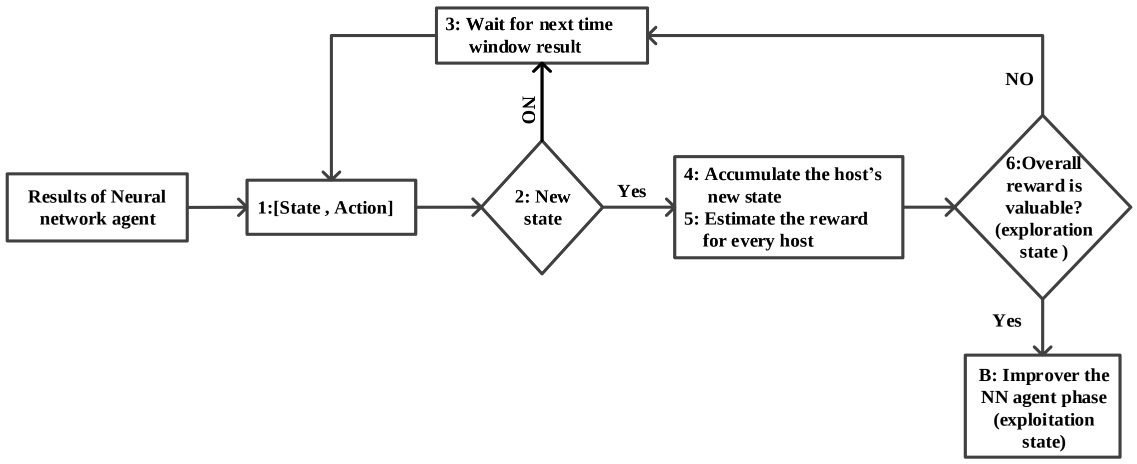
3: Wait for next time window result
NO
NO
Results of Neural network agent
1:[State , Action]
2: New state
Yes
4: Accumulate the host's new state
5: Estimate the reward for every host
6:Overall reward is valuable? (exploration state)
Yes
B: Improver the NN agent phase (exploitation state)

Figure 5.7 Extract new behaviour phase.

B. Improve the neural network agent using new extracted online behaviour:

7. Create a vector for the reward (new state) to use it to improve the system to detect new states of attacks.
8. Adopting new state and action by incremental training of the new neural network agent to add a new policy to the system.
9. Evaluate of the efficiency of the new neural network using 10-fold cross-validation. In addition, check if it success of satisfies the minimum requirements based on the cross-validation result of the new neural network agent and go to step 11.
10. If the system does no change the neural network policy, reset the reward vector of state and action and return to step 6 to check f it has valuable new rewards.

11. Test the new neural network using the reference dataset. If it succeeds in satisfying the minimum requirements for classifying the host inside the network, go to step 13.
12. If there are no changes in neural network policy, reset the reward vector of state and action and return to step 6 to check if it gives valuable new rewards.
13. Replace the last good configuration neural network (main agent) with the new neural network with incremental training using the new state and action that are extracted in the online phase, and reset the reward vector of state and action and return to step 6 to check if it gives a valuable new reward.

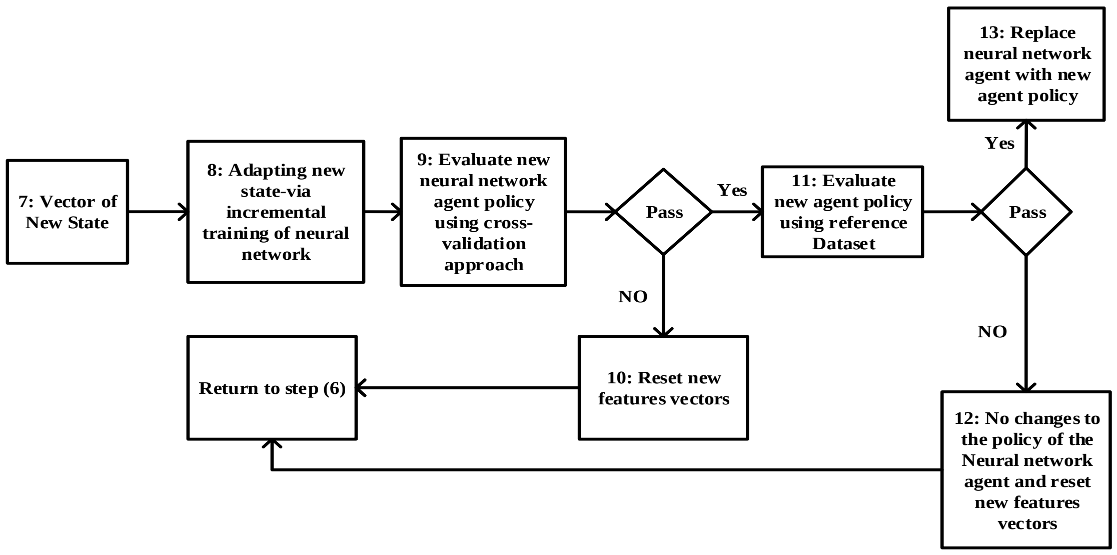
13: Replace neural network agent with new agent policy
Yes
7: Vector of New State
8: Adapting new state-via incremental training of neural network
9: Evaluate new neural network agent policy using cross-validation approach
Pass
Yes
11: Evaluate new agent policy using reference Dataset
Pass
NO
NO
Return to step (6)
10: Reset new features vectors
12: No changes to the policy of the Neural network agent and reset new features vectors

Figure 5.8 Improving the classifier agent.

Figure 5.7 and Figure 5.8 shows the main components of RL agents. The model for extract new behaviour (learning agent) and the model of adopting the new behaviour phases are demonstrated. In This study, a novel connection between RL and neural networks is given in order to resolve the control problems with dimensionality and the partially observable environment.

5.5 Experimental Results and Evaluation

5.5.1 Experiments Using Differences Sliding Time-window Size

This section gives an overview of the results of the proposed technique in Bot detection using the testing dataset (Table 3.1). We have conducted the experimental procedure that mentioned in Section 4.5.2 in Chapter 4. The results obtained from the analysis of the online experiment outcomes are summarized in Figure 5.9 to Figure 5.12.

As shown in Figure 5.9, the proposed approach using online evaluation gives the highest ACC, DR and F-measure rates of around 98.8%, 98.3% and 97.9% respectively using the 60-seconds time window; meanwhile, the lowest performance of the proposed approach using these measures was achieved with a 10 seconds time-window.

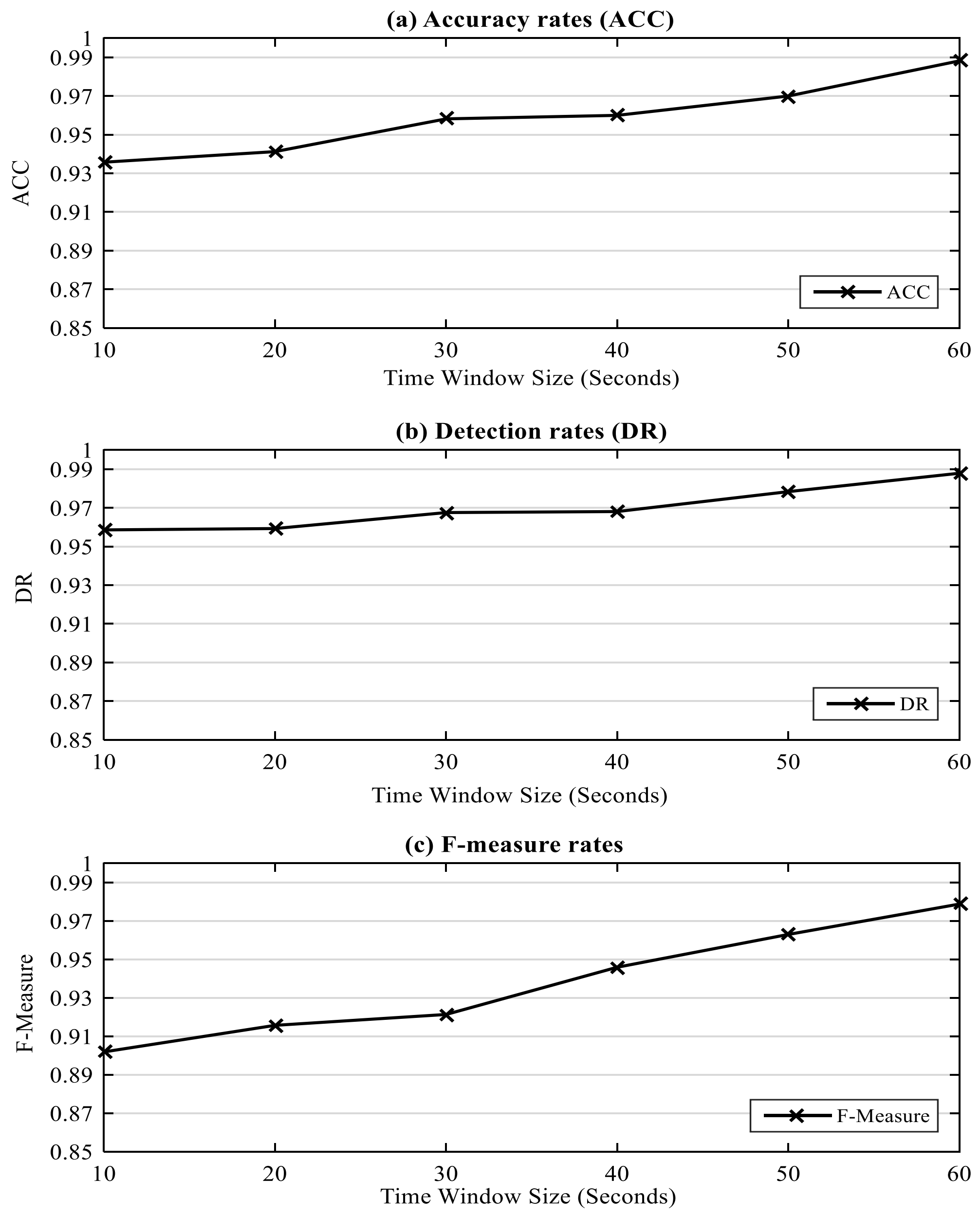

Figure 5.9 (a) ACC rates, (b) DR rates, (c) F-measure rates.

Figure 5.10 presents the performance measurement AUC and MCC for the imbalance dataset. The results show that the highest AUC and MCC rates were 99.96% and 95.6% respectively in the online testing dataset evaluation using a 60-seconds time-window.

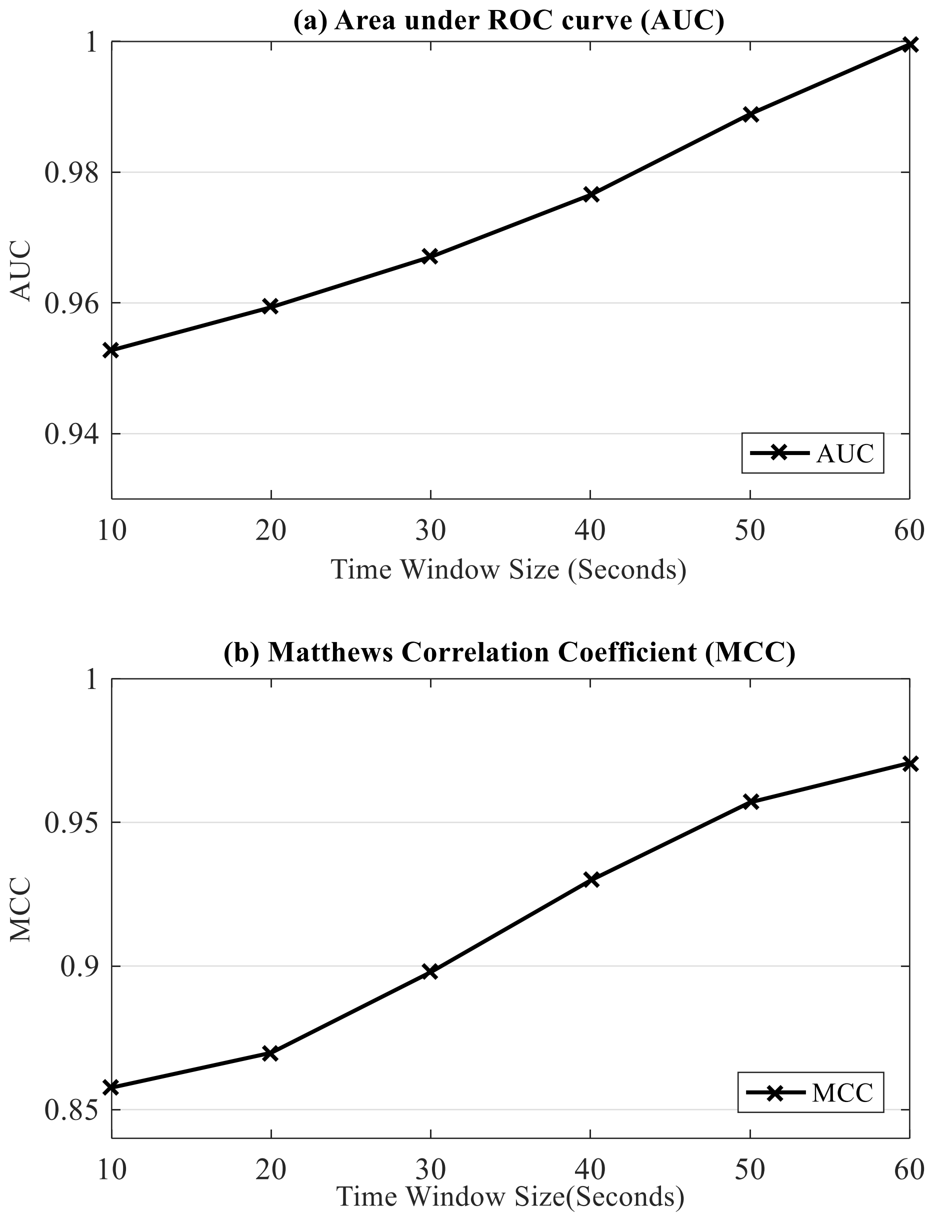

Figure 5.10 (a) AUC rates, (b) MCC rates.

In addition, the quality of outcomes of the proposed method based on time window size was compared using the RMSE and NDEI measures, and the 60-seconds time window achieved the best RMSE and NDEI rate around 0.093 and 0.187 respectively as shown in

Figure 5.11. Furthermore, the lowest FPR and FNR were given with a 60-seconds time window size as shown in Figure 5.12.

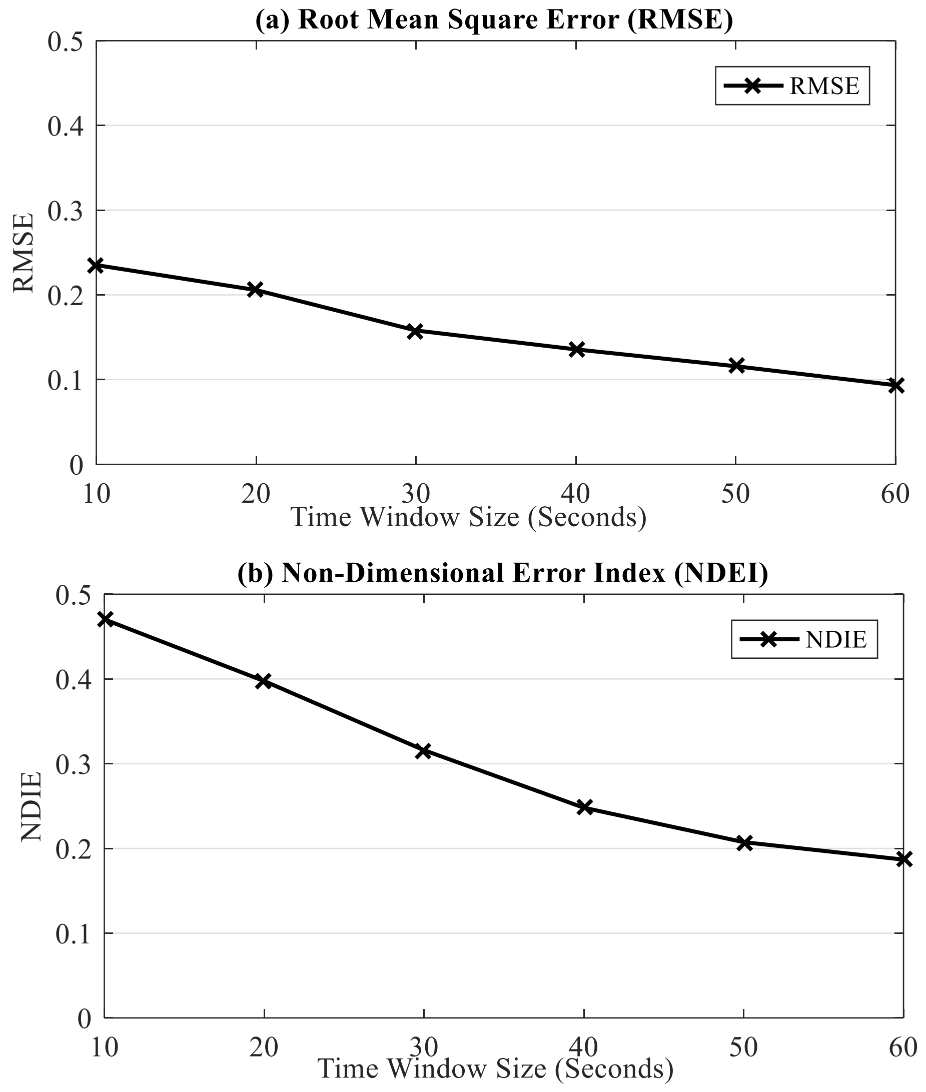

Figure 5.11 (a) RMSE rates, (b) NDEI rates.

The proposed approach achieves the best performance results at the 60-seconds time-window. Consequently, this size of time window is sufficient to collect Bots malicious behaviour and, therefore, achieve the best classification outcomes. In addition, to test the efficiency of the proposed approach in detecting P2P Bots, the ROC curve was plotted to show the trade-off between TPR and FPR. A perfect classifier would have an area under the curve (AUC) close to 1.0, where the x-axis represents FPR and the y-axis represents TPR.

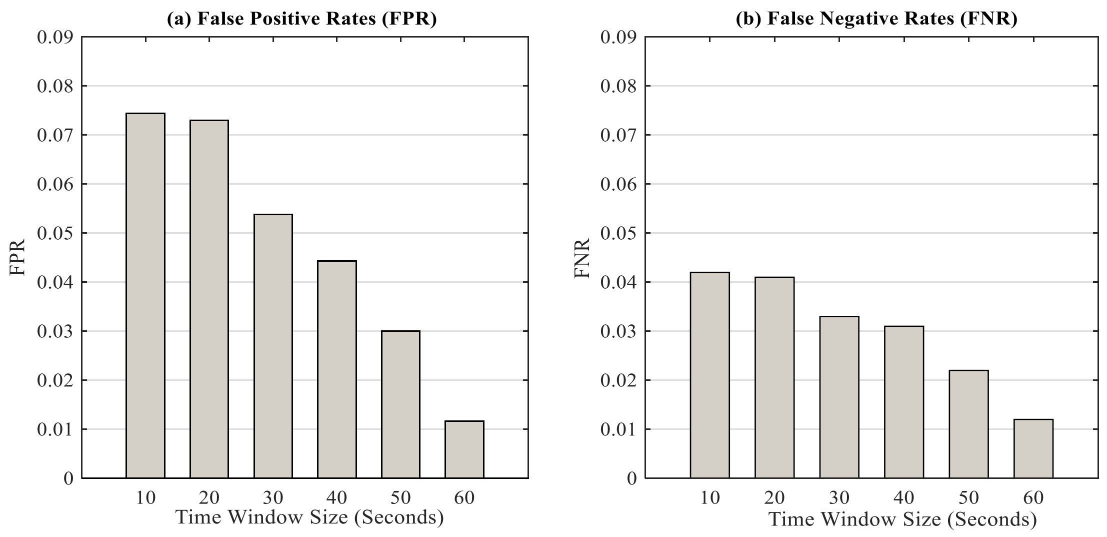

Figure 5.12 (a) FPR rates. (b) FNR rates.

Figure 5.13, plots the ROC for three time-windows 10, 30 and 60 as a sample in order to compare the performance of proposed approach in different time-window size. As shown in Figure 5.13 the 60-seconds time-window obtained the best performance in the AUC for both Bot and legitimate detection rates around 0.9916 and 0.9896 respectively. Therefore, it is found that the proposed approach performs well in classifying host inside the network traffic as a Bot or legitimate hosts.

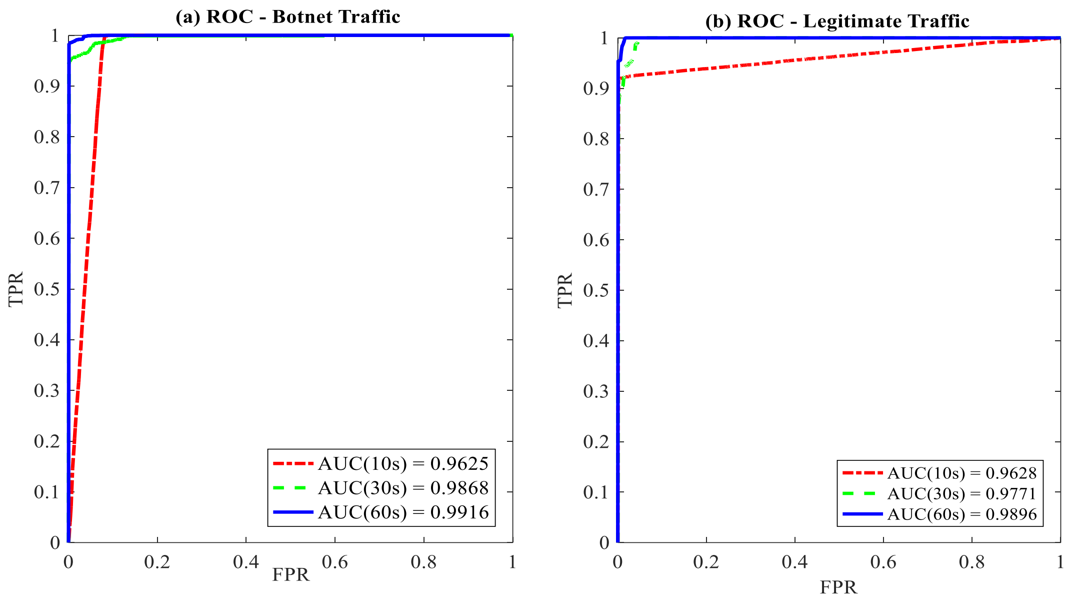

Figure 5.13 ROC comparison.

5.5.2 Testing on Zero-day Attack

To further evaluate the effectiveness of the proposed method with new P2P Botnet network traffic, Zeus, Waledac and Storm Bots were used to test the system for detection of zero-day attacks. It is evident from the results presented in Figure 5.14 that the technique adopted by our framework was powerful in detecting new P2P Bots types, with good accuracy rates. As shown in Figure 5.14, the detection rates for the Storm Bot and Waledac Bot using 60-seconds time-window were those higher than for the Zeus Bot 96.83%, 98.2% and 93.8% respectively. This was because for the testing and training dataset we used Storm and Waledac Bots, but from different dataset sources as shown in Table 3.1.

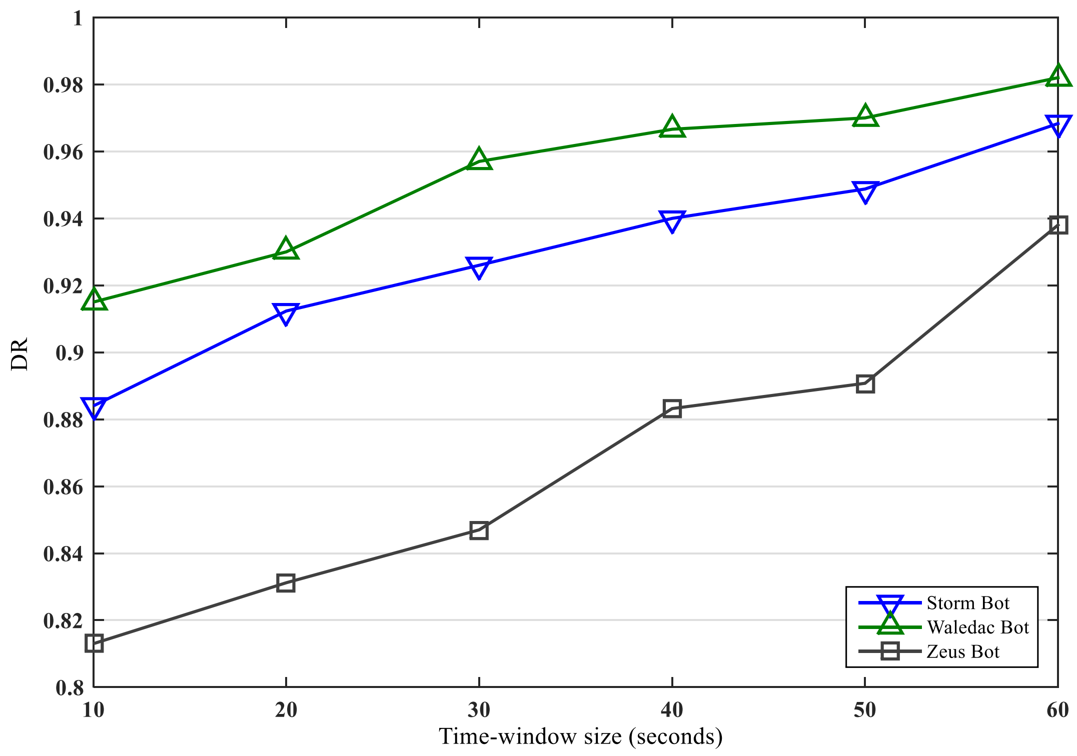

Figure 5.14 Detection rate (zero-day attack).

From Figure 5.15, a significant observation is that the approach gives low FPR for the Zeus, Storm and Waledac Bots at around 0.04, 0.07and 0.09 respectively using 60-seconds time-window. What is interesting in this result is that the proposed system is able to extract new features from the environment online and to utilize these features to enhance the system's on detection of novel types of Bot behaviour. The results of this experiment confirm that the proposed RL agent with proposed feature set can detect Bot even if it is a Zero-day attack.

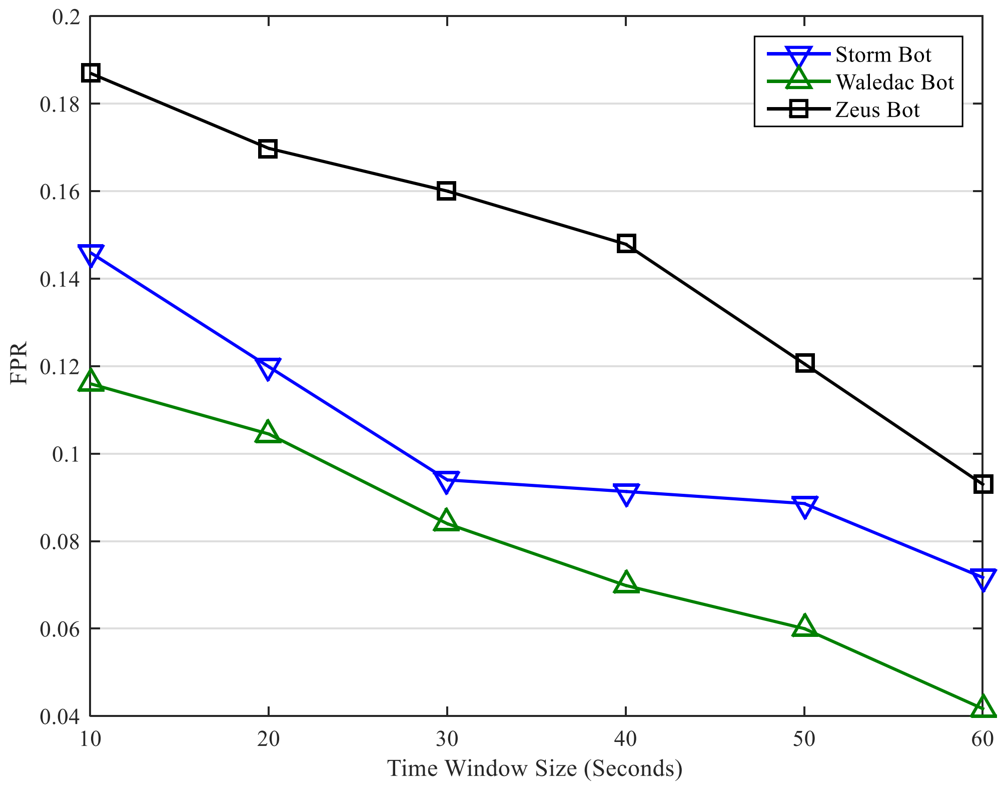

Figure 5.15 FPR (Zero-day attack).

5.5.3 Reinforcement Learning Model Evaluations

5.5.3.1 Evaluation the Efficiency of the Proposed Approach Based on Reference Dataset

The evaluation results for training the neural network agent in the online phase based on the reference dataset (train dataset) are demonstrated in Figure 5.16 to Figure 5.19. The x-axis represents the training index of neural network agent. It can be clearly seen that different performance measurements on the result from different time-window sizes. Based on these, results for 10s, 30s, and 60s time-windows are calculated. Therefore, the evaluations of the online agent with reference dataset give the highest average accuracy rate is 99.20% with standard deviation of 0.004 based on a 60-seconds time-window; meanwhile, the lowest average accuracy achieved with a 10-seconds time window was 95.92% and standard deviation 0.0143 as shown in Figure 5.16.

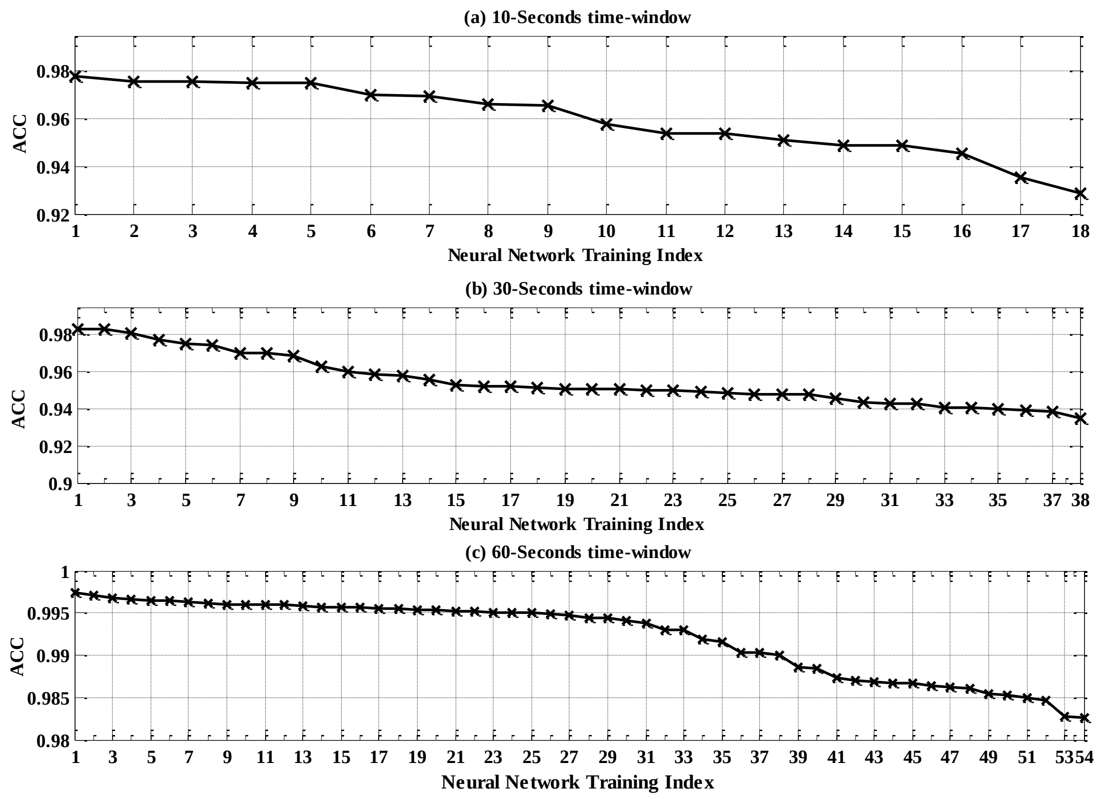

Figure 5.16 Online evaluations the ACC of a classifier based on a reference dataset.

Subsequently, the performances of the proposed approach according to time window size and reference dataset were compared based on AUC, and the 60-second time window achieved the best average AUC rates of 98.37% and the standard deviation around 0.0067 as shown in Figure 5.17. In addition, the lowest performance results given with a 10-second time-window.

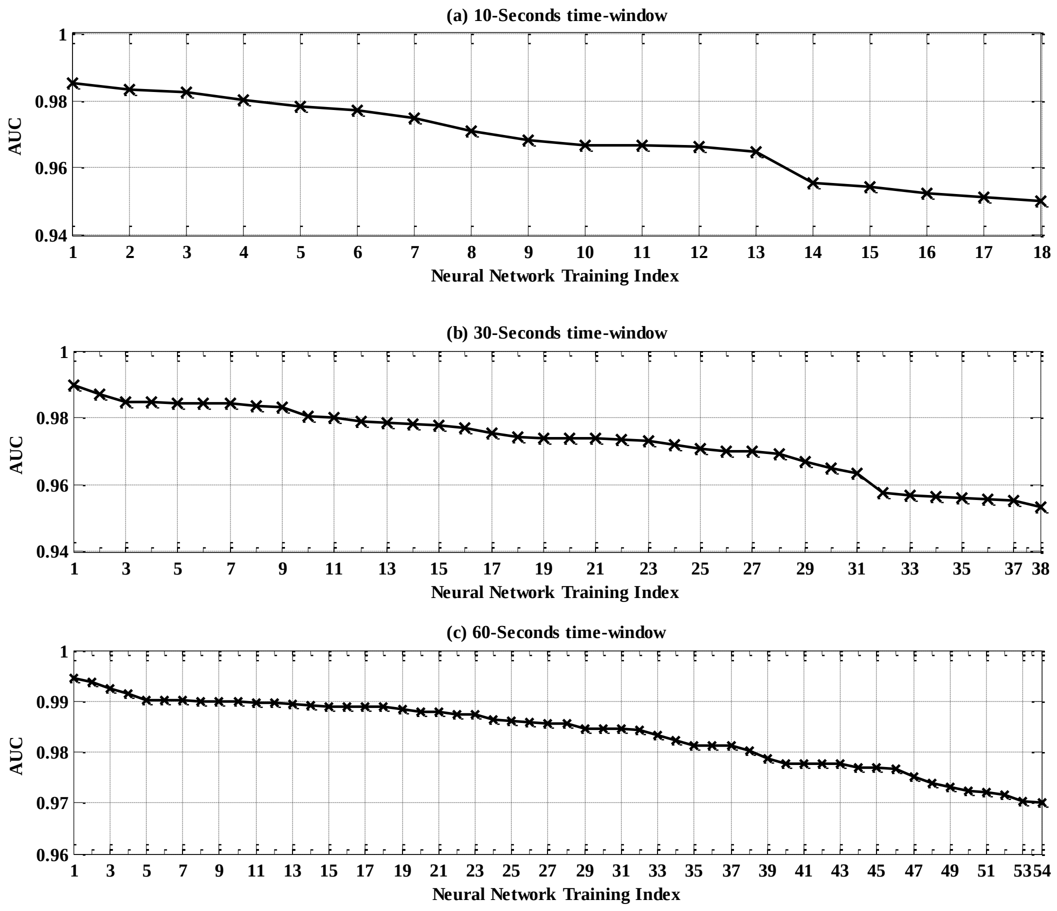

Figure 5.17 Online evaluations the AUC of a classifier based on a reference dataset.

In addition, Figure 5.18 compares the evaluation of MCC result, and the 60-second time-window achieves the best average rates around 94.9% with the standard deviation around 0.0224. AUC and MCC are considered the most reliable performance measures for imbalanced datasets.

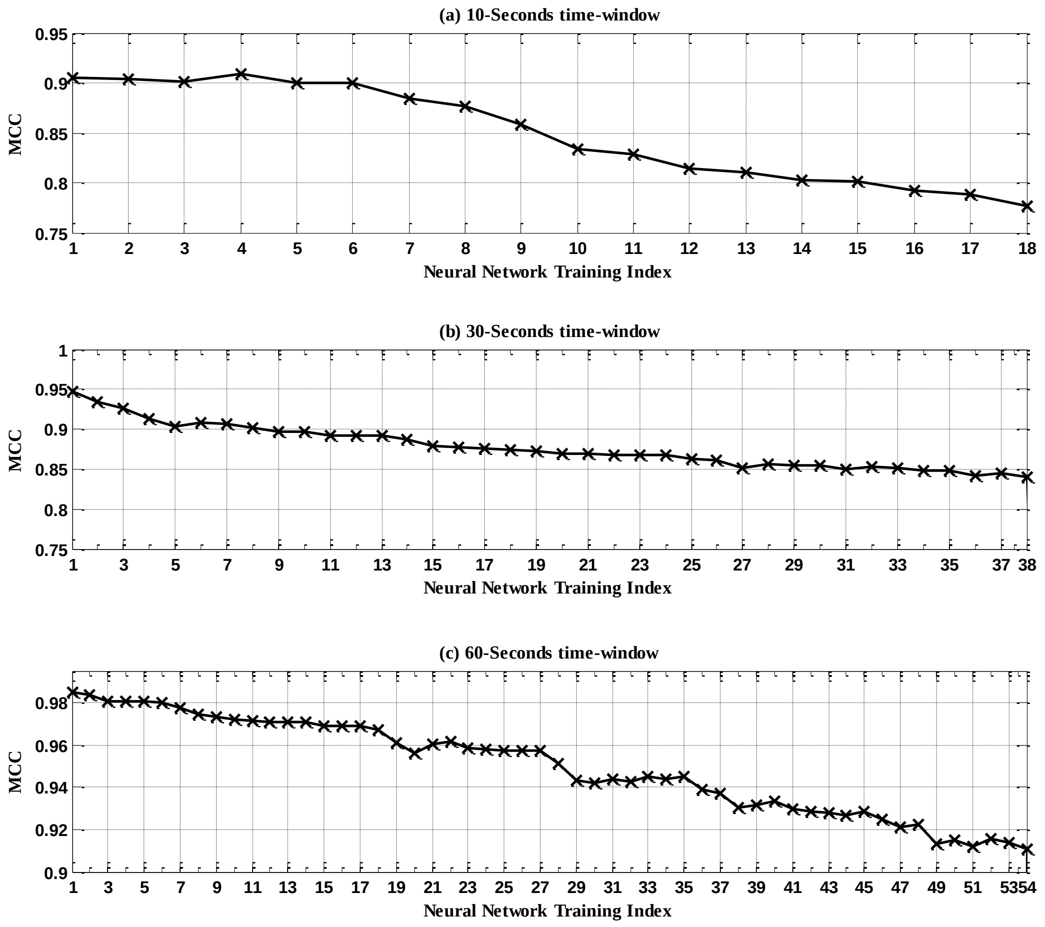

Figure 5.18 Online evaluations the MCC of a classifier based on a reference dataset.

In addition, the performances of the proposed approach according to time window size and reference dataset were compared based on the average RMSE, and the 60-second time window achieved the best average RMSE rates at around 0.044 with standard deviation around 0.0365 as shown in Figure 5.19. In addition, the lowest rate given with a 10-second time-window with average and standard deviation 0.14 and 0.291 respectively.

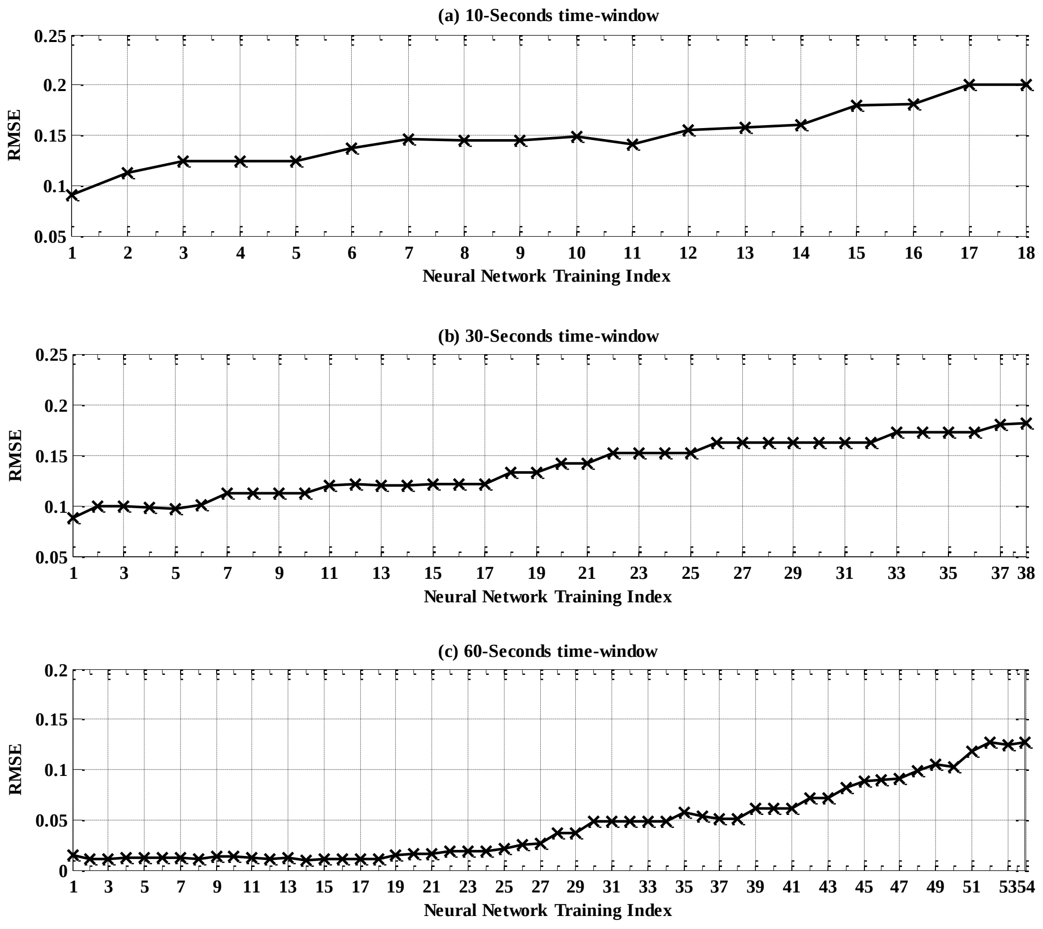

Figure 5.19 Online evaluations the RMSE of a classifier based on a reference dataset.

As shown in Figure 5.16, Figure 5.17 and Figure 5.18, the performance of the ACC, MCC, and AUC starts with high rates over the neural network training index, and then slightly decreases due to the probability of learning misclassified behaviours by the agent of malicious activity detector. Moreover, for the possibility of misclassified behaviours the RMSE increases over the neural network training index as shown in Figure 5.19.

Furthermore, the RL agent achieves the best performance over the 60-seconds time-window using reference dataset. By then, the 60-seconds time-window is sufficient to collect Bots malicious behaviours and it has low numbers of misclassified activities. In addition, the evaluation experiment results for time window 20, 40 and 50 seconds are demonstrated in Appendix A.

5.5.3.2 Evaluation the Efficiency of the Proposed Approach based on Updated Dataset

This section gives an overview of the evaluation results of the neural network agent in Bot detection using the updated dataset with new behaviours. The results obtained from the analysis of the online experiment outcomes are summarized in Figure 5.20 to Figure 5.23. As shown in Figure 5.20, the evaluation of the proposed approach using updated dataset online evaluation gives the best average of accuracy 98.26 with standard deviation around 0.0032 using the 60-seconds time window; meanwhile, the lowest evaluation of accuracy was achieved with a 10-seconds time-window with average 94.48% and standard deviation around 0.0126.

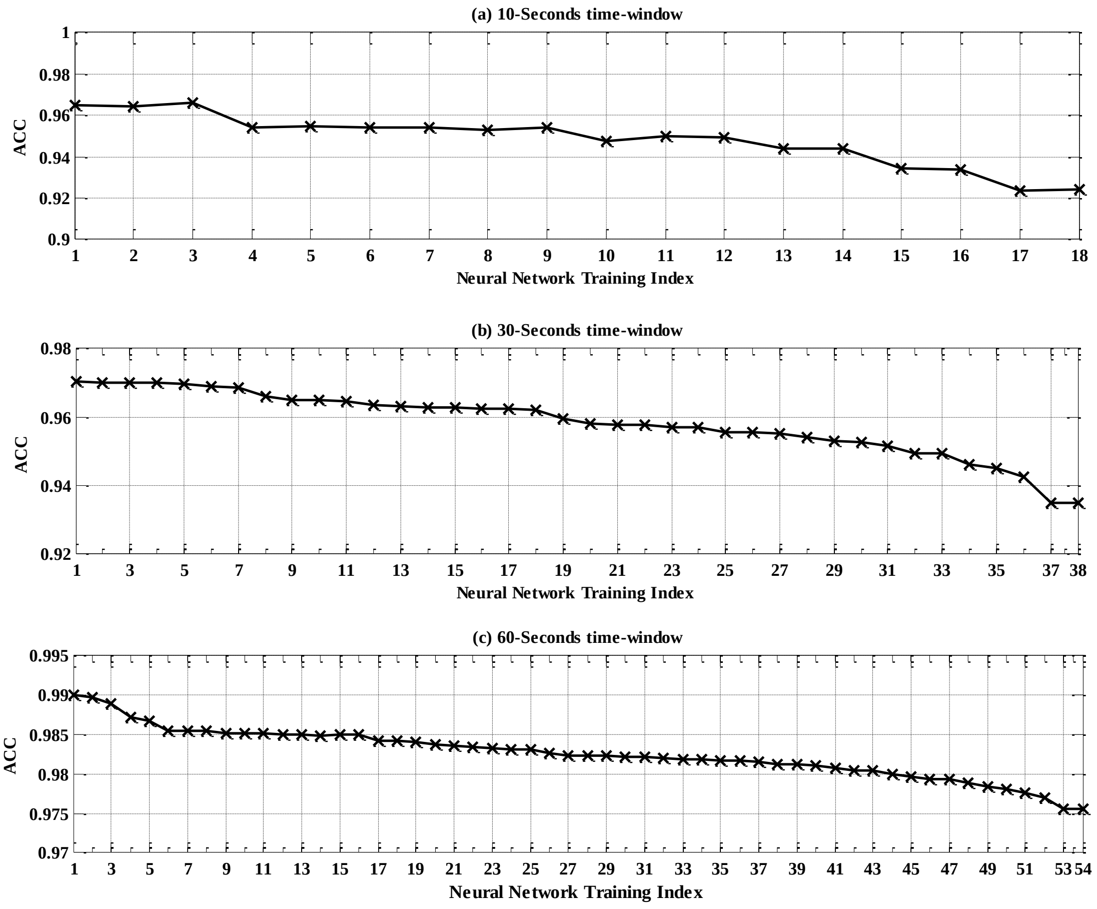

Figure 5.20 Online evaluations the ACC of a classifier based on an updated dataset.

Figure 5.21 and Figure 5.22 present the performance measurement AUC and MCC for the updated dataset. The results show that the highest average AUC and MCC rates were above 98.55% and above 95.22% respectively in the online testing evaluation using a 60-seconds time-window.

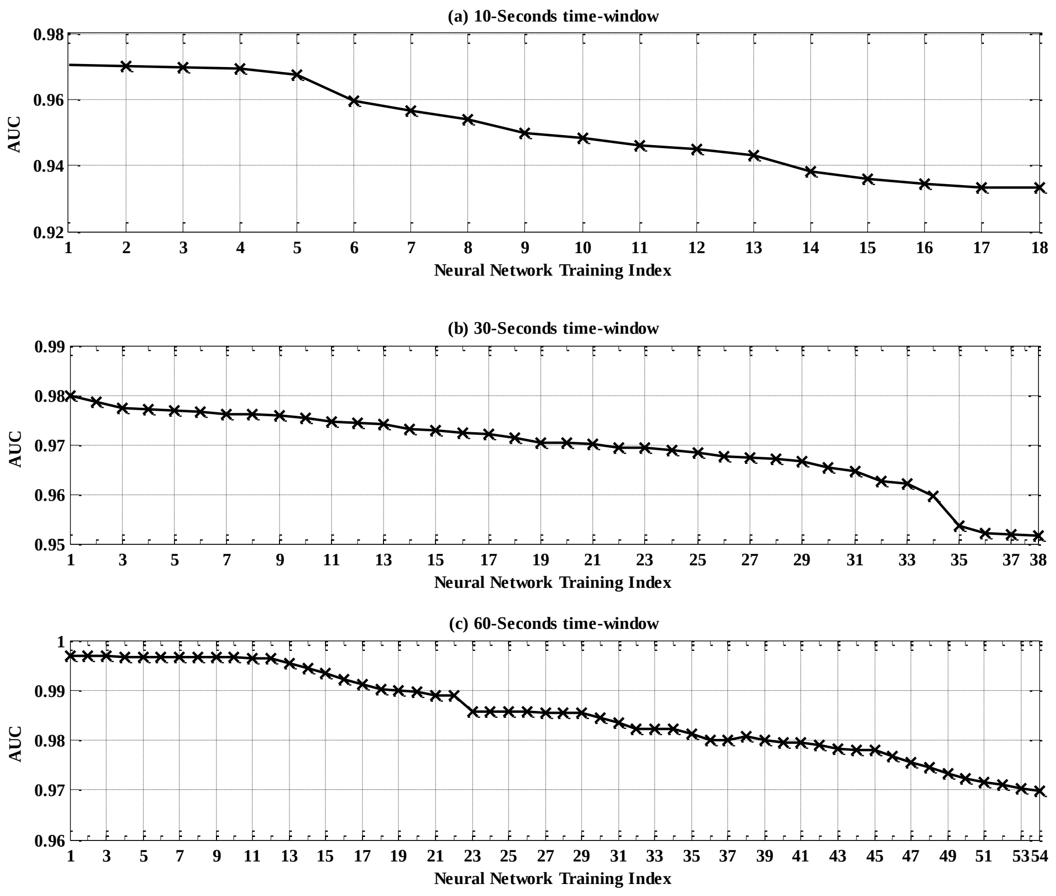

Figure 5.21 Online evaluations the AUC of a classifier based on an updated dataset.

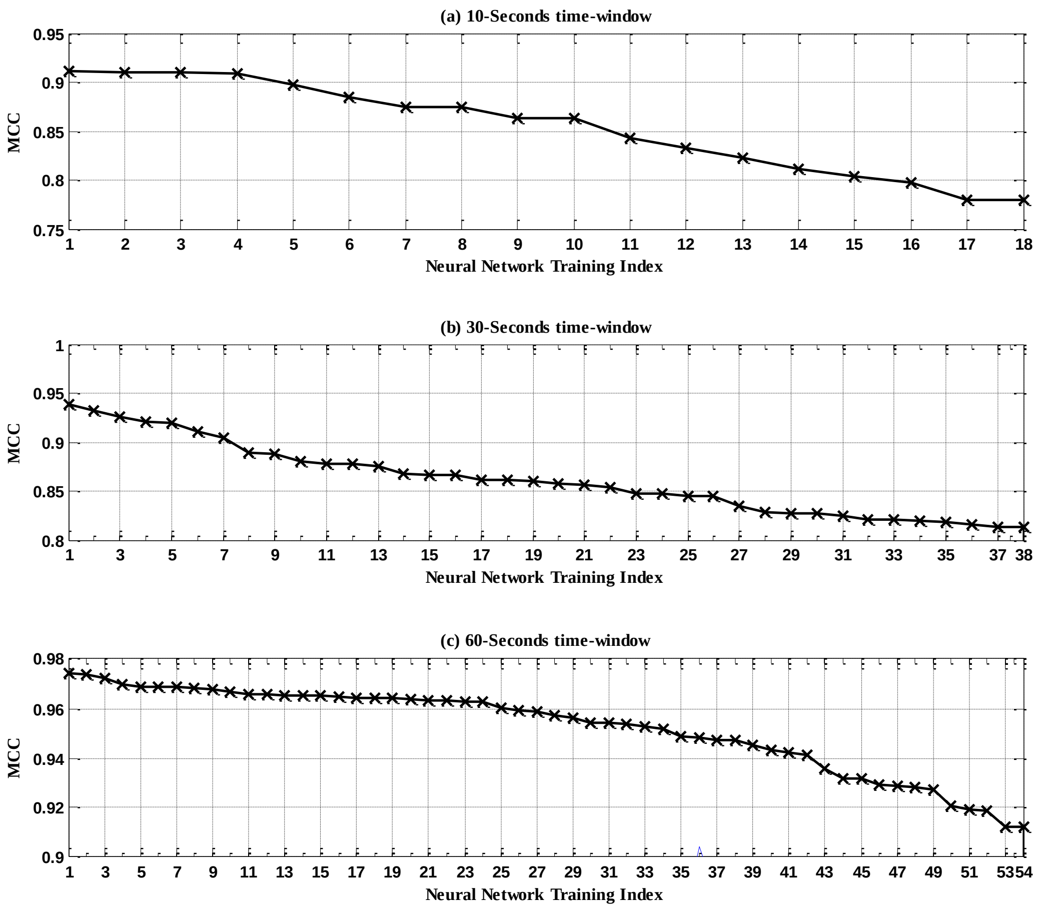

Figure 5.22 Online evaluations the MCC of a classifier based on an updated dataset.

The quality of outcomes of the proposed method based on time window size and the updated dataset is compared using the RMSE measure, and the 60-seconds time-window achieved the best average RMSE around 0.1255 and standard deviation of 0.0251 as shown in Figure 5.23. Furthermore, the lowest rate was given with a 10-seconds time-window size with average of 0.1729

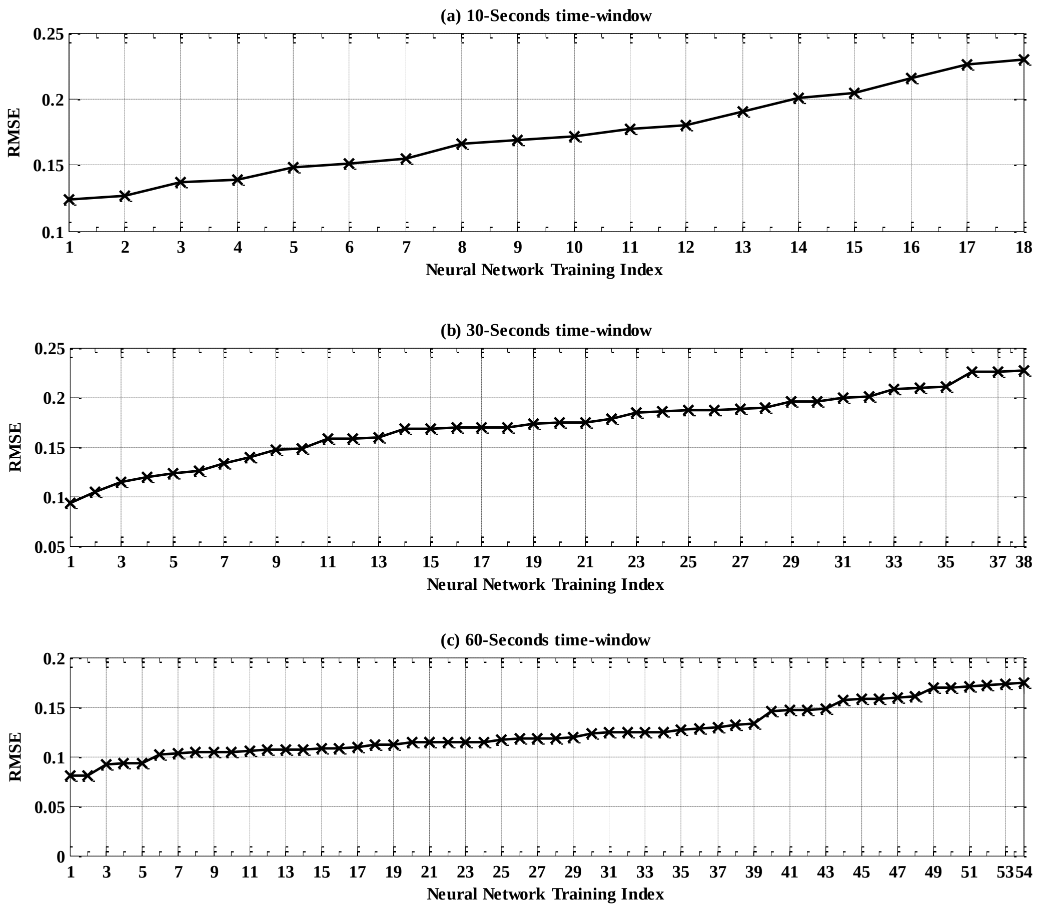

Figure 5.23 Online evaluations the RMSE of a classifier based on an updated dataset.

As shown in Figure 5.20, Figure 5.21 and Figure 5.22, the performance of the ACC, MCC, and AUC starts with high rates over the neural network training index, and then slightly decreases due to the probability of learning misclassified behaviours by the agent of malicious activity detector. Moreover, for the possibility of misclassified behaviours the RMSE increases over the neural network training index as shown in Figure 5.23. Moreover, the RL agent achieves the best performance over the 60-seconds time-window using updated dataset. By then, the 60-seconds time-window is sufficient to collect Bots malicious behaviours and it has low numbers of misclassified activities. In addition, the evaluation experiment results for time window 20, 40 and 50 seconds are demonstrated in Appendix B.

5.5.3.3 Evaluation Based on the Learning Rate

This section gives an overview of the evaluation results of the RL agent in extraction new behaviour using the test dataset. The results gathered from the analysis of the online experiment outcomes are summarized in Figure 5.24 to Figure 5.26. The x-axis represents the training index of the neural network agent and the y-axis represents the accumulative value of new Bot behaviour. It can be clearly seen that different numbers of the accumulated new Bot behaviours are extracted from different time-window sizes. Based on these, results for 10s, 30s, and 60s time-windows are calculated. Therefore, the evaluations of online agent to find new behaviour gives the highest accumulative number of new Bot behaviours is 4902 based on a 60-seconds time-window as shown in Figure 5.26; meanwhile the worst case to find new Bot behaviour with a 10-seconds time window is 934 as shown in Figure 5.25

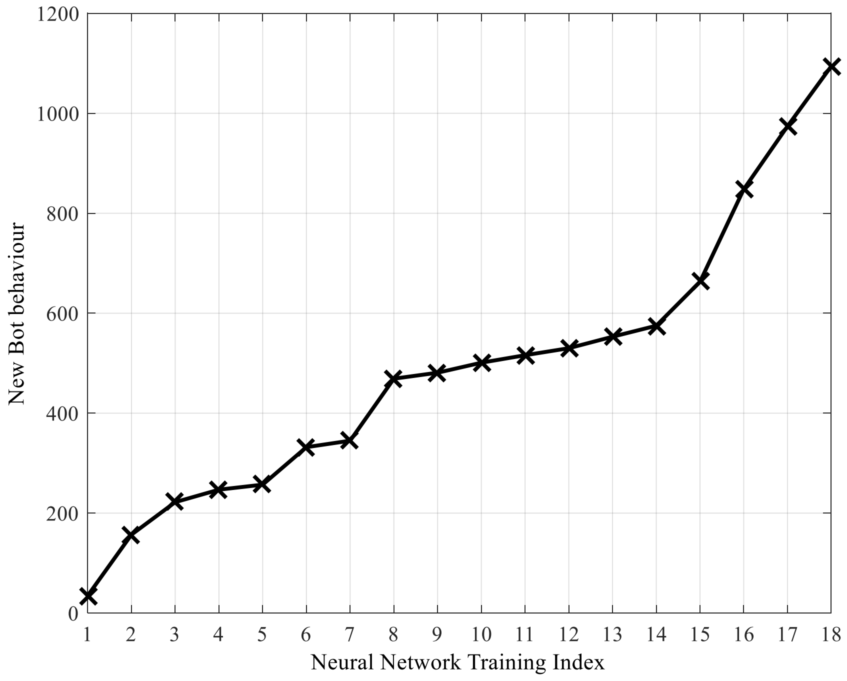

Figure 5.24 Evaluations of the extract new behaviours based on 10s time-window.

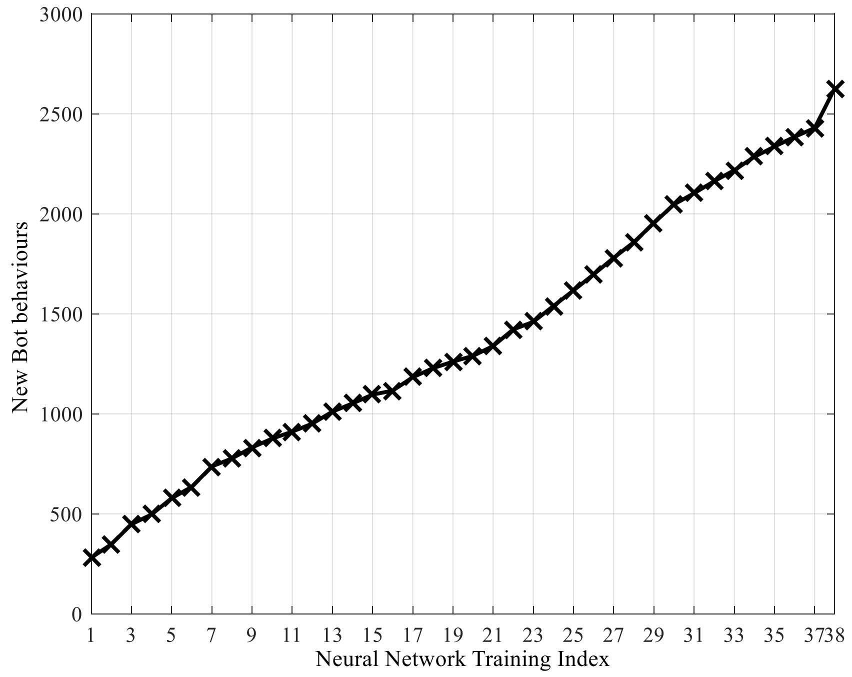

Figure 5.25 Evaluations of the extract new behaviours based on 30s time-window.

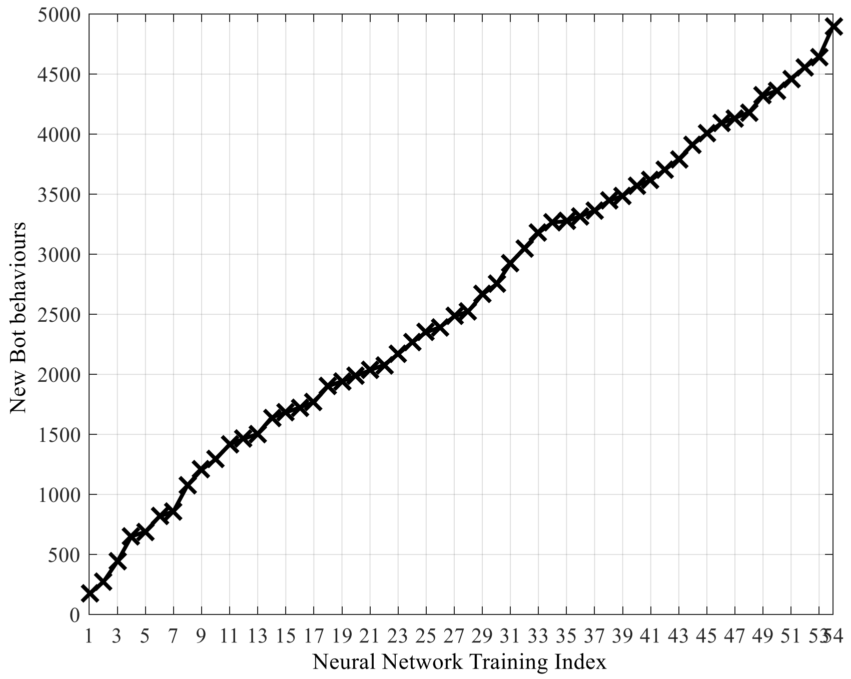

Figure 5.26 Evaluations of the extract new behaviours based on 60s time-window.

From the empirical results of the evaluation to extract new behaviour (learning rate) as shown in Figure 5.26 the 60-seconds time-window has the best result to find new Bot behaviour that improve the neural network agent online. In addition, as shown in above figures the RL agent has the maximum training index over the 60-seconds time-window due to this size of time window sufficient to collect Bots malicious behaviour and, therefore, discover the highest number of new Bot behaviours which improve the RL agent to detect a Zero-day attack as discussed in section 5.5.2.

Figure 5.27 gives an overview of the proposed technique results in the Bot detection based test dataset and the reference neural network agent (reference neural network: is the first neural network which is trained based on the offline dataset). The overall results obtained from the analysis of an online experiment outcome are summarized in Figure 5.27. As shown in Figure 5.27(a), Figure 5.27(b) and Figure 5.27(c), the evaluation of performance reference neural network using test data set gives the highest accuracy, detection and F-measure rate around 75%, 73% and 70%, respectively using 60-second time-window; meanwhile the lowest accuracy was achieved on 10-second time-window around 68%. Figure 5.27(e) and Figure 5.27(f) presents the imbalance dataset performance measurement AUC and MCC. The results show that the highest AUC and MCC rates were 90% and 85% respectively on testing dataset evaluation through 60-second time-window. In addition, the quality of outcomes of the proposed method based on time window size is compared using the RMSE and NDEI, the 60-second time-window achieves the best RMSE and NDEI rate around 0.28 and 0.30 respectively as shown in Figure 5.27(g) and Figure 5.27(h). Furthermore, the lowest FPR was given on 60-second time-window size.

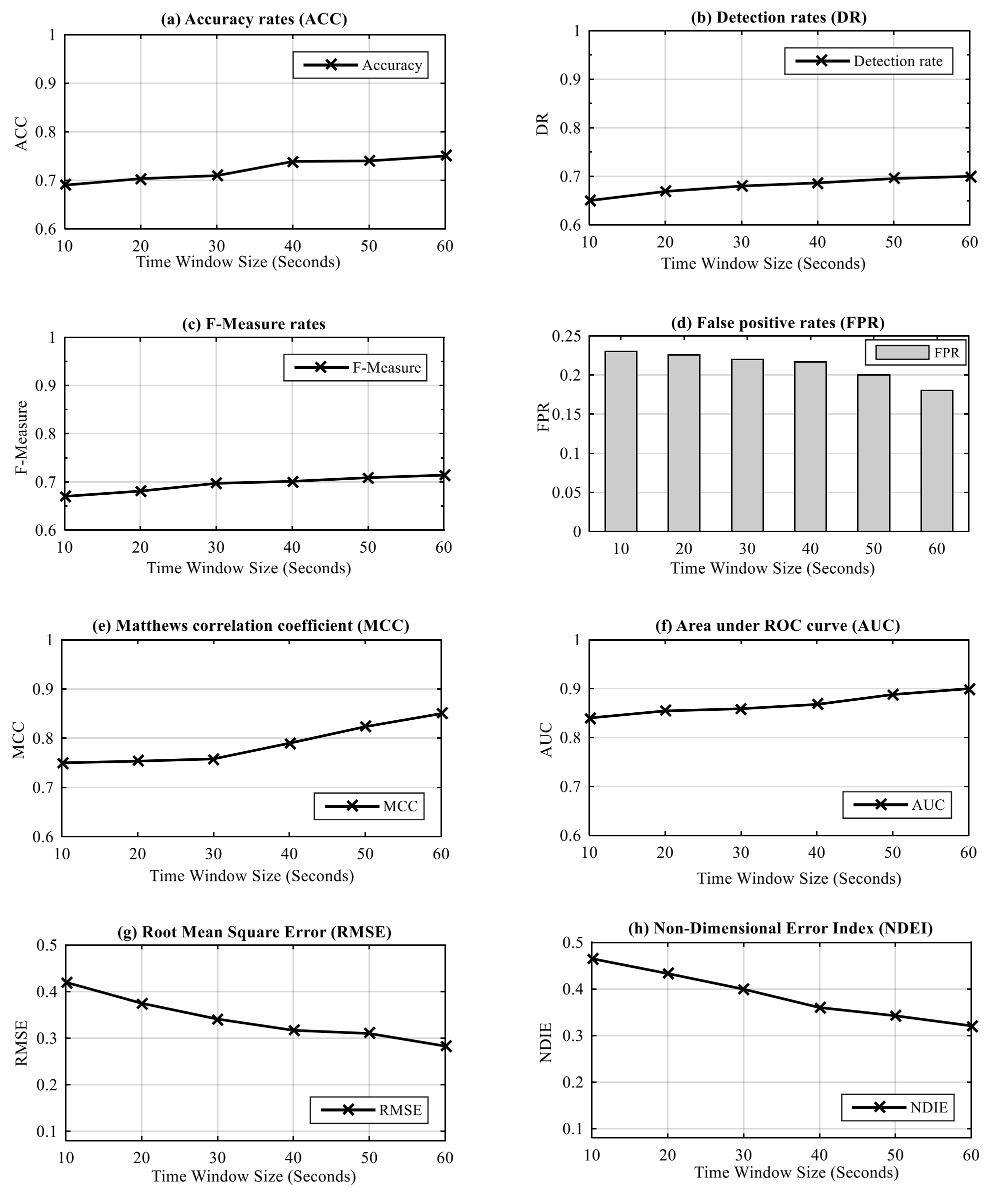

Figure 5.27 Test the online system based reference neural network.

As shown in above Figure the reference neural network with testing dataset has a bad performance outcome. However, the primary aim of the evaluation using reference neural network is to prove that the proposed approach with RL agent can detect P2P bots and able to learn new behaviour in order to improve the detection system over time. More details for the experiment parameters are provided in Appendix C.

5.6 Discussion

There are an enormous variety of RL problems depending on the respective environments and objectives. In practice, any solution for a dynamic problem using RL, such as Botnet detection, must be able to deal with the partial observability of the problem and the dimensionality of the data as well as the size of training data, as shown in Figure 5.28

Data with high dimensionality rates normally come from huge volumes of input into a system, which may affect the solution of the problems in the dynamic environments. For that reason, RL methods which can deal with high dimensionality and that are easily scalable are required. All table-based methods are excluded, because these methods are inefficient in problems with huge dimensionality. Instead an efficient and accurate function approximation is required.

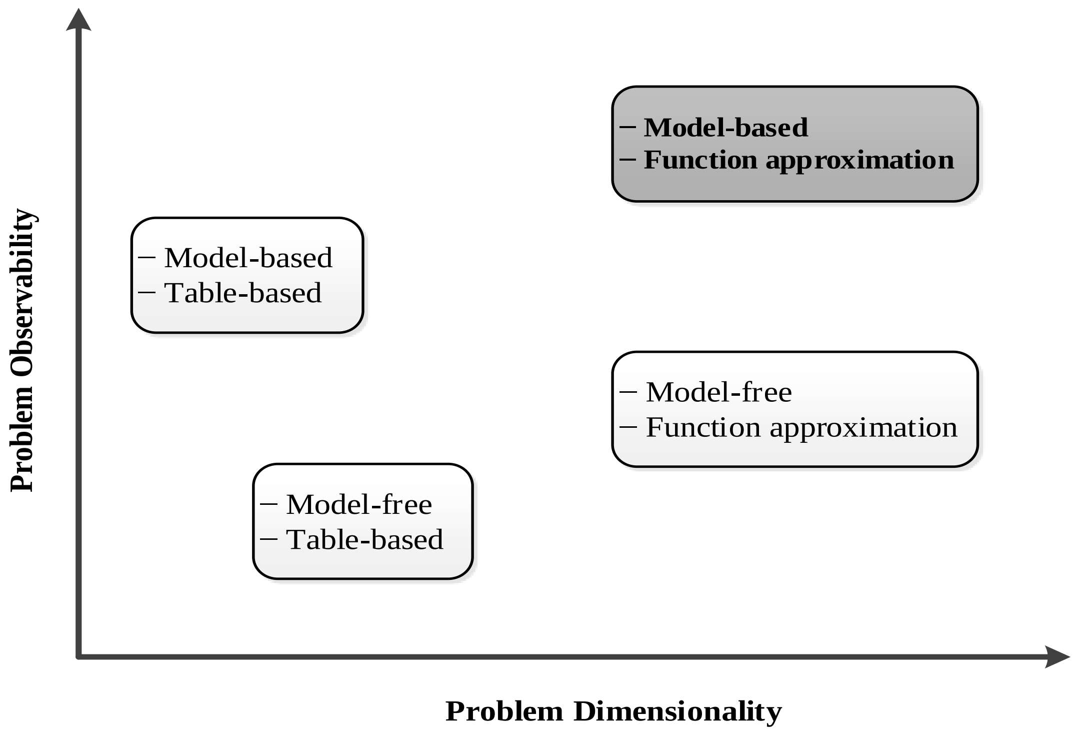

Figure 5.28 RL Characteristic.

In dynamic real-world environments, the situation is only partially observable since it is either an inaccessible environment or too costly to inspect all states and actions. In such situations, model-based strategies are beneficial as they first build the system's dynamics using current knowledge and utilize the current state to predict unobserved states. However, the quality of the first model is crucial.

In the present research, a model-based learning approach utilizes information collected through the training (offline phase) very effectively. Since the agent tries to learn a model of the states of the environment, thus it can combine the knowledge from multiple experiences. Besides the function approximation is applied overcome the storage problem with table-based approaches and to achieve data efficiency by generalizing to about unseen states.

However, The Botnet detection method chosen should satisfy the requirement of novelty detection, adaptability and early detection. Based on these measures, a neural network with a resilient back-propagation learning algorithm is adopted as a classification technique. This has robust capabilities for nonlinear system identification and control due to an inherent ability to approximate arbitrary nonlinear problems. Moreover, using the resilient back-propagation learning algorithm minimizes the harmful effects of volumes of fractional derivatives, and it increases the adoption rate. In addition, utilizing the RL approach improves the capability of the proposed system to detect a zero-day attack.

Table 5.1 shows the results of the comparison of our results with those of research using the same dataset in the offline phase that used by Zhao et al. (Zhao et al., 2013) and in online phase evaluation we used the same dataset as used by Babak et al. (Babak et al., 2014). The table also shows that the Bots detection and FPR using the proposed approach are better than those gained by previous solutions. Moreover, the proposed system is an online technique. Additionally, our approach differs from previous ones because the analysis is not performed on all network traffic such the studies (Babak et al., 2014; Zhao et al., 2013), which they analysis the whole network traffic to detect Bot malicious behaviours.

Table 5.1 Comparison with other published approaches.

Approaches		**FPR**	**Detection rate**	**Traffic reduction rate**
Babak et al. (2014)		0.1%	99.5%	0%
Zhao et al. (2013)		2.1%	98.1%	0%
Proposed approach	**Online**	0.012%	98.30%	40% -70%
	Offline	0.01%	99.1%	

5.7 Summary

In this chapter, a novel combination of neural networks and reinforcement learning were introduced in the design of an efficient Bot detection method. Practicality in solving highdimensional and partially observable RL problems in dynamic environments requires a model-based approach to identify Botnet malicious activity on the network. In addition, a technique was developed to achieve a balance between exploration and exploitation for the RL agent. In practice, a dynamic controller is constructed that obtains an optimal dynamic control policy under a RL framework. For the controller and its learning method, neural networks were used. The experiments with real Bot network traffic samples show that the controller succeeds in learning the optimal policy for a task of Bot detection.

6 CONCLUSIONS AND FUTURE WORK

This chapter presents the thesis conclusions and summarizes its unique contributions along with suggesting directions for future work. The main conclusion of the research is presented in Section 6.1. A summary of contributions is given in Section 6.2. Section 6.3 presents difficulties and solutions, and the limitations of the proposed approach are outlined in Section 6.4. Directions for future research are then indicated in Section 6.5.

6.1 Thesis Summary

Since the appearance of the Internet, network security has always been a primary interest of its users. Currently, Botnet detection is the most serious task in Internet security. Botnets can be utilized for many malicious activities such as DDoS, Spam and stealing sensitive information. Botnet detection, therefore, has assumed fundamental importance.

This thesis has presented our research on the Bot host detection using network traffic reduction with RL approach. In this thesis, Chapter 2 had reviewed the relevant background on Botnet phenomena and couple of related work on Botnet detection. Chapter 3 had presented our proposed contribution on the network traffic reduction. Chapter 4 had described our proposed connection-level feature set. Besides, the design of our offline Bot detection system, the experimental procedures, the results evaluation matrix and further discussions have been described in the chapter.

In Chapter 5, experiments are conducted to test the efficiency and effectiveness of the designed RL method. The RL algorithm for Bot detection is evaluated from different perspective using real world datasets. Furthermore, several methods of evaluation are used that cover balanced and imbalanced datasets. The results of the assessment show the efficiency of the proposed approach to deal with different types of Bot traffic and if the approach can detect zero-day attacks using the proposed RL algorithm.

6.2 Summary of Key Contributions

The significance of the proposed system lies in the following aspects. Firstly, it can detect P2P Bots even when their malicious activities are hidden, and without inspecting packet payloads. Secondly, the approach is capable of online detection based on a powerful mechanism for traffic reduction and short detection time-windows. Finally, a RL methodology used in the proposed approach increases the ability of the system to evolve based on the environment evolving.

Combining the use of network traffic reduction and RL approach gives our solution a valuable contribution to the field of Botnet detection. The experiments carried out involved testing and comparing with other research work based on the same datasets.

The first contribution is traffic reduction approach. The result revealed that using the traffic reduction approach achieved better reduction rate and had a considerable effect on online Bot detection. Besides, the traffic reduction approach improvises the efficiency of the proposed solution to work as an online Bot detection system and to deal with massive volumes of network traffic.

The second contribution is the connection-level feature set. To achieve earlier Bot detection and bypass the encrypted network traffic, connections-based detection mechanism was designed and implemented which utilizes the information in the header of TCP control packets. The evaluation result of the proposed connection-level feature set using offline model shows that our feature set achieved better accuracy and detection rate. Moreover, the performance of the proposed feature set is evaluated using offline Bot detection model and compared with existing detection methods, and achieves better results using the same dataset.

The third contribution is the online RL model. To achieve adaptability in the proposed approach, a new model-based RL algorithm was designed and implemented. The experimental results revealed that using RL approach with traffic reduction method achieved high accuracy and detection rates compared with existing results using the same dataset. In addition, the solution has shown the ability to learn rapidly new attack patterns online. This important benefit supports intrusion detection systems and enhances their ability to detect zero-day attacks without the need for continuously external updates.

6.3 Difficulties and Solutions

Despite all concerted efforts to reduce the influence of Botnets, improvements in Botnet evasion techniques are rapidly growing, which makes Botnet detection a very difficult task for the Internet security community (FBI, 2011; IBM, 2013; Plohmann et al., 2011). Botnets are becoming more complicated, employing a diversity of evasion methods such as protocol evasion techniques, rootkits, advanced executable packers and moving away from IRC to VOIP, HTTP, IPV6 and P2P protocols and networks. These evasion strategies enhance the survivability of Botnets and increase the rates of infection of new hosts.

There are three principal difficulties in the classification of host behaviour: Firstly, the network traffic is continuous, which indicates that it is persistent and features will change over time. Furthermore, Botnets dynamically change via Bot updates or altering their operation in various life cycle stages after receiving instructions from a Botmaster. These phenomena are termed concept drift and this is currently a serious issue for any detection method (Dries & Rückert, 2009). Therefore, the proposed framework adopts the idea of RL to improve the system dynamically over time. Secondly, there is always the risk of a new Botnet emerging on a network. It's spread may be stealthy, such as in zero-day attacks, and the behaviour of the host might seem like legitimate behaviour, and the difficulty to detect malicious activities if the classifier not trained for this behaviour previously. These cases generate a problem of novelty detection for detection models. Therefore, the proposed framework continually extracts new features to improve detection rates over time. Thirdly, evaluating the entire network traffic in real-time is a computationally expensive task due to the speed of network traffic. Therefore, the proposed approach uses a traffic reduction method, which helps to set up a more lightweight, and speedy online detection method.

6.4 Limitations

In general, the major challenge for detection Botnet using data mining techniques is obtaining the training dataset. The universality and precision of the classifier depend on the training data sets quality. A diversity and illustrative training dataset are hard to obtain and to create one due to the time consuming and resource. In addition, the majority of the available Botnet dataset is formed in academic experiment source due to the security and privacy issue, it is very difficult for the researcher to get a Botnet traces from other such as corporate networks.

6.5 Future Research Directions

Our online Bot detection approach with its proof-of-concept design and implementation could be able to address the real-time objective. However, further research has to investigate the challenging of real-time implementation. Nevertheless, machine learning methods could be applied to a real-time solution.

The following list contains summaries of several research topics that can be pursued in the near future as a continuation of the research work presented in this thesis:

1. Discovery of further Botnet features. Based on P2P Bot communications, P2P applications and an analysis of the literature, 16 host features were created. It is possible to add informative connection-based features by analysing Botnets traffic using the UDP protocol. These features may be valuable for increasing the performance of any future Botnet detection system.
2. Use different feature selection algorithms. Although the feature selection algorithms that used in this work helped to minimize the vectors dimensions of the feature set without greatly reducing the performance of the detection approach, other feature selection algorithms could perhaps be utilized to gain a better feature subset.
3. Botnet detection in new trends, platforms and infrastructures. Many Botnets that work on smartphones have been classified, adding another threat to personal information. Consequently, exploring the experience gained from identifying Botnets on network to reduce their effects in developing infrastructures will be worthy of investigation in the future.
4. New Types of Botnet Attacks. To reserve their Botnets, attackers always attempt to make Botnet C&C connections as hidden as possible. Therefore, new types of Botnets have begun to adopt the social networks as their new communication channel. Investigating how this communication protocol operates, and how Botnets utilize this channel, could be a new direction to continue the present work.
5. An interesting direction might be to combine the connection-level feature with the host-level feature sets and using ensemble parallel classifiers. Such a combination

could add to the analysis of Botnet traffic which might improve the accuracy of the Botnet detection approach.

6. Another area of interest might be to investigate the possibility of replacing the neural network in proposed strategy with other machine learning methods such as those employing unsupervised learning (for example, clustering algorithms).

REFERENCES

AbuHmed, T., Mohaisen, A., & Nyang, D. (2007). Deep packet inspection for intrusion detection systems: A survey. Magazine of Korea Telecommunication Society, 24, 25-36.

Al-Hammadi, Y., & Aickelin, U. (2010). Behavioural Correlation for Detecting P2P Bots. Paper presented at the Second International Conference on Future Networks (ICFN), Sanya, Hainan.

Al Shalabi, L., & Shaaban, Z. (2006). Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix. Paper presented at the International Conference on Dependability of Computer Systems, Washington, DC, USA.

Alder, R., Burke, J., Keefer, C., Orebaugh, A., Pesce, L., & Seagren, E. S. (2007). Chapter 4 - Introducing Snort. In R. Alder, J. Burke, C. Keefer, A. Orebaugh, L. Pesce & E. S. Seagren (Eds.), How to Cheat at Configuring Open Source Security Tools (pp. 181-212). Burlington: Syngress.

Alpaydin, & Ethem. (2014). Introduction to machine learning: MIT press.

Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., & Feamster, N. (2010). Building a Dynamic Reputation System for DNS. Paper presented at the USENIX security symposium.

Babak, R., Roberto, P., Andrea, L., & Kang, L. (2014). PeerRush: Mining for unwanted P2P traffic. Journal of Information Security and Applications, 19(3), 194-208. doi: http://dx.doi.org/10.1016/j.jisa.2014.03.002

Bacher, P., Holz, T., Kotter, M., & Wicherski, G. (2005). Know your enemy: Tracking botnets. Published on the Web: The Honeynet Project and Research Alliance.

Baecher, P., Koetter, M., Holz, T., Dornseif, M., & Freiling, F. (2006). The Nepenthes Platform: An Efficient Approach to Collect Malware. In D. Zamboni & C. Kruegel (Eds.), Recent Advances in Intrusion Detection (Vol. 4219, pp. 165-184): Springer Berlin Heidelberg.

Bailey, M., Cooke, E., Jahanian, F., Yunjing, X., & Karir, M. (2009). A Survey of Botnet Technology and Defenses. Paper presented at the Cybersecurity Applications & Technology Conference for Homeland Security, Washington, DC.

Balas, E., & Viecco, C. (2005). Towards a third generation data capture architecture for honeynets. Paper presented at the In Proceedings of the 2005 IEEE Workshop on Information Assurance and Security, NY, USA.

Barford, P., & Yegneswaran, V. (2007). An Inside Look at Botnets. In M. Christodorescu, S. Jha, D. Maughan, D. Song & C. Wang (Eds.), Malware Detection (Vol. 27, pp. 171-191): Springer US.

Barsamian, A. V. (2009). Network characterization for botnet detection using statistical-behavioral methods. (Ph.D. thesis), Thayer School of Engineering Dartmouth College, Hanover, New Hampshire.

Barto, & Andrew. (1998). Reinforcement learning: An introduction: MIT press.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. Systems, Man and Cybernetics, IEEE Transactions on, SMC-13(5), 834-846. doi: 10.1109/TSMC.1983.6313077

Batchelder, D., Blackbird, J., Felstead, D., Henry, P., Jones, J., & Kulkarni, A. (2014). Microsoft Security Intelligence Report: Microsoft.

Bertsekas, D. P., & Tsitsiklis, J. (1996). Neuro-Dynamic Programming (1st ed.): Athena Scientific.

Boger, Z., & Guterman, H. (1997). Knowledge extraction from artificial neural network models. Paper presented at the EEE International Conference on Systems, Man and Cybernetics, Orlando, FL, USA.

Boshmaf, Y., Muslukhov, I., Beznosov, K., & Ripeanu, M. (2013). Design and analysis of a socialBotnet. Computer Networks, 57(2), 556-578.

Brafman, R. I., & Tennenholtz, M. (2003). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. J. Mach. Learn. Res., 3, 213-231. doi: 10.1162/153244303765208377

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. Belmont, California: Wadsworth, Inc.

Chao, L., Wei, J., & Xin, Z. (2009). Botnet: Survey and Case Study. Paper presented at the Fourth International Conference on Innovative Computing, Information and Control (ICICIC), Kaohsiung.

Chen, C.-M., & Lin, H.-C. (2015). Detecting botnet by anomalous traffic. Journal of Information Security and Applications, 21, 42-51. doi: http://dx.doi.org/10.1016/j.jisa.2014.05.002

Chiang, K., & Lloyd, L. (2007). A case study of the rustock rootkit and spam bot. Paper presented at the The First Workshop in Understanding Botnets, Cambridge, MA.

Choi, H., & Lee, H. (2012). Identifying botnets by capturing group activities in DNS traffic. Computer Networks, 56(1), 20-33. doi: 10.1016/j.comnet.2011.07.018

Choi, H., Lee, H., & Kim, H. (2009). BotGAD: detecting botnets by capturing group activities in network traffic. Paper presented at the Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, Dublin, Ireland.

Cisco. (2012). Introduction to Cisco IOS NetFlow: CISCO White Paper.

Cooke, E., Jahanian, F., & McPherson, D. (2005). The zombie roundup: Understanding, detecting, and disrupting botnets. Paper presented at the Proceedings of the USENIX SRUTI Workshop.

Cover, T. M., & Thomas, J. A. (2012). Elements of information theory: John Wiley & Sons.

Cristianini, N., & Shawe-Taylor, J. (2000). An introduction to support vector machines and other kernel-based learning methods: Cambridge university press.

Crites, R., & Barto, A. (1998). Elevator Group Control Using Multiple Reinforcement Learning Agents. Machine Learning, 33(2-3), 235-262. doi: 10.1023/a:1007518724497

Dae-il, J., Kang-yu, C., Minsoo, K., Hyun-chul, J., & Bong-Nam, N. (2010). Evasion technique and detection of malicious botnet. Paper presented at the International Conference for Internet Technology and Secured Transactions (ICITST), London, UK.

Dagon, D. (2005). Botnet detection and response, the network is the infection. Paper presented at the OARC Workshop. http://www.caida.org/workshops/dns-oarc/200507/slides/oarc0507-Dagon.pdf

Dan, L., Yichao, L., Yue, H., & Zongwen, L. (2010). A P2P-Botnet detection model and algorithms based on network streams analysis. Paper presented at the International Conference on Future Information Technology and Management Engineering (FITME), Changzhou, China.

Daswani, N., & Stoppelman, M. (2007). The anatomy of Clickbot. A. Paper presented at the Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets.

Davis, C. R., Fernandez, J. M., & Neville, S. (2009). Optimising sybil attacks against P2P-based botnets. Paper presented at the the 4th International Conference on Malicious and Unwanted Software, Montreal, QC.

Demarest, J. (2014). Statement before the Senate Judiciary Committee, Subcommittee on Crime and Terrorism. Washington, D.C.: FBI Retrieved from https://www.fbi.gov/news/testimony/taking-down-botnets.

Dittrich, D., & Dietrich, S. (2008). Discovery techniques for P2P botnets Stevens Institute of Technology CS Technical Report 2008 (Vol. 4).

Dittrich, D., & Dietrich, S. (2008). P2P as botnet command and control: A deeper insight. Paper presented at the 3rd International Conference on Malicious and Unwanted Software (MALWARE), Univ. of Washington, Washington, DC, USA.

Drake, A. W. (1962). Observation of a Markov process through a noisy channel. Massachusetts Institute of Technology.

Dries, A., & Rückert, U. (2009). Adaptive concept drift detection. Statistical Analysis and Data Mining, 2(5-6), 311-327. doi: 10.1002/sam.10054

Dshield.org. (2013). Most attacked Port reports. Retrieved Aug, 2013, from http://www.dshield.org/portreport.html.

Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv., 44(2), 1-42. doi: 10.1145/2089125.2089126

Espinosa, J., & Vandewalle, J. (2000). Constructing fuzzy models with linguistic integrity from numerical data-AFRELI algorithm. IEEE Transactions on Fuzzy Systems, 8(5). doi: 10.1109/91.873582

Falliere, N., Murchu, L., & Chien, E. (2011). W32. Stuxnet Dossier.

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874. doi: 10.1016/j.patrec.2005.10.010

FBI. (2011). Another pleads guilty in botnet hacking conspiracy. FBI National Press Office.

Fedynyshyn, G., Chuah, M., & Tan, G. (2011). Detection and Classification of Different Botnet C&C Channels. In J. A. Calero, L. Yang, F. Mármol, L. García Villalba, A. Li & Y. Wang (Eds.), Autonomic and Trusted Computing (Vol. 6906, pp. 228-242): Springer Berlin Heidelberg.

Feily, M., Shahrestani, A., & Ramadass, S. (2009, 18-23 June 2009). A Survey of Botnet and Botnet Detection. Paper presented at the Third International Conference on Emerging Security Information, Systems and Technologies. SECURWARE '09.

Feinberg, E. A., Shwartz, A., & Altman, E. (2002). Handbook of Markov decision processes: methods and applications: Kluwer Academic Publishers Boston, MA.

Felix, J., Joseph, C., & Ghorbani, A. (2012). Group Behavior Metrics for P2P Botnet Detection. In T. Chim & T. Yuen (Eds.), Information and Communications Security (Vol. 7618, pp. 93-104): Springer Berlin Heidelberg.

Freiling, F., Holz, T., & Wicherski, G. (2005). Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In S. di Vimercati, P. Syverson & D. Gollmann (Eds.), Computer Security – ESORICS 2005 (Vol. 3679, pp. 319-335): Springer Berlin Heidelberg.

García, S., Zunino, A., & Campo, M. (2014). Survey on network-based botnet detection methods. Security and Communication Networks, 7(5), 878-903. doi: 10.1002/sec.800

Garg, S., Singh, A. K., Sarje, A. K., & Peddoju, S. K. (2013). Behaviour analysis of machine learning algorithms for detecting P2P botnets. Paper presented at the 15th International Conference on Advanced Computing Technologies (ICACT).

Gass, S., & Fu, M. (2013). Markov Property Encyclopedia of Operations Research and Management Science (pp. 941-942). USA: Springer

Goebel, J., & Holz, T. (2007). Rishi: identify bot contaminated hosts by IRC nickname evaluation. Paper presented at the Proceedings of USENIX HotBots Cambridge, MA.

Goerzen, J. (2004). Domain Name System Foundations of Python Network Programming (pp. 65-85): Apress.

Grizzard, J. B., Sharma, V., Nunnery, C., Kang, B. B., & Dagon, D. (2007). Peer-to-peer botnets: overview and case study. Paper presented at the Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets, Cambridge, MA.

Gu, G., Perdisci, R., Zhang, J., & Lee, W. (2008). BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. Paper presented at the USENIX Security Symposium.

Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. (2007). BotHunter: detecting malware infection through IDS-driven dialog correlation. Paper presented at the Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, Boston, MA. http://portal.acm.org/citation.cfm?id=1362915#

Gu, G., Zhang, J., & Lee, W. (2008). BotSniffer: Detecting botnet command and control channels in network traffic. Paper presented at the 15th Annual Network & Distributed System Security Symposium, San Diego.

Guofei, G., Yegneswaran, V., Porras, P., Stoll, J., & Wenke, L. (2009). Active Botnet Probing to Identify Obscure Command and Control Channels. Paper presented at the Annual Computer Security Applications Conference (ACSAC), Honolulu, HI.

Gurney, K. (1997). An introduction to neural networks. Bristol,USA: Taylor & Francis.

Gusella, R. (1991). Characterizing the variability of arrival processes with indexes of dispersion. IEEE Journal on Selected Areas in Communications, 9(2), 203-211. doi: 10.1109/49.68448

Han, K.-S., & Im, E. (2012). A Survey on P2P Botnet Detection. In K. J. Kim & S. J. Ahn (Eds.), Proceedings of the International Conference on IT Convergence and Security 2011 (Vol. 120, pp. 589-593): Springer Netherlands.

Han, K.-S., Lim, K.-H., & Im, E.-G. (2009). The Traffic Analysis of P2P-based Storm Botnet using Honeynet. Journal of the Korea Institute of Information Security and Cryptology, 19(4), 51-61.

Hannah, K., & Gianvecchio, S. (2015). Zeuslite: a tool for botnet analysis in the classroom. J. Comput. Sci. Coll., 30(3), 109-116.

Harmon, M. E., & Harmon, S. S. (1996). Reinforcement learning: a tutorial. WL/AAFC, WPAFB Ohio, 45433.

Hegna, A. (2010). Visualizing Spatial and Temporal Dynamics of a Class of IRC-Based Botnets. (PhD thesis), Norwegian University of Science and Technology. Retrieved from http://ntnu.diva-portal.org/smash/record.jsf?pid=diva2:353050

Holz, T., Steiner, M., Dahl, F., Biersack, E., & Freiling, F. C. (2008). Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. LEET, 8(1), 1-9.

Huy, H., Xuetao, W., Faloutsos, M., & Eliassi-Rad, T. (2013). Entelecheia: Detecting P2P botnets in their waiting stage. Paper presented at the IFIP Networking Conference, Brooklyn, NY.

Ianelli, N., & Hackworth, A. (2005). Botnets as a vehicle for online crime. CERT Coordination Center, 1(1), 28.

IBM. (2013). IBM X-Force 2012 Trend and Risk Report: IBM Security Systems.

Igel, C., Toussaint, M., & Weishui, W. (2005). Rprop Using the Natural Gradient. In D. H. Mache, J. Szabados & M. G. de Bruin (Eds.), Trends and Applications in Constructive Approximation (pp. 259-272). Basel: Birkhäuser Basel.

Iivari, J., Hirschheim, R., & Klein, H. K. (1998). A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. Information Systems Research, 9(2), 164-193. doi: doi:10.1287/isre.9.2.164

Jaber, M., Cascella, R. G., & Barakat, C. (2011). Can We Trust the Inter-Packet Time for Traffic Classification? Paper presented at the IEEE International Conference on Communications (ICC) Kyoto, Japan.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. ACM Comput. Surv., 31(3), 264-323. doi: 10.1145/331499.331504

Jian, K., & Jun-Yao, Z. (2009). Application Entropy Theory to Detect New Peer-to-Peer Botnet with Multi-chart CUSUM. Paper presented at the Second International Symposium on Electronic Commerce and Security (ISECS), Nanchang, China.

Jiang, H., & Shao, X. (2012). Detecting P2P botnets by discovering flow dependency in C&C traffic. Peer-to-Peer Networking and Applications, 1-12. doi: 10.1007/s12083-012-0150-x

John, J. P., Moshchuk, A., Gribble, S. D., & Krishnamurthy, A. (2009). Studying Spamming Botnets Using Botlab. Paper presented at the 6th USENIX symposium on Networked systems design and implementation (NSDI), Boston, Massachusetts.

Jun, L., Shunyi, Z., Yanqing, L., & Junrong, Y. (2008). Real-Time P2P Traffic Identification. Paper presented at the IEEE Global Telecommunications Conference, New Orleans, USA.

Junjie, Z., Perdisci, R., Wenke, L., Sarfraz, U., & Xiapu, L. (2011). Detecting stealthy P2P botnets using statistical traffic fingerprints. Paper presented at the IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), Hong Kong.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1–2), 99-134. doi: http://dx.doi.org/10.1016/S0004-3702(98)00023-X

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 237-285.

Kang, B. B., Chan-Tin, E., Lee, C. P., Tyra, J., Kang, H. J., Nunnery, C., . . . Yongdae Kim. (2009). Towards complete node enumeration in a peer-to-peer botnet. Paper presented at the Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, Sydney, Australia.

Karasaridis, A., Rexroad, B., & Hoeflin, D. (2007). Wide-scale botnet detection and characterization. Paper presented at the Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets.

Kreibich, C., & Crowcroft, J. (2004). Honeycomb: creating intrusion detection signatures using honeypots. SIGCOMM Comput. Commun. Rev., 34(1), 51-56. doi: 10.1145/972374.972384

Kristoff, J. (2005). Botnets, detection and mitigation: DNS-based techniques. NU Security Day, 23.

Lashkari, A., Ghalebandi, S., & Reza Moradhaseli, M. (2011). A Wide Survey on Botnet. In H. Cherifi, J. Zain & E. El-Qawasmeh (Eds.), Digital Information and Communication Technology and Its Applications (Vol. 166, pp. 445-454): Springer Berlin Heidelberg.

Li, H., Hu, G., & Yang, Y. (2012). Research on P2P Botnet Network Behaviors and Modeling. In C. Liu, L. Wang & A. Yang (Eds.), Information Computing and Applications (Vol. 307, pp. 82-89): Springer Berlin Heidelberg.

Limmer, T., & Dressler, F. (2009). Flow-based TCP connection analysis. Paper presented at the IEEE 28th International Performance Computing and Communications Conference (IPCCC), Scottsdale, AZ.

Liu, L., Chen, S., Yan, G., & Zhang, Z. (2008). BotTracer: Execution-Based Bot-Like Malware Detection. In T.-C. Wu, C.-L. Lei, V. Rijmen & D.-T. Lee (Eds.), Information Security (Vol. 5222, pp. 97-113): Springer Berlin Heidelberg.

Livadas, C., Walsh, R., Lapsley, D., & Strayer, W. T. (2006). Usilng Machine Learning Technliques to Identify Botnet Traffic. Paper presented at the Proceedings 2006 31st IEEE Conference on Local Computer Networks, Tampa, FL.

Lu, W., Rammidi, G., & Ghorbani, A. A. (2011). Clustering botnet communication traffic based on n-gram feature selection. Computer Communications, 34(3), 502-514. doi:10.1016/j.comcom.2010.04.007

Ludl, C., McAllister, S., Kirda, E., & Kruegel, C. (2007). On the Effectiveness of Techniques to Detect Phishing Sites. In B. Hämmerli & R. Sommer (Eds.), Detection of Intrusions and Malware, and Vulnerability Assessment (Vol. 4579, pp. 20-39): Springer Berlin Heidelberg.

Marpaung, J. A. P., Sain, M., & Hoon-Jae, L. (2012). Survey on malware evasion techniques: State of the art and challenges. Paper presented at the 14th International Conference on Advanced Communication Technology (ICACT), PyeongChang.

Masud, M. M., Al-khateeb, T., Khan, L., Thuraisingham, B., & Hamlen, K. W. (2008). Flow-based identification of botnet traffic by mining multiple log files. Paper presented at the First International Conference on Distributed Framework and Applications, Penang, Malaysia

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA) - Protein Structure, 405(2), 442-451. doi: http://dx.doi.org/10.1016/0005-2795(75)90109-9

Mille, A., Horne, R., & Potter, C. (2014). information security breaches survey (technical report). PriceWaterhouseCoopers.

Mitchell, T. M. (1997). Machine Learning: McGraw-Hill, Inc.

Mukamurenzi, N. M. (2008). Storm worm: A p2p botnet. (Master of Science in Communication Technology), Norwegian University of Science and Technology, Norway.

Mullaney, C. (2012). Android.Bmaster: A Million-Dollar Mobile Botnet. from http://www.symantec.com/connect/blogs/androidbmaster-million-dollar-mobile-botnet

Nagaraja, S., Houmansadr, A., Piyawongwisal, P., Singh, V., Agarwal, P., & Borisov, N. (2011). Stegobot: A Covert Social Network Botnet. In T. Filler, T. Pevný, S. Craver & A. Ker (Eds.), Information Hiding (Vol. 6958, pp. 299-313): Springer Berlin Heidelberg.

Nguyen, H., Petrović, S., & Franke, K. (2010). A Comparison of Feature-Selection Methods for Intrusion Detection. In I. Kotenko & V. Skormin (Eds.), Computer Network Security (Vol. 6258, pp. 242-255): Springer Berlin Heidelberg.

Nigrin, A. (1994). Book review: Neural Networks for Pattern Recognition (Vol. 5): MIT Press.

Nilsson, N. J. (1996). Introduction to machine learning. An early draft of a proposed textbook.

Nogueira, A., Salvador, P., & Blessa, F. (2010). A Botnet Detection System Based on Neural Networks. Paper presented at the Fifth International Conference on digital Telecommunications (ICDT), Athens, TBD, Greece.

Nummipuro, A. (2007). Detecting P2P-controlled bots on the host. Paper presented at the Seminar on Network Security, Espoo, Helsinki.

Pelleg, D., & Moore, A. W. (2000). X-means: Extending K-means with Efficient Estimation of the Number of Clusters. Paper presented at the Seventeenth International Conference on Machine Learning (ICML).

Perdisci, R., Guofei, G., & Wenke, L. (2006). Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems. Paper presented at the Sixth International Conference on Data Mining (ICDM), Hong Kong.

Petersen, M. N. (2014). Detecting network intrusions. (M.Sc.), Technical University of Denmark.

Pham, V.-H., & Dacier, M. (2011). Honeypot trace forensics: The observation viewpoint matters. Future Generation Computer Systems, 27(5), 539-546.

Plohmann, D., Gerhards-Padilla, E., & Leder, F. (2011). Botnets: Detection, measurement, disinfection & defence. The European Network and Information Security Agency (ENISA).

Provos, N. (2003). Honeyd-a virtual honeypot daemon. Paper presented at the 10th DFN-CERT Workshop, Hamburg, Germany.

Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming: John Wiley & Sons.

Rafique, M. Z., & Caballero, J. (2013). FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In S. J. Stolfo, A. Stavrou & C. V. Wright (Eds.), Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25,

2013. Proceedings (pp. 144-163). Berlin, Heidelberg: Springer Berlin Heidelberg.

Rajab, M. A., Zarfoss, J., Monrose, F., & Terzis, A. (2006). A multifaceted approach to understanding the botnet phenomenon. Paper presented at the Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, Rio de Janeriro, Brazil.

Ramachandran, A., & Feamster, N. (2006). Understanding the network-level behavior of spammers. SIGCOMM Comput. Commun. Rev., 36, 291-302. doi: 10.1145/1151659.1159947

Ramachandran, A., Feamster, N., & Dagon, D. (2006). Revealing botnet membership using DNSBL counter-intelligence. Paper presented at the Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet - Volume 2, San Jose, CA.

Razi, M. A., & Athappilly, K. (2005). A comparative predictive analysis of neural networks (NNs), nonlinear regression and classification and regression tree (CART) models. Expert Systems with Applications, 29(1), 65-74. doi: http://dx.doi.org/10.1016/j.eswa.2005.01.006

Rgio S. C. Silva, Rodrigo M. P. Silva, Raquel C. G. Pinto, & Ronaldo M. Salles. (2013). Botnets: A survey. Comput. Netw., 57(2), 378-403. doi: 10.1016/j.comnet.2012.07.021

Riedmiller, M. (2005). Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In J. Gama, R. Camacho, P. Brazdil, A. Jorge & L. Torgo (Eds.), Machine Learning: ECML 2005 (Vol. 3720, pp. 317-328): Springer Berlin Heidelberg.

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: the RPROP algorithm. Paper presented at the the IEEE International Conference on Neural Networks, San Francisco, USA.

Rodríguez-Gómez, R. A., Maciá-Fernández, G., & García-Teodoro, P. (2013). Survey and taxonomy of botnet research through life-cycle. ACM Computing Surveys (CSUR), 45(4), 45. doi: 10.1145/2501654.2501659

Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. Paper presented at the 13th USENIX conference on System administration, Seattle, Washington.

Roughan, M., Sen, S., Spatscheck, O., & Duffield, N. (2004). Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. Paper presented at the Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, Taormina, Sicily, Italy.

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., Felix, J., Hakimian, P. (2011). Detecting P2P botnets through network behavior analysis and machine learning. Paper presented at the Ninth Annual International Conference on Privacy, Security and Trust (PST), Montreal, QC.

Sang-Kyun, N., Joo-Hyung, O., Jae-Seo, L., Bong-Nam, N., & Hyun-Cheol, J. (2009). Detecting P2P Botnets Using a Multi-phased Flow Model. Paper presented at the Third International Conference on Digital Society, Cancun, Mexico

Scanlon, M., & Kechadi, T. (2012). Peer-to-Peer Botnet Investigation: A Review. In J. J. Park, V. C. M. Leung, C.-L. Wang & T. Shon (Eds.), Future Information Technology, Application, and Service (Vol. 179, pp. 231-238): Springer Netherlands.

Schaal, S., & Atkeson, C. G. (1994). Robot juggling: implementation of memory-based learning. Control Systems, IEEE, 14(1), 57-71. doi: 10.1109/37.257895

Schäfer, A. M. (2008). Reinforcement Learning with Recurrent Neural Networks. (PhD thesis), University of Osnabrück.

Schiller, C., & Binkley, J. R. (2011). Botnets: The killer web applications: Syngress.

Seewald, A. K., & Gansterer, W. N. (2010). On the detection and identification of botnets. Computers &amp; Security, 29(1), 45-58. doi: 10.1016/j.cose.2009.07.007

Selvaraj, K. (2014). A Brief Look at Zeus/Zbot 2.0. Symantec Security Response. Retrieved 18-Jan-2016, from http://www.symantec.com/connect/blogs/brief-look-zeuszbot-20

Seungwon, S., Zhaoyan, X., & Guofei, G. (2012). EFFORT: Efficient and effective bot malware detection. Paper presented at the INFOCOM, 2012 Proceedings IEEE, Orlando, FL.

Shen, Z., & Wang, H. (2009). Network Data Packet Capture and Protocol Analysis on Jpcap-Based. Paper presented at the International Conference on Information Management, Innovation Management and Industrial Engineering, Xi'an, China.

Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security, 31(3), 357-374. doi: http://dx.doi.org/10.1016/j.cose.2011.12.012

Singh, S., Jaakkola, T., Littman, M., & Szepesvári, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. Machine Learning, 38(3), 287-308. doi: 10.1023/A:1007678930559

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. Operations Research, 21(5), 1071-1088.

Staniford, S., Hoagland, J. A., & McAlerney, J. M. (2002). Practical automated detection of stealthy portscans. Journal of Computer Security, 10(1), 105-136.

Stevanovic, M., Revsbech, K., Pedersen, J., Sharp, R., & Jensen, C. (2012). A Collaborative Approach to Botnet Protection multidisciplinary Research and Practice for Information Systems. In G. Quirchmayr, J. Basl, I. You, L. Xu & E. Weippl (Eds.), (Vol. 7465, pp. 624-638): Springer Berlin / Heidelberg.

Stinson, E., & Mitchell, J. C. (2007). Characterizing Bots' Remote Control Behavior. Paper presented at the Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Lucerne, Switzerland.

Stock, B., Go, x, bel, J., Engelberth, M., Freiling, F. C., & Holz, T. (2009). Walowdac - Analysis of a Peer-to-Peer Botnet. Paper presented at the European Conference on Computer Network Defense (EC2ND), Milan, Italy.

Stover, S., Dittrich, D., Hernandez, J., & Dietrich, S. (2007). Analysis of the Storm and Nugache Trojans: P2P is here. USENIX; login, 32(6), 18-27.

Strayer, W. T., Walsh, R., Livadas, C., & Lapsley, D. (2006). Detecting Botnets with Tight Command and Control. Paper presented at the 31st IEEE Conference on Local Computer Networks, Tampa, USA.

Swets, J. A. (2014). Signal detection theory and ROC analysis in psychology and diagnostics: Collected papers: Psychology Press.

Symantec Corporation. (2014). Symantec Internet security threat report (Vol. 19).

Szymczyk, M. (2009). Detecting Botnets in Computer Networks Using Multi-agent Technology. Paper presented at the Fourth International Conference on Dependability of Computer Systems, Brunow, Germany.

TAX, D. (2001). One-class classification. (PhD thesis), TU Delft University.

TcpReplay. (2014). TCPReplay (Version 4.0.1). Retrieved from http://tcpreplay.synfin.net

Tegeler, F., Fu, X., Vigna, G., & Kruegel, C. (2012). BotFinder: finding bots in network traffic without deep packet inspection. Paper presented at the Proceedings of the 8th international conference on Emerging networking experiments and technologies, Nice, France.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation, 6(2), 215-219.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Commun. ACM, 38(3), 58-68. doi: 10.1145/203330.203343

Timothy, S. W., David, L., Robert, W., & Carl, L. (2008). Botnet Detection Based on Network Behavior. In W. Lee, C. Wang & D. Dagon (Eds.), Botnet Detection (Vol. 36, pp. 1-24): Springer US.

Trend-Micro. (2006). Taxonomy of botnet threats, A Trend Micro White Paper.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. Expert Systems with Applications, 36(10), 11994-12000. doi: http://dx.doi.org/10.1016/j.eswa.2009.05.029

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. Automatic Control, IEEE Transactions on, 42(5), 674-690. doi: 10.1109/9.580874

Tyagi, A. K., & Aghila, G. (2011). A wide scale survey on botnet. International Journal of Computer Applications, 34(9), 9-22.

Tyagi, R., Paul, T., Manoj, B. S., & Thanudas, B. (2015, 17-20 Dec. 2015). A novel HTTP botnet traffic detection method. Paper presented at the 2015 Annual IEEE India Conference (INDICON).

Ullah, I., Khan, N., & Aboalsamh, H. A. (2013). Survey on botnet: Its architecture, detection, prevention and mitigation. Paper presented at the 10th IEEE International Conference on Networking, Sensing and Control (ICNSC), Oslo, Norwegia.

Van der Putten, P., & Van Someren, M. (2004). A Bias-Variance Analysis of a Real World Learning Problem: The CoIL Challenge 2000. Machine Learning, 57(1-2), 177-195. doi: 10.1023/B:MACH.0000035476.95130.99

Villamarin-Salomon, R., & Brustoloni, J. C. (2008). Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic. Paper presented at the 5th IEEE Consumer Communications and Networking Conference (CCNC). Las Vegas, USA.

Wang, K., Huang, C.-Y., Lin, S.-J., & Lin, Y.-D. (2011). A fuzzy pattern-based filtering algorithm for botnet detection. Computer Networks, 55(15), 3275-3286. doi: 10.1016/j.comnet.2011.05.026

Wang, P., Sparks, S., & Zou, C. C. (2010). An advanced hybrid peer-to-peer botnet. Dependable and Secure Computing, IEEE Transactions on, 7(2), 113-127.

Wang, P., Wu, L., Aslam, B., & Zou, C. (2015). Analysis of Peer-to-Peer Botnet Attacks and Defenses. In D. Król, D. Fay & B. Gabryś (Eds.), Propagation Phenomena in Real World Networks (Vol. 85, pp. 183-214): Springer International Publishing.

Wang, X., Qiu, W., & Zamar, R. H. (2007). CLUES: A non-parametric clustering method based on local shrinking. Computational Statistics & Data Analysis, 52(1), 286-298. doi: http://dx.doi.org/10.1016/j.csda.2006.12.016

Watkins, C. J. C. H. (1989). Learning from delayed rewards. (Ph.D. thesis), University of Cambridge England.

Weaver, N., Paxson, V., Staniford, S., & Cunningham, R. (2003). A taxonomy of computer worms. Paper presented at the Proceedings of the 2003 ACM workshop on Rapid malcode, Washington, DC, USA.

Weaver, R. (2010). A probabilistic population study of the Conficker-C botnet. Paper presented at the Proceedings of the 11th international conference on Passive and active measurement, Zurich, Switzerland.

Wechsler, & Harry. (2000). Learning from Data: Concepts, Theory and Methods, Vladimir Cherkassky and Filip Mulier, John Wiley, New York, 1998. International Journal of Robust and Nonlinear Control, 10(9), 747-748. doi: 10.1002/1099-1239(20000730)10:9<747::AID-RNC507>3.0.CO;2-5

Wei, L., Tavallaee, M., Rammidi, G., & Ghorbani, A. A. (2009). BotCop: An Online Botnet Traffic Classifier. Paper presented at the Seventh Annual Communication Networks and Services Research Conference(CNSR), Moncton, NB.

Wen-Hwa, L., & Chia-Ching, C. (2010). Peer to Peer Botnet Detection Using Data Mining Scheme. Paper presented at the the international Conference on Internet Technology and Applications, Wuhan, China.

Werbos, P. J. (1992). Neural networks and the human mind: new mathematics fits humanistic insight. Paper presented at the the IEEE International Conference on Systems, Man and Cybernetics, 1992, Chicago, USA.

Wireshark. (2015). Wireshark (Version 1.12.8). Retrieved from http://www.wireshark.org

Witten, I. H., & Frank, E. (2005). Data Mining: Practical machine learning tools and techniques (Second ed.): Morgan Kaufmann.

Xiaomei, D., Fei, L., Xiaohua, L., & Xiaocong, Y. (2010). A novel Bot detection algorithm based on API call correlation. Paper presented at the Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Yantai, Shandong.

Yang, W., Fang, B.-X., Liu, B., & Zhang, H.-L. (2004). Intrusion detection system for high-speed network. Computer Communications, 27(13), 1288-1294. doi: http://dx.doi.org/10.1016/j.comcom.2004.03.001

Yen, T.-F. (2011). Detecting stealthy malware using behavioral features in network traffic. (Ph.D), Carnegie Mellon University.

Yen, T.-F., & Reiter, M. (2008). Traffic Aggregation for Malware Detection. Paper presented at the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessmen, Paris, France.

Yuanyuan, Z., Xin, H., & Shin, K. G. (2010). Detection of botnets using combined host- and network-level information. Paper presented at the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Chicago, USA.

Zeidanloo, H. R., Bt Manaf, A., Vahdani, P., Tabatabaei, F., & Zamani, M. (2010). Botnet detection based on traffic monitoring. Paper presented at the International Conference on Networking and Information Technology (ICNIT), Manila,Philippines.

Zeidanloo, H. R., Shooshtari, M. J. Z., Amoli, P. V., Safari, M., & Zamani, M. (2010). A taxonomy of Botnet detection techniques. Paper presented at the 3rd IEEE International Conference onComputer Science and Information Technology (ICCSIT), Chengdu, China.

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: A New Data Clustering Algorithm and Its Applications. Data Mining and Knowledge Discovery, 1(2), 141-182. doi: 10.1023/a:1009783824328

Zhang, W., & Dietterich, T. G. (1996). High-performance job-shop scheduling with a *time delay TD(λ) network*. Paper presented at the Advances in neural information processing systems, Cambridge.

Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A., & Garant, D. (2013). Botnet detection based on traffic behavior analysis and flow intervals. Computers & Security, 39, Part A(0), 2-16. doi: http://dx.doi.org/10.1016/j.cose.2013.04.007

Zhaosheng, Z., Guohan, L., Yan, C., Zhi, F., Roberts, P., & Keesook, H. (2008). Botnet Research Survey. Paper presented at the 32nd Annual IEEE International computer Software and Applications, Turku.

APPENDICES

A Appendix A

A.1 Evaluation the efficiency of the proposed approach using reference dataset for 20, 40 and 50 seconds time windows.

The evaluation results for training the neural network agent in the online phase based on the reference dataset (train dataset) are demonstrated in Figure A.1 to Figure A.4. It can be clearly seen that different performance measurements on the result from different time-window sizes. Based on these, results for 20s, 40s, and 50s time-windows are calculated.

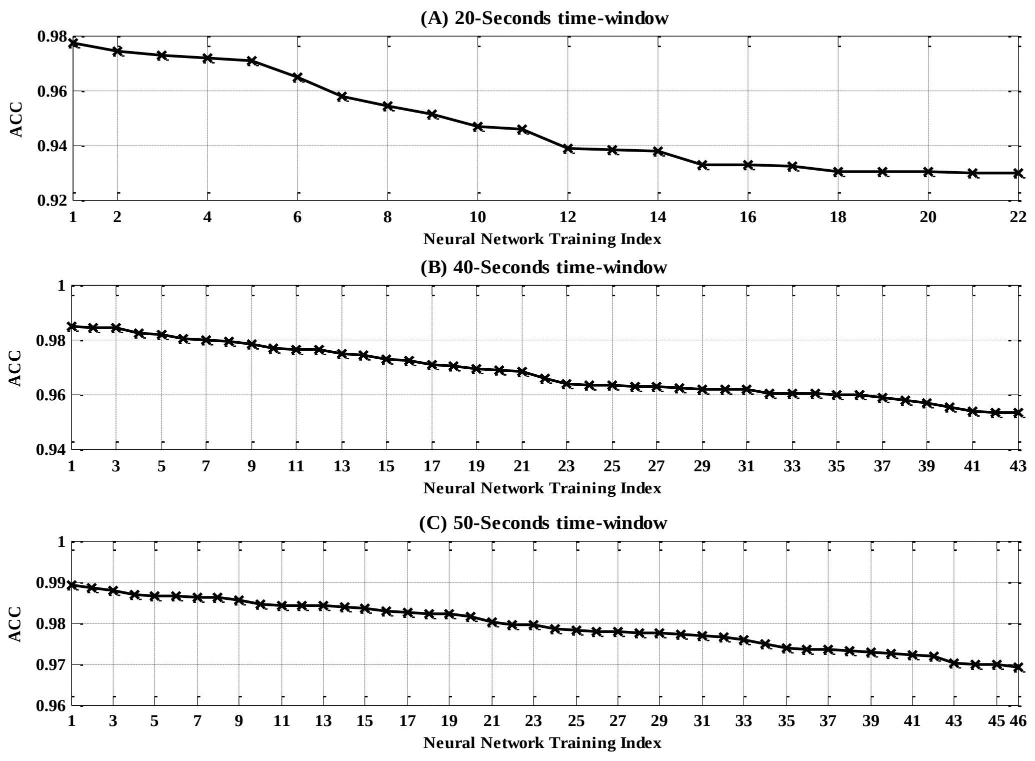

Figure A.1 Online evaluations the ACC of a classifier based on a reference dataset.

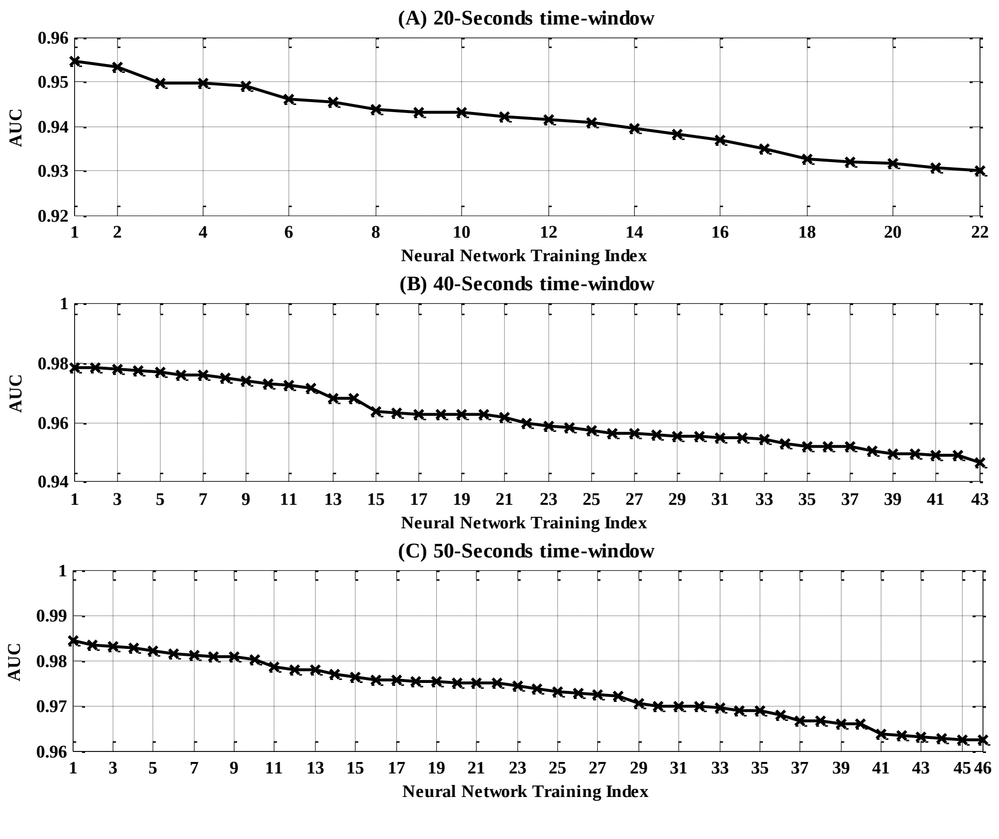

Figure A.2 Online evaluations the AUC of a classifier based on a reference dataset.

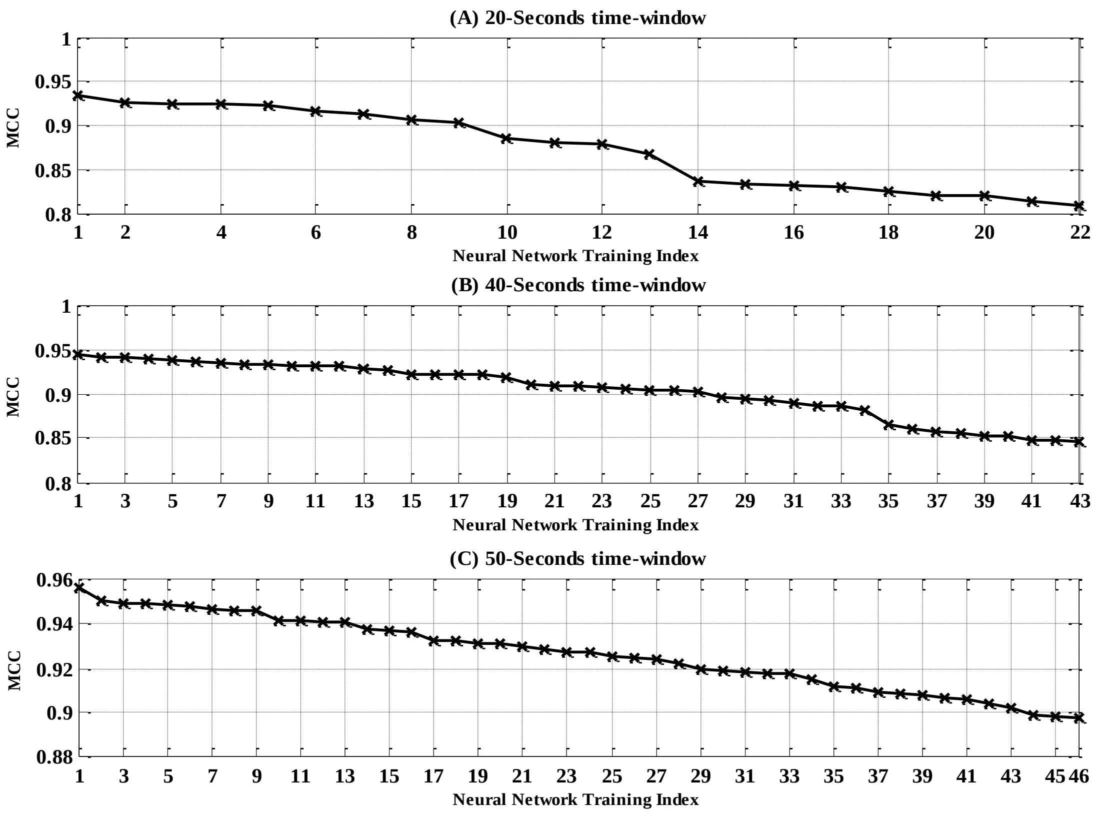

Figure A.3 Online evaluations the MCC of a classifier based on a reference dataset.

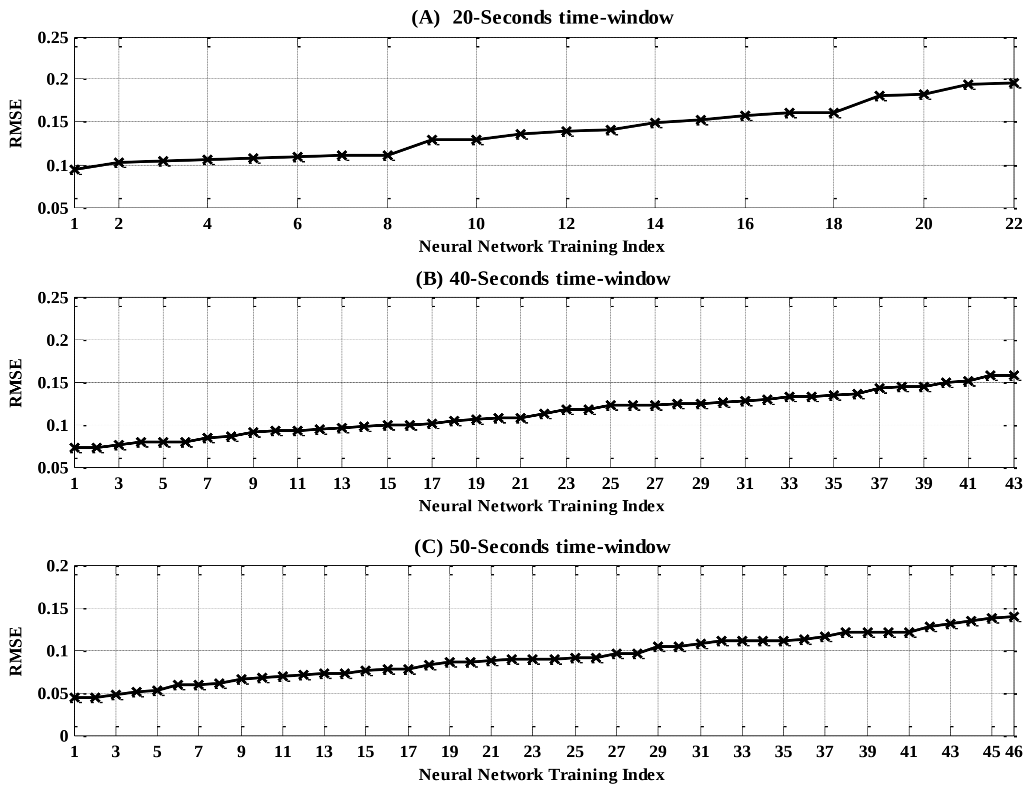

Figure A.4 Online evaluations the RMSE of a classifier based on a reference dataset.

B APPENDIX B

B.1 Evaluation the efficiency of the proposed approach using updated dataset for 20, 40 and 50 seconds.

This section gives an overview of the evaluation results of the neural network agent in Bot detection using the updated dataset with new behaviours. The results obtained from the analysis of the online experiment outcomes are summarized in Figure B.1 to Figure B.4 for 20s, 40s, and 50s time-windows are calculated.

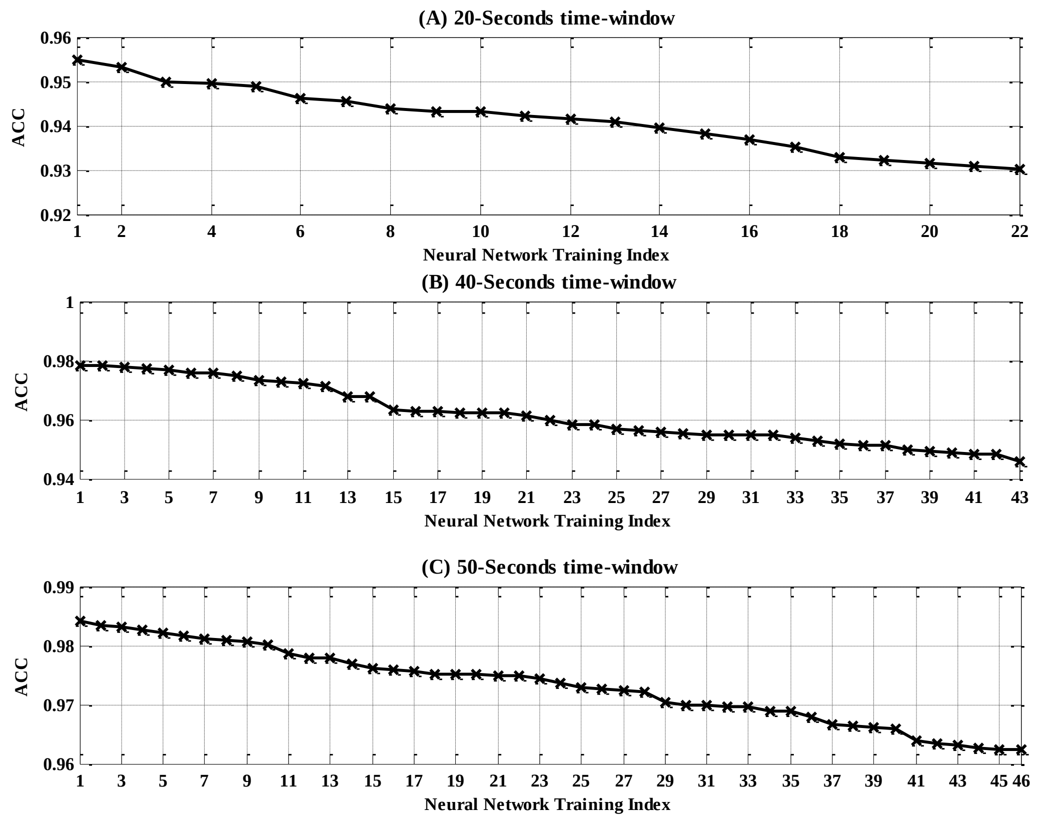

Figure B.1 Online evaluations the ACC of a classifier based on an updated dataset.

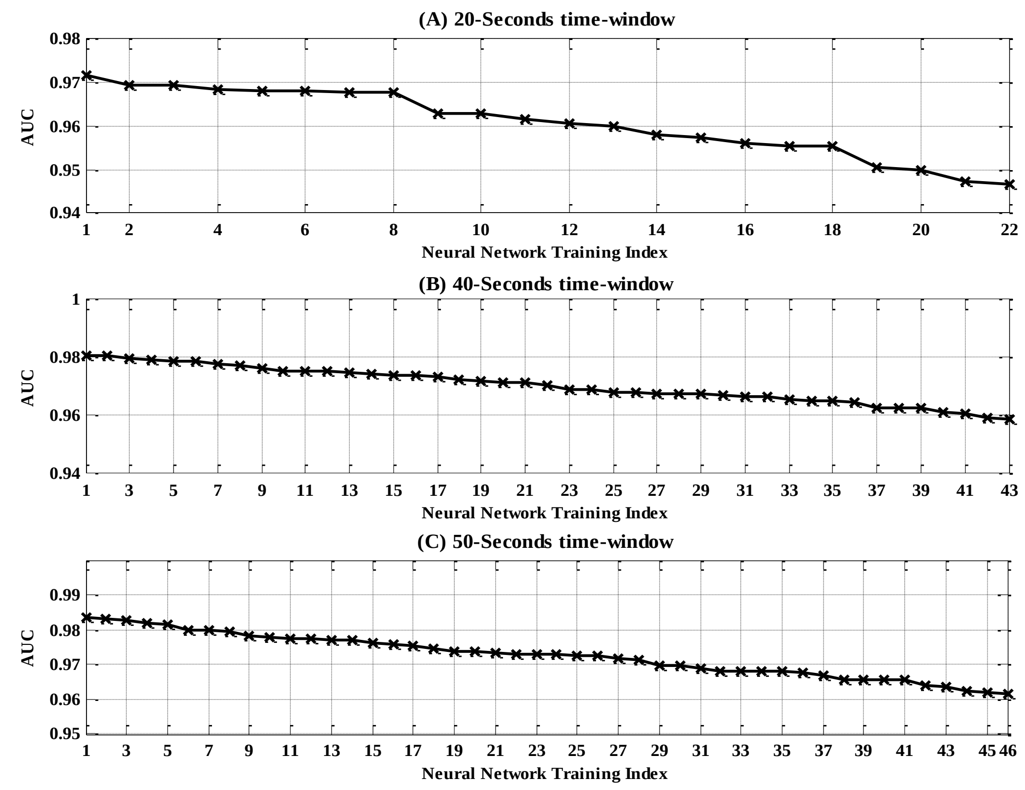

Figure B.2 Online evaluations the AUC of a classifier based on an updated dataset.

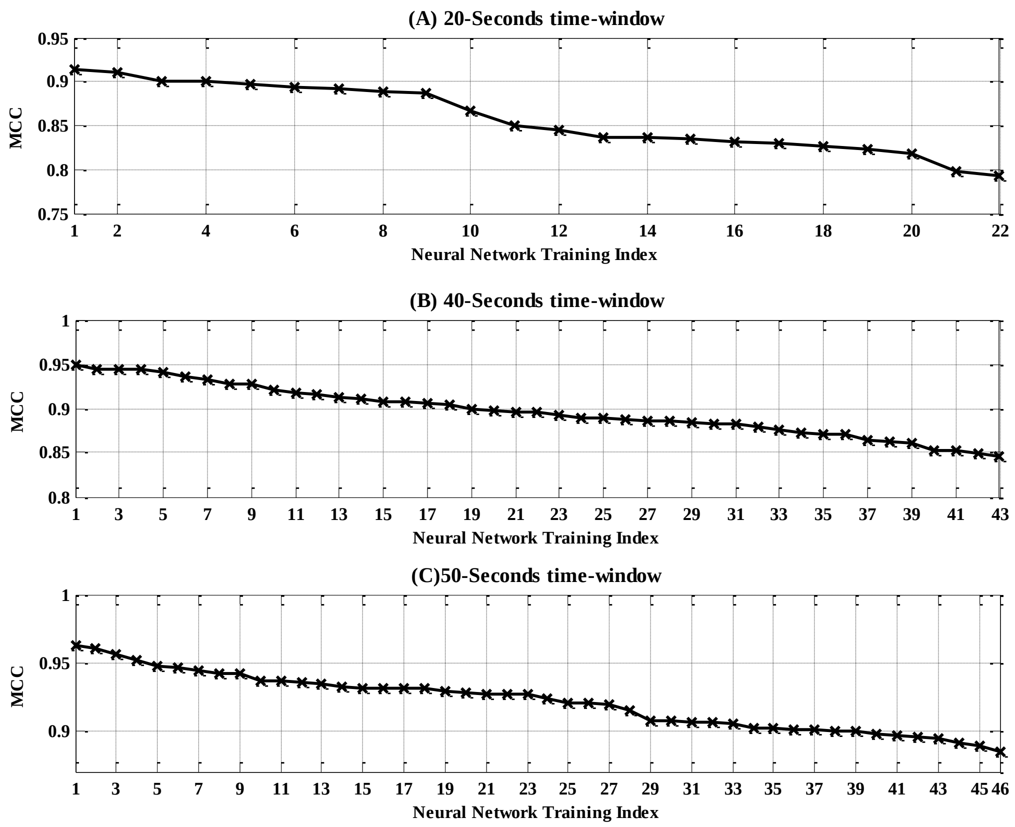

Figure B.3 Online evaluations the MCC of a classifier based on an updated dataset.

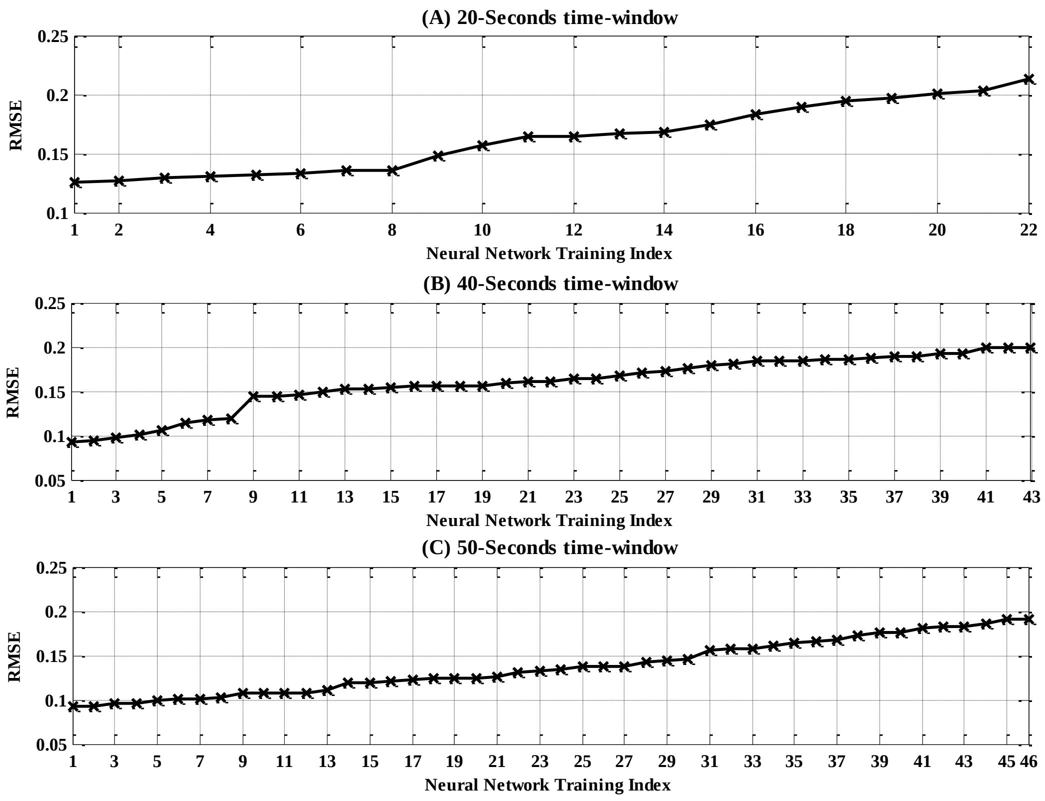

Figure B.4 Online evaluations the RMSE of a classifier based on an updated dataset.

C List of Experiment Parameters

1. The minimum level to accept the new neural network (after retrained) based on the reference dataset.

Table C.1 Evaluation parameter of new neural network using reference dataset

Evaluation Method	**Parameter value**
AUC	>0.90
MCC	>0.80
ACC	>0.90
RMSE	<0.20

2. The minimum level to accept the new neural network (after retrained) based on the updated dataset.

Table C.2 Evaluation parameter of new neural network using updated dataset

Evaluation Method	**Parameter value**
AUC	>0.90
MCC	>0.80
ACC	>0.90
RMSE	<0.20

3. The threshold number used of exploration step per host is five.
4. The value of threshold factor that allow the proposed system to change state form exploration to exploitation.

$$Ts \geq 0.15$$

5. Neural network parameter.

Neural network parameter	**Parameter value**
Number of layers	5
Input layer neurons	16
output layer neurons	2
Neuron per hidden layer	10
Learning function	Resilient Backpropagation
epochs	1000
goal	1e-5
Error function of	MSE

6. Cross-validation 5-fold was used.

D IMPLEMENTATION FRAMEWORK

1. Traffic capture module.

a) Pseudo code

```
import java.util.ArrayList;
import java.util.List;
import org.jnetpcap.Pcap;
import org.jnetpcap.PcapIf;
import org.jnetpcap.packet.PcapPacket;
import org.jnetpcap.packet.PcapPacketHandler;
import org.jnetpcap.protocol.network.Ip4;
public class PackageCapture
{
public static void main(String[] args)
{
List<PcapIf> alldevs = new ArrayList<PcapIf>();
StringBuilder errbuf = new StringBuilder
int r = Pcap.findAllDevs(alldevs, errbuf);
if (r != Pcap.OK || alldevs.isEmpty())
{
 System.err.printf("Can't read list of devices, error is %s",
errbuf.toString());
return;
 }
System.out.println("Network devices found:");
int i = 0;
for (PcapIf device : alldevs)
{
String description = (device.getDescription() != null) ?
device.getDescription():"Nodescription
available";.out.printf("#%d: %s [%s]\n", i++,
device.getName(),description);
}
PcapIf device = alldevs.get(0); // Get first device in list
System.out.printf("\nChoosing '%s' on your behalf:\n",
 (device.getDescription() != null) ? device.getDescription()
: device.getName());
int snaplen = 64 * 1024; // Capture all packets, no trucation
int flags = Pcap.MODE_PROMISCUOUS; // capture all packets
int timeout = 10 * 1000; // 10 seconds in millis

Pcap pcap = Pcap.openLive(device.getName(), snaplen, flags,
timeout, errbuf);

if (pcap == null) {System.err.printf("Error while opening device
for capture: "+ errbuf.toString());

return;

}

PcapPacketHandler<String> jpacketHandler = new
PcapPacketHandler<String>()

{

public void nextPacket(PcapPacket packet, String user) {

byte[] data = packet.getByteArray(0, packet.size()); // the
package data

byte[] sIP = new byte[4];

byte[] dIP = new byte[4];

Ip4 ip = new Ip4();

if (packet.hasHeader(ip) == false) {

return; packet

}

ip.source(sIP);

ip.destination(dIP);

String sourceIP = org.jnetpcap.packet.format.FormatUtils.ip(sIP);

String destinationIP =
org.jnetpcap.packet.format.FormatUtils.ip(dIP);

System.out.println("srcIP=" + sourceIP + " dstIP=" +
destinationIP + " caplen=" + packet.getCaptureHeader().caplen());

}

};

pcap.loop(10, jpacketHandler, "jNetPcap");

pcap.close();

}

}
```

2. Traffic reduction module.

a) Input: network packets.

b) Output: TCP control packets.

c) Pseudo code:

```
function contrl_packets = Traffic_Reducton(raw)
  temp = raw;
  control_pak = [];
  Res=[];
  for j = 1:size(temp,1)
    SYN=~isempty(strfind(cell2mat(temp(j,8)),'[SYN]'));
    ACK=~isempty(strfind(cell2mat(temp(j,8)),'[ACK]'));
    FIN=~isempty(strfind(cell2mat(temp(j,8)),'[FIN]'));
    RST=~isempty(strfind(cell2mat(temp(j,8)),'[RST]'));
    SACK=~isempty(strfind(cell2mat(temp(j,8)),'[SYN, ACK]'));
    FACK=~isempty(strfind(cell2mat(temp(j,8)),'[FIN, ACK]'));
    RACK=~isempty(strfind(cell2mat(temp(j,8)),'[RST, ACK]'));
     if (ACK | SYN | FIN | RST |SACK|FACK|RACK)
      control_pak = cat(1, control_pak, temp(j, :));
      Res = cat(1, Res, temp(j, :));
   end
  end
  save('Cont_packet.mat', 'control2');
  contrl_packets=control_pak;
end
```

3. Connection-level features extraction module.

a) Input: Time window size and TCP control packets.

b) Output: Connection-level features list.

c) Pseudo code:

```
function extract_flow = extract(control_raw,time_w)
temp2 = control_raw;
temp = temp2;
CPT = [];
Res = [];
while ~isempty(temp)
x=[];
   TIME_rACK=[];  % reactive ACK packets - time sequence per connection.
    TIME_sACK=[]; % send ACK packets - time sequence per connection.
    TIME_rSYN=[];  % receive SYN packets - time sequence per connection.
    TIME_sSYN=[];% send SYN packets - time sequence per connection.
    paket_seq=[]; % packets sequence per connection.
    start_Conn=0; % start time of connection.
    cp = 0;    % total number of packet.
    NsendPacket=0; % total number of send packets.
    NrecivePacket=0; % total number of receive packets.
    TotalSendByte=0; %total number of send Bytes per connection.
    TotalReciveByte=0;%total number of receive Bytes per connection.
    NsendSyn=0;% number of send SYN packets per connection.
    NreciveSyn=0;% number of receive SYN packets per connection.
    NsendAck=0; %number of send ACK packets per connection.
    NreciveAck=0;%number of receive ACK packets per connection.
    NsendSynAck=0;%number of send SYN ACK packets per connection.
    sACK=0; %number of send ACK=1 packets per connection.
    NreciveSynAck=0;%number of receive SYN ACK packets per connection.
    rACK=0;  %number of receive ACK=1 packets per connection.
    NsendDupAck=0;%number of send double ACK packets per connection.
    NreciveDupAck=0;%number of receive double ACK packets per connection.
    NsendFinAck=0;%number of send FIN ACK packets per connection.
    NreciveFinack=0;%number of receive FIN ACK packets per connection.
    SendRSTack=0;%number of send RST ACK packets per connection.
    RecivdRSTack=0; %number of receive RST ACK packets per connection.
    avg_time=0; % Avg. time between packets.
    defrent_time=0; % time between connections.
    avgLengthSendPacket=0;% Avg. length of send packets.
    avgLengthRecivedPacket=0;%Avg. length of receive packets.
    avgLengtPacket=0;% Avg. length of packets.
    SendFailedConnection =0;% number of send fail per connection.
    TotalRConnection=0;% Total number of receive connections.
     RecivdRST=0; % number of receive RST packets per connection.
     RecivdRSTack=0;%  number  of  receive  RST  ACK  packets  per
connection.
     SendRSTack=0;% number of send RST ACK packets per connection.
     SendRST=0;% number of send RST packets per connection.
     scanCount=0;% number of scanning activates.
    % % % % % % % % % % % % % % % % % % % % % % % % % % % %
    cpt = 0;
    temp = temp2;
    host_A=temp(1, 2);
    host_B=temp(1, 3);
    port_A=temp(1, 4);
    port_B=temp(1, 5);
    start_Con=cell2mat(temp(1,6));
   s1 = strcat(temp(1, 2), temp(1, 3),temp(1, 4),temp(1, 5));
    for j = 1:size(temp, 1)
        s3 = strcat(temp(j, 2), temp(j, 3),temp(j, 4),temp(j, 5));
        s4 = strcat(temp(j, 3), temp(j, 2),temp(j, 5),temp(j, 4));
        if strcmp(s3, s1) % send packet
            timePacket=[timePacket cell2mat(temp(j,6))];
            Res = cat(1, Res, temp(j, :));
            NsendPacket=NsendPacket+1;
            TotalSendByte=TotalSendByte+cell2mat(temp(j,7));
            if ((~isempty(strfind(cell2mat(temp(j,8)),
           '[SYN]'))&(isempty(strfind(cell2mat(temp(j,8)),'[TCP
             Retransmission]')))...
             & (isempty(strfind(cell2mat(temp(j,8)),'[TCP Out-Of-
                Order]')))))
               NsendSyn=NsendSyn+1;
              paket_seq=[paket_seq 's1,'];
              TIME_sSYN=[TIME_sSYN num2str(cell2mat(temp(j,6)))';'];
              timesyn=cell2mat(temp(j,6));
            end
            if (~isempty(strfind(cell2mat(temp(j,8)), '[SYN,
            ACK]'))&& ~isempty(strfind(cell2mat(temp(j,8)), 'Seq=0
             ')))
                NsendSynAckseq0=NsendSynAckseq0+1;
                paket_seq=[paket_seq '1s2,'];
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[SYN, ACK]'))
                NsendSynAck=NsendSynAck+1;
                 timeSrstack=cell2mat(temp(j,6));
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[ACK]'))
                NsendAck=NsendAck+1;
            end
           if (~isempty(strfind(cell2mat(temp(j,8)), '[FIN,
            ACK]'))&& ~isempty(strfind(cell2mat(temp(j,8)),'Seq=1
            Ack=1 ')))
                NsendFinAck=NsendFinAck+1;
                paket_seq=[paket_seq '1s7,'];
                timeSfinack=cell2mat(temp(j,6));
           end
            if ~isempty(strfind(cell2mat(temp(j,8)), 'Dup'))
                NsendDupAck=NsendDupAck+1;
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[RST]'))
                SendRST=SendRST+1;
            end


            if (~isempty(strfind(cell2mat(temp(j,8)), '[RST]'))&&
           ~isempty(strfind(cell2mat(temp(j,8)), 'Seq=1 ')))
```

```
                SendRSTseq1=SendRSTseq1+1;
                paket_seq=[paket_seq '1s4,'];
                timeSrst=cell2mat(temp(j,6));
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[RST, ACK]'))
                SendRSTack=SendRSTack+1;
            end
            if (~isempty(strfind(cell2mat(temp(j,8)), '[RST,
             ACK]'))&& ~isempty(strfind(cell2mat(temp(j,8)), 'Seq=1
             ')))
             SendRSTackSeq1=SendRSTackSeq1+1;
             paket_seq=[paket_seq '1s5,'];
             timeSrstack=cell2mat(temp(j,6));
            end

            ACK1=~isempty(strfind(cell2mat(temp(j,8)),'[ACK]'));
            ack_seq=~isempty(strfind(cell2mat(temp(j,8)),'Seq=1
            Ack=1 '));
            dumack=isempty(strfind(cell2mat(temp(j,8)),'Dup '));
            keepAliveAck=isempty(strfind(cell2mat(temp(j,8)),'[TCP
            Keep-Alive] '));

            if(ACK1 & ack_seq & dumack & keepAliveAck)
             sACK=sACK+1;
             TIME_sACK=[TIME_sACK num2str(cell2mat(temp(j,6))) ','];
             timeSack=cell2mat(temp(j,6));
             paket_seq=[paket_seq '1s3,'];
            end
            temp2(j-cpt, :) = [];
            cpt = cpt + 1;
            cp = cp + 1;
        elseif strcmp(s4, s1) % receive packet
            Res = cat(1, Res, temp(j, :));
            timePacket=[timePacket cell2mat(temp(j,6))];
            NrecivePacket=NrecivePacket+1;
            TotalReciveByte=TotalReciveByte+cell2mat(temp(j, 7));
            if ((~isempty(strfind(cell2mat(temp(j,8)), '[SYN]'))&
           (isempty(strfind(cell2mat(temp(j,8)),'[TCP Out-Of-Order]'
            )))))
             NreciveSyn=NreciveSyn+1;
             paket_seq=[paket_seq 'r1,'];
             TIME_rSYN=[TIME_rSYN num2str(cell2mat(temp(j,6))) ';'];
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[ACK]'))
                NreciveAck=NreciveAck+1;
            end
             if (~isempty(strfind(cell2mat(temp(j,8)),'[SYN, ACK]'
              ))&& ~isempty(strfind(cell2mat(temp(j,8)),'Seq=0 ')))
                 NreciveSynAckseq0=NreciveSynAckseq0+1;
                 paket_seq=[paket_seq '1r2,'];
                  timeRsynack=cell2mat(temp(j,6));
             end


            if ~isempty(strfind(cell2mat(temp(j,8)), '[SYN, ACK]'))
                NreciveSynAck=NreciveSynAck+1;
            end
            if ~isempty(strfind(cell2mat(temp(j,8)), 'Dup'))
                NreciveDupAck=NreciveDupAck+1;
            end
             if ~isempty(strfind(cell2mat(temp(j,8)), '[RST]'))
                RecivdRST=RecivdRST+1;
            end
             if (~isempty(strfind(cell2mat(temp(j,8)), '[RST]'))&&
                ~isempty(strfind(cell2mat(temp(j,8)), 'Seq=1 ')))
                RecivdRSTseq1=RecivdRSTseq1+1;
                paket_seq=[paket_seq '1r4,'];
                timeRrst=cell2mat(temp(j,6));
             end
            if ~isempty(strfind(cell2mat(temp(j,8)), '[RST, ACK]'))
                RecivdRSTack=RecivdRSTack+1;
            end
          if (~isempty(strfind(cell2mat(temp(j,8)), '[RST, ACK]'))&&
           ~isempty(strfind(cell2mat(temp(j,8)), 'Seq=1 ')))
               RecivdRSTackseq1=RecivdRSTackseq1+1;
                paket_seq=[paket_seq '1r5,'];
                timeRrstack=cell2mat(temp(j,6));
            end
           if (~isempty(strfind(cell2mat(temp(j,8)), '[FIN,
            ACK]'))&& ~isempty(strfind(cell2mat(temp(j,8)),'Seq=1
            Ack=1 ')))
                NreciveFinack=NreciveFinack+1;
                paket_seq=[paket_seq '1r7,'];
           end

            ACK2=~isempty(strfind(cell2mat(temp(j,8)),'[ACK]'));
            ack_seq=~isempty(strfind(cell2mat(temp(j,8)),'Seq=1
            Ack=1 '));
            dumack=isempty(strfind(cell2mat(temp(j,8)),'Dup '));
            keepAliveAck=isempty(strfind(cell2mat(temp(j,8)),'[TCP
            Keep-Alive] '));
            if(ACK2 & ack_seq & dumack & keepAliveAck)
            rACK=rACK+1;
            TIME_rACK=[TIME_rACK num2str(cell2mat(temp(j,6))) ','];
            paket_seq=[paket_seq '1r3,'];
            end
            temp2(j-cpt, :) = [];
            cpt = cpt + 1;
            cp = cp + 1;
        end
    end
    temp = temp2;
    if (NsendPacket>0)
        avgLengthSendPacket= TotalSendByte/NsendPacket;
    end
    if(NrecivePacket>0)
        avgLengthRecivedPacket=TotalReciveByte/NrecivePacket;
    end
  avgLengtPacket=(avgLengthSendPacket+avgLengthRecivedPacket)/2;
```

```
  if (NsendSynAckseq0==cp)
    synackscan=NsendSynAckseq0;
  end
  % send connection info.
  SendFailedConnection1 =(NsendSyn-sACK);
SendFailedConnection2=SendRSTackSeq1+SendRSTseq1+RecivdRSTackseq1+Re
civdRSTseq1+synackscan;
  if (SendFailedConnection1>=SendFailedConnection2)
      SendFailedConnection=SendFailedConnection1;
  else
      SendFailedConnection=SendFailedConnection2;
  end
  % Received connection info.
    RecivedFailedConnection1 =(NreciveSyn-rACK);
    RecivedFailedConnection12=RecivdRSTackseq1+RecivdRSTseq1;
        if (RecivedFailedConnection1>=RecivedFailedConnection12)
      RecivedFailedConnection=RecivedFailedConnection1;
  else
      RecivedFailedConnection=RecivedFailedConnection12;
    end
  conection_Duration=timePacket(size(timePacket,2))-timePacket(1);

if ~isempty(TIME_sSYN)
     TIME_sSYN_vector=((TIME_sSYN))';
     num_TIME_sSYN_vector=size(TIME_sSYN_vector,2);
 if (num_TIME_sSYN_vector>1)
        for i = 2:num_TIME_sSYN_vector
            defrent_time=defrent_time+(TIME_sSYN_vector(i)-
TIME_sSYN_vector(i-1));
         end
       avg_time=defrent_time/(num_TIME_sSYN_vector-1);
     end
 end
  x=[time_w,start_Con,host_A,host_B,port_A,port_B,...
       paket_seq,...
       TIME_sSYN,...
```

```
        TIME_rSYN,...

        cp,...

        NsendPacket,...

        NrecivePacket,...

        TotalSendByte,...

        TotalReciveByte,...

        NsendSyn,...

        NreciveSyn,...

        NsendAck,...

        NreciveAck,...

        NsendDupAck,...

        NreciveDupAck,...

        avgLengthSendPacket,...

        avgLengthRecivedPacket,...

        avgLengtPacket,...

        SendFailedConnection,...

        RecivedFailedConnection,...

        sACK,...

        rACK,...

        NsendSynAck,...

        NreciveSynAck,...

        TotalSendByte+TotalReciveByte,...

        NrecivePacket/cp,...

avgLengthSendPacket/((avgLengthSendPacket+avgLengthRecivedPacket)/2)
,...
       (NsendSyn-sACK)/NsendSyn,...

        NsendSyn-NreciveSynAck,...

        NsendFinAck,...

        NreciveFinack,...

        SendRSTack,...

        RecivdRSTack,...

        avg_time,...

        NsendSynAckseq0,...

        SendRSTseq1,...

        SendRSTackSeq1,...
        NreciveSynAckseq0,...
        RecivdRSTseq1,...
        RecivdRSTackseq1,...

SendRSTseq1+SendRSTackSeq1+RecivdRSTseq1+RecivdRSTackseq1+synackscan
,...
       conection_Duration];
         CPT = cat(1, CPT, x);
  cp = 0;
end
 extract_flow=CPT;
end
```

4. Host features extraction module.

a) Input: connection features.

b) Output: Host features.

c) Pseudo code:

```
  function extract_IP = extract_host_features flow,time_w)
  temp2 = flow;
  temp = temp2;
  CPT = [];
  Res = [];
  Targets=[];
  paket_seq=[];
  xx=[];
  while ~isempty(temp)
      cpt=0;
      cp=0;
      Nsendflow=0; %number of send flows.
      NsendCon=0;  %number of send connections.
      NreciveCon=0;%number of receive connections.
      NsendSyn=0;  %number of send SYN packets.
      NSfailCon=0; %number of send failed connection.
      fail_conn=0; %total number of failed connections.
      timeSeq=[];  % packets time sequence.
      timeSeq1=[]; %packets time sequence.
      avg_flow_time=0; % average time flow between flow.
      sendConseq=0; %receiver IP address.

      portASeq=[]; % sender ports sequence.

      portBSeq=[]; % receiver ports sequence.

      defrent_time=0; % time between connections.

   %%%%%%%%%%%%%%%%% inter packet %%%%%%%

      clinet_synAck=[];  %  Number  of  send  SYN  ACK  packets  in  a
  connection per host.

      clinet_synRst=[];  %  Number  of  send  SYN  ACK  packets  in  a
  connection per host.

      clinet_synRstack=[];%  Number  of  send  RST  ACK  packets  in  a
  connection per host.

      server_synRst=[];   %  Number  of  receive  RST   packets  in  a
  connection per host.

      Server_synRstack=[];% Number of receive RST ACK  packets in a
  connection per host.

      srver_finackRst=[];% Number of receive FIN ACK  packets in a
  connection per host.

      server_synSynack=[];% Number of receive SYN ACK  packets in a
  connection per host.

   %%%%%%%%%%%% number send, receive and total control packet %%%

     total_control=[];

     send_control=[];

     recive_control=[];

    %%%%%%%%%%%%%%%%%%%%%%%% connection - duration %%%%%%%%%%%%%%

    connections_Durations=[];

    Avg_connections_Duration=0;

    %%%%%%%%%%%%%%%%%%%%%%% port - severity %%%%%%%%%%%%%%%%%%%%%

    portSH=0;

    portSL=0;

    portDH=0;

    portDL=0;

      host =temp(1,3);

     for j = 1:size(temp, 1)

        hostSend= temp(j,3);

    if strcmp(host, hostSend)

        Nsendflow=Nsendflow+1;% total of connections

        if  (cell2mat(temp(j,15))>0  || cell2mat(temp(j,46))>0  ||

         cell2mat(temp(j,47))>0  )

            timeSeq=[timeSeq num2str(cell2mat(temp(j,2))) ';' ];
```

```
             timeSeq1=[timeSeq1 cell2mat(temp(j,2))];
             end
        sendConseq=[sendConseq cell2mat(temp(j,4)) ';' ];% destination
   IP address
        portASeq=[portASeq, temp(j,5)];% sender ports.
        portBSeq=[portBSeq, temp(j,6)];% destination ports.
        NsendCon=NsendCon+cell2mat(temp(j,26)); % number send Packets
   Start Ack.
       NreciveCon=NreciveCon+cell2mat(temp(j,27)); % number of Packets
   receive start ack.
       NSfailCon=NSfailCon+cell2mat(temp(j,24));  %  number  of  fail
   connection.
      NsendSyn=NsendSyn+cell2mat(temp(j,15));  %  number  send  packets
   start SYN.
     fail_conn=fail_conn+cell2mat(temp(j,24));
      ailconnseq =[failconnseq num2str(cell2mat(temp(j,24))) ','];
        %%%%%%%%%%%%%%%%%%%%%%%%%%% inter packet %%%%%%%%%%%%%%%%
      clinet_synAck=[clinet_synAck (cell2mat(temp(j,48))) ','];
      clinet_synRst=[clinet_synRst (cell2mat(temp(j,49))) ','];
      clinet_synRstack=[clinet_synRstack (cell2mat(temp(j,50))) ','];
      server_synRst=[server_synRst (cell2mat(temp(j,51))) ','];
      Server_synRstack=[Server_synRstack (cell2mat(temp(j,52))) ','];
      srver_finackRst=[srver_finackRst (cell2mat(temp(j,53))) ','];
      server_ack_rest=[server_ack_rest (cell2mat(temp(j,54))) ','];
      server_synSynack=[server_synSynack (cell2mat(temp(j,55))) ','];
   %%%%%%%%%%%%%%%%%%%%%%%%%%%Ports severity %%%%%%%%%%%%%%%%
      if (find (cell2mat(temp(j,5))== portHseverity))
        portSH=portSH+1;
      else
        portSL=portSL+1;
       end
       if (find (cell2mat(temp(j,5))==  portHseverity))
        portDH=portDH+1;
      else
        portDL=portDL+1;
       end
      %%%%%%%%%%%%%%%%%%%%%%%% connection duration %%%%%%%%%%%%%%
```

```
  connections_Durations=[connections_Durations
num2str(cell2mat(temp(j,56))) ','];

 %%%%%%%%%%%%%% number of sent received control packet %%%%%%%%%%

total_control=[total_control num2str(cell2mat(temp(j,10))) ','];

send_control=[send_control num2str(cell2mat(temp(j,11))) ','];

recive_control=[recive_control num2str(cell2mat(temp(j,12))) ','];

          temp2(j-cpt, :) = [];

          cpt = cpt + 1;

          cp = cp + 1;

  end

 end

 temp = temp2;

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Avg_clinet_synAck=mean(clinet_synAck);

    Avg_clinet_synRst=mean(clinet_synRst);

    Avg_clinet_synRstack=mean(clinet_synRstack);

    Avg_server_synRst=mean(server_synRst);

    Avg_Server_synRstack=mean(Server_synRstack);

    Avg_srver_finackRst=mean(srver_finackRst);

    Avg_server_ack_rest=mean(server_ack_rest);

    server_synSynack=mean(server_synSynack);

    Avg_connections_Duration=mean(connections_Durations);

avg_clinet_interarival=(Avg_clinet_synAck+Avg_clinet_synRst+Avg_cl
inet_synRstack)/3;

avg_server_interarival=(Avg_server_synRst+Avg_Server_synRstack+Avg
_srver_finackRst+Avg_server_ack_rest+server_synSynack)/5;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   NportASeq=size(portASeq,2); % number of using port at sender

   NdefPortA=size(unique(cell2mat(portASeq)),2);

   NportBSeq=size(portBSeq,2); % number of using port at receiver.

   NdefPortB=size(unique(cell2mat(portBSeq)),2);

Nsend_Def_flow=size(unique(strread(sendConseq,'%s','delimiter',';'
)),1); % number of send different IP address.

    rateDeffIP=Nsend_Def_flow/Nsendflow;

%%%%%%%%%%%%%%%%%%% Avg time between connection per second %%%%%%

  if ~isempty(timeSeq)

      TIME_flow_vector=(timeSeq1);
```

```
         num_TIME_flow_vector=size(TIME_flow_vector,2);
     if (num_TIME_flow_vector>1)
            for i = 2:num_TIME_flow_vector
               time=[time                num2str(TIME_flow_vector(i)-
TIME_flow_vector(i-1)) ','] ;
               defrent_time=defrent_time+(TIME_flow_vector(i)-
TIME_flow_vector(i-1));
             end
          avg_flow_time=defrent_time/(num_TIME_flow_vector-1);
   end
   end
   if Nsendflow>NsendSyn
       conn=Nsendflow;
   else
      conn= NsendSyn;
   end
%%%%%%% different time between connection time- entropy %%%%%%
if size(str2num(time),2)>1>1
tim_EntropyResult= Entropy(fix(str2num(time)'));
else
tim_EntropyResult=0;
end
%%%%%%%%%%%%%%%%% entropy control packet number %%%%%%%%%%%%%%
if size(str2num(total_control),2)>1
total_control_EntropyResult= Entropy(fix((total_control)'));
else
 total_control_EntropyResult=0;
end
if size(str2num(send_control),2)>1
send_control_EntropyResult= Entropy(fix((send_control)'));
else
 send_control_EntropyResult=0;
end
if size(str2num(recive_control),2)>1
recive_control_EntropyResult= Entropy(fix((recive_control)'));
else
```

```
  recive_control_EntropyResult=0;
 end
     result=[time_w,host,Nsendflow,...
     NsendCon,NreciveCon,...send  and  receive  connections  with
 ACk(1).
    NportASeq,NportBSeq,... total number of send and receive ports.
     NdefPortA,NdefPortB,...total number unique of send and receive
 ports.
     Nsend_Def_flow,....
     NSfailCon,...
     NsendSyn,...
     NSfailCon/conn,...
     sendConseq,...
     avg_flow_time,...
     rateDeffIP,...
     NdefPortB/NportBSeq,...
     NdefPortA/NportASeq,...
     time,...
     timeSeq,...
     fail_conn,...
     failconnseq,...
     clinet_synAck,...
     clinet_synRst,...
     clinet_synRstack,...
     server_synRst,...
     Server_synRstack,...
     srver_finackRst,...
     server_ack_rest,...
     server_synSynack,...
     total_control,...
     send_control,...
     recive_control,...
     connections_Durations,...
     portSH,...
     portSL,...
     portDH,...
     portDL,...
     portSL/(portSL+portSH),... %rate of low severity source port
 number
    portDL/(portDL+portDH),...% rate of low severity destination
 port number
    Avg_clinet_synAck,...
    Avg_clinet_synRst,...
    Avg_clinet_synRstack,...
    Avg_server_synRst,...
    Avg_Server_synRstack,...
    Avg_srver_finackRst,...
    Avg_server_ack_rest,...
    server_synSynack,...
    Avg_connections_Duration,...
    total_control_EntropyResult,...
    send_control_EntropyResult,...
    recive_control_EntropyResult,...
    avg_clinet_interarival,...
    avg_server_interarival];...
 CPT=cat(1, CPT, result); % results
  cp = 0;
 end
   extract_IP=CPT;
  end
```

5. Entropy algorithm model.

a. Input: Number of control packets per connection.

b. Output: Connection entropy value.

c. Pseudo code:

```
function Entropy = Entropy(X)
[n m] = size(X);
H = zeros(1,m);
for Column = 1:m,
    % Assemble observed alphabet
    Alphabet = unique(X(:,Column));
    % Housekeeping
    Frequency = zeros(size(Alphabet));
     % Calculate sample frequencies
    for symbol = 1:length(Alphabet)
        Frequency(symbol) = sum(X(:,Column) == Alphabet(symbol));
    end
    % Calculate sample class probabilities
    P = Frequency / sum(Frequency);
    % Calculate entropy in bits
    % Note: floating point underflow is never an issue since we are
    %   dealing only with the observed alphabet
    H(Column) = -sum(P .* log2(P));
 end
Entropy=H;
end
```

6. Reinforcement learning agent

a) Input: Host state (neural network outcomes).

b) Output: new updated neural network.

c) Pseudo code:

```
function check_NN_sataus()
   if (newBotitem+newNormalitem)/a >=threshold
      alldataset2= cat(1,alldataset,newdatasetitem);
%%%%%%%%%%%%% evaluation cross-validation for new dataset %%%%
result_cross_newDataset_online2=cross_valadition(alldataset2(:,19:20
)',alldataset2(:,1:16)');

result_cross_newDataset_online=cat(1,result_cross_newDataset_online,
result_cross_newDataset_online2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%train NEW neural network   %%%%%%%%%
    net_test = patternnet([10 10 10]);
    net_test.trainFcn = 'trainrp';
    net_test.trainParam.epochs=500;
    net_test.trainParam.showWindow=false;
    net_test.trainParam.goal=1e-10;
   net_test.divideParam.trainRatio = 100/100;
   net_test.divideParam.valRatio = 0/100;
   net_test.divideParam.testRatio = 0/100;
   net_test.trainParam.showWindow=false;
   net_test.trainParam.showCommandLine = false;
 [net_test,tr]=
train(net_test,alldataset2(:,1:16)',alldataset2(:,19:20)');
     Y = net_test(alldataset2(:,1:16)');
    error=Y-alldataset2(:,19:20)';
result_cross_newDataset_online;
all_result_newDataset_online2=result_evaluation(alldataset2(:,19:20)
',Y,0,0)';
all_result_newDataset_online   =   cat(1,all_result_newDataset_online,
all_result_newDataset_online2);
% %%%%%%%%%%%%%%% make decision for change neural network %%%%%
if (result_cross_newDataset_online2(1,7)>0.95 &&
    result_cross_newDataset_online2(1,11)>0.5)
    net_last_good = net_test;
    alldataset= cat(1,alldataset,newdatasetitem);
    newdatasetitem=[]; %
    newBotitem=0;%
    newNormalitem=0;%
    totalNumber=0;
    Y1 = net_last_good(olddataset(:,1:16)');
    all_result_oldDataset_online2=result_evaluation(olddataset
     (:,17:18)',Y1,0,0)';
    all_result_oldDataset_online =
cat(1,all_result_oldDataset_online,all_result_oldDataset_online2);%%
 else
    newdatasetitem=[];
    newBotitem=0;
    newNormalitem=0;
    totalNumber=0;
    net_fail_index=net_fail_index+1;
end
%%%%%%%%%%%%%%%% system reset when missing old dataset%%%%%%%%
if (all_result_oldDataset_online2(1,7)<=0.90)
  net_last_good=net_ref;
  net_rest_index=net_rest_index+1;
  alldataset=olddataset;
  newdatasetitem=[]; %  reset new dataset
  newBotitem=0;      %  reset botnet item counter
  newNormalitem=0;%  reset normal item counter
  totalNumber=0;
end
 end
end
```

E Publications

1. Alauthaman, M., Aslam, N., Hossain, M.A.and Alasem, R, "Investigation on Peer-to-Peer Botnet using TCP Control Packets and Data Mining Techniques". Seventh International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2013).
2. Alauthaman, M., Aslam, N., Hossain, M.A., Alasem, R. and Zhang, L. "A P2P Botnet Detection Scheme based on Decision Tree and Adaptive Multi-layer Neural Networks". Neural Computing and Applications. (Accepted)