Hierarchical TCP Network Traffic Classification with Adaptive Optimisation

by

Xiaoming Wang

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy
of
Loughborough University

3rd November 2010

Loughborough University

Thesis Access Form

Copy No……..……**Location**…….……..…

Author……..……..…….

Title……..

Status of access OPEN / RESTRICTED / CONFIDENTIAL

Moratorium Period:……years,ending……../……200…….

Conditions of access approved by (CAPITALS):……

Director of Research (Signature)……...……...

Department of……...……

Author's Declaration: *I agree the following conditions:*

OPEN access work shall be made available (in the University and externally) and reproduced as necessary at the discretion of the University Librarian or Head of Department. It may also be copied by the British Library in microfilm or other form for supply to requesting libraries or individuals, subject to an indication of intended use for non-publishing purposes in the following form, placed on the copy and on any covering document or label.
*The statement itself shall apply to **ALL** copies:*

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Restricted/confidential work: All access and any photocopying shall be strictly subject to written permission from the University Head of Department and any external sponsor, if any.

Author's signature…….Date……...……...……...

users declaration: for signature during any Moratorium period (Not Open work): ***I undertake to uphold the above conditions:***			
Date	Name (CAPITALS)	Signature	Address

Research Student Office, Academic Registry
Loughborough University, Leicestershire, LE11 3TU, UK
Switchboard: +44 (0)1509 263171 Fax: +44 (0)1509 223938

Loughborough University

Certificate of Originality

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgements or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a higher degree.

. . . . . .

Xiaoming Wang

3rd November 2010

Dedicated to my parents

Acknowledgements

First of all, I would like to thank my supervisor, Prof. David Parish, for his encouragement, guidance and inspiration. This work could not have been accomplished without his support and vital advice. More importantly, besides specific knowledge, my research ability has been greatly enhanced thanks to David.

I am grateful to my colleagues, Dr. Shiru De Silva, Dr. John Whitley, Dr. Konstantinos Kyriakopoulos and all other researchers in the High Speed Networks Group. Shiru gave invaluable input at the beginning of my research, and great support on Linux and Perl. Thanks to John for sharing his vast knowledge of networking and Linux. I also thank John and Konstantinos for their useful ideas, opinions and advice on this work.

I would like to acknowledge Martin Sykora for sharing his experience of data mining with me. Although we were working on completely different research fields, he provided lots of suggestions and support on data mining experiments.

I thank my friend Robert Archer for reviewing the manuscript and his extremely valuable comments.

Special thanks go to Yusi Liu from Alpari UK Limited and Dr. Jingbo Wang from Skyworks Solutions, Inc. for their constant encouragement and continued support.

Abstract

Nowadays, with the increasing deployment of modern packet-switching networks, traffic classification is playing an important role in network administration. To identify what kinds of traffic transmitting across networks can improve network management in various ways, such as traffic shaping, differential services, enhanced security, etc. By applying different policies to different kinds of traffic, Quality of Service (QoS) can be achieved and the granularity can be as fine as flow-level. Since illegal traffic can be identified and filtered, network security can be enhanced by employing advanced traffic classification.

There are various traditional techniques for traffic classification. However, some of them cannot handle traffic generated by applications using non-registered ports or forged ports, some of them cannot deal with encrypted traffic and some techniques require too much computational resources. The newly proposed technique by other researchers, which uses statistical methods, gives an alternative approach. It requires less resources, does not rely on ports and can deal with encrypted traffic. Nevertheless, the performance of the classification using statistical methods can be further improved.

In this thesis, we are aiming for optimising network traffic classification based on the statistical approach. Because of the popularity of the TCP protocol, and the difficulties for classification introduced by TCP traffic controls, our work is focusing on classifying network traffic based on TCP protocol. An architecture has been proposed for improving the classification performance, in terms of accuracy and response time. Experiments have been taken and results have been evaluated for proving the improved performance of the proposed optimised classifier.

In our work, network packets are reassembled into TCP flows. Then, the statistical characteristics of flows are extracted. Finally the classes of input flows can be determined by comparing them with the profiled samples. Instead of using

only one algorithm for classifying all traffic flows, our proposed system employs a series of binary classifiers, which use optimised algorithms to detect different traffic classes separately. There is a decision making mechanism for dealing with controversial results from the binary classifiers. Machining learning algorithms including k-nearest neighbour, decision trees and artificial neural networks have been taken into consideration together with a kind of non-parametric statistical algorithm — Kolmogorov-Smirnov test. Besides algorithms, some parameters are also optimised locally, such as detection windows, acceptance thresholds. This hierarchical architecture gives traffic classifier more flexibility, higher accuracy and less response time.

Keywords: *traffic classification, traffic identification, application detection, traffic characteristic, traffic feature, machine learning, data mining, artificial intelligence.*

Contents

List of Figures

List of Tables

Glossary

A

ACK Acknowledgment.

ANN Artificial Neural Network.

ATM Asynchronous Transfer Mode.

C

CRC Cyclic Redundancy Check.

CSMA/CD Carrier Sense Multiple Access with Collision Detection.

CSV Comma-Separated Values.

CWR Congestion Window Reduced.

D

DDoS Distributed Denial-of-Service.

DiffServ Differentiated Services.

DMZ Demilitarized Zone.

DNS Domain Name System.

DoS Denial-of-Service.

DPI Deep Packet Inspection.

E

ECE ECN (Explicit Congestion Notification) -Echo indicates.

F

FCS Frame Check Sequence.

FIN Finish.

FP False Positive.

FTP File Transfer Protocol.

G

GoS Grade of Service.

GUI Graphical User Interface.

H

HPC High Performance Computing service in Loughborough University.

HTML Hyper Text Markup Language.

HTTP Hypertext Transfer Protocol.

I

IANA Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IDS Intrusion Detection System.

IETF Internet Engineering Task Force.

IMAPS Internet Message Access Protocol.

IntServ Integrated Services.

IP Internet Protocol.

IPS Intrusion Prevention System.

IPv4 Internet Protocol version 4.

IRC Internet Relay Chat.

ISP Internet Service Provider.

K

k-NN k-Nearest Neighbour.

K-S Kolmogorov-Smirnov.

M

MAC Media Access Control.

MS-RDP Microsoft Remote Desktop Protocol.

MSN Microsoft Network.

MSS Maximum Segment Size.

MTU Maximum Transmission Unit.

N

NAT Network Address Translation.

O

OS Operating System.

OSI Open System Interconnection.

P

P2P Peer-to-Peer.

POP3 Post Office Protocol version 3.

PSH Push.

PSTN Public Switched Telephone Network.

Q

QoS Quality of Service.

R

RFC Request For Comments.

RST Reset.

RSVP Resource Reservation Protocol.

RTSP Real Time Streaming Protocol.

S

SMTP Simple Mail Transfer Protocol.

SNMP Simple Network Management Protocol.

SSH Secure Shell.

SVM Support Vector Machine.

SYN Synchronise.

T

TCP Transmission Control Protocol.

TCP/IP Transmission Control Protocol and Internet Protocol.

ToS Type of Service.

TP True Positive.

TTL Time To Live.

U

UDP User Datagram Protocol.

URG Urgent.

V

VoIP Voice over Internet Protocol.

W

WAN Wide Area Network.

CHAPTER 1

Introduction

1.1 Background

With the continuous development of information and communication technologies in the last few decades, computer networks have made a tremendous impact on our society. The Internet, which interconnects enormous computer networks, brings us a new way of living. We rely on it for business, education, entertainment, social activities, etc. As a result, efficiency and security of the Internet are critical not only for the industry itself, but also for the benefits of our whole society.

According to Moore's law, the processing speed of digital electronic devices grows exponentially, as well as computing performance per unit cost and network capacity [Moo65]. Nielsen argued that the bandwidth of the Internet available to end-users grows by 50% annually [Nie98]. The Internet industry is booming due to the low cost of communication and computing power. It has been reported that Internet traffic doubled every year in the past decade [MIN09]. At the same time, more applications and services have been introduced into this thriving market, which are demanding more network resources. In short, the amount of traffic and

number of applications delivered on the Internet is increasing every minute.

Although the bandwidth of networks is restricted by the network infrastructures and the development of semiconductor technologies, managing the expanding traffic more effectively could noticeably improve network performance. Since traffic classification could provide vital information for network management, building an accurate and efficient traffic classifier becomes our main objective. In addition, network traffic classification could also help improve network security, authorisation and accounting. In the following section, the motivation of developing traffic classifiers will be stated in detail.

1.2 Motivation

Network management covers management functions of configuration, fault tolerance, performance, security and accounting, which make networks more reliable, efficient and secure [Ram98]. Traffic classification could greatly assist network management in terms of performance, security, authorisation and accounting, all of which will be discussed in the following subsections.

1.2.1 Network Performance Management

1.2.1.1 Quality of Service

Traditional Public Switched Telephone Networks (PSTN) utilise circuit-switching networks, whose performance is called Grade of Service (GoS) and measured by rejected calls, noise, echo and so on. Since dedicated communication channels are allocated to users, the performance of the PSTN is independent among users. Due to the low bandwidth provided, normally only a single service runs on a PSTN connection, like voice or fax. As a result, it is rare to see multiplexing in a PSTN channel [LGW99] [ID06]. Packet-switching used in computer networks multiplexes the network traffic at the packet level, which provides variable bandwidth and efficiency of transmitting digital signals. However, the resources of packet-switching networks available to a single application could be affected by the usage of other applications running on the same network due to the nature of packet-level bandwidth sharing. It is very difficult to measure the general per-

formance of computer networks based on user experience. Besides the subjective attitudes, the experience of different applications cannot easily be quantified and compared. Therefore, we measure the network performance from the perspective of applications. The term 'Quality of Service' (QoS) is used for referring to the network performance related to the application requirements and the technologies that enable the network resources to be reserved for achieving the performance. The parameters normally used in QoS for measuring network performance are shown below:

Throughput
: This is the average rate of messages successfully delivered over a communication channel, usually measured in bits per second (bit/s or bps).

Delay
: This is the time taken for a bit of data to travel across the network between two endpoints.

Jitter
: This is expressed as the packet delay variation.

Packet Loss Rate
: This is the rate of packet loss occurring during transmission.

Packet Error Rate
: This is the rate of incorrect data packets transferred.

1.2.1.2 QoS in TCP/IP

Unfortunately, the commonly used TCP/IP protocol suite, known for providing best-effort services, which is dominant in computer networks and the Internet, does not have strong support for QoS. The Open System Interconnection (OSI) seven-layer reference model, the TCP/IP reference model, and the corresponding protocols are shown in Figure 1.1 [ITU94] [Bra89].

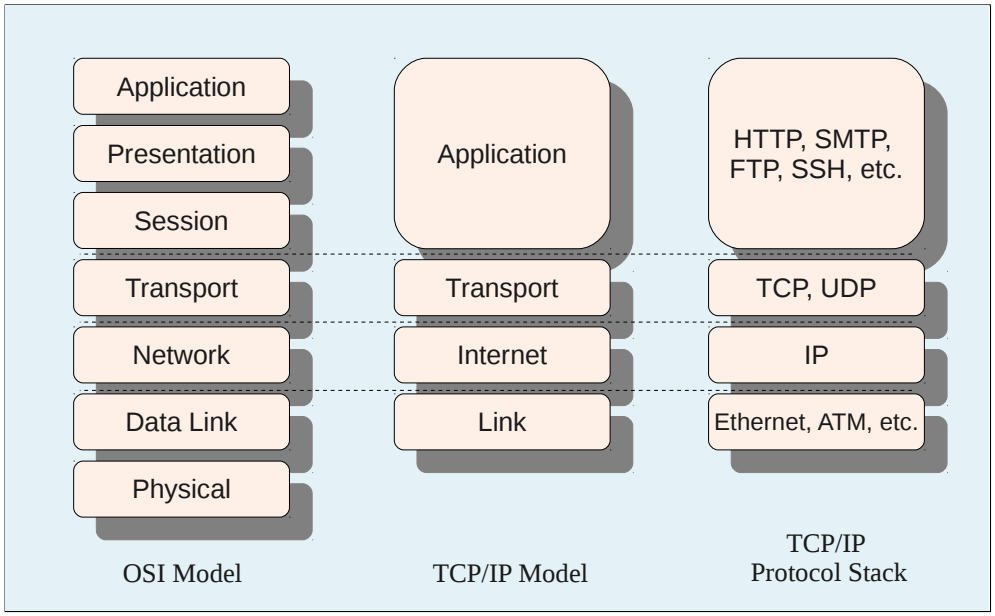

Figure 1.1: OSI and TCP/IP Reference Models and Protocol Stack

The transport layer protocols TCP and UDP do not provide strong QoS to the application layer. Although there is a Type of Service (ToS) field in the IP header, this field is not in common use [LGW99]. Recently, the IETF redefined the ToS field for supporting Differentiated Services (DiffServ) in RFC 2474. However, only coarse-grained traffic management based on traffic classes is provided in DiffServ [BBCD98], which cannot guarantee QoS for single applications. For the link layer, the Asynchronous Transfer Mode (ATM) only supports QoS at the virtual circuit level, which also cannot ensure network performance unless a virtual circuit is allocated to a special application stream [Bla98]. In addition, compared to Ethernet, high cost and lack of standardisation caused the slow deployment of ATM [Cla98]. On the other side, as a widely deployed link layer protocol, Ethernet employs Carrier Sense Multiple Access with Collision Detection (CSMA/CD) as the medium sharing scheme, which allows the hosts to send frames as soon as the medium is free. As a result, by using CSMA/CD, Ethernet cannot give fair access, let alone guarantee performance.

There is an approach defined in RFC 1633 for performing QoS on networks, which is called Integrated Services (IntServ). Unlike DiffServ, IntServ can support flow-level fine-grained QoS [BCS94]. It uses the RSVP protocol for sending the notification of resource reservation to network nodes. Network nodes residing in the flow transfer path could reserve network resources for providing guaranteed QoS as requested. However, not all routers support RSVP, and as an application layer

protocol, it also needs the support of application programmes.

Traffic shaping could be employed in network nodes for controlling the network traffic in order to manage network performance [BBCD98]. More specifically, at network nodes such as routers and gateways, traffic shaping takes some actions, such as delay or drop, on groups of packets for optimising network performance [IT04]. Traffic classification is needed when applying traffic shaping at the application level, which allows traffic shaping mechanisms to treat different kinds of traffic flows with different policies for optimising the QoS of applications [FH98]. After determining which classes the traffic flows belong to, network nodes process the different classes of traffic flows based on pre-defined profiles. For example, traffic flows belonging to real-time applications like VoIP or on-line games could obtain a higher processing priority than normal application flows when travelling through the network nodes.

Therefore, identifying the type of traffic flows is important for networks which provide differential services to applications. If the type of traffic flows can be determined, we can map the flows into different classes for providing different grades of service with DiffServ. Mapping traffic flows into ATM virtual channels could also achieve application level QoS. Traffic shaping based on applications can be implemented as well.

1.2.2 Network Security

The Internet has to face different kinds of threats with different purposes all around the world. A recent example of a high profile attack is that of a breached security system allowing access to details of approximately 5.6 million Visa and Mastercard accounts [Kat03]. This event shows that hackers have already got the ability to break into networked financial systems. It is not difficult to imagine that other financial organisations would be targeted by hackers as well, and our identities and monetary possessions are no longer safe. In addition, network attacks have been used as weapons by terrorists and hostile countries [PM04]. This could cause great economic loss, and even damage to national infrastructures such as communications, power supply, military command systems and so on. Therefore, great attention must be paid to the security of networks in order to protect us from cyber criminals.

1.2.2.1 Firewall

Firewalls are used in campus networks as standard for blocking unauthorised access to internal networks. The first generation firewall, known as a packet filters, was developed in 1988 [Mog89]. It filters packets based on a combination of source address, destination address, transport layer protocol and port number. The second generation firewall, known as a stateful filter, keeps track of the connection states of traffic flows, which could help prevent Denial-of-Service (DoS) attacks and attacks related to the TCP/IP stack [JC98]. The third generation firewall, known as an application layer firewall, could understand some protocols (such as DNS, FTP, etc.) based on packet contents [RSS01].

However, firewalls cannot guarantee network security. Packet filters can not handle IP spoofing or port spoofing. Although application layer firewalls can identify some applications based on packet contents, network performance is degraded by the inspection process [SHHP00]. Moreover, sometimes there is no significant signature in attacking packets, because the exploits used in the attacks may be system defects, network or OS misconfigurations, in which case a legitimate packet is indistinguishable from a malicious packet. In addition, networks may be compromised before firewall rules are updated.

1.2.2.2 Intrusion Detection Systems

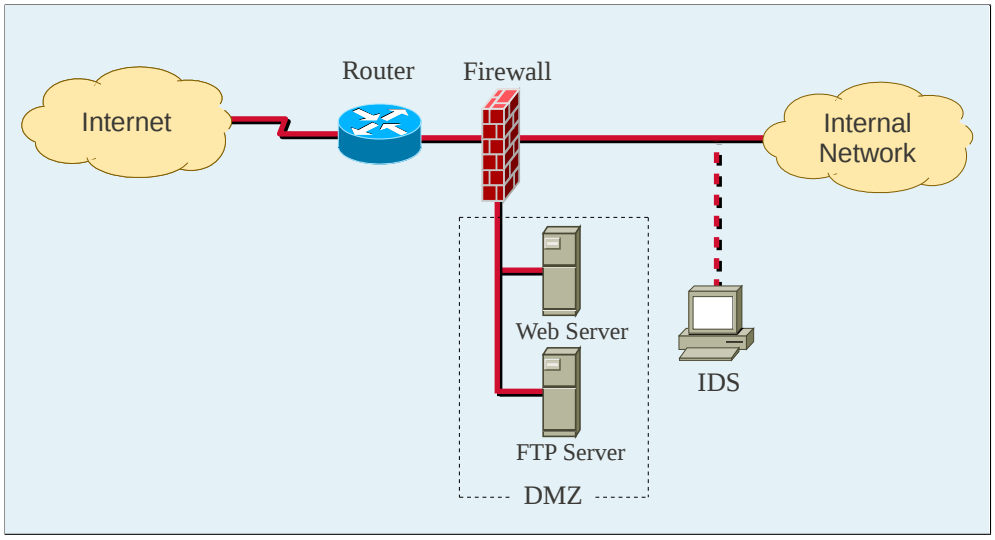

Figure 1.2: A Typical Campus Network with IDS

The Intrusion Detection System (IDS) has been invented for monitoring suspicious network and/or system activities [Den87]. A typical campus network with IDS is shown in Figure 1.2. The servers reside in the Demilitarized Zone (DMZ), which are treated with different policies to internal networks by firewalls. The hosts in the DMZ should be able to accept connections from the Internet for providing services like HTTP, FTP, and normally only necessary ports required by these services are opened for the DMZ. For internal networks, the firewall rules are more complex. Connections launched from outside should be examined in order to ensure only trusted connections are accepted. Due to different kinds of applications running on internal networks, connections launched by internal hosts should be allowed except in the cases of malicious ones. The port number does not help so much in this situation, as we cannot ensure the actual services are running with registered port numbers. For example, if an employee requests accepting SSH from outside to his hosts, then we create a firewall rule for opening port 22. However, he could use this port to accept connections from his friends for on-line gaming. The problem becomes more serious when dealing with outgoing connections. Although application layer firewalls could identify some types of traffic packets, encrypted traffic or unprofiled traffic creates difficulties.

As shown in Figure 1.2, IDSs are normally deployed in internal networks and are well protected, or they are configured as passive sniffers. Unlike firewalls, IDSs do not need to make on-line decisions of allow or deny for each packet or connection[1]. They mainly focus on analysing suspicious activities, which are found by the relationships between traffic flows, packets and system logs. IDSs can give further protection as intrusion activities can sometimes only be detected when combining these elements. They also have the ability to give administrators clues related to unknown system weaknesses [SM07].

IDSs being widely used today can reconstruct TCP flows, and examine the contents of flows more accurately than application layer firewalls, which are based on packet contents [Pax99] [Roe99] [Sou]. Since IDSs do not have responsibility for making real-time decisions, network performance is not affected. However, examining contents of flows will bring forth some issues related to privacy. Additionally, reconstructing TCP flows has high computational complexity and memory consumption. Advanced traffic classification could provide information about the traffic flows to the IDS. Ideally, traffic should be classified without examining the data fields. Consequently, the users' privacy will be protected and processing

[1]Intrusion Prevention System (IPS) is considered an extension of IDS, which can co-operate with firewalls to control access when predefined suspicious activities are detected [ZLZ04].

speed will be much faster.

1.2.3 Network Authorisation and Accounting

Since the type of traffic cannot be determined efficiently and accurately, network abuse always exists. For example, illegal traffic such as the downloading of copyrighted material in a company's network or the use of P2P software without the ISP's permission cannot be fully blocked due to port spoofing or encryption used for traffic masking. These kinds of traffic are normally resource-hungry. They not only break the organisation's policies, but also downgrade the network performance. As a result, network resources that are supposed to be reserved for business purposes are wasted, and the illegal traffic also confuses the network administrators' understanding of the network's capacity that is truly required by legal usage. Some ISPs are trying to throttle P2P file-sharing traffic because of its high bandwidth consumption [Bra08]. However, before applying differential policies to applications, they need to be identified.

Traffic classification can be applied for implementing accurate network authorisation based on applications, which only blocks illegal traffic flows including flows using port spoofing or encryption. It can also be used for fine grade traffic accounting at the flow level. Currently, network traffic is analysed and accounted based on port numbers, which is recognised as not being accurate enough. By using traffic classification engines, a true image of network usage can be obtained, which can be used as a consideration in upgrading network hardware, revising usage policies, market analysis, etc.

1.3 Contribution Highlights

Traditional network classifications are either based on layer 3 addresses and layer 4 ports in packet headers or contents transmitted in packets. However, information about addresses and ports can be easily forged, which means classification results become unreliable. On the other hand, TCP flows must be rebuilt before matching the contents with profiled traffic samples. This process needs a relatively high computational complexity and memory consumption. Moreover, the privacy of network users is threatened and encrypted traffic cannot be handled.

Recently, classifications based on statistical characteristics of traffic flows have been proposed by researchers. This work has explored the factors that affect performance of statistical classifications in terms of classification accuracy and speed. It has been proved that, classification methods, algorithms, detection windows and acceptance thresholds have the relations with classification performance.

A hierarchical system based on statistical classification has been proposed by this thesis, which is consisted by a series of parallel detectors for detecting different traffic classes separately and a mechanism for making final decisions. The system has been optimised by using different algorithms and parameters. Since the parallel detectors are independent with each other, they are optimised locally. A validating dataset is used for searching the optimised algorithms and parameters for detecting each kind of traffic. Then, our optimised model is tested by an unseen testing dataset.

Performance comparisons have been performed by testing the our proposed system and unoptimised classifiers proposed by other researchers. Since the datasets used for comparing the performances are identical, the performances are fully comparable. The results show that our optimised classifier has higher accuracy and less response time.

This work illustrates a practical approach of integrating different machine learning and statistical test algorithms into a single classifier. The independent parallel detectors enable classifier to take advantages of different algorithms and parameters when detecting different traffic classes. More flexibilities for deploying in the real world can be obtained due to the hierarchical architecture of the proposed system.

In summary, the work in this thesis provides a way of classifying network traffic with multiple optimised statistical algorithms and configurations, which has better performance than using only single global algorithms and configurations.

1.4 Chapter Outlines

This thesis is organised as follows.

Chapter 2 introduces the background of TCP/IP, which is related to traffic classification. Potential fields in the protocol stacks which can be used for traffic

classification are investigated. The methods of traffic classification are described, including current widely deployed methods and statistical methods. The pros and cons of these methods are also included in this chapter.

Chapter 3 describes the algorithms used in this work for statistical classification in details. Both parametric and non-parametric algorithms are considered. The details of feature extraction and similarity comparison are covered.

Chapter 4 covers the acquisition of datasets and preliminary tests, which includes the processes of obtaining the datasets from our test bed and some experimental preliminary tests for researching the performance of different algorithms and parameters.

Chapter 5 illustrates the architecture of proposed novice hierarchical classification. The processes of training, validating and testing are demonstrated, where the optimisation of algorithms and parameters are included. The implementation of the proposed system is also described.

Chapter 6 shows the result evaluations of the proposed system. The process of obtaining the optimised algorithms and parameters for detecting each traffic class are illustrated and the final classification results are listed. The process of optimisation is divided by two steps: firstly both parametric and non-parametric algorithms are considered separately; then overall optimisation is performed by selecting the best algorithms and parameters. The evaluations are also based on parametric optimisation, non-parametric optimisation and overall optimisation.

Chapter 7 illustrates the performance improvements of our proposed system compared to controlled experiments. Classifiers using a single algorithm and parameter are used as control classifiers for performance comparison.

Chapter 8 provides a summary of conclusions of this work and recommendations for future work.

CHAPTER 2

Traffic Classification Techniques

In this chapter, packet encapsulation and an overview of the TCP/IP protocol stack will be given initially. They will show which kinds of information contained in the network traffic are available for traffic classification. Different methods of traffic classification that have been implemented practically, together with their advantages and disadvantages, will be described in Section 2.3. Statistical traffic classification, which we mainly focus on, will be introduced in Section 2.4, where classifying traffic using statistical parametric values and non-parametric distribution tests will be discussed.

2.1 Packet Encapsulation

Referring to the TCP/IP reference model shown in Figure 1.1, information transmitted on networks using the TCP/IP protocol stack is encapsulated in three layers before being put on physical wires, which are the transport layer (OSI layer 4), internet layer (OSI layer 3) and link layer (OSI layer 2). TCP and UDP are transport layer protocols that are used to encapsulate the application layer (OSI layer 5–7) payload into TCP/UDP packets. IP, which acts as an internet

layer protocol, encapsulates TCP/UDP packets into IP packets and IP packets are encapsulated by link layer protocols. Link layer segments, commonly known as frames, can be captured entirely from networks for traffic classification. The encapsulation structure of a link layer frame, which can be captured from the wire, is illustrated in Figure 2.1.

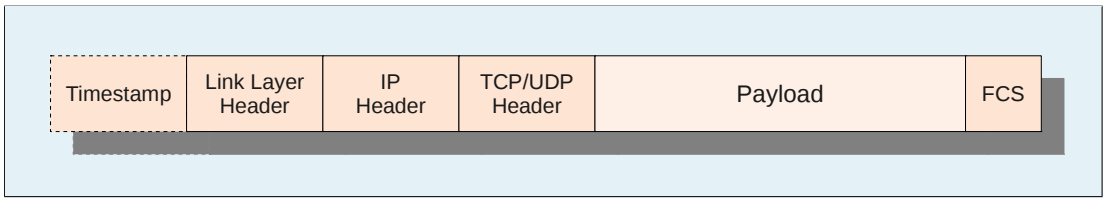

Figure 2.1: Captured Frame Structure

Normally, there is an additional time-stamp field added on by packet capturing software, which keeps records of the time when the packets were captured. Although this field of information is not actually generated by hosts and not being transmitted on the networks, it could be used for calculating packet inter-arrival time, which is highly related to the activities of applications [MZ05a] [CDGS07b] [TTNC02].

The link layer header contains the Media Access Control (MAC) address of the next network node that the packet travels to, and some controlling information related to the link layer. This information is irrelevant to applications, and it is changed step by step by network devices in the transfer path, as well as the link layer footer normally recognised as the FCS or CRC, which is used for verifying the data.

The IP and TCP/UDP headers contain much information for implementing traffic classification. Some fields are related to applications, such as port numbers, and some are useful for reconstructing flows. The issues related to IP and TCP/UDP protocols and their header fields will be discussed in the following subsections.

2.2 TCP/IP Protocol Stack Overview

2.2.1 IPv4

The internet layer uses the IP protocol to exchange packets, which provides host-to-host connectivity. Some header fields in IP packets may be changed when passing through OSI layer 3 devices, such as routers, gateways, etc. The structure of the IPv4[1] packet header is shown in Figure 2.2.

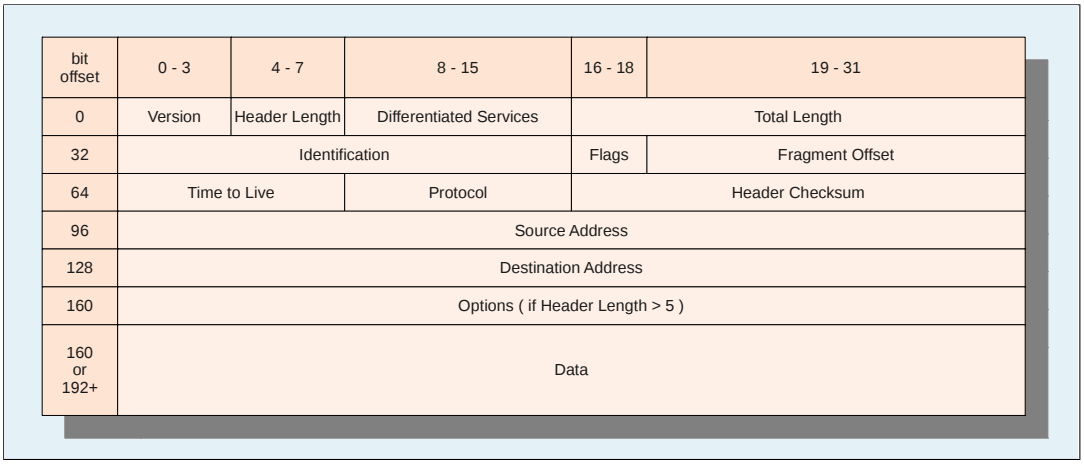

Figure 2.2: IP Packet Header

The *Version* field is fixed based on the version of the IP stack used by the host's operating systems, which is *4* for IPv4. The *Header Length* and *Total Length* are quite useful for traffic classification, and can be used to calculate the length of the application layer payload. It is recognised that there are relations between the type of applications and the payload length, which will be discussed in detail in Section 2.4 [PBL+03] [MZ05a] [CDGS07b] [TTNC02]. The *Differentiated Services* field is defined in [BBCD98]. These bits can be used for indicating the types of the traffic or the priorities of the traffic. However, it is tagged by operating systems rather than applications. Besides the lack of bits for representing the applications, it can be easily forged by malicious users. Consequently, we do not take it into consideration for traffic classification. The *Identification*, *Flags* and *Fragment Offset* fields are used for the fragmentation and reassembly of IP packets. Fragmentation and reassembly occur when the packet size is larger than the Maximum Transmission Unit (MTU) of the physical networks that the packets are travelling to. Ethernet is widely deployed for end-users' access, which uses an

[1]IP refers to IP version 4 in this thesis if not stated otherwise.

MTU of 1500 bytes, and is supported by most physical networks. In addition, fragmentation can downgrade network performance. As a result, it is rare to see fragmentation used nowadays, and is not our concern. The *Time to Live* (TTL) field contains a number which indicates how many hops the packet has been travelled through, which is changed by each router. The header checksum is changed as well. Except for identifying Distributed Denial-of-Service (DDoS) with TTL, these two fields are not useful for traffic classification. The *Protocol* field indicates the transport layer protocols used by the packets, such as TCP, UDP, ICMP, etc. It can be used for further analysis of the packets. The *Source Address* and *Destination Address* are IP addresses of the packets' source and destination. Although they may be changed due to NAT, they are kept the same on public networks before entering firewalls, and they may provide information about the applications. This will be discussed in Section 2.3.2 and Section 2.3.3. IP addresses are part of TCP sockets, which are used for distinguishing traffic flows. Therefore, they must be used for reassembling packets into traffic flows. The variable length of the *Options* field contains some optional information related to routing, which is not taken into consideration for traffic classification either.

To sum up, *Header Length* and *Total Length* are useful for payload length calculation, the *Protocol* field is needed for further decapsulation and analysis, and *Source Address* & *Destination Address* may provide some information about the traffic and they must be used for reassembling packets into traffic flows.

2.2.2 TCP

As a transport layer protocol, Transmission Control Protocol (TCP) provides connectivity between two processes. Reliable and ordered transmission of data streams can be guaranteed by using TCP, which is known as a connection-oriented protocol. The structure of the TCP header is shown in Figure 2.3. The fields of the header will be discussed briefly, followed by some mechanisms of flow control that are related to traffic classification.

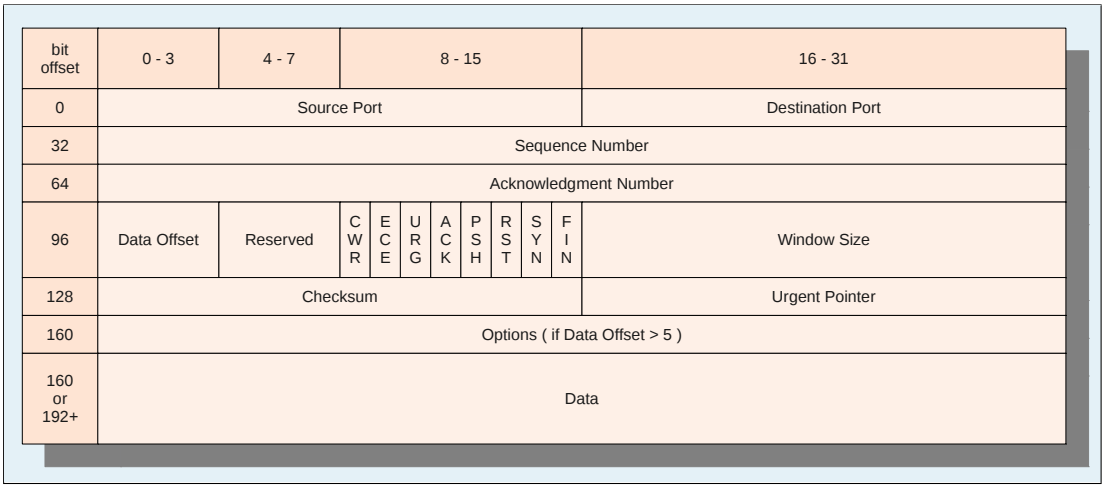

Figure 2.3: TCP Packet Header

Hosts for transferring and receiving network traffic distinguish different flows based on port numbers, and pass flows to proper processes according to port numbers. A combination of a *Source Port*, a *Destination Port*, a *Source IP Address* and a *Destination IP Address* is called a socket pair, which is a unique identification of a traffic flow in a network. Due to applications sometimes using special port numbers, traffic may be identified based on port numbers, which will be discussed more specifically in Section 2.3.1. In order to ensure reliability and the correct ordering of data, *Sequence Number* is used for identifying each transferring byte of data, and the *Acknowledgment Number* for acknowledging each successfully-received byte. These two 32-bit fields are needed when reassembling traffic flows. Because of the variable length of the *Options* field, *Data Offset* is needed for indicating the boundary of the packet header and application payload, which is required for calculating the length of application payload.

Following the *Reserved* field are eight one-bit flags, some of which are used for maintaining the connections and some of which are used for flow control. As a connection-oriented protocol, TCP establishes a connection before transferring data and terminates the connection when transferring is completed. It uses a three-way handshake to establish a connection, and a four-way handshake or three-way handshake to terminate a connection. The *SYN* and *FIN* flags are for handshaking when establishing and terminating a connection. The *RST* flag indicates to the remote host to disconnect a connection immediately without a handshake. *PSH* tells the recipient host to pass the content of packets to applications as soon as possible. *ACK* indicates that there is a valid acknowledgment carried in the field of *Acknowledgment Number*. *URG* together with *Urgent Pointer* allow applications

to send out-of-band data. It has been argued that *PSH* and *URG* can be used for traffic classification [MZ05a]. However, although applications request the setting of *PSH*, different OSs use different ways for dealing with the flag. Some implementations of the TCP stack ignore the applications' requests, some of them set the bit based on their buffer [Mic] [IBM]. Consequently, we do not do traffic classification with *PSH* flag. Because applications rarely use out-of-band data [GY08] [Tan03], it is meaningless to take *URG* into consideration for traffic classification as well. *CWR*, *ECE* and *Window Size* are used for traffic control, and their settings are depending on the implementation of the TCP stack and the OS's buffer management. The optional field of *Options*, which includes timestamp, maximum segment size, window scale, etc. for extra functionality or enhancing performance is also dependent on the implementation of the TCP stack.

The fields mentioned above related to OS or TCP implementation could be used as a fingerprint for determining the type of operation systems [Fyo98], but they cannot be used for application layer traffic classification. The TCP mechanisms of flow control and Nagle's algorithm may alter some parameters discussed above, which may affect the traffic classification. They will be considered in the following subsections.

2.2.2.1 Flow Control

TCP controls traffic flow using a sliding-window mechanism. For the purposes of preventing host buffer overflows, hosts advertise the window size in packets, which indicates how many bytes can be sent by the remote host without acknowledgment. Delayed acknowledgment as defined in [Bra89] allows the receiver to send piggyback ACKs, which includes ACKs in normal data packets instead of sending separate ACK packets. This mechanism prevents receivers sending ACKs immediately after receiving a packet. If the receiver has no data to send, it will send an ACK after 500 ms. Besides the hosts' buffer sizes, TCP window size also depends on the buffer sizes of intermediate network nodes, the capacity of the links and the traffic load of the links. In order to fully utilise the bandwidth, slow-start, congestion avoidance, fast retransmit and fast recovery algorithms are employed in modern TCP implementations, which maintain a congestion window [Ste97]. The actual amount of data allowed for sending without acknowledgment depends on the received window size and congestion window, which is given by

$$allowed\_window = min(received\_window, congestion\_window)$$

When a connection is established, slow-start increases the congestion window exponentially until a predefined threshold is reached, and then congestion avoidance increases the congestion window linearly until a packet loss is detected. Fast recovery adjusts the congestion window when a loss of packets is detected. Fast retransmit lets the sender retransmit a packet if the acknowlegment is not received in an estimated round-trip time.

2.2.2.2 Nagle's Algorithm

Nagle's algorithm, named after its creator John Nagle, is a mechanism for improving TCP efficiency by coalescing a number of small buffered messages [Nag84]. Some applications repeatedly emit data in small chunks, sometimes 1 byte in size, which wastes network resources due to a large proportion of packet overhead (4000%). For transferring 1 byte of useful information in a packet, at least 40 bytes of header is needed (20 bytes for IPv4, 20 bytes for TCP), which yields only 2.4% efficiency. In addition, lots of small packets in transit at the same time may cause intermediate network nodes to be overloaded by processing, reordering or dropping. Nagle coined the term 'congestive collapse' for this phenomenon.

The algorithm works by coalescing small outgoing messages, and sending them all at once. More specifically, the sender keeps buffering messages until the messages can be encapsulated into a full packet, which is worth sending at once. Additionally, the buffering time is restricted to 200 ms for keeping a reasonable response time.

Nagle's algorithm can increase TCP efficiency when applications generate lots of small message chunks, such as Telnet sessions. However, it increases the response time. Coalescing messages also mask the original message length, which adds difficulty in terms of identifying traffic flows when classifying traffic based on packet size.

2.2.3 UDP

Unlike TCP protocol, the User Datagram Protocol (UDP) is a stateless transport layer protocol, which transfers data without establishing a connection. Hence, there are no hand-shaking dialogues for establishing or terminating a transmission

process. UDP does not guarantee reliability, ordering or integrity. The data carried by UDP may be corrupted, out of order, duplicated or may even go missing without notice. However, UDP packets have a smaller overhead, which increases network efficiency. Because there are no traffic control or congestion avoidance algorithms, UDP has lower latency than TCP due to simpler processing. The structure of a UDP header is shown in Figure 2.4.

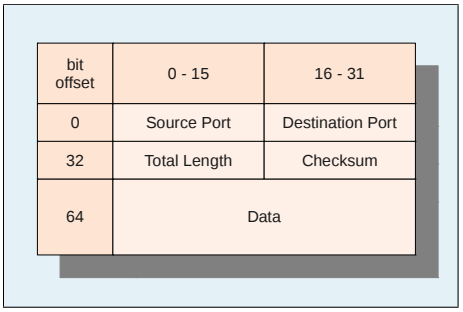

Figure 2.4: UDP Packet Header

Like TCP, *Source Port* and *Destination Port* may be used for traffic classification, as well as packet size and packet inter-arrival time. Some researchers have shown that, distributions or statistics of packet size and packet inter-arrival time could be used for UDP traffic classification [HP08] [CLW03] [PBL+03]. Since there is no traffic control and no Nagle's algorithm in UDP, packet size and packet inter-arrival time correspond with applications' actual behaviour, which has not been affected by traffic control algorithms. Hence, classifying UDP traffic is easier than classifying TCP traffic. Considering TCP traffic consumes more than 80% of the WAN resources [ARZ09], this thesis mainly focuses on classifying TCP traffic.

2.3 Current Classification Methods

2.3.1 Port Number

As we discussed in the last section, both TCP and UDP have 16-bit integer fields for source port and destination port. Servers run different services on different ports. Both servers and clients distinguish traffic flows by socket pairs, which are a combination of source port, destination port, source IP address and destination IP address. Therefore, clients must know the servers' port of a specific service before establishing a connection with TCP or starting to exchange data with UDP.

Fortunately, the Internet Assigned Numbers Authority (IANA) is responsible for assigning port numbers to applications. Then, clients can connect to servers with specific ports, which belong to desired applications. As a result, port numbers can be used for identifying applications [GM01]. The IANA port assignments are divided into three ranges, which are listed in Table 2.1 [IAN10].

Table 2.1: IANA Port Assignments

Range	Port Numbers
Well Known Ports	0 – 1023
Registered Ports	1024 – 49151
Dynamic and/or Private Ports	49151 – 65535

The *Well Known Ports* are assigned to commonly used services, such as Telnet (port 23), SMTP (port 25) and HTTP (port 80). On most operating systems, this range of ports can only be used by system (or root) processes or processes executed by privileged users. For the purpose of ensuring the access of basic services, ports in this range are not normally used for purposes other than IANA's definition. Therefore, identifying traffic based on *Well Known Ports* has a relatively high rate of precision. However, a service could still be configured to use a different port than that which it is assigned to. For example, the HTTP service could be configured using port 8080 instead of port 80. In addition, unused *Well Known Ports* in local hosts may be allocated to some applications, that use dynamic ports [RP94]. Furthermore, malicious traffic can be intentionally masked easily by port spoofing. Consequently, identifying traffic with *Well Known Ports* cannot guarantee accurate results.

The *Registered Ports*, ranging from 1024–49151, are normally configured to be used by ordinary processes or programmes on most operating systems. Companies register port numbers for their programmes with IANA for dedicated use. As a result, numerous programmes can communicate on the Internet straight away with their registered ports. However, registering ports with IANA is only for the convenience of the community, which is not forced. Therefore, unregistered programmes may arbitrarily use ports within this range, or they may use random ports which lie in this range. In addition, even port numbers of registered programmes can be changed by users. For example, TCP port 1993 has been assigned to Cisco SNMP. A TCP flow communicating with port 1993 cannot be 100% confirmed as being a Cisco SNMP flow, because other programmes may be

manually configured to use this port or use that port by random selection. We cannot also assume that all Cisco SNMP traffic uses TCP port 1993, due to it being configurable.

It is impractical for every programme to be assigned a port number, because of the limited number of ports and the increasing large number of applications. Additionally, some programmes require dynamic port allocation. These programmes only use port numbers temporarily for the duration of a single connection. After the communication session has finished, the ports can be reused. Therefore, IANA reserves some port numbers as *Dynamic and/or Private Ports*, which range from 49151–65535. These ports could be used by unregistered programmes without conflicting with well known services or registered services. They could be used by programmes that need ephemeral ports.

Although it has been reported that classification accuracy can reach around 70% based on port numbers [KBB+04], applications using reconfigurable port numbers, unregistered port numbers and dynamic port allocation still exist. Moreover, intentional port spoofing is commonly used by malicious traffic. Therefore, traffic classification based on port numbers is not reliable, especially for classifying illegal traffic.

2.3.2 Packet Classification

Besides source and destination port numbers from the transport layer, some other information in the packet header has been used for filtering network packets, including source and destination IP addresses and TCP flags. Practically, a list of rules are applied for packets in order to employ different access policies to different sorts of packets. This technique cannot be considered as traffic classification, but it can be used for detecting malicious packets. The first generation firewalls, packet filters, mentioned in Section 1.2.2 benefit from packet classification [Cis02]. For example, SSH connections from outside networks should not be present on a Web server, else there is probably an attack occurring. Therefore, a rule that blocks packets with destination port 22 from outside IP addresses to the Web server can be set in firewalls in order to filter potentially malicious packets.

Packet classification could be employed for providing address based QoS to some degree. Due to fast processing speeds, nowadays it is normally deployed in network

cores. It groups packets based on the range of IP addresses, then applies different policies [MF01].

Additional information of IP addresses may help for identifying some applications. For example, packets with the destination address of Google.com and TCP port 80 have a high confidence of being HTTP traffic. For identifying other traffic, the same issues as with using port numbers are present.

2.3.3 Stateful Inspection

By grouping TCP packets based on sockets and examining the TCP flags, TCP traffic flows can be rebuilt. More specifically, TCP flags SYN and FIN used for handshaking are tracked. Hence, inspection systems have the awareness of establishment and termination of TCP connections. Although UDP is a connectionless protocol, UDP traffic flows can be rebuilt as well based on UDP sockets. However, stateful inspection does not reorder TCP packets or deal with duplicated packets.

After traffic flows are obtained, the payload of the application layer can be examined. Services following the standard document normally have well-known distinct signatures that are used for identifying applications. Through protocol analysis, undocumented proprietary applications or protocols can also be identified as long as distinct patterns could be found in the packets. Lots of traffic could be classified with this approach, because protocols always use some magic numbers to identify themselves. Take HTTP version 1.1 as an example, the string *HTTP/1.1* is always included in the first response packet from the server.

Stateful inspection has the ability to examine the information retrieved from packets at OSI layer 3–7, which gives a high rate of accuracy [Raj05]. However, the application payload cannot be examined beyond packet boundaries, because the packets are not reordered, and duplicated packets may exist. In addition, matching packet contents with the database of application signatures has high computational complexity. Furthermore, it cannot handle encrypted traffic flows, and there may be privacy issues involved.

2.3.4 Deep Packet Inspection

Deep Packet Inspection (DPI) is currently used widely by numerous enterprises, ISPs and governments in a variety of applications [Ben09], which enables network equipment to perform full content inspections. Instead of examining on a packet by packet basis, DPI can inspect TCP flows as a whole by reassembling TCP packets into flows with the consideration of out-of-order packets, error packets and duplicated packets.

The first step is the same as for stateful inspection. DPI tracks TCP flows with the help of TCP flags and sockets. Secondly, packets are sent to a reorder engine for sorting packets into their original order based on sequence numbers in the TCP header. Then, the whole contents of two flows can be obtained, which are the flow of client to server and the flow of server to client. Flow contents could be matched with the signature database, and different policies could be applied for matched flows.

The matching processes have higher precision than only using packets contents, because signatures contained in the flows could be matched even if they are split by packets. Signatures longer than packet size can be used, which would also increase accuracy. The patterns used by the L7-filter for matching FTP traffic using regular expressions are shown below [l7p09]. From the documents of the software, using longer patterns for matching with more packets could obtain higher precision [l7h09].

FTP Matching Pattern for 1 Packet:

```
^220[\x09-\x0d -~]*ftp
```

FTP Matching Pattern for 2 Packets:

```
^220[\x09-\x0d -~]*\x0d\x0aUSER[\x09-\x0d -~]*\x0d\x0a
```

FTP Matching Pattern for 3 Packets:

```
^220[\x09-\x0d -~]*\x0d\x0aUSER[\x09-\x0d -~]*\x0d\x0a331
```

Many applications can be detected by DPI. Currently 112 applications are supported by the L7-filter[2], including traditional services such as FTP, DNS, HTTP,

[2] L7-filter is a open source DPI tool.

etc. and newly released applications like games, VoIP, etc. [l7p09] Additional application support can be added by customising the patterns.

However, the processing speed is slowed by reordering packets and longer matching patterns. This not only increases the computational complexity, but also increases memory consumption [PL06]. Additionally, analysing protocols manually for getting the matching patterns is recognised as being difficult. Overmatching and undermatching still exist even when using relatively long patterns. As with stateful inspection, DPI also has issues related to privacy and encrypted traffic.

2.4 Statistical Classification Methods

The current traffic classification methods discussed in Section 2.3 have their advantages. Content based classification methods have high accuracy, and packet header based classification methods have fast processing speed. However, classifying traffic using the contents of packets or flows requires high computing power and lots of memory for on-line examination. Profiling traffic into matching patterns is also difficult, and encrypted traffic cannot be classified. Moreover, in some circumstances, inspecting the payload of the application layer is prohibited due to privacy issues. For packet header based classification methods, port spoofing and IP spoofing decrease the accuracy, and packet headers normally do not provide enough information.

The central idea of statistical classification is identifying traffic flows using their statistical characteristics instead of examining the application layer contents. Hence, tunnelled or encrypted traffic may be identified [CDGS07a]. Since reordering packets are not needed and the matching process is simpler, statistical methods have a higher processing speed than DPI. It does not depend on port numbers or IP addresses, which could be forged. Moreover, profiling sample traffic flows is much easier. As a result, the work of this thesis has focused on statistical classification methods, and the optimisation related to them.

There are two approaches which identify traffic with different statistical features and algorithms. An overview of these two approaches will be given in the following subsections.

2.4.1 Parametric Classification

Parametric classification groups packets into flows based on TCP flags and TCP sockets. It might not reorder the packets and not consider duplicated packets, because out-of-order or duplicated packets do not greatly change the statistical characteristics of traffic flows. This could increase the processing speed and require less memory. A series of statistical values can be extracted from each flow. Therefore, flows could be classified based on training samples using a variety of algorithms. There are two things that we need to be concerned with: one is what kinds of parametric values should be used to discriminate the traffic flows; the other is the classification algorithms.

The parameters, known as 'attributes' in data mining, are normally calculated based on the information in the packet headers, which avoids breaking the privacy of network users. Moore *et al.* have proposed a list of 249 possible attributes, which could be used as discriminators for flow-based classification [MZ05a]. Some of them are extracted from the information of the data link layer, such as mean number of bytes in Ethernet frames. Some of them are calculated from the data highly depending on TCP/IP implementations, such as the count of the packets with PSH, average advertised window size, etc. As discussed in Section 2.2, these kinds of attributes are not actually dependant upon the classes of applications, but they may increase the classification accuracy. For example, the accuracy of detecting some games may be increased with the help of these attributes, because some games are probably only published on specific OSs, and these attributes include the fingerprint of OSs. In this thesis, for calculating the attributes, we only use the information directly related to applications, which are derived from packet size and packet inter-arrival time. The attributes we used will be described in Section 3.1.

The period for calculating the statistical values should also be considered; referred to as the detection window in this work. It will be shown that higher accuracy is not definitely brought about by a longer detecting period in Section 4.3, and a small detection window could give quicker classification speed. Therefore, optimising the detection window will become one of our main objectives.

After the attributes have been determined, every flow can be described by a multi-dimensional vector with a class name. As a result, the candidate flows could be classified based on a training set. This is a generic supervised classification problem

in the field of data mining. Different machine learning algorithms have been proposed for performing this job, such as Bayesian networks [MZ05b], k-nearest neighbours [SSM07], decision trees [WY08] [LM07], artificial neural networks [TC97] [NSV06] and SVMs [Zho08]. In this thesis, several algorithms are employed, which will be described in Section 3.2.

For content-based traffic classification, matching patterns should be discovered in advance through manual protocol analysis. Whereas, for parametric statistical classification, the training set consists of sample instances, which could be generated by sample traffic easily. A sample instance consists of a vector and a known class name, and a testing instance has the vector only.

2.4.2 Non-parametric Distribution Test

The other approach is distinguishing the traffic flows by non-parametric distributions instead of using numeric values in the parametric classification method. Parish *et al.* have argued that packet size distributions are distinct among different applications using UDP, and they can be used for identifying UDP traffic flows [PBL+03]. This approach has also been implemented for TCP traffic with the consideration of the effects caused by Nagle's algorithm by Bo Li in his Ph.D. thesis [Li07].

A packet size distribution can be built for each flow, which represents the frequency spectrum of the packet payload sizes appearing in the flow. The possible packet sizes are divided into bins, and the numbers of packets whose sizes fall into these bins are counted. Then, a normalised packet size distribution can be calculated for matching with a database, which is expressed as packet size bins versus normalised possibilities. The bin size is a factor which may affect the identification accuracy. Larger sized bins have a higher tolerance for distribution changes within classes, and smaller sized bins could distinguish more classes but have a lower tolerance. For example, the first few packets of HTTP flows may be similar, because software should follow the standard of the HTTP protocol. The sizes of these packets are probably not identical, due to the different implementations of the HTTP protocol by different software. Increased bin size may have higher tolerance of these differences, but eliminate some statistical characteristics. For simplicity, we use the smallest unit (one byte) per bin in our experiments. However, a genetic algorithm has been proposed, which has been proven to optimise the bin sizes

[SSM07].

Comparing the candidate distribution with distribution profiles in the database is another issue. There are some statistical algorithms which could be used. Correlation coefficient, the nearest neighbour with Euclidean distance and Chi-square test have been employed and tested in [PBL+03]. All of them could fulfil the task of identifying the packet size distributions very well, and have similar performance in terms of accuracy. As a result, we do not pay too much attention to the distribution comparison algorithms. The Kolmogorov-Smirnov (K-S) test has been used in the work of this thesis, which will be described in Section 3.4.

However, some network traffic generated by different applications may share the same distribution patterns, which are not distinguishable by this approach. By using distributions, it also eliminates the information of the flows in the time domain, which may help traffic classification. Moreover, since the similarities between candidate distributions and all distributions in the database must be calculated one by one, the comparison process needs more computational power than some algorithms using parametric values, which only process the classification once with a trained model. Nagle's algorithm may alter the distribution patterns by coalescing small packets [Li07]. Also, the same issue of detection window as parametric classification still exists, the number of packets for building the distribution would affect the classification accuracy. Optimising the detection window will increase the processing speed and may increase the accuracy, which will be discussed in Section 4.3.

2.5 Summary

In this chapter, packet encapsulation has been presented for the purpose of finding out what kind of information can be retrieved from the captured packets. Then an overview of TCP/IP has been presented, which includes the analysis of the relationships between application behaviours and packet contents. Finally, current traffic classification methods and statistical traffic classification methods have been discussed briefly.

Since optimising the statistical traffic classification is our main concern, the features and algorithms we used in this work will be presented in more depth in the next chapter.

CHAPTER 3

Algorithms for Statistical Classification

Optimising traffic classification using statistical methods is our main objective, which includes employing multiple algorithms. Therefore, this chapter covers the fundamentals of the classification algorithms that we used in this work. The attributes and the classification algorithms used by parametric classification will be presented, followed by the distributions and the distribution comparison algorithm used by non-parametric classification.

3.1 Attributes for Parametric Classification

As we discussed in Section 2.4.1, the statistics of each flow can be represented by a multi-dimensional vector. The dimensions of the vector are called attributes in the field of data mining. The attributes related to traffic classes should be discovered for classification.

Redundant attributes can be removed using data mining techniques. Some attributes are not directly related to application behaviour, which differs between OSs, protocol stacks etc., which may increase classification accuracy. Hence, using

more attributes, better classification performance might be obtained in practice. Although 249 statistical attributes for traffic classification have been proposed in [MZ05a], most of them are not directly related to the behaviours of specific traffic classes. Some statistical attributes described in [MZ05a] are calculated from layer 2 frame sizes, IP and TCP options, TCP flags, TCP window sizes, etc. As we discussed in Section 2.1 and Section 2.2, these kinds of information comes from layer 2 hardware or TCP/IP implementations instead of applications. Considering this work does not involve optimising the attributes, only 26 attributes that are directly related to application behaviour are used in our research, which are derived from packet size and packet inter-arrival time as discussed in Section 2.4.1. This will give us a experiments faster, easier evaluation and more constant classification performance.

Referring to Section 2.2, although TCP traffic control and Nagle's algorithm may alter the packet inter-arrival time and packet size, they are still claimed as having strong relations with the application activities by many researchers [DO01] [PBL+03] [MZ05a] [Li07]. Therefore, the attributes used for parametric classification are calculated based on these two parameters, which are shown in Table 3.1.

TCP flows can be split into two directions, which are defined as *client→server* and *server→client*. Since we do not actually know who is the client or server, the host who initiates the connection is arbitrarily defined as client in this work. The first two attributes, *pc_ratio* and *bc_ratio*, stand for the ratio of packet count and byte count between these directions. The remaining attributes can be divided into three parts, where eight statistics retrieved from each direction or both are included. Packet size and packet inter-arrival time are used for calculating these statistical attributes.

Every TCP flow can be described by a vector with these 26 dimensions, together with a class name. Sampling flows with known class names are used as training data. Then, network traffic classification can be turned into supervised classification of these vectors in a hyper-space.

3.2 Algorithms for Parametric Classification

For the purpose of classifying the vectors that stand for traffic flows, the following supervised classification algorithms are used in this work — k-nearest neigh-

Table 3.1: Attributes Used in Parametric Classification

Short	Long	Direction
pc_ratio	Ratio of packet count between two directions	n/a
bc_ratio	Ratio of byte count between two directions	
ps_min	Minimum packet size	both
ps_max	Maximum packet size	
ps_avg	Average packet size	
ps_var	Variance of packet size	
it_min	Minimum packet inter-arrival time	
it_max	Maximum packet inter-arrival time	
it_avg	Average packet inter-arrival time	
it_var	Variance of packet inter-arrival time	
ps_cs_min	Minimum packet size	client→server
ps_cs_max	Maximum packet size	
ps_cs_avg	Average packet size	
ps_cs_var	Variance of packet size	
it_cs_min	Minimum packet inter-arrival time	
it_cs_max	Maximum packet inter-arrival time	
it_cs_avg	Average packet inter-arrival time	
it_cs_var	Variance of packet inter-arrival time	
ps_sc_min	Minimum packet size	server→client
ps_sc_max	Maximum packet size	
ps_sc_avg	Average packet size	
ps_sc_var	Variance of packet size	
it_sc_min	Minimum packet inter-arrival time	
it_sc_max	Maximum packet inter-arrival time	
it_sc_avg	Average packet inter-arrival time	
it_sc_var	Variance of packet inter-arrival time	

bour, decision trees and artificial neural networks. The reason of selecting these three algorithms is because k-nearest neighbour is the simplest machine learning algorithm, artificial neural networks is the most sophisticated algorithm and decision trees is proposed by many other researchers used for classifying network traffic. They will be described briefly in the following subsections.

3.2.1 k-Nearest Neighbour

A kind of instance-based classifier has been defined, named k-Nearest Neighbour (k-NN), which classifies vectors based on the closest training examples in the attribute space. Firstly, the k-NN selects k cases from the examples, which are

the closest to the target case in terms of vector distance. Usually, Euclidean distance is used as the measurement of the distance between vectors, given by Equation 3.1.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{3.1}$$

Then, a vote is made among these k cases. Finally, the solution to the target case is given by the result from the voting. An example of k-NN dealing with a two-dimensional classification is illustrated in Figure 3.1.

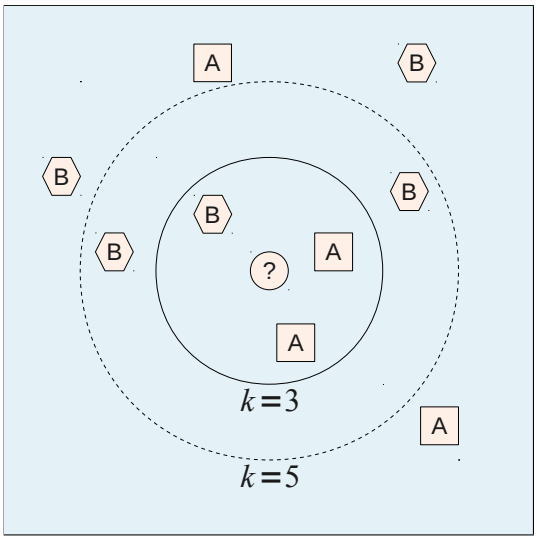

Figure 3.1: An Example of k-NN Classification

To determine the class of the case in the centre, k is chosen first. In the case shown in Figure 3.1, while $k = 3$, three previous cases are selected. As a consequence, the target case will be tagged with class A, since there are two class A cases among the selected three nearest neighbours. The situation changes when five closest cases are chosen, as circled by the dashed line in the figure, where class B is the output. It is not difficult to work out that large k reduces the noise but makes class boundaries less distinct and vice versa.

A k-NN classifier can operate without an actual training process, but it still relies on a stored training set for classification. Since a distance between the candidate vector and every sample in the training set must be calculated, the real-time classification speed is not very fast, especially when the training set is large.

3.2.2 Decision Trees

The process of constructing a decision tree for classification can be expressed recursively. First, an attribute is selected to be placed at the root node and branches for each possible value range. The training set is split into subsets, one for each possible value range of the attribute. Then, the process is repeated recursively for each branch, using only the instances that reach the branch. If all instances at a node have the same class, the developing process is stopped at that node [WF05a].

For numeric attributes, it is common to set the split points at the halfway between the values that delimit the boundaries of two classes. For recursively building the nodes and branches for efficient decision trees, we must determine which attributes should be selected as nodes, and which split points should be used as branches. Entropy impurity has been defined in Equation 3.2 for exhaustive searching of the nodes and split points, where n is the number of classes and P_i is the probability of ith class.

$$i = -\sum_{i=1}^{n} P_i \log_2 P_i \qquad (bits) \tag{3.2}$$

The process of constructing a tree can be demonstrated in Figure 3.2, where the dataset has six samples of A, five samples of B and there are 2 dimensions for each instance — x_1 & x_2. The entropy impurity of the initial dataset can be calculated by

$$i_0[6,5] = -(\frac{6}{11} \times \log_2 \frac{6}{11} + \frac{5}{11} \times \log_2 \frac{5}{11}) = 0.9940\ bits$$

In Figure 3.2 (a), x_1 has been selected as the root node and 0.3 has been selected as split point. The entropy impurity of each leaf can be calculated by

$$i_{a1}[3,1] = -(\frac{3}{4} \times \log_2 \frac{3}{4} + \frac{1}{4} \times \log_2 \frac{1}{4}) = 0.8113\ bits$$

$$i_{a2}[3,4] = -(\frac{3}{7} \times \log_2 \frac{3}{7} + \frac{4}{7} \times \log_2 \frac{4}{7}) = 0.9852\ bits$$

Then, the average of these two leaves can be calculated by

$$i_a([3,1][3,4]) = \frac{4}{11} \times i_{a1}[3,1] + \frac{7}{11} \times i_{a2}[3,4] = 0.9220\ bits$$

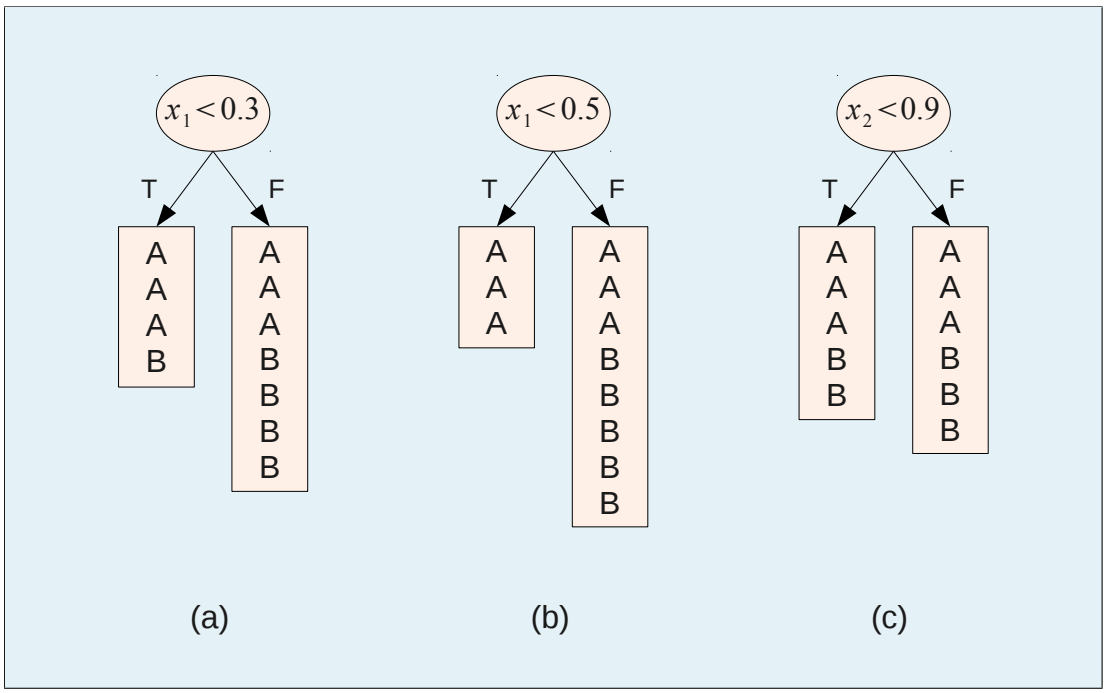

Figure 3.2: An Example of Decision Tree Construction

The information gain of this tree can be found by

$$g_a = i_0 - i_a = 0.0721 \; bits$$

The way forward is clear. The information gain of different attributes and split points can be calculated. In the situations of (b) and (c) in Figure 3.2, the gains are

$$g_b = 0.2999 \; bits$$

$$g_c = 0.0072 \; bits$$

Since the tree in (b) mostly reduces the impurity, or has the highest information gain, x_1 is selected as the root node and 0.5 is selected as its split point. We continue recursively on each leaf until all subsets cannot be split further.

Practically, a pruning mechanism is involved in commonly used C4.5 decision trees[1]. It prunes sub-trees or raises sub-trees to higher nodes for keeping trees simpler and preventing over-fitting to the training set [WF05b] [DHS01].

After a decision tree has been constructed based on training samples, it can be used for classification. Target vectors are input from the root node, and they follow the branches until they reach the end leaves, where their classes can be

[1]C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan [Qui92].

obtained.

3.2.3 Artificial Neural Networks

The central concept of Artificial Neural Networks (ANNs) is modelling the relationships between inputs and outputs with a group of interconnected artificial neurons. For classification, ANNs approach functions that can separate the training vectors. This process is called training. Therefore, trained ANNs can be used for classifying unknown vectors.

From a biological viewpoint, billions of neurons make up a brain and enable the ability of thinking. Like a brain, an ANN is comprised of a number of nodes that are interconnected and each connection has a numeric value, called a weight. As in a brain, these nodes could incept information from the outside world or from other nodes, and they have outputs to excite other nodes. For each node, the operations are simple, which are illustrated in Figure 3.3.

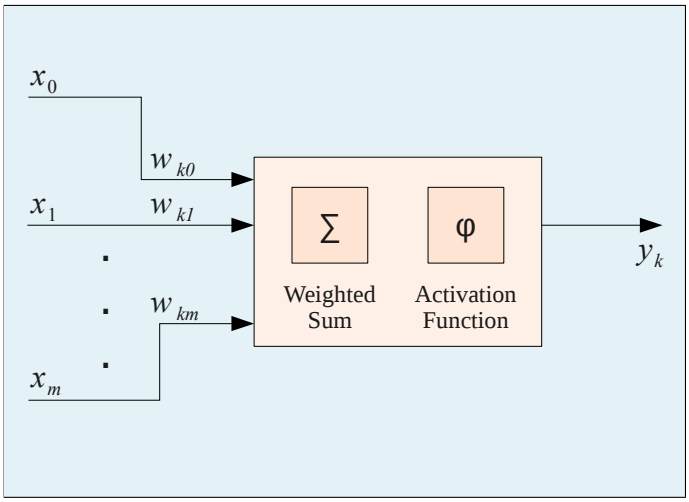

Figure 3.3: A Neural Node

Firstly, the weighted sum of all the inputs is calculated. Secondly, a non-linear activation function is used to produce an output value. Finally, the output value is exported to other nodes. This process can be expressed by Equation 3.3, where y_k is the output of the node k, φ is the activation function, w_{kj} is the weight for jth input of the node k and x_j is the jth input.

$$y_k = \varphi(\sum_{j=0}^{m} w_{kj} x_j) \tag{3.3}$$

In order to interconnect the artificial neural nodes, a feed-forward structure is normally used even though there are rampant back-connections in real brains [RN95]. In the feed-forward structure, the neural nodes are arranged in layers, and the propagation is one way from the input layer to the output layer, which is shown in Figure 3.4. It has been proved that an ANN without a hidden layer could represent any linear function; with one hidden layer, it could represent any continuous function; with two hidden layers, even discontinuous functions could be represented [RN95] [MP88]. For classifying multiple classes, multiple nodes of the output layer can be employed for each class, and the output of each node in output layer y is the confidence for the corresponding class. The input vector can be determined as the class with the highest confidence.

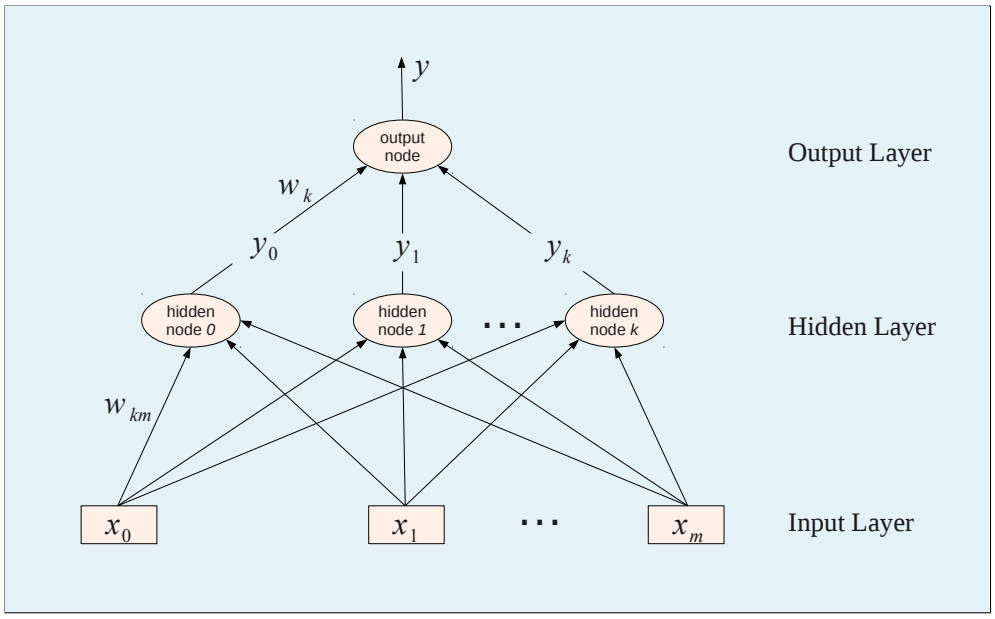

Figure 3.4: A Neural Network

For training the ANN, a back-propagation learning mechanism is used for determining the weights w, which was demonstrated by Rumelhart et al. in [RHW86]. Training samples with known classes are input into the ANN, and the errors in the output nodes are calculated from the output layer to the input layer. Then, the back-propagation algorithm revises the weights in order to minimise these errors.

As soon as an ANN is trained, the candidate vectors with unknown classes can be input to the model for classification.

3.3 Distributions for Non-parametric Tests

As discussed in Section 2.4.2, using distributions of packet size and packet inter-arrival time as traffic discriminators have been proposed by researchers. Traffic types can be identified, because applications generate packets with patterns. However, packet inter-arrival time is affected greatly by network conditions. Jitter, retransmission and packet loss change the inter-arrival time in some degree, which masks the original behaviours of applications. In our research, packet size distributions are used, because they are more stable than distributions of packet inter-arrival time.

Although bin sizes of distributions may affect the classification accuracy, one byte per bin is used in our research for simplicity. Figure 3.5 and Figure 3.6 illustrate the packet size distributions of a Telnet flow and an IMAPS flow respectively, where the frequencies have been normalised.

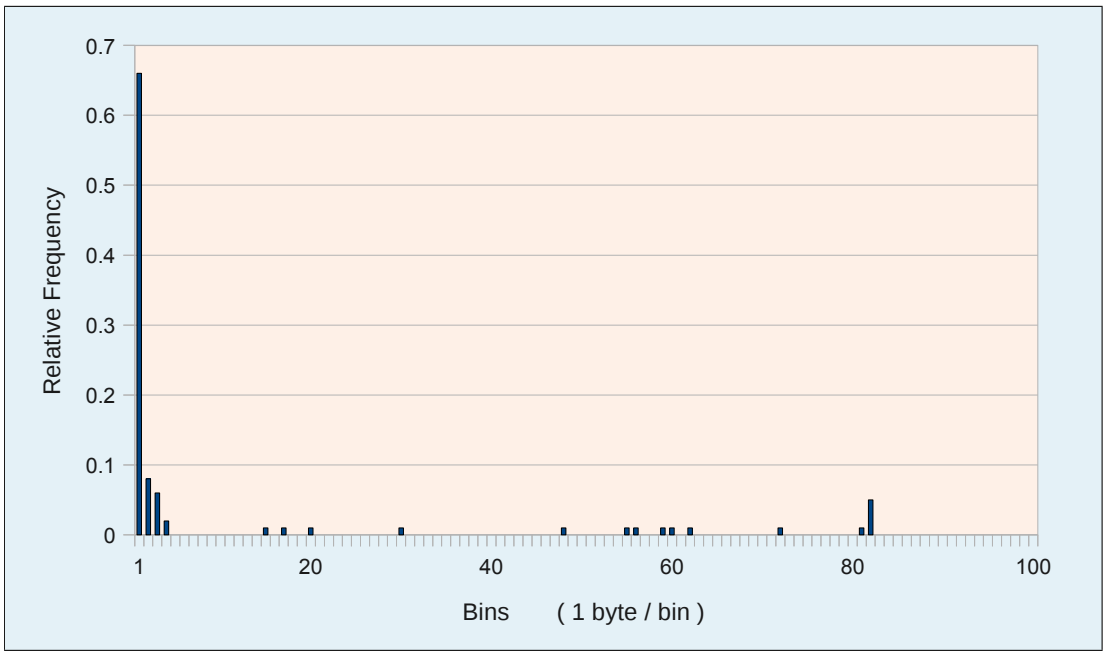

Figure 3.5: Packet Size Distribution of a Telnet Flow

In the figures, the x axes are bins that indicate the amount of bytes in packet payloads, and the y axes are the normalised frequency count for a given bin. Obviously, these two discrete distributions from different applications have different patterns. In order to compare distributions, an algorithm for quantifying the similarities among distribution patterns is needed.

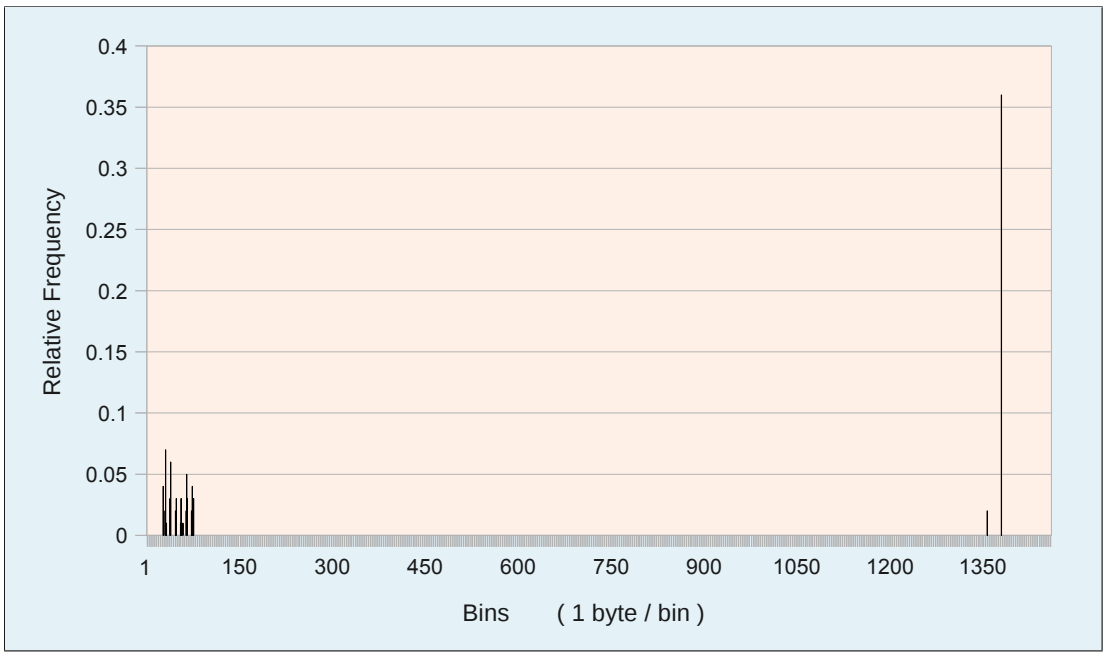

Figure 3.6: Packet Size Distribution of an IMAPS Flow

3.4 Comparison Algorithm for Non-parametric Tests

The Kolmogorov-Smirnov (K-S) test has been used in this work for calculating the similarity between two distributions, assuming there are two normalised probability distributions $P(x)$ and $P'(x)$, which are given in Figure 3.7.

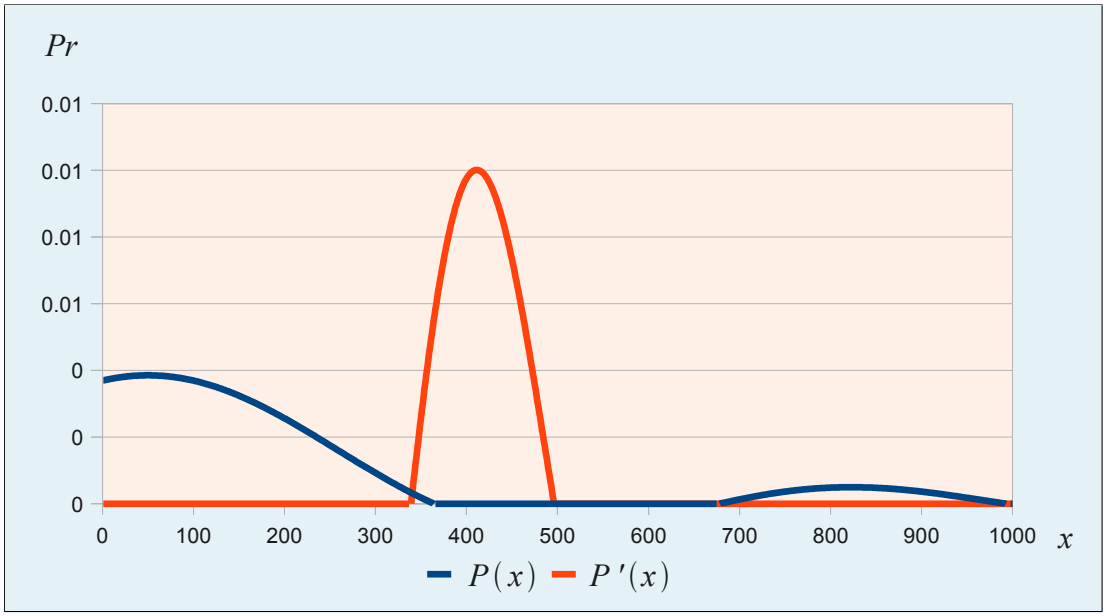

Figure 3.7: A Frequency Distribution

The normalised cumulative distributions can be calculated by Equation 3.4.

$$F(x) = \int_{-\infty}^{x} P(x)dx \tag{3.4}$$

Therefore, the dissimilarity of these two distribution patterns is defined by the maximum distance between their cumulative distributions, which can be expressed by Equation 3.5 and is shown in Figure 3.8 [CLR67]. The $\sup(S)$ is defined to be the smallest real number that is greater than or equal to every number in S.

$$D_{max} = \sup_{x}(|F(x) - F'(x)|) \tag{3.5}$$

It is not difficult to discover that the dissimilarity D_{max} is in the range of $[0, 1]$, which can easily be used for searching the most similar distribution in the database.

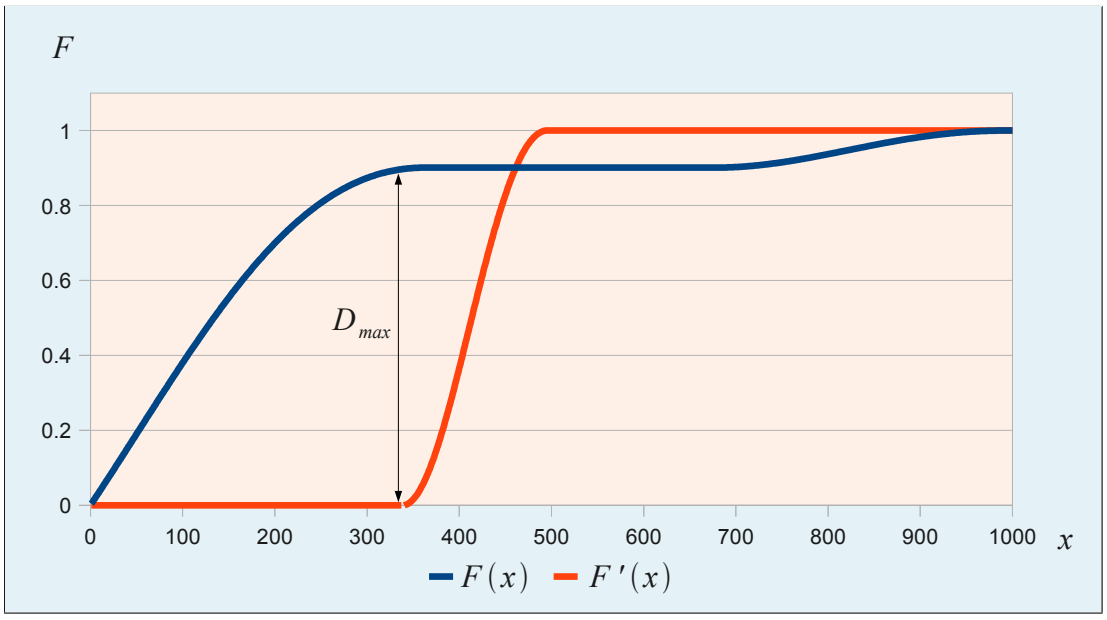

Figure 3.8: Distance Between Two Cumulative Distributions

Since packet size distributions are discrete, the cumulative distributions of packet size are shown in Equation 3.6

$$F(x) = \sum_{1}^{x} P(x) \tag{3.6}$$

Considering the two examples of a Telnet flow and an IMAPS flow given in the

previous subsection, their cumulative distributions and the dissimilarity can be calculated, which are illustrated in Figure 3.9.

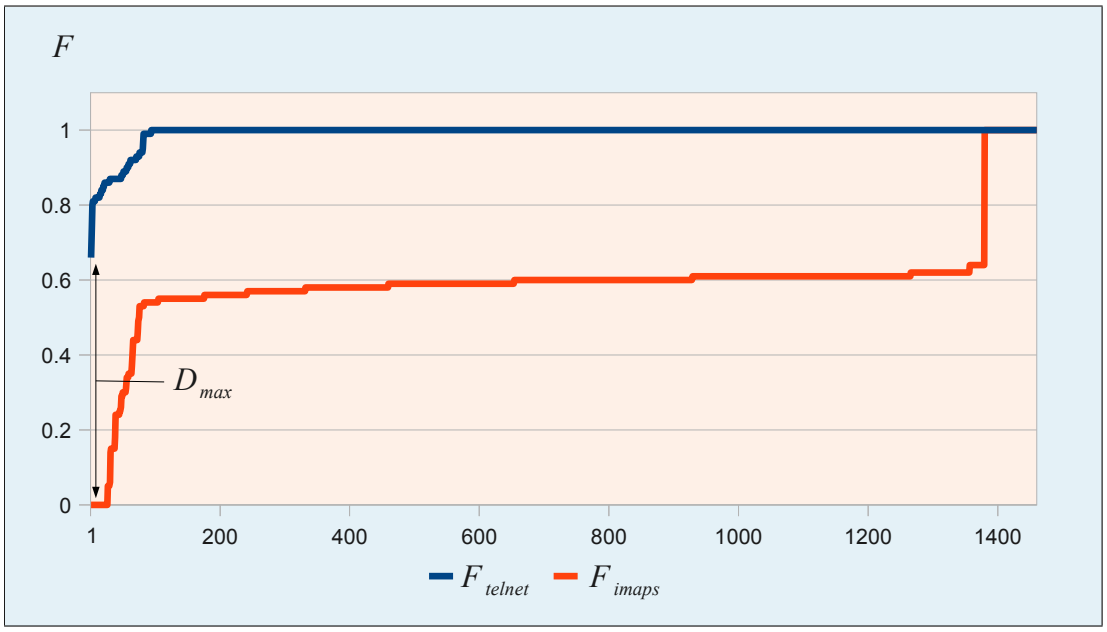

Figure 3.9: Cumulative Distributions of a Telnet and an IMAPS Flow

From the figure, there are a large proportion of packets with 1380 bytes in the IMAPS flow. This is caused by bulk data transfer, which tries to transmit as many bytes as possible per packet. Although normally the MTU of Ethernet is fixed at 1500 bytes, the Maximum Segment Size (MSS) of the application layer may change, because the TCP and IP header have option fields with variable length as discussed in Section 2.2. More specifically, in the case of the IMAPS flow, the operating system encapsulates the application payload with 20-byte IP headers without options, and 100-byte TCP headers, where there are 80-byte TCP options. As a result, only 1380 bytes of the payload are transmitted in a packet. This can greatly affect the distribution comparison, because the same application running on two different operating systems or TCP/IP implementations may generate distributions with a large distance. For example, in Figure 3.10, F_{imaps} is generated by the flow with an MSS of 1380 bytes and F'_{imaps} is generated by the flow with the MSS of 1460 bytes, which is the largest possible MSS. Although both of the flows are generated from IMAPS and the distribution patters are quite similar, they have a large distance D_{max} because of the different MSSes.

Because a large proportion of packets with MSS exist in TCP traffic flows for bulk data transfer, a mechanism has been developed in our work for overcoming this problem. Basically, the mechanism treats MSS as a special bin. MSS packets from different flows should be counted in this bin. Practically, for each traffic flow,

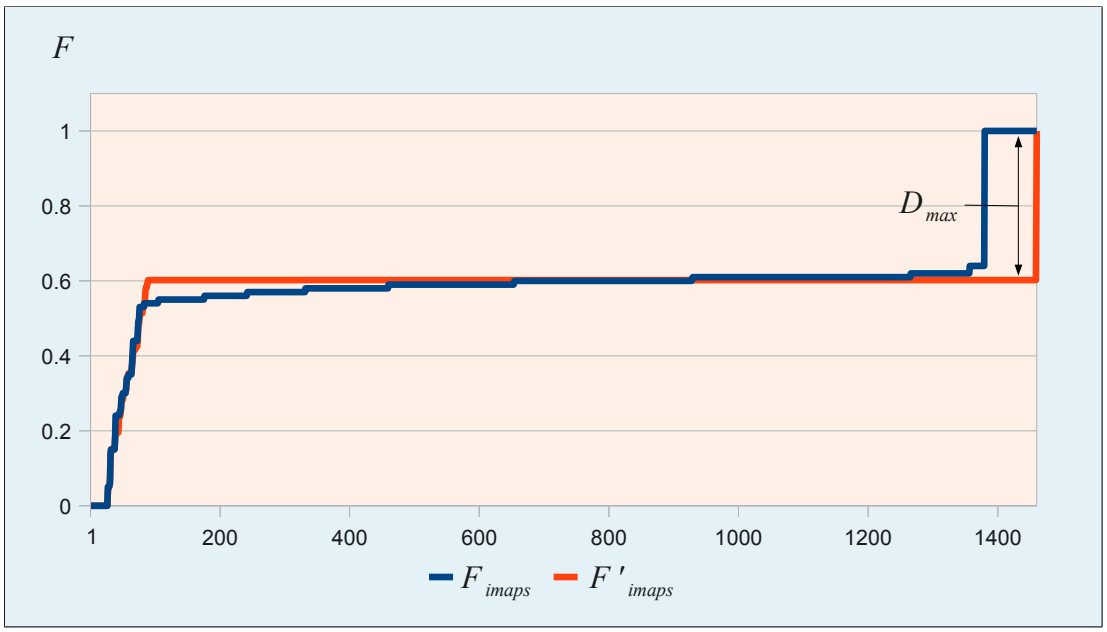

Figure 3.10: Cumulative Distributions of IMAPS Flows with Different MSS

the MSS is recorded, and then the sizes of packets with this MSS are arbitrarily counted as 1460 bytes.

3.5 Summary

In this chapter, an introduction to the algorithms to be used in this work has been given, which includes algorithms for parametric classification and comparison algorithms for distribution classification. Attributes used for parametric classification and distributions used for non-parametric tests are also described.

In the next chapter, the preliminary test will be presented. The limitations of using a single algorithm will be covered together with the introduction to the optimisation.

CHAPTER 4

Datasets & Preliminary Tests

At the beginning of this chapter, generating the traffic data and post-processing are discussed. After data for evaluation are prepared, preliminary tests are taken, where the classification performance of different algorithms is examined. Performance of different detection windows is also explored.

4.1 Data for Preliminary Evaluation

4.1.1 Acquisition of Raw TCP Data

The data used for evaluation and testing must be obtained carefully. For classifying network traffic, the classes of the traffic in the evaluation data must be correctly known; otherwise the optimisation and evaluation are misleading. Taking into consideration the fact that the traffic classes of flows are not included in any public datasets, we have captured traffic data for evaluation with a test bed; this ensures that the classes of traffic are recorded correctly.

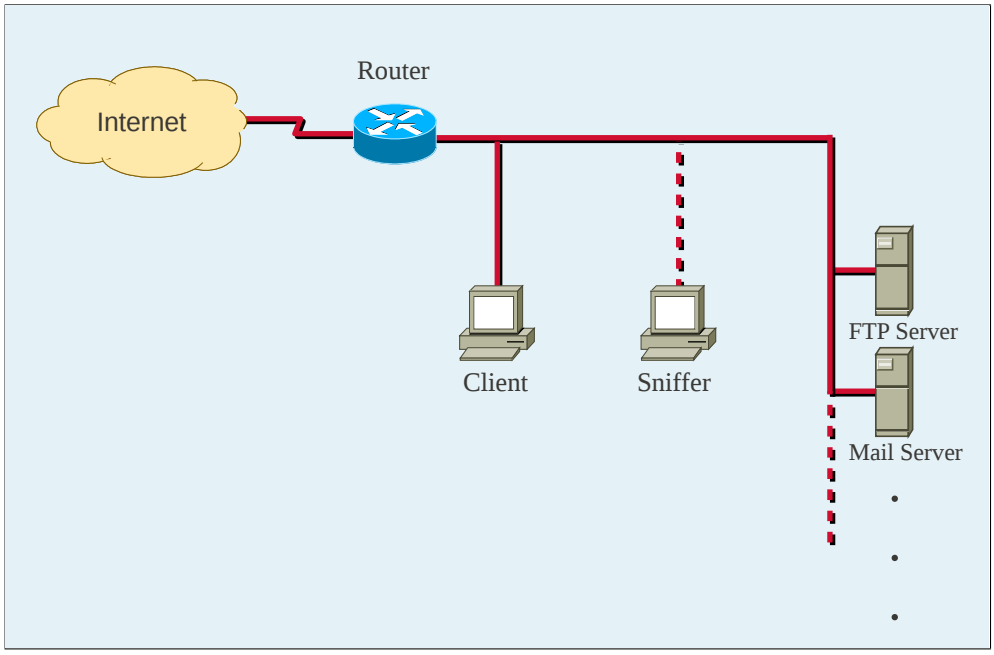

Figure 4.1: System Architecture for Collecting Data

The logical experimental architecture for collecting data is illustrated in Figure 4.1. The data is produced by the client communicating with servers. The sniffer is used to capture and record all the traffic between the client and the servers. The servers located inside the internal network should be set up with various software for simulating a practical environment. By visiting sites on the Internet, the conditions of networks can be taken into consideration. In practice, the sniffer was deployed in the same machine as client and some servers were also share machines. Therefore, when client using Linux platform, Tcpdump powered by pcap library was used for capturing the packets in the test bed, where Ethernet with 1500 MTU was deployed. As discussed in Section 2.1, a captured raw packet includes a MAC layer header, an IP header, a TCP header and payload data. A time-stamp is added automatically in front of the packet and stored in pcap files by Tcpdump for recording the time of capture. Since the information of payload is not required in statistical traffic classification, only headers of packets were captured and stored into pcap files by using the following command in a Linux environment, which sniffed TCP packets on *eth0* and stored them to *filename.pcap* [tcp09].

```
tcpdump -i eth0 tcp -w filename.pcap
```

Although there are IP and TCP options with variable lengths, Tcpdump can automatically adjust the capturing length for storing the packet headers only. For the

applications running under Windows, Wireshark is used by the sniffer for capturing the packets. Wireshark is a GUI protocol analyser software, which can capture network traffic and produce pcap files as well [wir].

Because certain applications have to follow the specified protocol standards, in term of traffic content, there are only minor differences among applications which implement the same protocol. This makes it almost impossible to determine the exact applications. Therefore, network traffic was classed based on protocols. In total, ten classes were considered, including FTP[1], FTP-data[2], IMAPS, IRC, MS-RDP, POP3, RTSP, SMTP, SSH and Telnet. An extra class called 'Others' was also generated, which consisted of various traffic of other protocols including HTTP.

For the purpose of generating these traffic streams, servers were set up in the test bed, and clients with different software were used. In order to simulate a realistic environment, different kinds of user behaviour were employed, such as attaching files in E-mail, different screen resolutions in remote desktop, browsing directories with FTP, etc. In addition, considering the real network situations, we also captured some traffic by visiting sites on the Internet, which may have included traffic generated by different server side software. They could also include the influences of network conditions such as delay, jitter, retransmission, etc., which may affect the traffic statistical characteristics by TCP traffic control and Nagle's algorithm. The traffic classes, client software and servers we used are listed in Table 4.1.

4.1.2 TCP Flow Reconstruction

As we were classifying network traffic at the flow level, TCP flows should be reconstructed with captured packets. A Perl script was developed for the purpose of this task. We did not mind the script language having a relatively slow processing speed, for the experimental stage. As off-line classification was evaluated in our research, the reconstructed TCP flows were stored in plain text files for our tests.

The script named *build_connection.pl* listed in Appendix C uses the Net::Pcap module, which provides an interface to the pcap library [AT]. The module provides

[1]FTP stands for FTP control traffic.
[2]FTP-data stands for data traffic of FTP.

Table 4.1: Traffic Classes and Used Applications

Traffic Class	Client Agent	Server
FTP	Linux Built-in CuteFtp	vsftpd
FTP-data	Linux Built-in CuteFtp	vsftpd
IMAPS	Thunderbird Evolution	postfix googlemail.com hotmail.com lboro.ac.uk
IRC	xchat mIRC	ircd-irc2 Various Internet Sites
MS-RDP	Windows Built-in rdesktop	Windows Built-in
POP3	Thunderbird Evolution	postfix googlemail.com hotmail.com lboro.ac.uk
RTSP	totem RealPlayer Windows Media Player	Helix Server Various Internet Sites
SMTP	Thunderbird Evolution	postfix googlemail.com hotmail.com lboro.ac.uk
SSH	openssh-client PuTTY	openssh-server
Telnet	Linux Built-in PuTTY	Linux Built-in
Others (inc. HTTP)	Various	Various
HTTP	Google Chrome Firefox Internet Explorer	apache2 Various Internet Sites

scripts which have the ability to conveniently read packets in binary pcap files, which are generated by Tcpdump or Wireshark. The NetPacket module is also employed for decoding the Ethernet, IP, TCP headers and time-stamps of packets, which are stored in binary format [PWC].

The script creates a file when a packet with SYN flag is detected. Then, the

information from packets with the same socket or reverse socket[3] is stored in that file. Flows are determined to be terminated as soon as an RST or FIN flag is detected. In addition, there are two thresholds in the script. One is max_idle for terminating the flows when no packets have been detected after a period of time; the other one is max_time for limiting the whole TCP flows in a certain period of time. These two thresholds restrict the flows to a reasonable size, and also make the process of reconstruction faster. In our work, max_idle is set to 600 seconds, which is proposed by Dunigan and Ostrouchov in [DO01], and max_time is set to 3600 seconds. The following is an example of a plain text file in CSV format, where information of TCP flows is stored.

```
@,10.0.0.101,4121,10.0.0.102,3389,9701286,11787,120.236094,ms-rdp
0.000705,C-S,19
0.107169,S-C,11
0.107909,C-S,428
0.108485,S-C,333
0.108548,C-S,12
0.108581,C-S,8
0.108773,S-C,11
0.108829,C-S,12
0.109015,S-C,15
0.109078,C-S,12
0.109302,S-C,15
0.109351,C-S,12
0.109506,S-C,15
...
```

The first field in the summary line — @ is the delimitation for the flow, and the following four fields are a socket pair consisted of source IP address, source TCP port, destination IP address and destination port, which is based on the first packet with a SYN flag. The sixth field is the total counted bytes of the payloads, the seventh is the total packets processed and the eighth is the total duration. The final field is the class of this flow. Because port spoofing is not used when generating the traffic, the class can be determined by TCP destination port. The following lines are details of packets in the flow, which start with a relative time-stamp, followed by direction and payload length. Only packets with an application layer payload are recorded. Packets for TCP handshaking, or only

[3]If a TCP socket is srcIP : srcPort → dstIP : dstPort, the reverse socket is defined as dstIP : dstPort → srcIP : srcPort.

for acknowledgment are not included. As we mentioned before, the first direction of the first packet with SYN is defined as client to server. Therefore, the direction of packets with reversed socket is server to client. The payload length is given by

$$payload\_len = ip\_len - ip\_hlen - tcp\_hlen$$

where ip_len stands for total length of IP packet, ip_hlen and tcp_hlen are IP header length and TCP header length respectively. Referring to the structures of the IP packet header and the TCP packet header shown in Figure 2.2 and Figure 2.3, ip_len is given by the field of *Total Length* in the IP header, ip_hlen is given by the field of *Header Length* in the IP header and tcp_hlen is given by the field of *Data Offset* in the TCP header. Since the IP and TCP header length are counted in words, they are multiplied by 4 when calculating the payload length in bytes.

4.1.3 Discriminator Calculation

After the TCP flows have been built, the CSV files are sent into another two Perl scripts, which are listed in Appendix C, one called *matrix.pl* for calculating the attributes matrix for parametric classification, and the other called *len_ dist.pl* for building the normalised cumulative packet size distributions for non-parametric classification. The principles of these discriminators have been discussed in Section 3.3 and Section 3.4, and the calculation processes are straight forward with the application of simple mathematics. The output files are in CSV format, in which each TCP flow is represented by a line of attribute values or a series of distribution values followed by a class name.

Besides CSV files containing TCP flows, another parameter is input into these two scripts, called 'detection window', which specifies how many packets are taken into consideration for discriminator calculation. This parameter will be optimised for better classification performance in Section 6.1.2 and Section 6.1.3.

For the purpose of preventing bias in classifiers and easy evaluation, the number of traffic flows among classes should be equal [BBM03]. We balanced the flow instances by a method of random undersampling, which randomly picked 30 TCP flows for each traffic class. The exception to this was the 'Others' class, which was treated specially, with 900 traffic flows.

4.2 Preliminary Tests for Different Algorithms

Initially, some preliminary tests with different algorithms based on the processed data were performed. As mentioned before, many machine learning algorithms have been proven to classify network traffic. Therefore, we focused on the performance of different algorithms for detecting different classes, which could help to optimise classification with multiple algorithms.

First, the data was randomly split into two parts, which were the training dataset and the testing dataset. The percentage of the training dataset was 33% of the total. Then the files containing the flows were sent into the scripts for generating discriminators. Since the detection window was not considered at this stage, attributes or distributions were calculated based on full TCP flows. For the training dataset, the class names were given to classifiers, and class names were set to 'unknown' for the testing dataset. Finally, the training dataset and the testing dataset were input into classifiers for evaluation.

As discussed in Section 3.2, k-NN, decision tree and ANN were employed in our work. Weka has been used for implementing these algorithms, which is an open-source data mining tool in Java [wek]. In order to undertake automatic classification, a Bash script has been written for reading input files, executing Java classes and writing output files. The parameters for these algorithms were also specified in this Bash script. Since optimising the parameters for these algorithms was not our concern here, recommended values were used. For k-NN, one nearest neighbour was used, which was the simplest way. For decision tree, a pruned tree with minimum 2 instances per leaf was used for preventing over-fitting. For ANN, the parameters were $(attributes + classes)/2$ hidden layers, 0.3 learning rate, 0.2 momentum and 500 epochs training time.

For non-parametric classification, the K-S test was employed as was described in Section 2.4.2. Another Perl script called *classify_ks.pl* was developed in order to fulfil this task, which is listed in Appendix C. For every input, the distances between the incoming distribution and all distributions in the training dataset were calculated. Then the distribution in the training dataset with the smallest distance was selected as a match, and the input distribution was marked as the same class as the matched one.

The classification accuracies for different algorithms are given in Table 4.2, which

are calculated by Equation 4.1

$$A = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)} \tag{4.1}$$

where TP_i means True Positive for the ith class, FP_i means False Positive for the ith class and accuracy is the ratio of correctly classified instances to all instances among all classes.

Table 4.2: Overall Classification Accuracies for Different Algorithms (Full Flow)

Class	Parametric Classification			Distribution
	k-NN	Decision Tree	ANN	K-S Test
Overall	0.91	0.95	0.91	0.86

The algorithms of parametric classifications have similar performance, because they use the the same attributes as discriminators. The performance of K-S classification using packet size distributions has a lower level of accuracy, so further research of the confusion matrix is needed for finding out the underlying reasons. Also, in order to gain a better understanding of the classification performance for different classes, the classification confusion matrices of different algorithms have been investigated, which are shown in Table 4.3, Table 4.4, Table 4.5 and Table 4.6, where the actual class is the row and the predicted class is the column.

Table 4.3: Confusion Matrix for k-NN Classifier (Full Flow)

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	16	0	1	0	1	0	2	0	0	0	0
b=RTSP	0	19	0	0	0	0	0	0	0	0	1
c=POP3	1	0	11	0	0	0	1	0	7	0	0
d=SMTP	0	0	0	19	0	1	0	0	0	0	0
e=SSH	0	0	0	0	17	0	3	0	0	0	0
f=FTP	0	0	0	0	0	17	3	0	0	0	0
g=IMAPS	1	0	4	0	1	0	7	3	4	0	0
h=IRC	0	4	0	0	0	0	3	13	0	0	0
i=Telnet	0	0	1	0	0	0	4	0	15	0	0
j=FTP-data	0	0	0	0	0	0	0	0	0	10	10
k=Others	1	2	1	5	1	0	1	3	0	4	582

Table 4.4: Confusion Matrix for Decision Tree Classifier (Full Flow)

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	16	0	0	0	0	0	2	0	0	0	2
b=RTSP	0	19	0	0	0	0	0	0	0	0	1
c=POP3	0	0	19	0	0	0	0	1	0	0	0
d=SMTP	0	0	0	19	0	0	1	0	0	0	0
e=SSH	0	0	0	0	20	0	0	0	0	0	0
f=FTP	0	0	0	0	0	20	0	0	0	0	0
g=IMAPS	0	0	0	0	0	0	15	0	0	0	5
h=IRC	0	0	4	0	0	0	0	16	0	0	0
i=Telnet	0	0	0	0	0	0	0	0	20	0	0
j=FTP-data	0	0	0	0	0	0	0	0	0	13	7
k=Others	1	3	1	0	2	3	2	2	0	2	584

Table 4.5: Confusion Matrix for ANN Classifier (Full Flow)

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	16	0	0	0	2	0	1	0	0	0	1
b=RTSP	0	18	0	0	0	0	0	0	0	0	2
c=POP3	0	0	6	0	3	0	10	0	1	0	0
d=SMTP	0	0	0	18	0	0	0	0	0	0	2
e=SSH	0	0	0	0	19	0	1	0	0	0	0
f=FTP	0	0	0	0	0	9	0	2	0	0	9
g=IMAPS	0	0	0	0	0	0	17	3	0	0	0
h=IRC	0	0	0	0	0	0	1	19	0	0	0
i=Telnet	0	0	1	0	0	2	1	0	16	0	0
j=FTP-data	0	0	0	0	0	0	2	0	3	0	15
k=Others	0	2	0	5	1	0	0	0	1	0	591

Table 4.6: Confusion Matrix for K-S Classifier (Full Flow)

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	13	1	0	4	0	0	0	1	0	0	1
b=RTSP	0	15	0	1	0	0	0	0	0	2	2
c=POP3	0	3	6	5	0	0	2	3	0	0	1
d=SMTP	0	0	4	12	0	0	0	2	0	2	0
e=SSH	1	0	0	0	19	0	0	0	0	0	0
f=FTP	0	0	0	0	0	20	0	0	0	0	0
g=IMAPS	1	0	0	0	0	0	17	0	0	0	2
h=IRC	0	0	0	0	0	1	1	16	0	0	2
i=Telnet	0	0	1	0	0	0	0	0	19	0	0
j=FTP-data	0	0	1	0	0	0	0	0	0	9	10
k=Others	16	14	5	8	0	0	4	7	2	1	543

From the tables above, the reason of the K-S classification having lower accuracy is caused by some traffic flows that should belong to the 'Others' class being misclassified. Further analysis of the distributions will be presented later in this subsection. Before investigating the distributions, we summarise the confusion matrices. Two concepts of evaluating the errors of classification have been defined in the data mining field; these are recall rate and precision.

For each class i, the recall rate is defined by correctly classified instances divided by actual instances, which can be expressed by Equation 4.2, where FN_i means False Negative for the ith class.

$$R_i = \frac{TP_i}{TP_i + FN_i} \tag{4.2}$$

From the confusion matrices, a recall rate for a class is calculated by the instances in the diagonal line divided by all the instances in that row.

The precision is defined by correctly classified instances divided by all instances classified as that class, which is given by Equation 4.3.

$$P_i = \frac{TP_i}{TP_i + FP_i} \tag{4.3}$$

From the confusion matrices, a precision for a class is calculated by the instances in the diagonal line divided by all the instances in that column.

Recall rate indicates the percentage of a classifier detecting the instances of a specific class out of all actual instances of that class. Precision is the percentage of correctly detected instances of specific class out of all predicted instances of that class, which is actually the classification confidence. The recall rates and precisions of different classifiers among classes are listed in Table 4.7 and Table 4.8.

Table 4.7: Classification Recall Rates for Different Algorithms (Full Flow)

Class	Parametric Classification			Distribution
	k-NN	Decision Tree	ANN	K-S Test
MS-RDP	0.80	0.80	0.80	0.65
RTSP	0.95	0.95	0.90	0.75
POP3	0.55	0.95	0.30	0.30
SMTP	0.95	0.95	0.90	0.60
SSH	0.85	1.00	0.95	0.95
FTP	0.85	1.00	0.45	1.00
IMAPS	0.35	0.75	0.85	0.85
IRC	0.65	0.80	0.95	0.80
Telnet	0.75	1.00	0.80	0.95
FTP-data	0.50	0.65	0.00	0.45
Others	0.97	0.97	0.99	0.91

Table 4.8: Classification Precisions for Different Algorithms (Full Flow)

Class	Parametric Classification			Distribution
	k-NN	Decision Tree	ANN	K-S Test
MS-RDP	0.84	0.94	1.00	0.42
RTSP	0.76	0.86	0.90	0.45
POP3	0.61	0.79	0.86	0.35
SMTP	0.79	1.00	0.78	0.40
SSH	0.85	0.91	0.76	1.00
FTP	0.94	0.87	0.82	0.95
IMAPS	0.29	0.75	0.52	0.71
IRC	0.68	0.84	0.79	0.55
Telnet	0.57	1.00	0.76	0.90
FTP-data	0.71	0.87	0.00	0.64
Others	0.98	0.98	0.95	0.97

Because some instances from the class 'Others' are misclassified to other classes, the K-S classifications have a relatively low precision rate. We can take a misclassified sample for further investigation. The packet size distribution of a HTTP flow shown in Figure 4.2 has been classified as POP3, which should belong to 'Others'. This HTTP flow has an overwhelming amount of packets with MTU size, which produces a fairly long bar at 1460 in the distribution chart. The reason is that HTTP traffic is actually comprised of transferring files of HTML, graphics, Flash, etc., except initial requests and handshakes. For the transferring of files with batch bytes, TCP implementations always encapsulate packets with MTU size. Therefore, the packet size distribution pattern contains little information entropy caused by the dominant number of MTU packets in the flow.

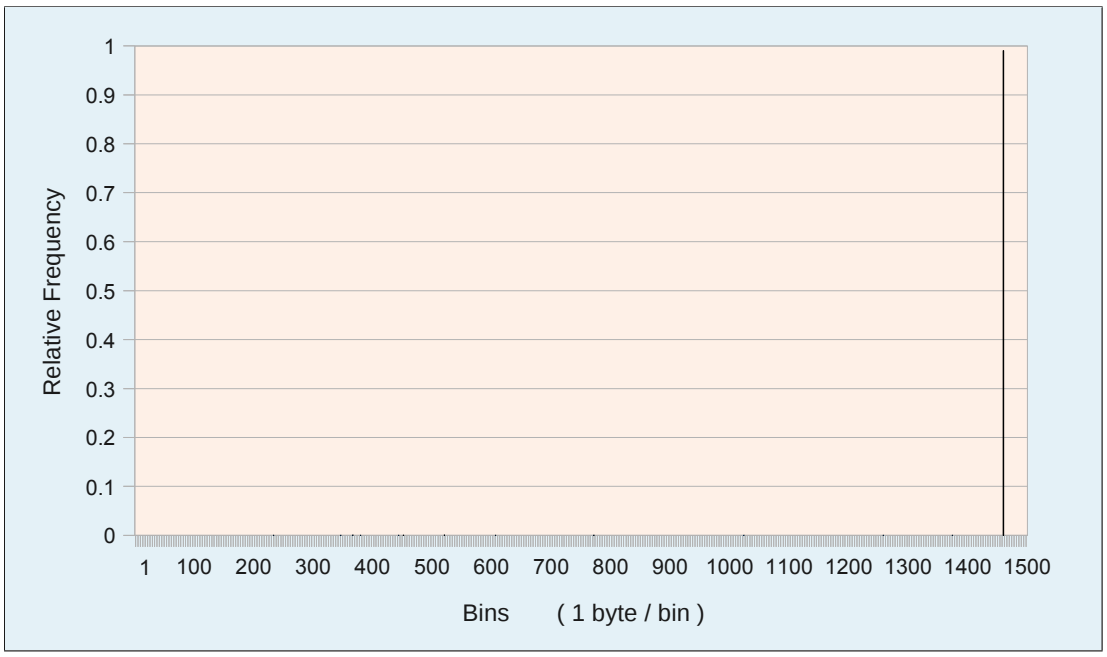

Figure 4.2: Packet Size Distribution of a Full HTTP Flow

Comparing the HTTP flow with one of the traffic flows in the POP3 class shown in Figure 4.3, they have similar patterns of packet size distributions. This POP3 traffic flow also has a large proportion of packets with MTU sizes because transferring files, which are attached in the E-mail, are involved. Although there are some packets in the POP3 flows laid in the frequencies of smaller sizes, the percentage is so small that they may be covered by noise. Consequently, the K-S classification may match the HTTP flow with this POP3 flow due to the small distance between them, and thus make a wrong classification.

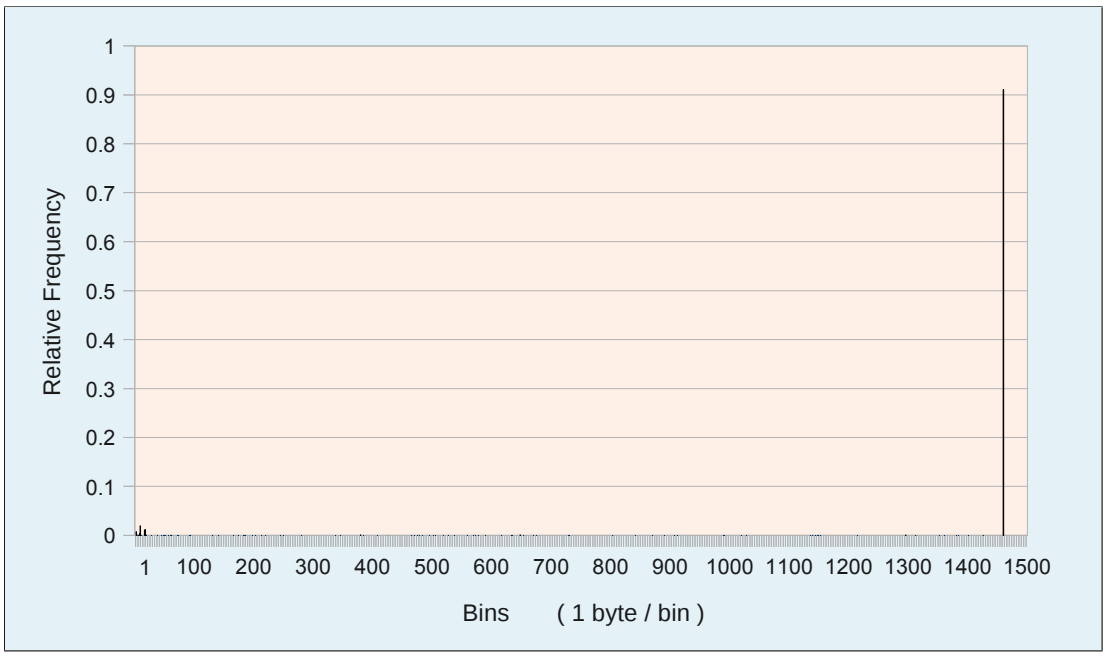

Figure 4.3: Packet Size Distribution of a Full POP3 Flow

The parametric values for these two flows, then, can be checked to see if parametric classification can help for distinguishing them. The values are listed in Table 4.9, where values are rounded to two decimal places; the unit of packet size is byte, and the unit of inter-arrival time is ms.

Since the two flows are generated by applications without human interventions, the values related to packet inter-arrival time are very small and similar. Although the maximum packet sizes and the maximum packet sizes of server to client are similar, the minimum packet sizes, the minimum packet sizes of server to client and packet size variations are distinct due to the different handshaking processes of HTTP and POP3. The average packet sizes in both directions and in the direction of server to client are similar, due to the large number of MTU packets, but the average packet sizes of client to server show significant differences because of the different requests sent by different protocols. As a result, the parametric classification could distinguish these two flows.

The different levels of accuracy produced by different parametric classification algorithms are caused by different methods of splitting the vectors in multi-dimensional space. K-NN can only split the hyperspace with one-dimensional linear functions, decision tree can use two-dimensional linear functions and ANN can use multi-dimensional non-linear functions. From their recall rates and precisions, there is no one best algorithm for classifying all the classes.

Table 4.9: Parametric Values of a HTTP and a POP3 Flow (Full Flow)

Attributes	HTTP (2 decimals)	POP3 (2 decimals)
ps_max	1460	1460
ps_min	234	3
ps_avg	1451.91	1352.97
ps_var	7871.69	131451.46
it_max	14.23	30.39
it_min	0	0
it_avg	0.02	0.04
it_var	0.27	0.07
ps_cs_max	452	28
ps_cs_min	347	6
ps_cs_avg	392.83	9.85
ps_cs_var	1619.81	1.17
it_cs_max	14.34	30.65
it_cs_min	0.01	0.38
it_cs_avg	6.77	1.5
it_cs_var	35.47	9.43
ps_sc_max	1460	1460
ps_sc_min	234	3
ps_sc_avg	1456.29	1385.42
ps_sc_var	3233.87	89985.11
it_sc_max	14.23	30.39
it_sc_min	0	0
it_sc_avg	0.02	0.04
it_sc_var	0.27	0.07
pc_ratio	0	0.02
bc_ratio	0	0

The K-S classification has relatively high recall rates and precision for SSH, FTP and Telnet as these protocols normally generate small packets by keystrokes or commands, which have distinct packet size distribution patterns. That is a similar situation as classifying traffic generated by games, which has been proposed by [PBL+03].

In conclusion, some applications may share packet size distribution, which makes

them hard to classify. Besides HTTP and protocols related to E-mail, other classes of traffic may contain batch byte transfer as well, such as SSH, FTP-data, etc. This decreases the accuracy of traffic classification with packet size distribution. On the other hand, packet size distributions could describe some network flows better than when using absolute parametric values, such as FTP control traffic, SSH, Telnet, etc. Moreover, different parametric classification algorithms produce different recall rates and precisions for different traffic classes. Consequently, it is worth trying to optimise the network traffic classifier with multiple classification algorithms and methods.

4.3 Preliminary Tests for Different Detection Windows

Besides algorithms, different detection windows may affect the classification accuracy. Wang et al. argued that using only a few initial packets in TCP flows could increase classification accuracy and speed, because the handshaking procedures of applications are distinct [WY08]. From this research, using the first 7 packets at the beginning of TCP flows produces the highest accuracy. However, only one single global detection window for all classes is considered in their research. In this subsection, the effects of detection windows for different classes and algorithms will be presented.

For the purposes of parametric classification, the relationship of classification accuracies versus detection windows is illustrated in Figure 4.4, where detection windows from 1 to 500 packets are considered. Among the three algorithms, the accuracies reach the peaks when only a few packets are taken into consideration for calculating the flow attributes, which matches the results shown in [WY08].

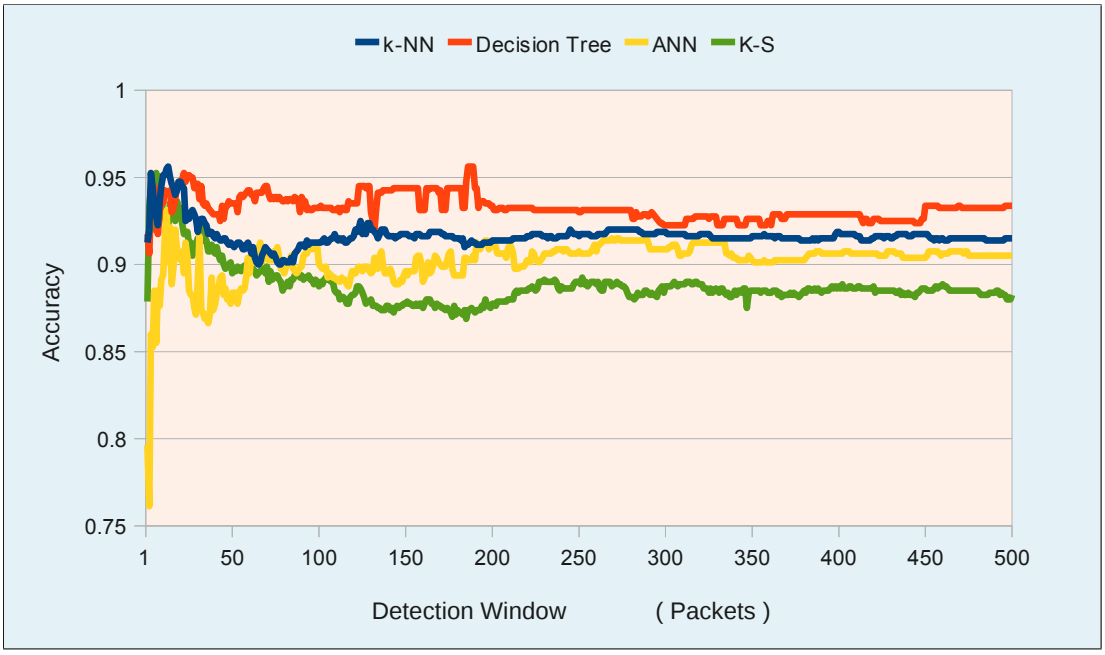

Figure 4.4: Accuracies of Classifications vs Detection Windows

Instead of one single global detection window, the classification performance of different classes versus detection windows can be figured out. As we discussed in the previous subsection, for each class, the classification performance can be expressed by the recall rate and precision. The recall rates specify the detection abilities of classes, and precisions give the confidence of predicted true positives. Since the class 'Others' contains varieties of traffic flows, its performance is not taken into consideration due to the existence of different characteristics. The recall rates versus detection windows among classes are shown in Appendix A, and precisions are shown in Appendix B.

From the diagrams shown in the appendices, different classifications have different performance in terms of recall rates and precisions. For instance, the ANN has a low recall rate and precision among all detection windows for FTP-DATA flows. However, the decision tree has a higher recall rate and the K-S has higher precision. The diagrams also indicate that the detection windows have a different manner of effects to classification performance. Flows like SSH, Telnet and FTP controls have better classification performance when detection windows are increased because the statistical characteristics of these kinds of flows remain stable, which generates more accurate discriminators when more packets are taken into consideration. For other classes, the detection windows affect the classification performance in terms of both recall rates and precisions. In the charts, some curves reach their peaks at few packets, some of them at hundreds of packets. For a single class, different

classification algorithms have different shapes.

For the example used in the previous subsection shown in Figure 4.2 and Figure 4.3, the packet size distributions of reduced detection window for K-S classification can be examined. The Figure 4.5 and Figure 4.6 illustrate the packet size distributions of the HTTP flow and POP3 flow respectively, where only first 20 packets are taken into account. The distribution patterns are more distinct compared with Figure 4.2 and Figure 4.3, where all packets are taken into account. By getting rid of the overwhelming amount of MSS packets, the K-S distance is increased from 0.08 to 0.65. However, the price of reducing the sampling period must be paid. A shorter sampling period could mean the statistics are more sensitive to packet retransmissions.

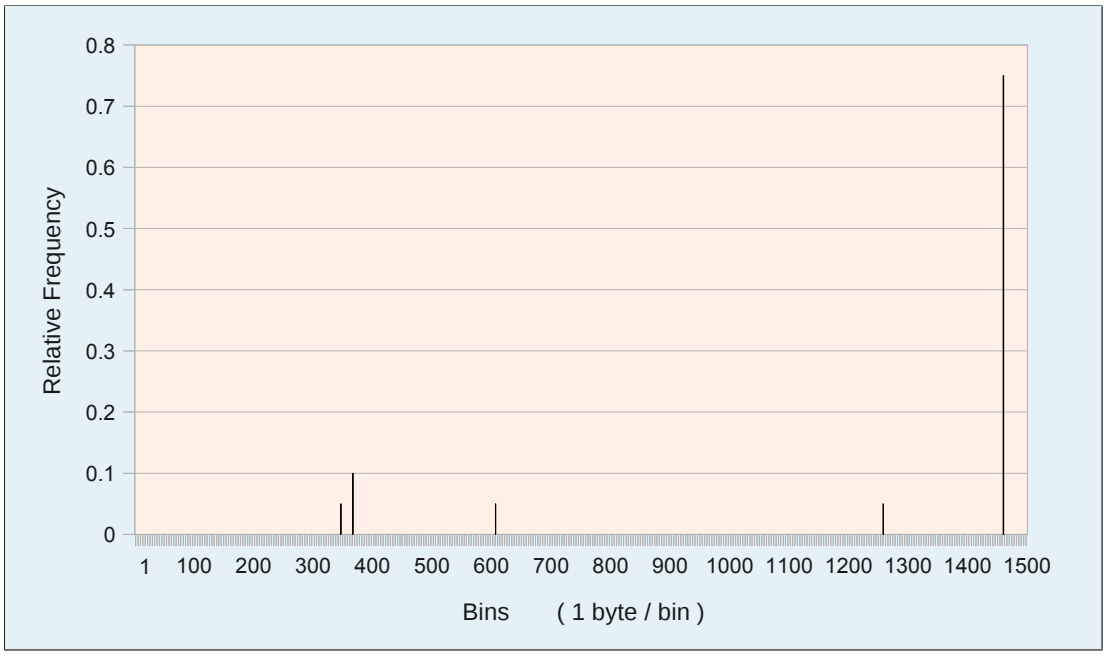

Figure 4.5: Packet Size Distribution of a HTTP Flow (20 Packets)

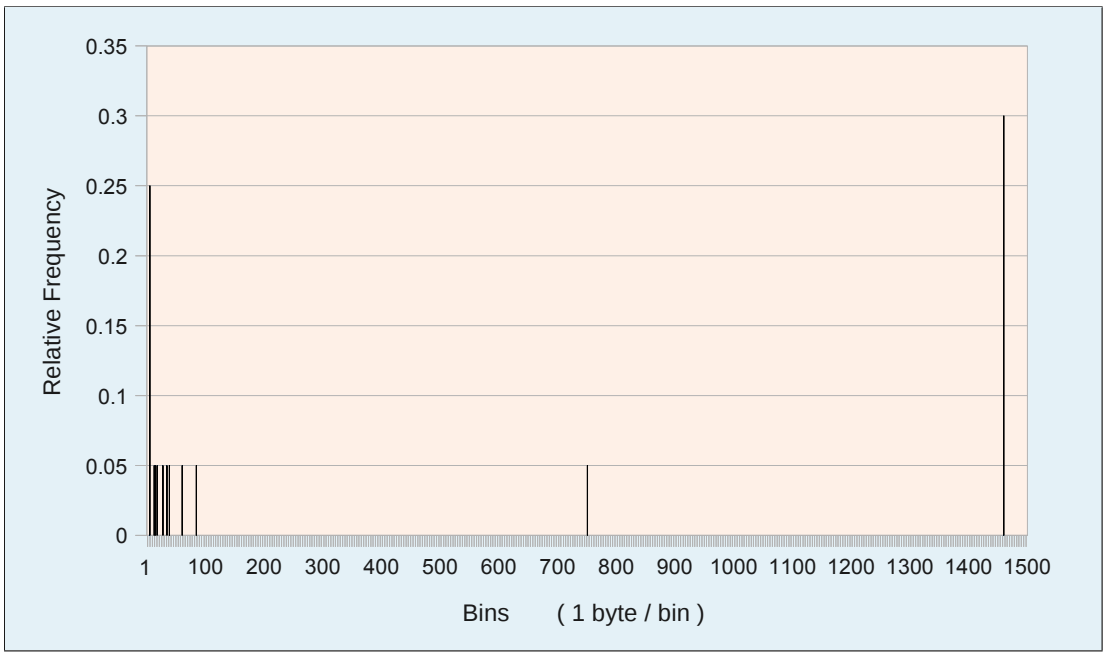

Figure 4.6: Packet Size Distribution of a POP3 Flow (20 Packets)

4.4 Summary

In this chapter, the steps of generating the data for evaluation are described, as well as the post-processing of the captured packets, which describes TCP flow reconstruction and discriminators calculation in detail.

The following subsection presents the preliminary tests of classification performance versus different algorithms. In the previous subsection, the concept of the detection window was introduced, which means the number of packets that are taken into account when calculating the discriminators. Then, the relationship between classification performance and detection windows among different algorithms was explored.

Within these tests, two indicators of performance are used, which are recall rate — representing the ability to detect specified classes, and precision — representing the confidence of predicted positives. The performance is evaluated by classes separately, which could be used for further optimisation.

CHAPTER 5

Hierarchical Classification

In this chapter, an architecture for a hierarchical traffic classifier will be proposed for detecting traffic classes differently. Then, the optimisation for traffic classification with different algorithms and detection windows will be presented. The datasets used for evaluating its performance are also covered in this chapter. In addition, practical implementation will be illustrated.

5.1 System Architecture

5.1.1 Overview

From the results of the preliminary tests in the previous chapter and the preliminary results shown in Appendix A and Appendix B, it has been shown that different algorithms and detection windows have different classification performance for traffic classes in terms of precisions and recall rates. In order to optimise the overall classification performance, instead of using only single algorithm and detection window, different ones can be applied for detecting different traffic classes.

That turns the classification system into a series of application detectors[1]. An overview of the proposed system architecture is illustrated in Figure 5.1.

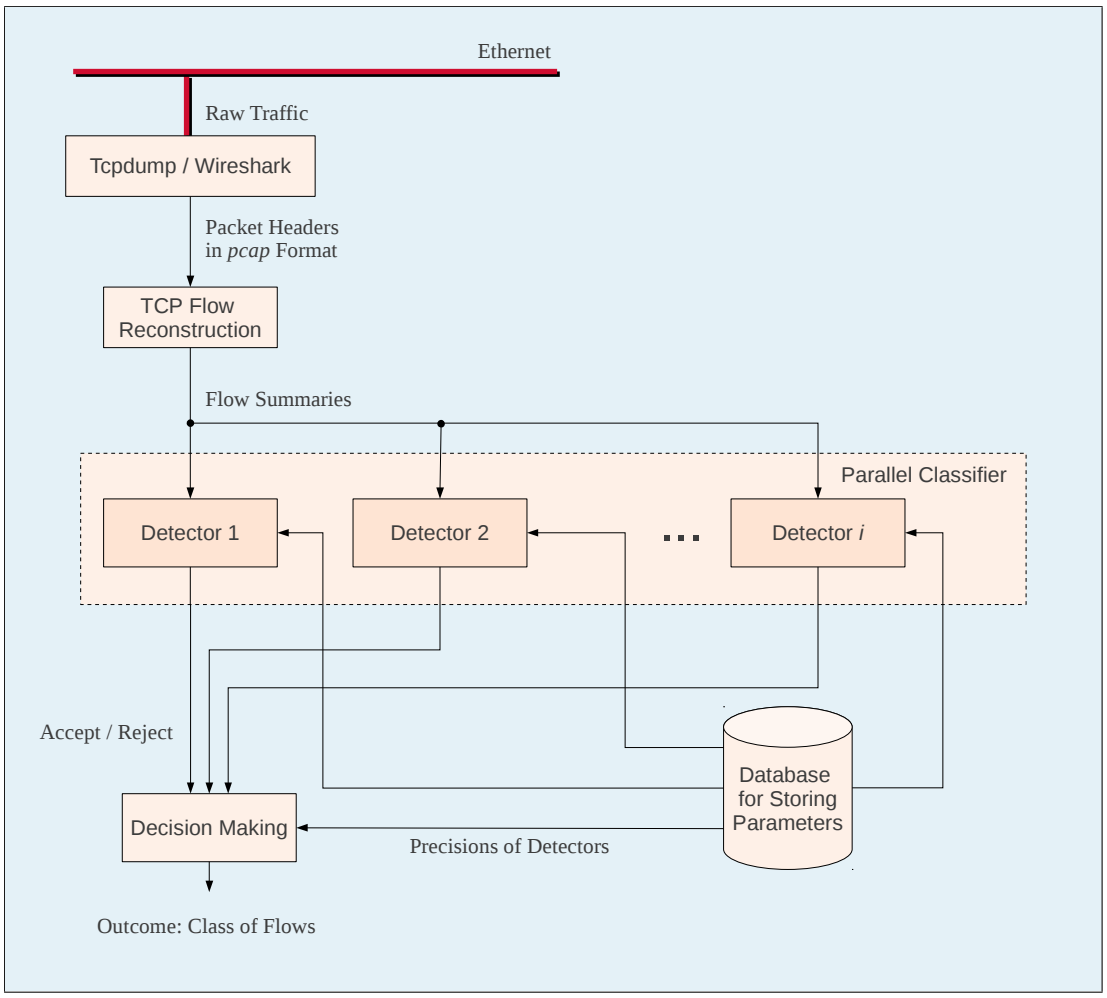

Figure 5.1: An Overview of the System Architecture

At the beginning, the same steps as those we used in the preliminary test are applied, the packet headers of raw data are captured from the Ethernet by Tcpdump or Wireshark, and they are reconstructed into TCP flows. As we discussed before, because only packet length, time-stamps and packet directions are needed for statistical classification, this information is extracted from packet headers and kept as TCP flow summaries for further processing. As a result, for each TCP flow, a summary containing size, time-stamps and directions of all packets can be obtained.

The flow summaries are then sent into a parallel classifier, which consists of a

[1]In this thesis, a detector refers to a binary classifier for separating instances of target class for the rest of instances.

series of detectors. Each detector distinguishes one class of traffic from the others. For classifying network traffic of i known classes, i detectors are required, and each flow summary is fed into them separately. There is a special class called 'Others', which indicates unknown traffic class or uninteresting traffic classes and it is not counted in the i. For the output of each detector, a binary result of *Accept / Reject* is produced. The parallel classifier can employ different classification techniques or configurations for detecting the required traffic classes differentially. It can also benefit from parallel computing for high processing speed. In addition, instead of profiling all possible traffic classes on networks, only interesting traffic classes need to be profiled. If all of the detectors output a reject, the input TCP flow will be classified as 'Others'.

Since different algorithms and detection windows used by detectors, and the parameters of the detectors may vary after retraining, a database has been employed for storing the related configurations for detectors. Detection widows and the traffic samples for K-S classification are stored in the database. There is an additional parameter used for K-S classification, called 'acceptance threshold'; this is also stored in the database, which will be explained later.

The binary outputs for the detectors are fed into a decision making mechanism, which produces the final results of the classification. Because of the binary outputs generated by independent detectors, there is no doubt that controversial outputs might be obtained. There are some rules or simple calculations involved in the decision making mechanism based on the previous performance of the detectors, which is also stored in the database.

5.1.2 Parallel Classifier

In order to employ different algorithms and configurations for classification, a parallel classifier is applied, which consists of detectors for each traffic class. As a result, totally i detectors are needed when recognising i traffic classes, and unrecognised traffic is classified as an additional special class called 'Others'. The details of a detector is shown in Figure 5.2.

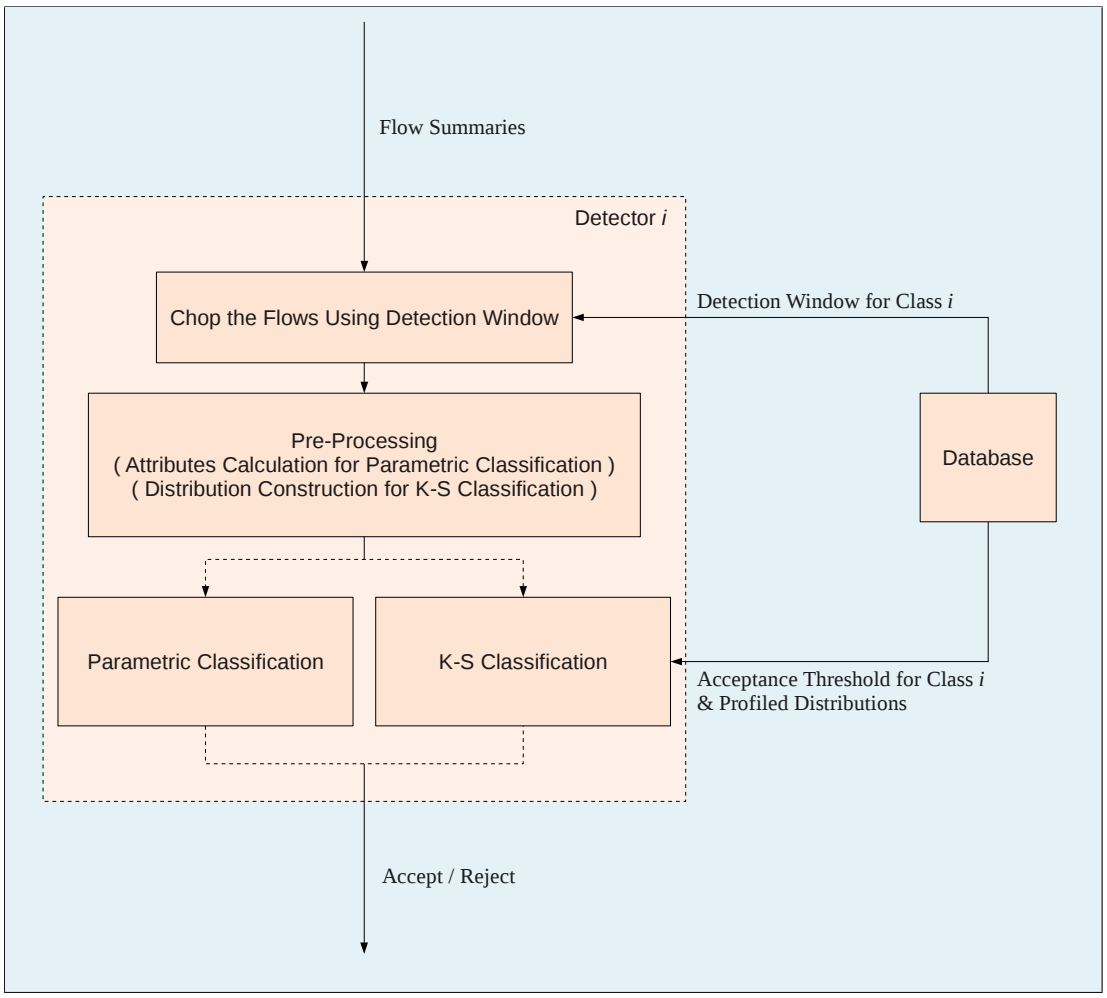

Figure 5.2: A Detector in the Proposed Parallel Classifier

Because only statistical information related to packet size and packet inter-arrival time is used in this work, the input flow summaries only contain the size, time-stamp and direction for each packet in the TCP flows. The summaries are then chopped based on the detection windows stored in the database, which only keeps a certain number of packets at the beginning of the TCP flows for further processing. Next, the chopped flow summaries are input into the pre-processing module, which extracts statistical characteristics. As we discussed in Section 3.1, for detecting traffic with parametric classification, the attributes of the flows must be calculated, which are represented as a numerical array of 26 elements for each TCP flow. The definitions of the elements have been shown in Table 3.1. For non-parametric classification, which is the K-S test used in this work, the normalised packet size distributions for traffic flows are constructed at this stage as shown in Section 3.3. Depending on which kinds of traffic classes to be detected, parametric classification or K-S non-parametric test is used, which produces a binary result of *Accept / Reject*. That means that the input TCP flow, whether it belongs to the traffic

class i or not. For parametric classifications, because the models are built at the training stage, no extra parameters or profiled samples are needed when classifying incoming flows. On the other hand, as a non-parametric test, K-S needs profiled distribution samples when classifying incoming flows. In addition, acceptance thresholds are introduced as critical values for acceptance. This information is also stored in the database.

There are i detectors working in parallel to form the parallel classifier for classifying i traffic classes. This architecture could help us to apply different algorithms and detection windows independently for detecting traffic classes. Although, a few algorithms are used in this work, adding new algorithms or changing them are easy, as well as adding new traffic classes. The increased computational complexity can be solved by employing distributed computing or parallel computing by hardware [SSB05]. DPI can be used as a separate detector for detecting traffic classes that do not have distinct statistical characteristics, and implemented by parallel hardware computing [DKSL04] [SGO+09]. As a result, the parallel classifier, which detects traffic classes independently, has great flexibility.

5.1.3 Acceptance Thresholds for K-S

In preliminary testing, classification algorithms classify the testing dataset into several traffic classes at one time. In contrast, in the proposed system, each detector has the responsibility of detecting only one class of traffic. In other words, each detector splits only one kind of traffic flows with all others. For parametric algorithms, we can change the class names of input training flows for building binary classifiers, which means we only keep class names that correspond to each detector and change the rest of class names to 'Others'. For non-parametric classification, for which K-S test is used in this work, besides changing the class names, acceptance thresholds are introduced for preventing acceptance with great dissimilarities.

In Figure 5.3, the flow chart of a K-S detector with an acceptance threshold is illustrated. The incoming distributions are compared with the profiled samples using the algorithm stated in Section 3.4. The sample having the least dissimilarity with the incoming distribution is picked as a match. For the i th detector, incoming distributions are only classified as the i th class or 'Others', which is represented as an *Accept* or a *Reject* outcome. If the matched sample belongs to 'Others', an

output of *Reject* is produced. If the matched sample belongs to the i th class, before outputting an *Accept*, the dissimilarity between the matched sample and incoming distribution is compared with the acceptance threshold. An *Accept* is output only if this dissimilarity is smaller than the threshold, otherwise a *Reject* is produced.

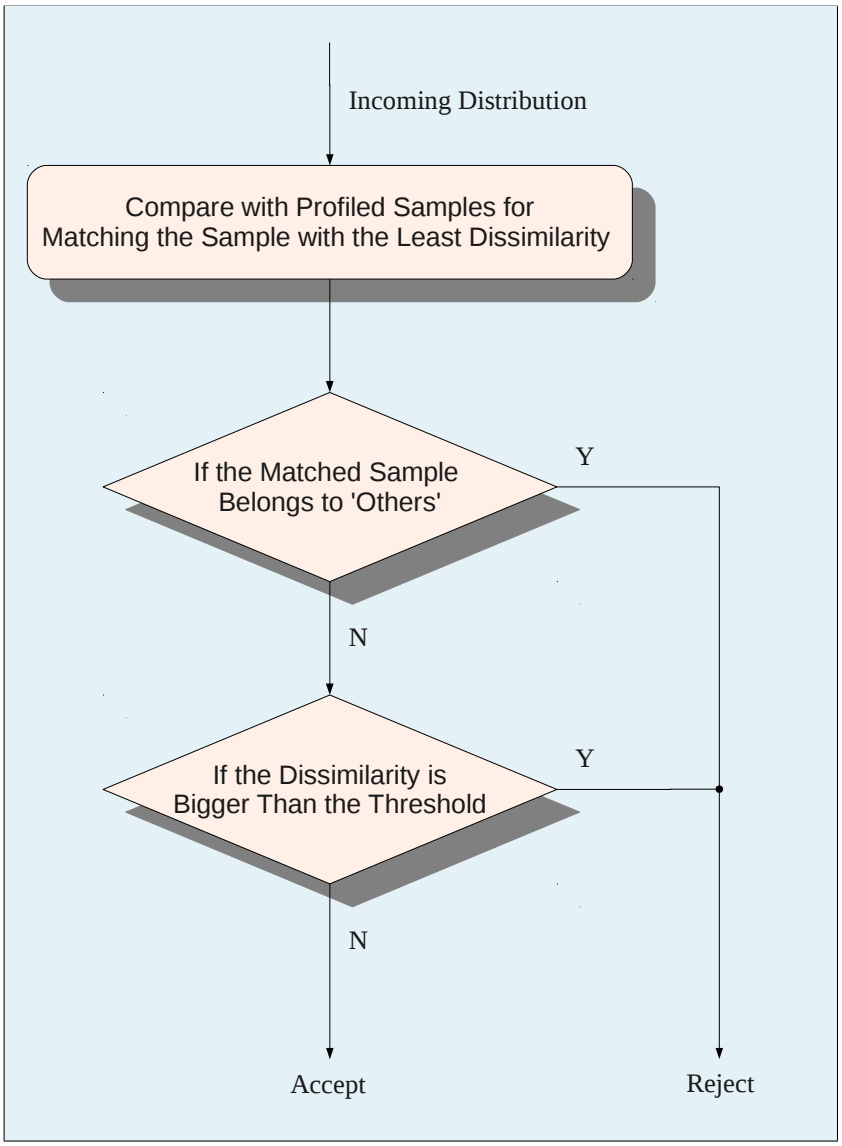

Figure 5.3: Acceptance Threshold in K-S Detector

Since acceptance thresholds determine the dissimilarity tolerance of outputting positives, there is no doubt that higher acceptance thresholds cause a higher numbers of false positives, and lower thresholds cause a higher number of false negatives. In addition, different classes of network traffic have different variations of packet size distribution. Therefore, an optimised acceptance threshold should be determined for each K-S detector. Because the dissimilarities produced by K-S

comparison range from 0 to 1, the acceptance thresholds should also range from 0 to 1. In order to reduce the computational complexity of searching the optimised values, the acceptance thresholds are tried in 0.1 intervals. Considering 0 and 1 are meaningless, nine acceptance thresholds in total are considered in this work.

Applying acceptance thresholds might prevent detectors outputting a false positive when unknown network traffic is encountered. For instance, if a flow produced by MSN Messenger is input into the system and this kind of traffic has not been profiled as a specified traffic class or unspecified traffic class in 'Others', there is a possibility that this flow may be matched with a sample, which belongs to other traffic classes instead of samples in the class of 'Others'. Therefore, a false positive is produced. By introducing acceptance thresholds, the matched samples need to be similar enough with the incoming flows to produce positives, which could solve this problem.

5.1.4 Decision Making Mechanism

In order to keep all detectors working independently in parallel, the results obtained from these detectors only accept or reject the hypothesis that the incoming traffic flows belong to corresponding classes. As a result, controversial results might obtained, as caused by multiple acceptances. The decision making mechanism is designed for solving this problem.

The simplest way is only outputting the class generated by the detector with the highest confidence. The confidences of the detectors are expressed by the classification precisions as we discussed in Section 4.2. The precisions stand for the rates of the true positives out of all predicted positives. Therefore, the detector with the highest precision gives acceptance with the highest confidence. The precisions are counted by previous classifications and stored in the database.

Although, normally only a single class name should be the final output for a incoming flow, outputting all possible classes may be useful in some situations. For network intrusion detection, all possible potential threats should be examined carefully. For example, if a flow is accepted by two detectors, and the traffic classifier is deployed within an internal network for security purposes, it is better to output both acceptances for further inspection even if the unsuspicious traffic class has higher confidence. On the other hand, the combination of multiple

acceptances may provide extra information for determining the traffic classes. Further processing algorithms, such as Bayesian networks, can be employed for higher accuracy. Due to uncertain usages of the classifier and the small amount of data, the decision making mechanism only outputs the final result with the highest confidence in this work. However, using multiple acceptances will be discussed when evaluating the performance.

5.2 Training, Validating & Testing

As was addressed in the previous section, there are some parameters that should be investigated before putting the whole system into practice. First of all, the best classification methods or algorithms for detecting each traffic class should be determined. Secondly, the best detection window for each class should be found. For detecting classes using the K-S distribution test, acceptance thresholds should also be found. In order to make a final decision, the precision of each detector need to be determined for use as detection confidences.

All of the parameters mentioned above should be worked out and stored in the database before testing the system performance with the actual dataset. Because these parameters are influenced by a variety of factors, which are not easy to determine analytically, they were determined empirically by an extra validating dataset.

Firstly, each detector is trained with a training dataset with all possible combinations of parameters. The detectors are configured with different algorithms including k-NN, decision tree, ANN and K-S. Different detection windows ranges from 1 packet to 500 packets are taken into account. For K-S classification, different acceptance thresholds from 0.1 to 0.9 with 0.1 intervals are considered. Then, trained detectors are tested with the validating dataset in order to find the optimised parameters. The classification performance of the validating dataset with optimised parameters can be obtained and stored in the database for decision making. Next, the detectors are retrained with the training dataset and the validating dataset. Finally, an unseen testing dataset is used for evaluating the performance of the optimised system. This process can be expressed by Figure 5.4.

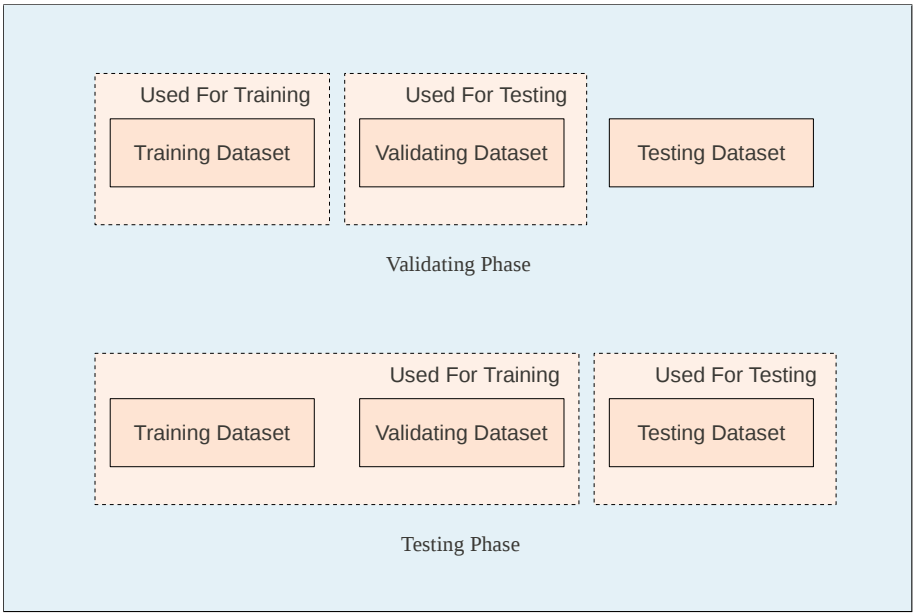

Figure 5.4: Validating and Testing Phases

In the validating phase, the detectors are trained with different detection windows. Therefore, the flow summaries are chopped into 1 to 500 packets before calculating the attributes or distributions. For training the parametric detection models, the calculated attributes for each flow are fed into the detectors. As we presented in Section 3.2, k-NN simply stores the samples and their classes, decision tree builds a series of decision rules based on attribute values and ANN adjusts the weights in the neural network. As we discussed in Section 3.3 and Section 3.4, for training non-parametric detectors, the distributions of the samples and their classes are constructed and stored in the database together with their classes. After training, detectors with different parameters are obtained. In order to get the optimised parameters, the validating dataset is tested with different trained models, and for each detector, the parameters with the best performance are selected. The input flows are chopped into sub-flows as the same detection windows as the trained models, then attributes or distributions are calculated, and they are fed into the detectors for testing. As stated in Section 4.2, there are two measurements for performance, which are recall rates and precisions. Recall rates indicate the detector abilities for detecting positives, and the precisions are the confidence of the detected positives. As a result, selecting the parameters based on recall rates or precisions may affect the overall performance of the whole system. This issue will be addressed more deeply in Section 6.1.

In the testing phase, for evaluating the performance of the proposed system, an additional testing dataset is used, which is totally unseen by the system. The

detectors are retrained with the training dataset and the validating dataset using optimised parameters. Then, the flows in the testing dataset are chopped with the optimised detection windows for each detector; then the attributes or distributions are calculated depending on the optimised algorithms. Next, they are sent to detectors, and binary results are produced. The optimised acceptance thresholds are used for producing the binary results when using K-S classification. Finally, the binary results for each flow are fed into the decision making mechanism, where final output is produced with the help of detectors' precisions obtained in the validating phase. Therefore, the overall performance of the proposed system can be evaluated by comparing the final outputs with the known answers.

5.3 Datasets for Optimisation Evaluations

As stated in Section 5.2, apart from the training dataset and the testing dataset, an extra validating dataset is required for determining the parameters. These three datasets should be sampled independently, which means TCP flows in the datasets should be generated by different operational behaviours, different network environments, different software, etc. Otherwise, the trained classification models and validated parameters would lose their generalities. In other words, the detecting models might be over-fitted with the training dataset and the optimised parameters might be over-fitted with the validating dataset.

The traffic data used for the evaluation is the same as we used in the preliminary test. Some of the traffic is captured in a test bed with different software, some of the traffic is captured on the Internet. Referring to Table 4.1, 10 different specified traffic classes are considered and one special class called 'Others' contains various unspecified traffic classes. Instead of splitting the dataset into two parts in the preliminary test, we randomly sampled the dataset equally into three parts as the training dataset, the validating dataset and the testing dataset. As a result, for each dataset, every specified traffic class has 10 TCP flows and there are 300 TCP flows in the unspecified class of 'Others', which are undersampled from a population that contains around 3000 TCP traffic flows. Random undersampling could ensure the independence of the datasets.

For building the binary traffic detectors, modifications are needed for the training dataset. After the TCP flows are reconstructed, the classes of flows are changed based on the detectors. For example, for training an IRC detector, 10 IRC samples

together with 300 flows of 'Others' are used as input, and the rest of 90 samples belong to other specified classes are changed to 'Others' before inputting to the detector. As a result, the IRC detector are trained with 10 IRC sample flows and 390 flows belong to all other kinds of traffic. Therefore, each trained detector only has the ability of detecting single specified traffic. For the validating phase, the class of all flows is simply marked as 'Unknown' and sent to detectors constructed by different parameters. By comparing the outputs of the detectors with the original classes of the flows, the parameters showing the best performance can be determined. For the purpose of testing the model, the classes of input flows are modified to 'Unknown' as well. Then, they go through the optimised parallel classifier and the decision making mechanism. Therefore, the performance of the proposed system can be evaluated by comparing the outputs with the original classes of the testing flows.

5.4 Implementation of the Proposed System

Although some parts of the system implementation have been briefly presented in the preliminary test, each step of implementing the proposed system will be presented more deeply in this section, including coding the programmes, flow charts of the programmes, and the steps of training, validating and testing.

Since only off-line classification is applied in this experimental system, and the datasets are relatively small, the datasets are stored in pcap files and the database is simply implemented by several plain text files, which are easy for accessing and maintenance. For actual implementation, exhaustive searching for the optimised parameters was used. Initially, we trained the detectors with a combination of parameters and different algorithms, then tested the detectors with the validating dataset and recorded the performance. This process was repeated until all parameters were tried. Then, the optimised parameters were selected based on the performance. Therefore, detectors were retrained with the training dataset and the validating dataset using optimised parameters. Finally, the testing dataset was input into the detectors and final results were produced by the decision making mechanism. A training and validating loop is shown in Figure 5.5, where only one detector is considered. The implementation of the testing process is shown in Figure 5.6. Most of the programmes are coded using Perl script, which are ended with *.pl* and listed in Appendix C. For building and testing parametric models, Weka was used, which is a Java based toolbox for data mining. Bash scripts were

used for calling the Java functions of Weka in command line, which makes it easy to embedded in our automatic system.

In Figure 5.5, only one detector is considered for training and validating. The parametric and non-parametric algorithms are both applied. Detection windows from 1 to 500 and acceptance thresholds from 0.1 to 0.9 with 0.1 intervals were considered. Therefore, for each parametric algorithm, 500 models with different detection windows were built and tested; for K-S classification, 4500 models with different detection windows and acceptance thresholds were built and tested. All of the results were stored in the *performance.csv*. Since 10 classes of traffic are considered in this work, 10 detectors have been built and this process was repeated for each detector.

For the testing phase shown in Figure 5.6, the parameters with the best performance were used for retraining the detectors with the training dataset and the validating dataset, including classification algorithm, detection window and acceptance threshold if the non-parametric algorithm is selected. The performance of the optimised parameters was also retrieved for decision making purposes. Then, the detectors were tested with testing dataset, and the decision making mechanism outputted the final results for evaluation.

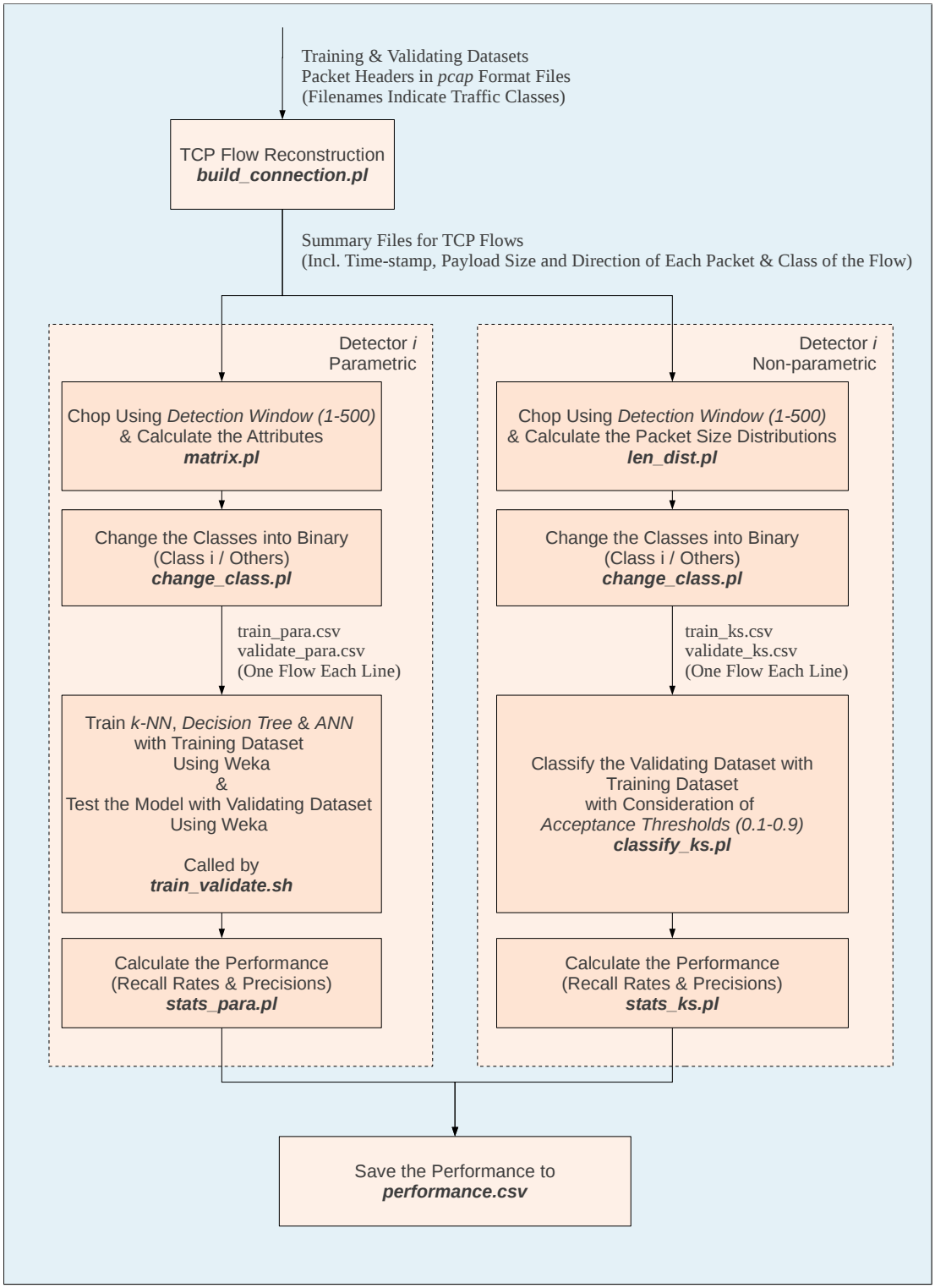

Figure 5.5: Implementation of Validating Phase

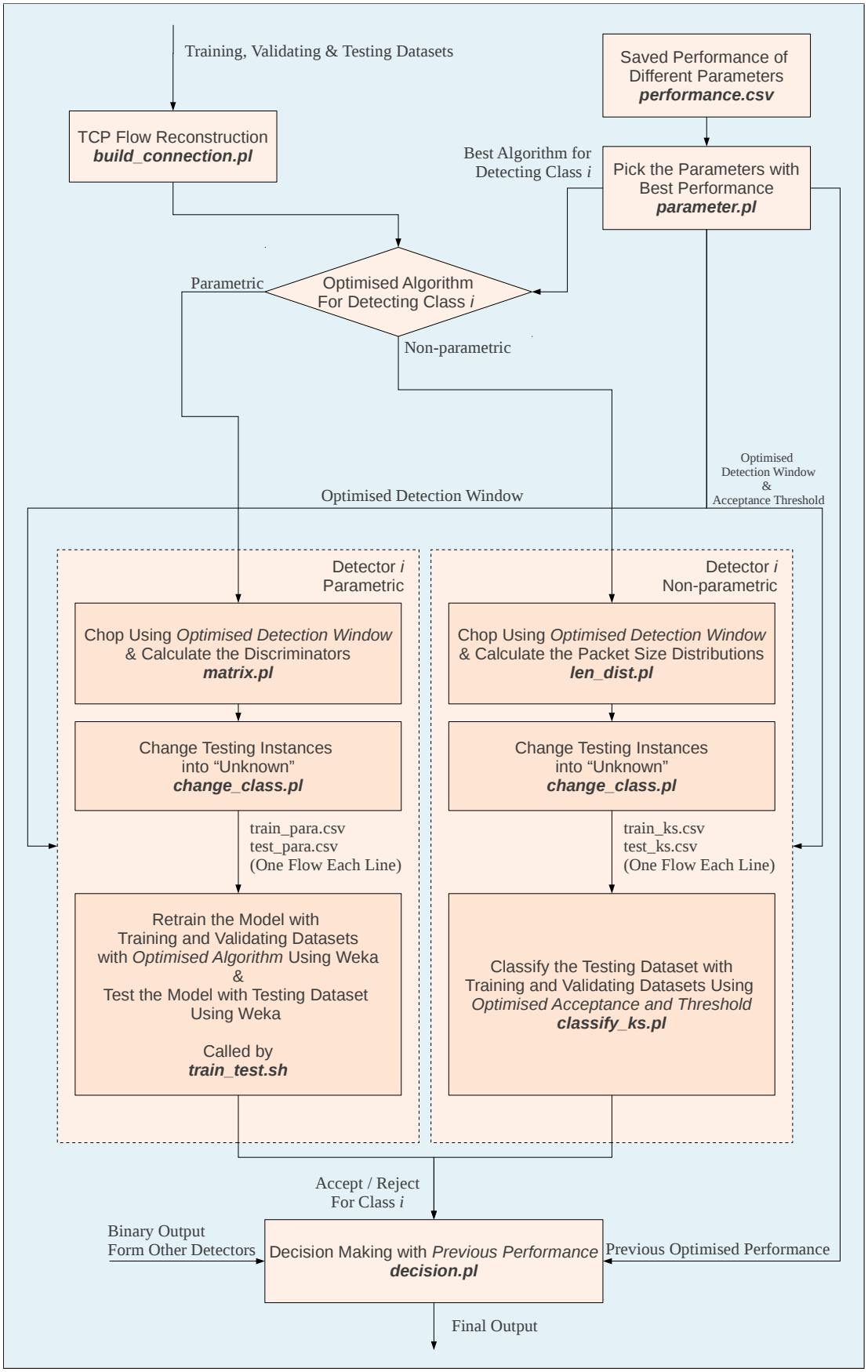

Figure 5.6: Implementation of Testing Phase

5.5 Summary

In this chapter, the architecture of the proposed hierarchical network classifier has been discussed, including the principles of parallel classifier and the decision making mechanism. By using detectors for dealing with different classes separately, different algorithms and parameters could be applied for detecting different traffic classes. The methodology of optimising these parameters is presented, together with the datasets used in the experiments. In the last part of this chapter, the practical implementation of training, validating and testing processes are illustrated.

In the next chapter, some experimental results will be evaluated and compared with single algorithm classification and unoptimised classification.

CHAPTER 6

Result Evaluations

In this chapter, the results obtained from the system presented in the previous chapter will be evaluated, including the optimised parameters produced by the validating phase and the classification results produced by the testing phase. In addition, the performance of the proposed system will be justified.

6.1 Optimised Parameters

As was previously stated, besides the best classification algorithm, the detection windows for both parametric classifications and non-parametric classifications should be optimised. Additionally, for non-parametric classifications, acceptance thresholds should be optimised. Then, for detecting each traffic class, the algorithm and the combination of parameters with best performance are selected as optimised parameters. In the following subsections, how to select the parameters is covered, and the optimisation of parametric classifications and non-parametric classifications will be discussed separately followed by overall optimisation among all algorithms and parameters.

6.1.1 Parameter Selection

As discussed in Chapter 4, in order to evaluate the performance, recall rates and precisions can be considered. In addition, because every detector is recognised as a binary classifier, recall rates and precisions for target traffic class and the class of 'Others' can be considered separately. Referring to Equation 4.2, for each individual detector, the recall rates for target class and the class of 'Others' can be defined more specifically in Equation 6.1 and Equation 6.2 respectively,

$$R_t = \frac{TP_t}{TP_t + FN_t} \tag{6.1}$$

$$R_o = \frac{TP_o}{TP_o + FN_o} \tag{6.2}$$

where TP is the detected true positives, FN is the detected false negatives and TP+FN is the all correct classified instances in the dataset with the correspondent class. Therefore, the recall rates stand for the ability of detecting correct instances among all positives for specific classes.

On the other hand, referring to Equation 4.3, the precisions for the target class and the class of 'Others' can be defined more specifically in Equation 6.3 and Equation 6.4 respectively,

$$P_t = \frac{TP_t}{TP_t + FP_t} \tag{6.3}$$

$$P_o = \frac{TP_o}{TP_o + FP_o} \tag{6.4}$$

where TP+FP is all the detected positives. Therefore, the precisions are the ratio of the correct detections to the total positives among all the detected positives, which can be considered as the confidence of outputting a positive for the specific class.

There is another measurement for presenting the general performance of detectors,

which is called accuracy and is shown in Equation 4.1. It stands for the overall correct classified instances out of all instances among all classes. More specifically, for our binary detectors, it can be defined by Equation 6.5.

$$A_b = \frac{TP_t + TP_o}{TP_t + FP_t + TP_o + FP_o} \tag{6.5}$$

It is not easy to discover that using overall accuracies to evaluate the performance of detectors are affected by the number of instances in traffic classes. Fortunately, as we presented in Section 5.3, in each dataset, there are 300 instances initially tagged as 'Others', and 10 instances each for 10 of the traffic classes. Therefore, after changing the classes for feeding into the binary detectors, there are 10 instances for the target class and 390 for the class of 'Others'. Then, the calculation of accuracy for each detector is simplified by

$$A_b = \frac{TP_t + TP_o}{400}$$

As a result, there is no bias when using the accuracies to measure the performance among the 10 detectors and accuracies could represent the general performance of the detectors.

Because the input instances for each binary detector are classified as either target class or 'Others', and we normally only care about the recall rates and precisions for the positives instead of rejected instances for the whole system, R_t and P_t can be used for effectively evaluating the recall rates and precisions for the detectors, which contribute the recall rates and precisions for the whole system. Therefore, by optimising the R_t and P_t, the recall rates or precisions of specific classes for the whole system can be optimised. For the same reasons, by optimising the accuracies of the detectors, which is A_b, the overall accuracy of the whole system can be optimised.

In this chapter, the evaluations mainly focus on the classification accuracy with the assumption that there is no requirement of recall rates or precisions for detecting specific traffic classes. Consequently, by selecting the algorithms and combinations of parameters with the highest accuracy for each binary detector, the accuracy of the whole system may be optimised.

6.1.2 Optimised Parameters for Parametric Classifications

There are three algorithms considered for parametric classifications, which are k-NN, decision tree and ANN. Referring to the previous chapter, 10 detectors are used for detecting 10 traffic classes. For each parametric algorithm and traffic class, detection windows from 1 to 500 were tried in the validating phase. The accuracies versus detection windows of the 10 parametric detectors are shown in Figure 6.1 – Figure 6.10. The consistency of the results is shown in my published paper [WP10a], where multiple experiments are performed.

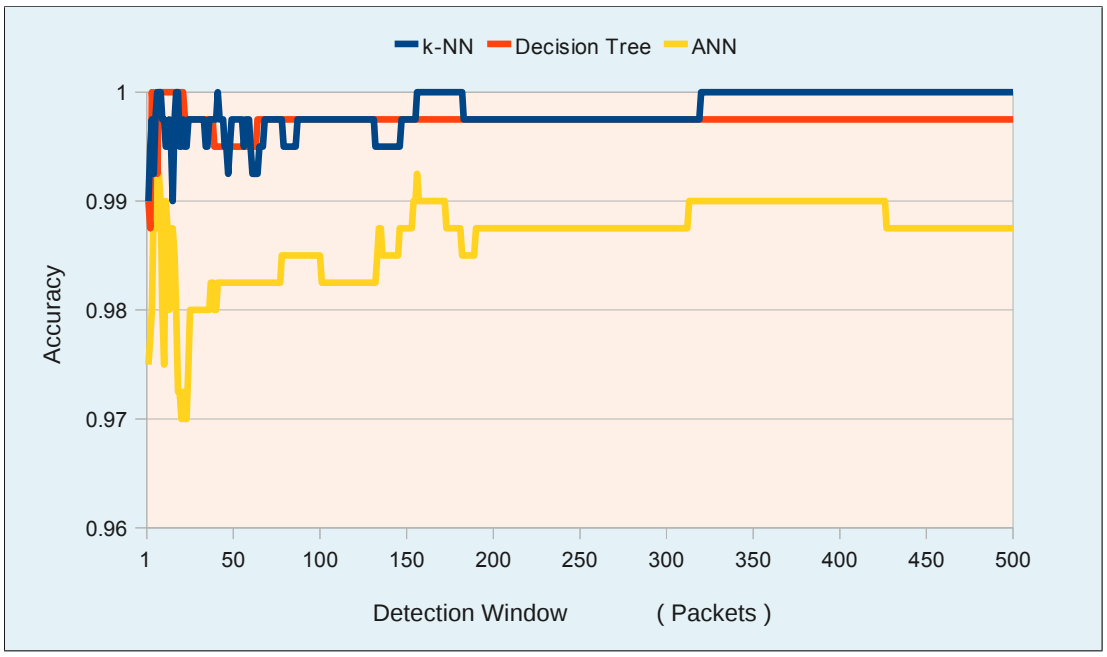

Figure 6.1: Accuracies of Parametric FTP Detectors with Different Algorithms vs Detection Windows

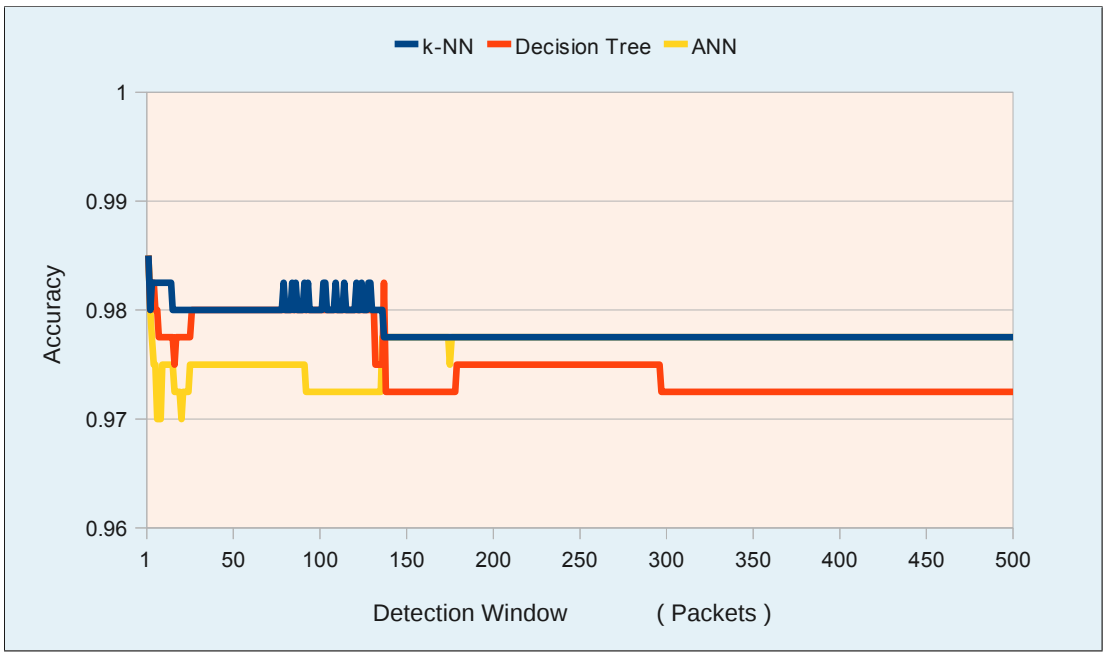

Figure 6.2: Accuracies of Parametric FTP-Data Detectors with Different Algorithms vs Detection Windows

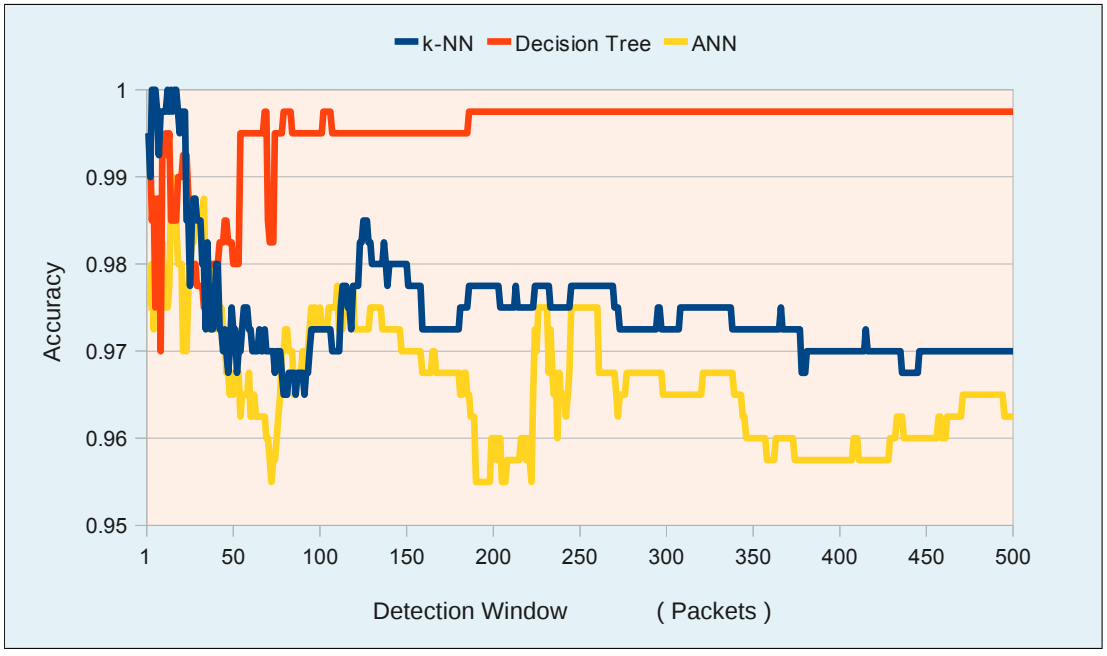

Figure 6.3: Accuracies of Parametric IMAPS Detectors with Different Algorithms vs Detection Windows

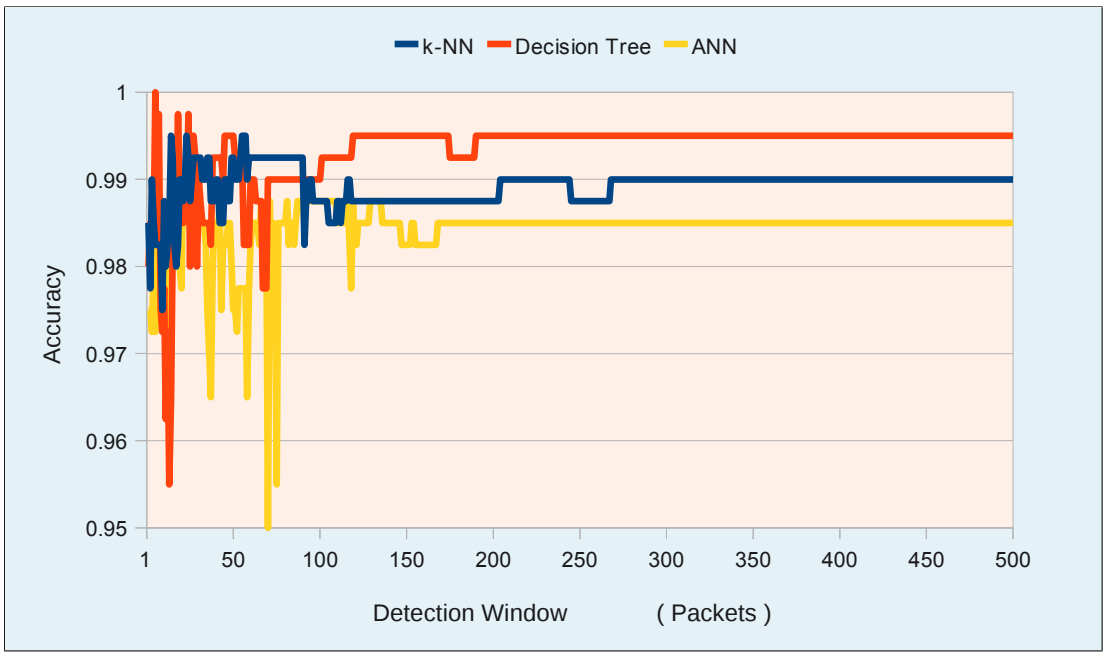

Figure 6.4: Accuracies of Parametric IRC Detectors with Different Algorithms vs Detection Windows

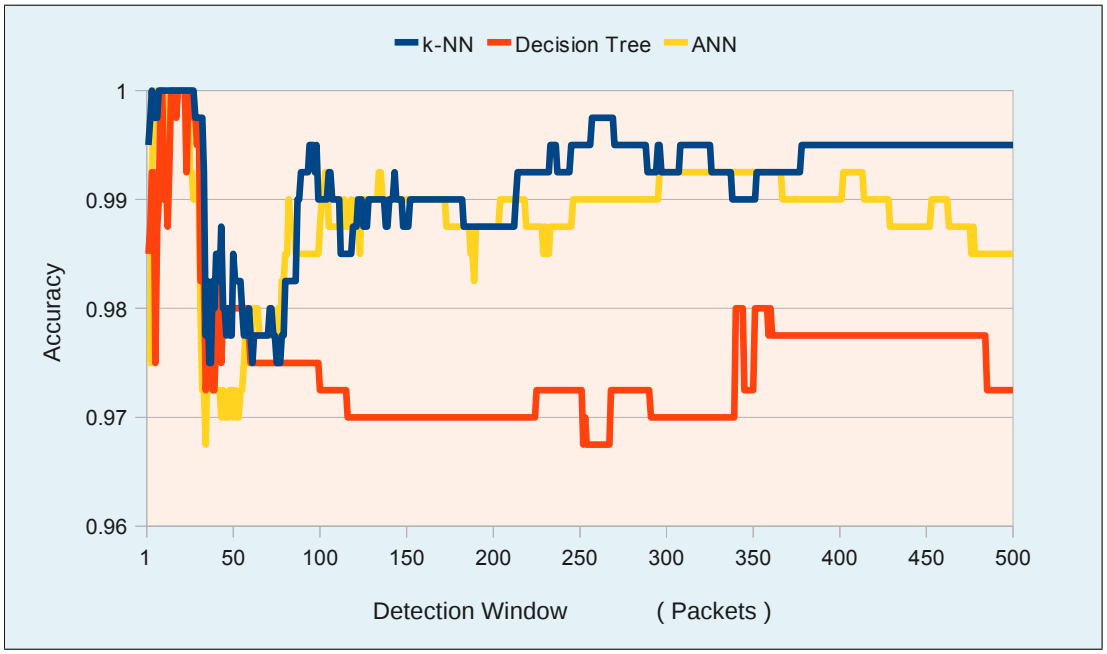

Figure 6.5: Accuracies of Parametric MS-RDP Detectors with Different Algorithms vs Detection Windows

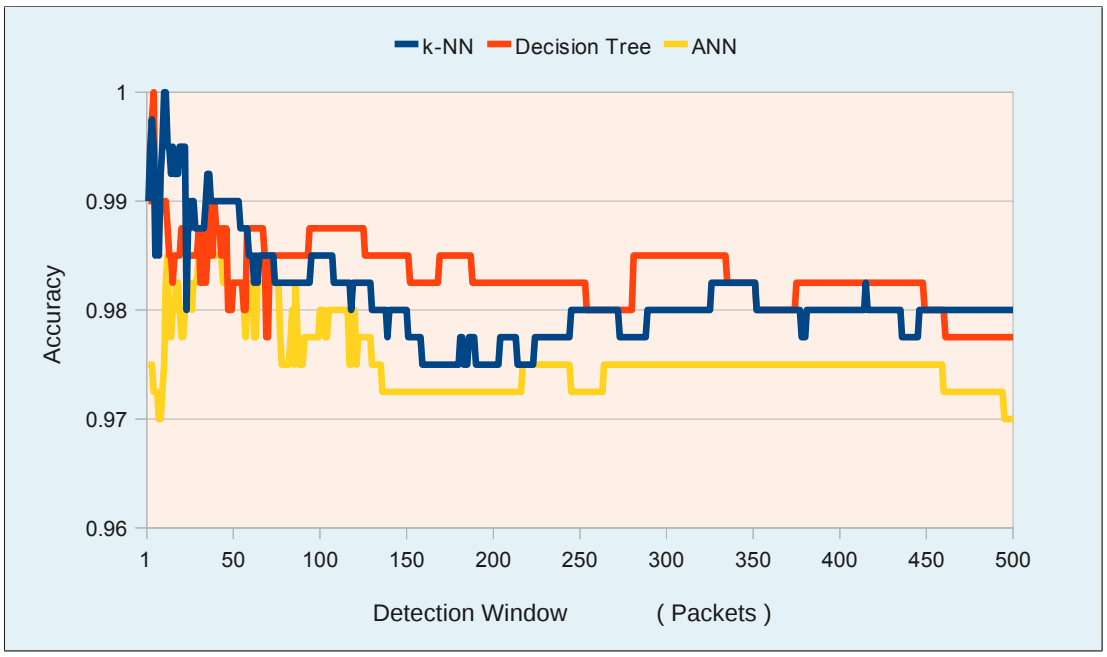

Figure 6.6: Accuracies of Parametric POP3 Detectors with Different Algorithms vs Detection Windows

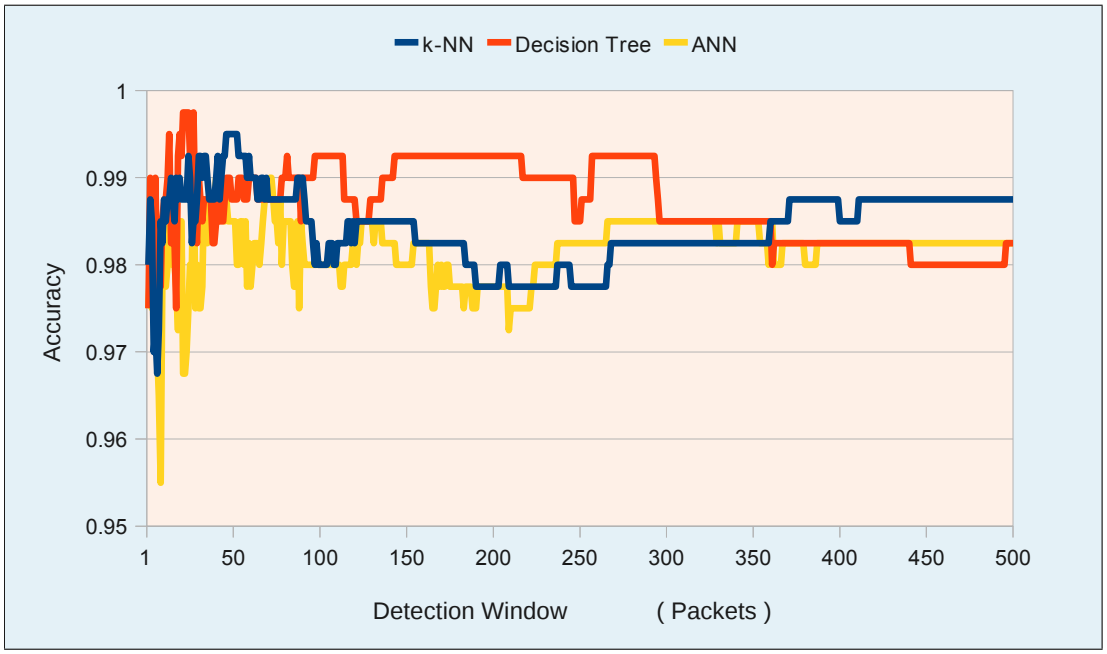

Figure 6.7: Accuracies of Parametric RTSP Detectors with Different Algorithms vs Detection Windows

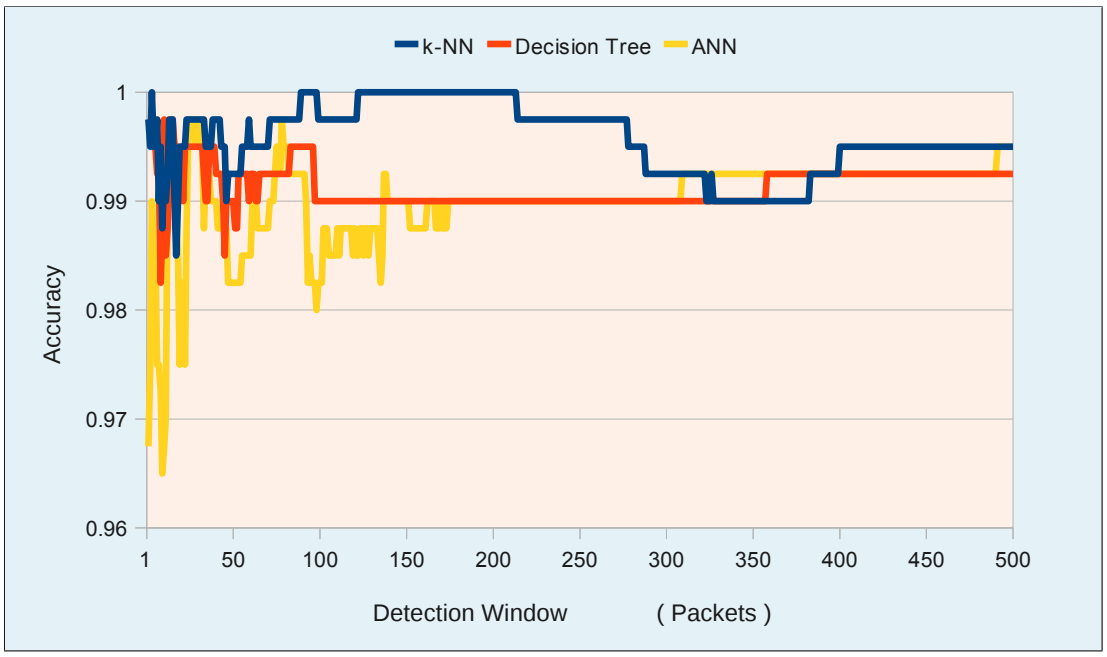

Figure 6.8: Accuracies of Parametric SMTP Detectors with Different Algorithms vs Detection Windows

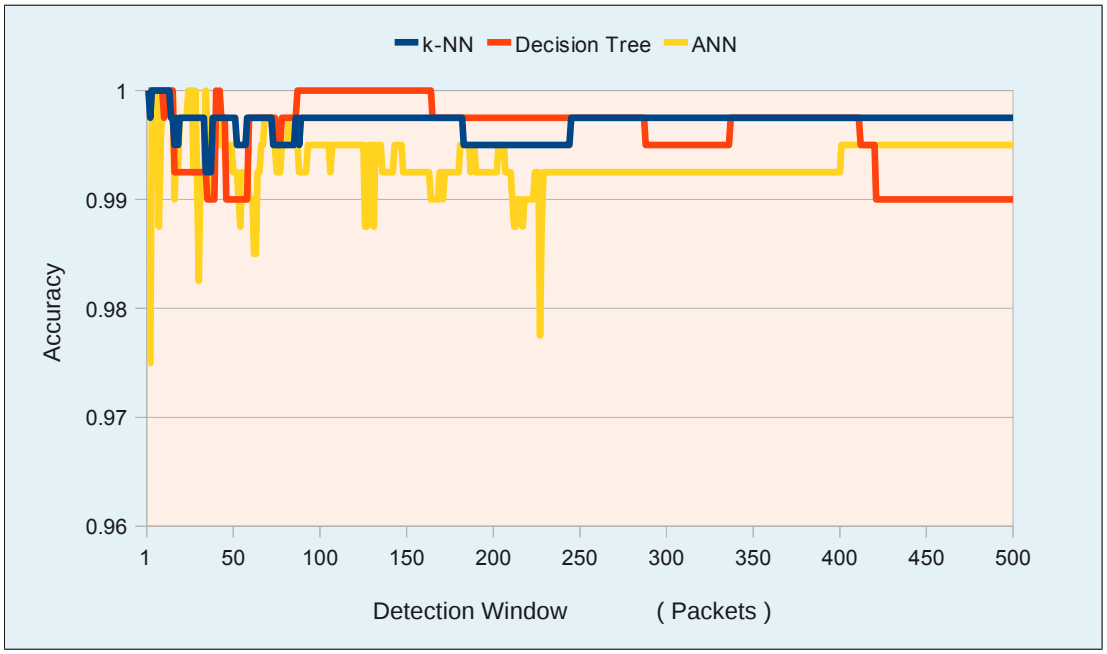

Figure 6.9: Accuracies of Parametric SSH Detectors with Different Algorithms vs Detection Windows

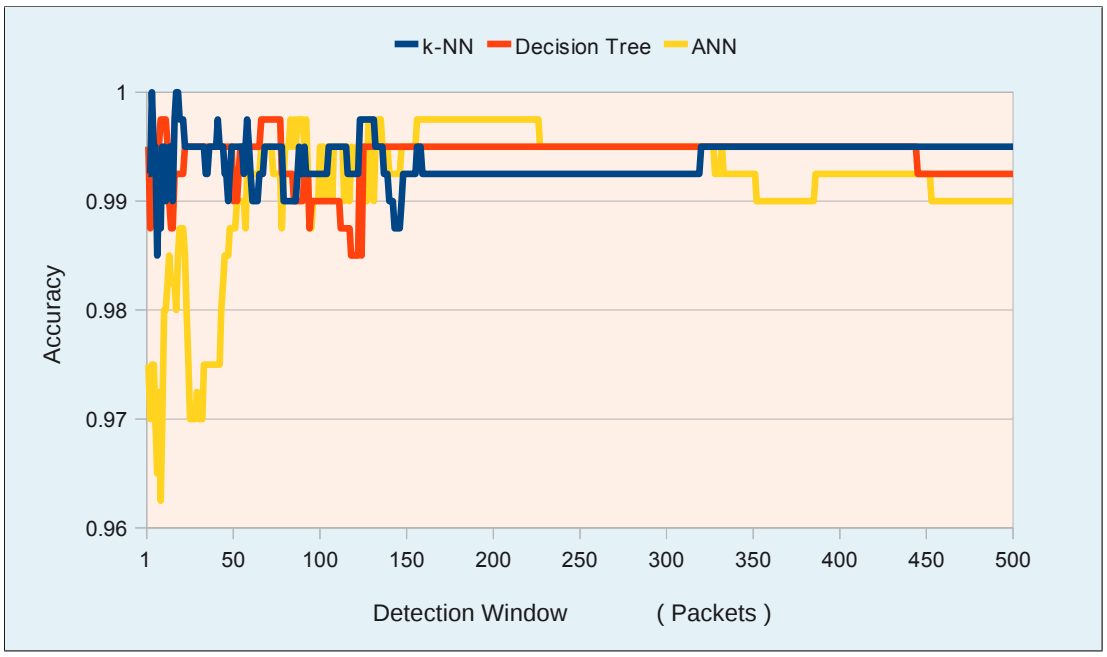

Figure 6.10: Accuracies of Parametric Telnet Detectors with Different Algorithms vs Detection Windows

In the figures shown above, the accuracies are quite high, and are normally above 96%. That is caused by 390 negative instances out of 400 in the validating dataset and the testing dataset. Therefore, the classifiers have very high performance when detecting them. However, the errors will add up after the binary results are fed to the decision making mechanism. The optimisation process for parametric classification selects the algorithm and detecting window with the highest accuracy for each detector.

From the figures, as we may have expected, different algorithms have different performance in some situations. The ANN does not perform very well compared with other two algorithms, especially when detecting the traffic of FTP, IMAPS, IRC, POP3 and SSH. This may be caused by the small number of training instances. Although ANN is supposed to have better regression ability than the other two algorithms, it needs more training instances to adjust the randomly generated neural nodes. The detection windows affect the accuracies as well. The figures show that using only several packets at the beginning of the flows could increase the classification accuracies as we discussed in Section 4.3.

In order to select the best algorithms and detection windows, accuracies should be the first concern, followed by detection speed. In other words, the combinations of algorithms and detection windows should make the detectors at the highest ac-

curacies, and thereafter shorter detection windows are preferred. From the figures, the accuracies are not stable, and sometimes they fluctuate significantly. This happens more frequently when the detection windows are small, because one extra packet may alter the statistical characteristics dramatically when the considered number of packets is small. Therefore, instead of considering the single point of detection windows with the highest accuracy, the average values of accuracies are considered, which means the accuracy curves are smoothed before selecting the detection windows. A smoothing factor f was introduced for this job, which smooths the curves in the following way as shown in Equation 6.6.

$$SA_n = \frac{\sum_{i=-f}^{f} A_{n+i}}{2f+1} \tag{6.6}$$

SA_n is the averaged accuracy for detection window n, which averages $2f+1$ of the original accuracies ranging from $[n-f, n+f]$. Therefore selecting the detection windows with the highest SA_n can produce values that are more stable. By using the averaged accuracies may eliminate some effects caused by retransmission, because only detection windows with the highest accuracy among several packets could be selected, which is less likely to be affected by a few retransmissions within the detection windows. In our experiments, f is arbitrarily set to 1 packet. Take SMTP for example, the original data in Figure 6.8 shows that detection accuracy reaches 100% when using 3 packets for the detection window and k-NN algorithm, but the accuracy is not very stable around the third packet. If the peak is caused by retransmissions in the training or validating datasets, the accuracy may drop dramatically when the testing dataset is applied. On the other hand, if there are retransmissions in the testing dataset instead of the training or validating dataset, the accuracy may also drop dramatically. However, considering the smoothed curve shown in Figure 6.11, using 90 packets as the detection window is preferred, which is much more stable.

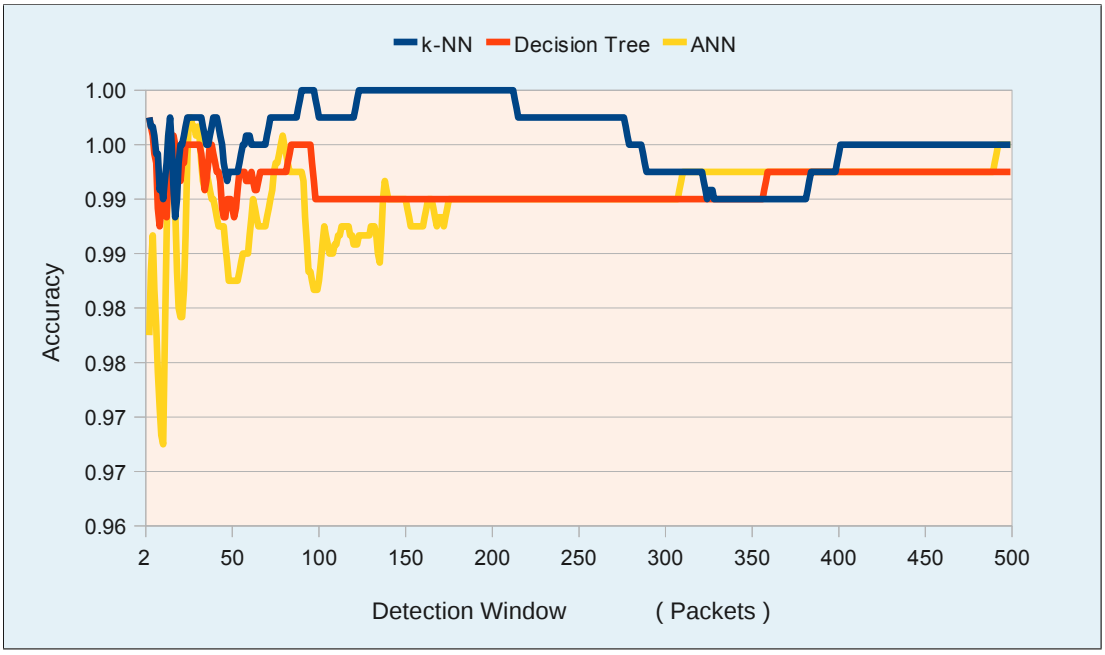

Figure 6.11: Smoothed Accuracies of Parametric SMTP Detectors with Different Algorithms vs Detection Windows

The optimised algorithm and detection window for each detector could, then, be determined with the averaged accuracies. Because a smoothing factor of 1 is used arbitrarily in this work, the optimised detection windows can be chosen from 2 – 499. There is a possibility that multiple algorithms might have the same averaged accuracies when using certain detection windows. For instance, when determining the parameters of the FTP-DATA detector, both k-NN and decision tree reach the maximum accuracy at 2 packets. Therefore, the average accuracy among all detection windows from 2 to 499 packets are considered. Because the k-NN has a higher average accuracy than the decision tree, using the k-NN with 2 packets detection window is selected. The parameters selected for all detectors are shown in Table 6.1, where the *accuracy* is the averaged accuracy values of selected parameters and the $confidence$ is the classification precision for the target classes with selected parameters. Then using these values obtained from validating phase, detectors could be built for testing with optimised algorithms and detection windows, and confidence can be used for final decision making.

We noticed that the classification confidence for FTP-Data is relatively low. By investigating the confusion matrix, there are some false positives produced, which is caused by FTP-Data traffic sharing the statistical characteristics with the traffic made by attaching files in E-mail, copying files with SSH, transferring files with HTTP, etc. Because these traffic is consisted of a very small proportion of packets

for hand shaking and an overwhelming number of MTU packets for transferring batch bytes.

Table 6.1: Optimised Parameters for Parametric Detectors

Detector	Algorithm	Window	Accuracy	Confidence
FTP	k-NN	7	100%	100%
FTP-Data	k-NN	2	98.25%	62.50%
IMAPS	k-NN	4	100%	100%
IRC	Decision Tree	6	99.58%	71.40%
MS-RDP	k-NN	8	100%	100%
POP3	k-NN	10	99.83%	100%
RTSP	Decision Tree	22	99.75%	90.90%
SMTP	k-NN	90	100%	100%
SSH	Decision Tree	2	100%	100%
Telnet	k-NN	17	99.92%	100%

6.1.3 Optimised Parameters for Non-parametric Classifications

As only K-S classification is used as the non-parametric algorithm for the detectors, the optimisation process is searching for the best parameters for the K-S detectors, which are detection windows and acceptance thresholds. Acceptance thresholds influence the accuracies of the detectors, because with the increase of the thresholds, the instances tend to be classified as false positives, and the possibilities of classified as false negatives increase with the decrease of the acceptance thresholds. The optimised acceptance threshold for detecting a specific traffic class is dependent on the distribution variance of the specific traffic class. In addition, the distributions of instances other than the specific target class affect the optimised acceptance threshold of the target class as well. More specifically, if the distribution variance of a target class is relatively large, increasing the acceptance threshold may help in reducing the false negatives. On the other hand, if some instances other than the target class have similar distributions with the instances belonging to the target class, a smaller acceptance threshold could reduce the false positives. Therefore, as for the detection windows, optimised acceptance thresholds for the detectors can be only found out by searching empirically, which is fulfilled by the validating phase in this work. If there are new types of classes to

be taken into consideration or new instances are added into the existing training dataset, revalidation is required.

Since exhaustive searching was used, detection windows from 1 to 500 packets and acceptance thresholds from 0.1 to 0.9 with 0.1 intervals were tried for each detector. In total 4500 combinations of these two parameters were used for validating the detectors. The High Performance Computing (HPC) service at Loughborough University, accelerated the validating process by parallel computing. The scripts were split for validating each acceptance threshold in a single thread. As a result, for 10 detectors and 9 considered acceptance thresholds for each detector, 90 jobs were submitted to HPC for calculating simultaneously.

As for selecting optimised parameters for parametric algorithms, the performance of each detector for each combination of parameters was measured by classification accuracy. The accuracies of the 10 detectors using different combinations of parameters are illustrated in Figure 6.12 – Figure 6.21. The consistency of the results is shown in my published paper [WP10a], where multiple experiments are performed.

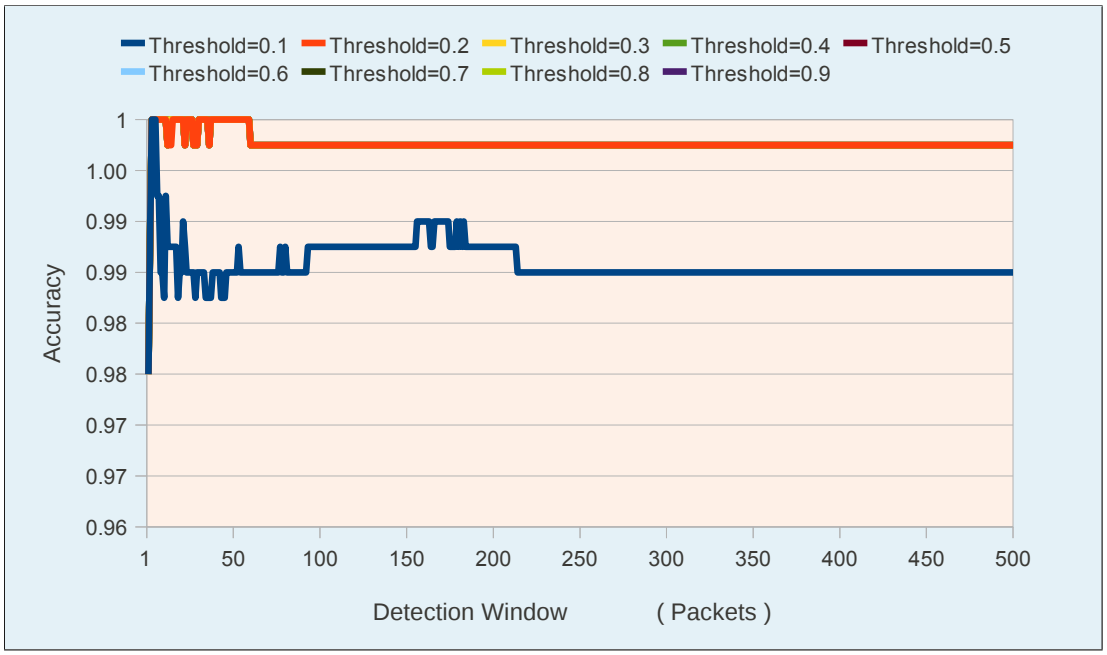

Figure 6.12: Accuracies of K-S FTP Detectors with Different Acceptance Thresholds vs Detection Windows

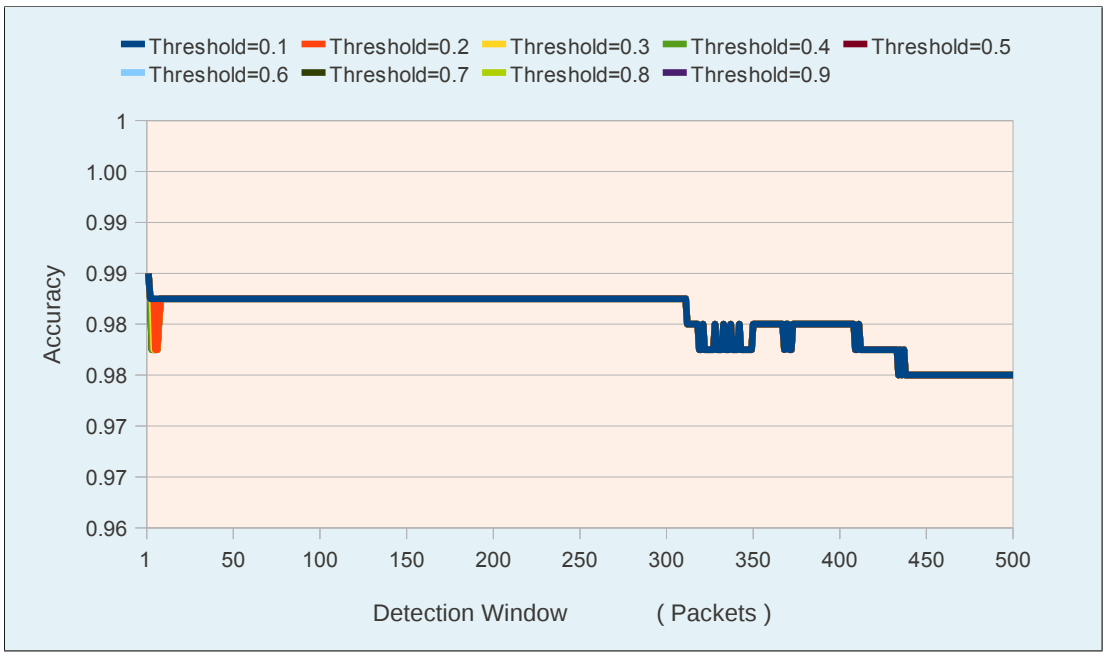

Figure 6.13: Accuracies of K-S FTP-Data Detectors with Different Acceptance Thresholds vs Detection Windows

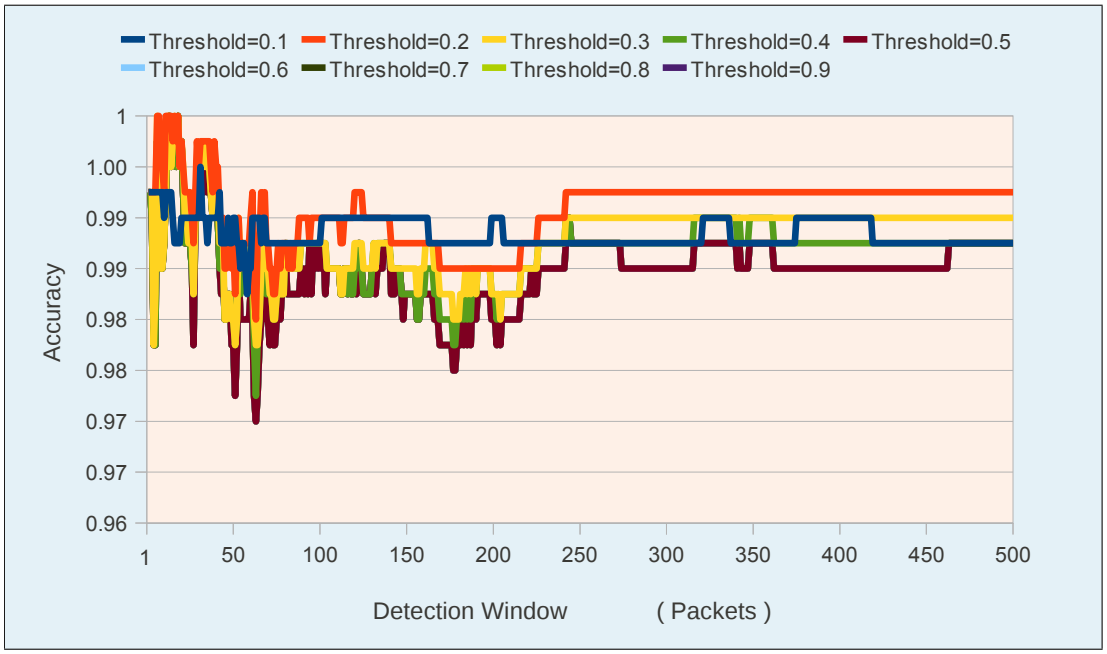

Figure 6.14: Accuracies of K-S IMAPS Detectors with Different Acceptance Thresholds vs Detection Windows

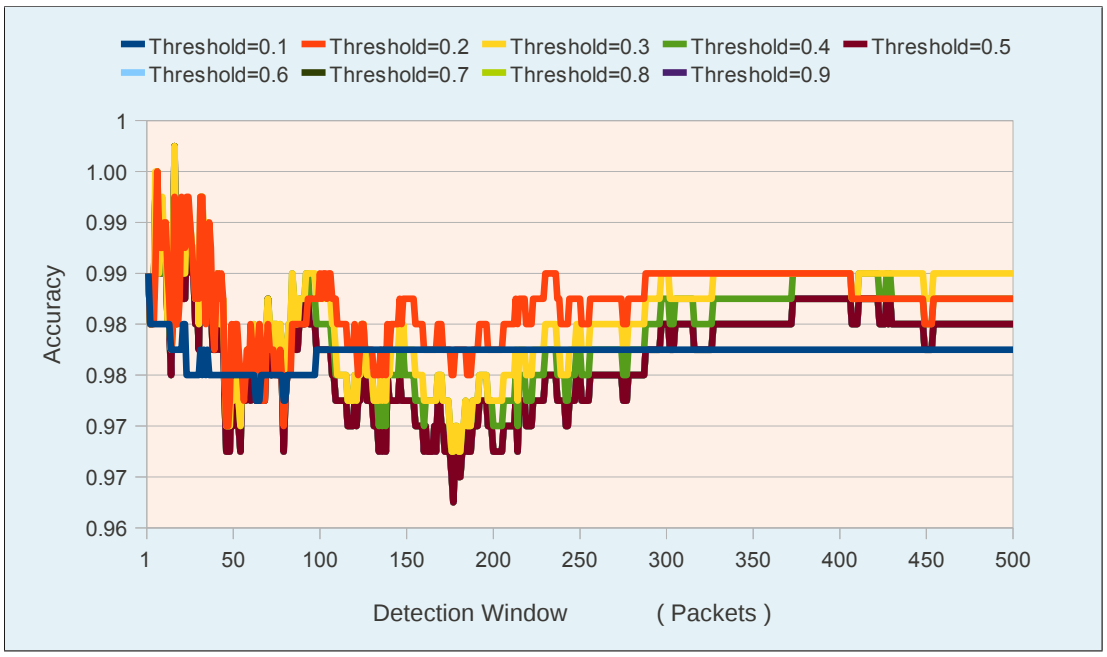

Figure 6.15: Accuracies of K-S IRC Detectors with Different Acceptance Thresholds vs Detection Windows

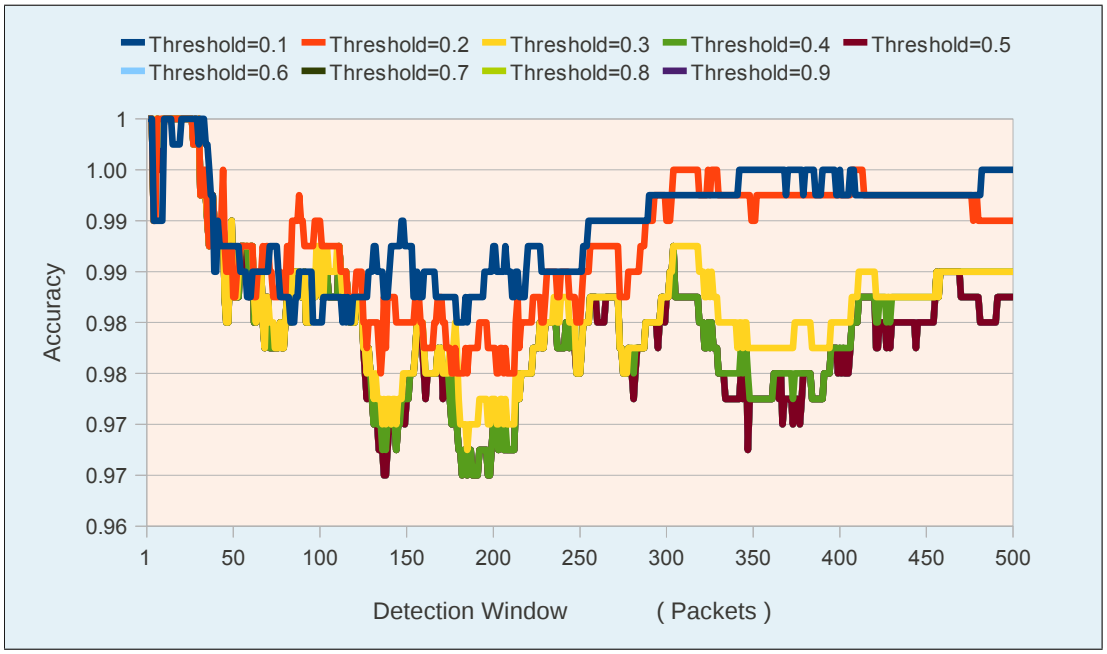

Figure 6.16: Accuracies of K-S MS-RDP Detectors with Different Acceptance Thresholds vs Detection Windows

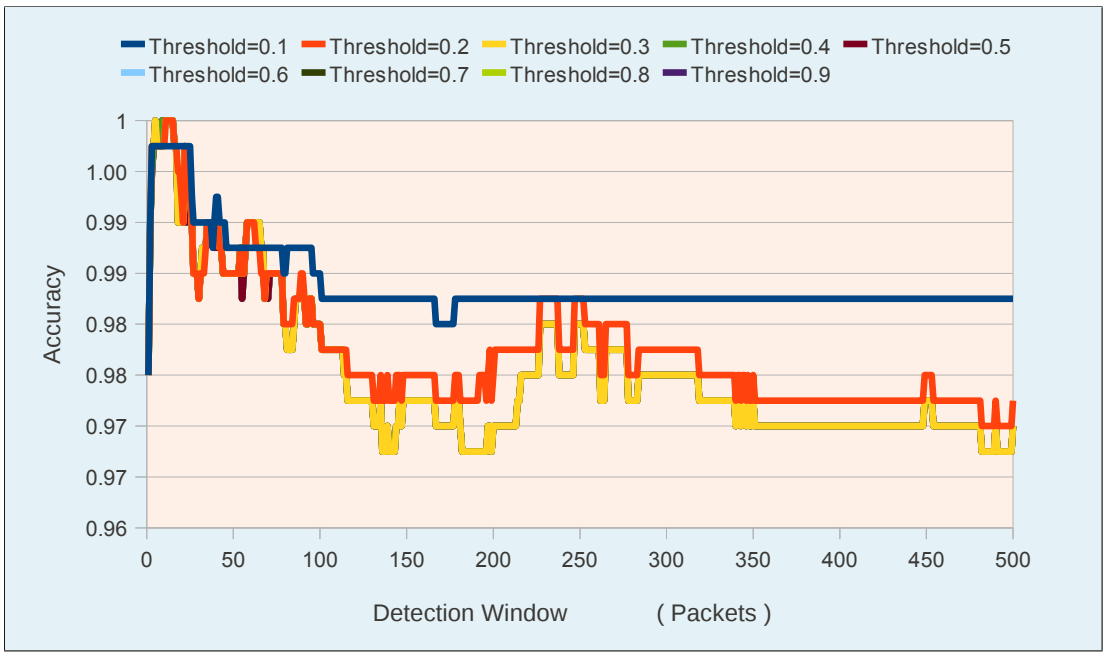

Figure 6.17: Accuracies of K-S POP3 Detectors with Different Acceptance Thresholds vs Detection Windows

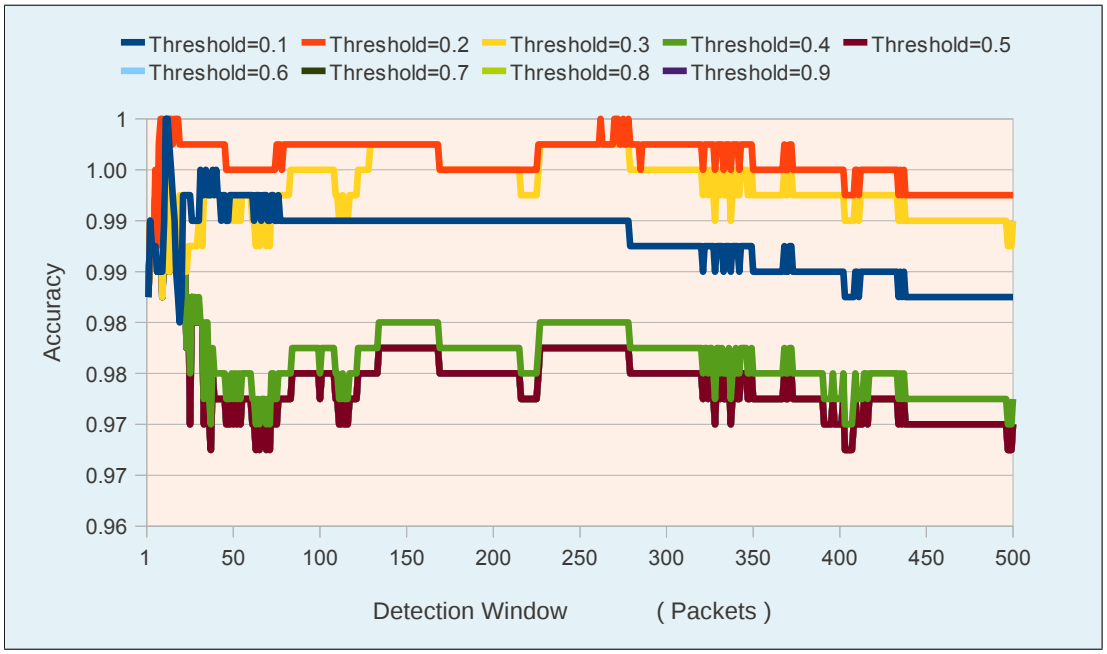

Figure 6.18: Accuracies of K-S RTSP Detectors with Different Acceptance Thresholds vs Detection Windows

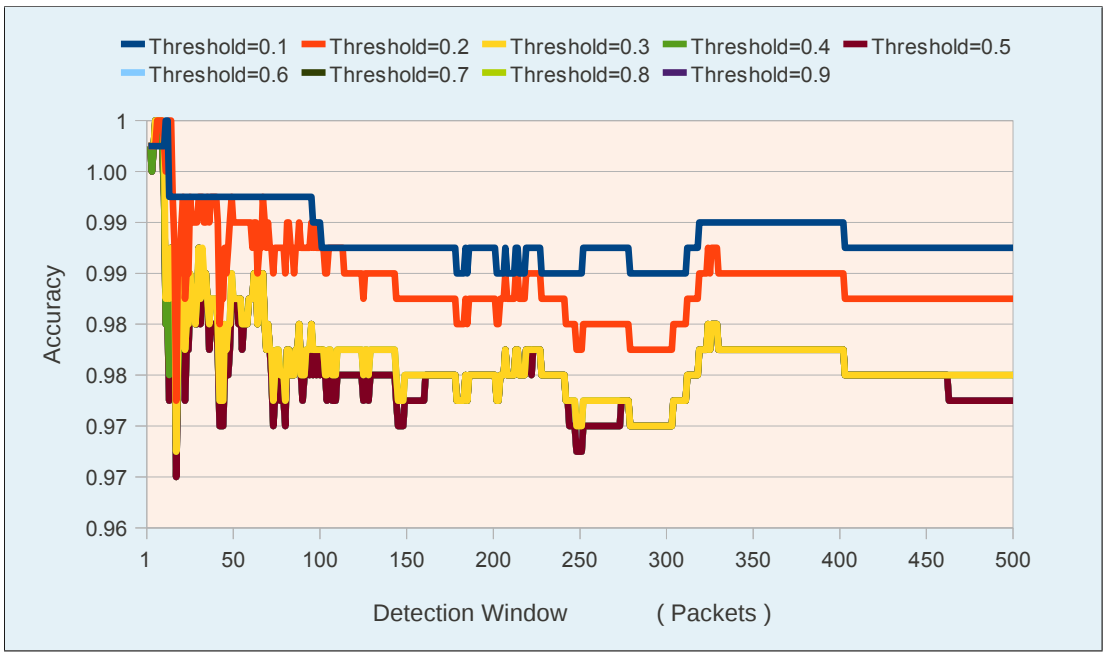

Figure 6.19: Accuracies of K-S SMTP Detectors with Different Acceptance Thresholds vs Detection Windows

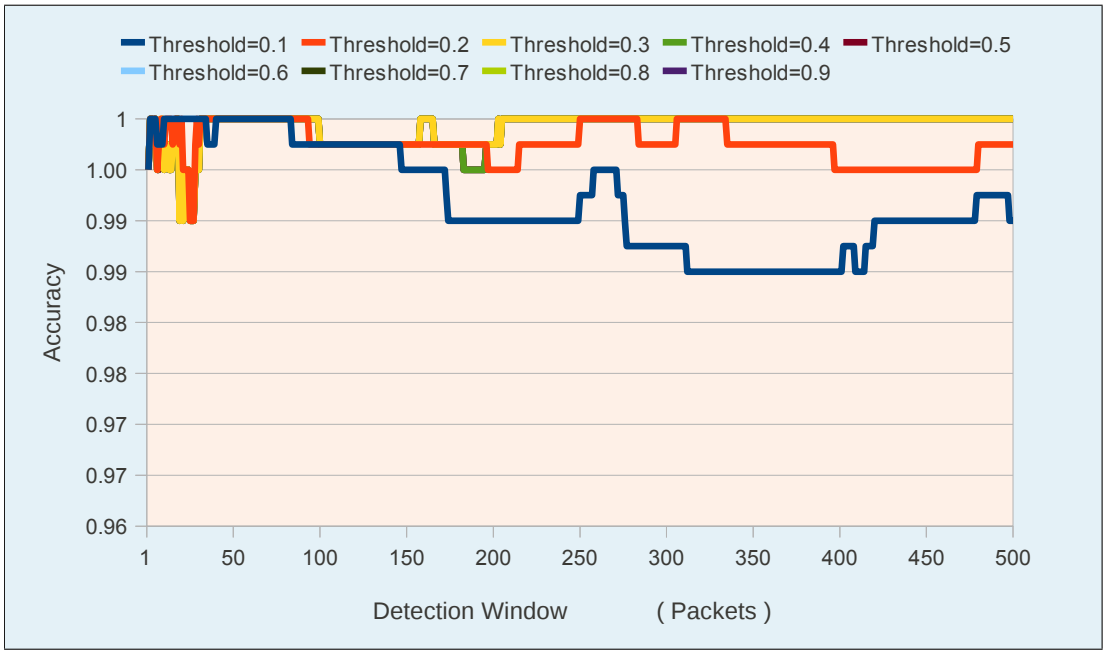

Figure 6.20: Accuracies of K-S SSH Detectors with Different Acceptance Thresholds vs Detection Windows

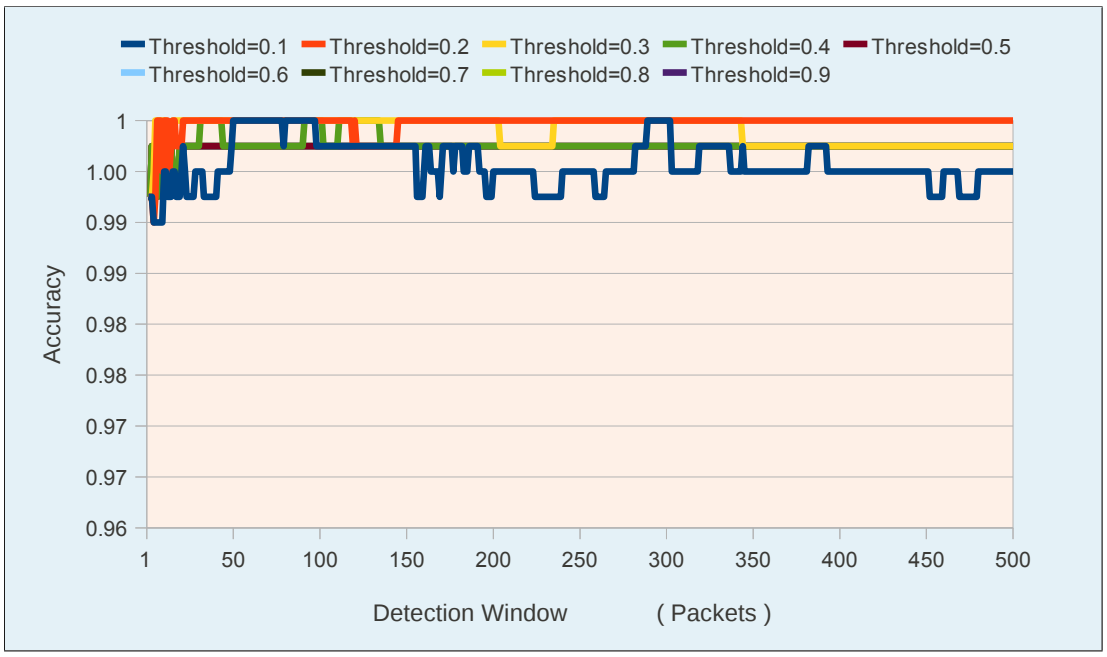

Figure 6.21: Accuracies of K-S Telnet Detectors with Different Acceptance Thresholds vs Detection Windows

As we expected, different acceptance thresholds of detectors shift the detecting accuracies except for the FTP-Data detector. This may be caused be by the low distribution variance of the FTP-Data traffic, because only batch bytes are transferred with MTU sized packets in FTP-Data flows, which cause the packet size distributions to appear identical. The uniqueness of the FTP-Data distributions prevents the other instances being classified as false positives even when the acceptance threshold is large. The acceptance thresholds for other detectors generally shift the curves of accuracies versus detection windows.

There is a similar problem as dealing with parameter selection for parametric classifications, which is that the curves are not very stable for easily selecting the optimised values. Therefore, the same algorithm for smoothing the curves must be applied as we used for parametric classifications. This is expressed in Equation 6.6. The smoothing factor f is also arbitrarily chosen to be 1 here.

For most of the detectors, we have multiple choices of acceptance thresholds for obtaining the maximised accuracies. The detection windows also have multiple choices for obtaining the maximised accuracies. These are caused by the limited number of flows in the datasets, and the limited number of traffic classes considered. However, parameters can still be optimised by applying several rules, which might increase detection accuracies and detection speed. For each de-

tector with detection window w and the acceptance threshold t, the smoothed accuracy of this detector can be expressed as SA_{wt}, where $w \in [2, 3, 4...499]$ and $t \in [0.1, 0.2, 0.3...0.9]$. The rules for selecting the best combination of w and t can be expressed as follows.

1. Select the w and t pair(s) with the highest SA_{wt}.
2. Select the w and t pair(s) with the smallest w.
3. Select the w and t pair(s) with the t, which gives the highest $\sum_{i=2}^{499} SA_{wt}$.
4. Select the w and t pair with the preference of t at the median value among selected pairs.

These rules are applied one by one until only one w and t pair is left. The first step ensures that the pair(s) producing the highest smoothed accuracies are selected. Then, from the selected pairs, the one(s) with the lowest detection window w are selected for faster detection. If there are still more than one pair, the average smoothed accuracies among all detection windows are considered in the third step, where the pair(s) with the t producing the highest average smoothed accuracies are preferred. If there are still some pairs available after the third step, we do not have enough information to determine the w and t for optimising the accuracy or classification speed. However, because we are focusing on the general classification performance and smaller t are more conservative for outputting an *Accept* and vice versa, the t at the median value among existing pairs is preferred at the final step, which balances the possibilities of outputting false positives and false negatives.

Finally, the optimised parameters, which are detection windows w_i and acceptance thresholds t_i, for the ith K-S detector can be determined, which are listed in Table 6.2. In the table, the optimised detection windows and acceptance thresholds are listed in the columns of *Window* and *Threshold* respectively. The columns of *Accuracy* and *Confidence* give the smoothed accuracies and precisions respectively, which are produced by the validating dataset using detectors with optimised w_i and t_i.

Table 6.2: Optimised Parameters for K-S Detectors

Detector	Window	Threshold	Accuracy	Confidence
FTP	4	0.6	100%	100%
FTP-Data	2	0.1	98.33%	80.00%
IMAPS	12	0.2	100%	100%
IRC	24	0.2	99.17%	81.82%
MS-RDP	2	0.1	100%	100%
POP3	10	0.4	100%	100%
RTSP	12	0.2	100%	100%
SMTP	6	0.3	100%	100%
SSH	3	0.2	100%	100%
Telnet	6	0.3	100%	100%

6.1.4 Overall Optimised Parameters

By combining the selected parameters for parametric classifications and non-parametric classifications shown in Table 6.2 and Table 6.1, overall optimised parameters can be obtained, which are shown in Table 6.3.

Table 6.3: Overall Optimised Parameters

Detector	Algorithm	Window	Threshold	Accuracy	Confidence
FTP	K-S	4	0.6	100%	100%
FTP-Data	K-S	2	0.1	98.33%	80.00%
IMAPS	k-NN	4	N/A	100%	100%
IRC	Decision Tree	6	N/A	99.58%	71.40%
MS-RDP	K-S	2	0.1	100%	100%
POP3	K-S	10	0.4	100%	100%
RTSP	K-S	12	0.2	100%	100%
SMTP	K-S	6	0.3	100%	100%
SSH	Decision Tree	2	N/A	100%	100%
Telnet	K-S	6	0.3	100%	100%

For each traffic class, the classification configuration with higher accuracy is pre-

ferred in the first place. Then, detection windows are considered. If two configurations have the same accuracy, a shorter detection window is preferred.

6.2 Classification Results

In the previous section, the optimised parameters for parametric classifications and non-parametric classifications have been obtained from the validating phase. In addition, the overall optimised parameters are also found. Then, detectors can be built, and the testing phase can be preformed as discussed in Section 5.2. The detectors are retrained with the training dataset and the validating dataset with optimised parameters, then the detectors are tested with the unseen testing dataset. In the following subsections, the issues around the decision making process will be covered and the final testing results will be illustrated, including optimised parametric classification, optimised non-parametric classification and overall optimised classification.

6.2.1 Final Decision Making

As discussed in Section 5.1.4, there is a decision making mechanism for dealing with controversial binary results output from the detectors. Essentially, the mechanism is based on the classification confidences of the detectors, which means the acceptance obtained from the detector with the highest confidence is preferred.

Because of the small amount of traffic flows considered in this work, many detectors have a confidence of 100% as shown in Table 6.1, Table 6.2 and Table 6.3. If more than one acceptance is produced by detectors with equal confidence, both acceptance are output to users due to the lack of information. However, in our experiments, this did not happen. Only one instance was accepted by both FTP and IRC detectors. Because the optimised FTP detector has higher confidence than the optimised IRC detector, the instance is classified as FTP flow.

6.2.2 Classification Results of Optimised Parametric Classifier

By applying the configurations listed in Table 6.1, binary detectors using only optimised parametric algorithms can be built and evaluated. The detectors are optimised in terms of different parametric classification algorithms and detection windows.

The general performance of optimised parametric classifier using the unseen testing dataset is listed as follows and the classification confusion matrix is shown in Table 6.4.

Correct Classified Instance:

$$\sum_i (TP_i + TN_i) = 392$$

Incorrect Classified Instance:

$$\sum_i (FP_i + FN_i) = 8$$

Classification Accuracy:

$$\frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = 98\%$$

Table 6.4: Confusion Matrix for Optimised Parametric Classifier

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	10	0	0	0	0	0	0	0	0	0	0
b=RTSP	0	9	0	0	0	0	0	0	0	0	1
c=POP3	0	0	10	0	0	0	0	0	0	0	0
d=SMTP	0	0	0	9	0	0	0	0	0	0	1
e=SSH	0	0	0	0	10	0	0	0	0	0	0
f=FTP	0	0	0	0	0	10	0	0	0	0	0
g=IMAPS	0	0	0	0	0	0	10	0	0	0	0
h=IRC	0	0	0	0	0	0	0	8	0	0	2
i=Telnet	0	0	0	0	0	0	0	0	9	0	1
j=FTP-data	0	0	0	0	0	0	0	0	0	9	1
k=Others	0	0	0	1	0	0	1	0	0	0	298

6.2.3 Classification Results of Optimised Non-parametric Classifier

By applying the configurations listed in Table 6.2, binary detectors using only non-parametric algorithm, which is K-S in this work, can be built and evaluated. The detectors are optimised in terms of detection windows and acceptance thresholds.

The general performance of optimised non-parametric classifier using the unseen testing dataset is listed as follows and the classification confusion matrix is shown in Table 6.5.

Correct Classified Instance:

$$\sum_i (TP_i + TN_i) = 388$$

Incorrect Classified Instance:

$$\sum_i (FP_i + FN_i) = 12$$

Classification Accuracy:

$$\frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = 97\%$$

Table 6.5: Confusion Matrix for Optimised Non-parametric Classifier

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	9	0	0	0	0	0	0	0	0	0	1
b=RTSP	0	10	0	0	0	0	0	0	0	0	0
c=POP3	0	0	10	0	0	0	0	0	0	0	0
d=SMTP	0	0	0	10	0	0	0	0	0	0	0
e=SSH	0	0	0	0	10	0	0	0	0	0	0
f=FTP	0	0	0	0	0	9	0	0	0	0	1
g=IMAPS	0	0	0	0	0	0	10	0	0	0	0
h=IRC	0	0	0	0	0	0	1	7	0	0	2
i=Telnet	0	0	0	0	0	0	0	0	10	0	0
j=FTP-data	0	0	0	0	0	0	0	0	0	5	5
k=Others	0	0	0	0	0	0	0	2	0	0	298

6.2.4 Classification Results of Overall Optimised Classifier

Finally, the overall optimised detectors can be built by applying the configurations listed in Table 6.3, where both parametric and non-parametric classification algorithms are considered. The detectors use the algorithms with the highest accuracies and smallest detection windows, and acceptance thresholds are also optimised when the K-S algorithm is used.

The general performance of the overall optimised classifier using the unseen testing dataset is listed as follows and the classification confusion matrix is shown in Table 6.6.

Correct Classified Instance:

$$\sum_i (TP_i + TN_i) = 390$$

Incorrect Classified Instance:

$$\sum_i (FP_i + FN_i) = 10$$

Classification Accuracy:

$$\frac{\sum_i (TP_i + TN_i)}{\sum_i (TP_i + TN_i + FP_i + FN_i)} = 97.5\%$$

Table 6.6: Confusion Matrix for Overall Optimised Classifier

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	9	0	0	0	0	0	0	0	0	0	1
b=RTSP	0	10	0	0	0	0	0	0	0	0	0
c=POP3	0	0	10	0	0	0	0	0	0	0	0
d=SMTP	0	0	0	10	0	0	0	0	0	0	0
e=SSH	0	0	0	0	10	0	0	0	0	0	0
f=FTP	0	0	0	0	0	9	0	0	0	0	1
g=IMAPS	0	0	0	0	0	0	10	0	0	0	0
h=IRC	0	0	0	0	0	0	0	8	0	0	2
i=Telnet	0	0	0	0	0	0	0	0	10	0	0
j=FTP-data	0	0	0	0	0	0	0	0	0	5	5
k=Others	0	0	0	0	0	0	1	0	0	0	299

From the performance shown above, taking into account both parametric and non-parametric algorithms does not provide higher classification accuracy. On the contrary, the accuracy is slightly decreased by 0.5%. The performance comparison among optimised classifiers can be analysed further, which is listed in Table 6.7.

Table 6.7: Performance Comparison among Optimised Classifiers

Optimisations	Accuracy	Max Window	Avg. Window
Only Parametric	98.00%	90	16.8
Only Non-parametric	97.00%	24	8.1
Overall Optimised	97.50%	12	5.4

From the table, the optimised parametric classifier has a slightly higher accuracy

than the optimised non-parametric classifier. On the other hand, the optimised parametric classifier requires bigger detection windows in terms of the maximum window and average window, which means it needs to examine more packets before decisions can be made. The overall optimised classifier uses detectors with high accuracies and small detection windows among both parametric and non-parametric algorithms. Although the accuracy of the overall optimised classifier can not go higher, the maximum detection window and average detection windows are reduced compared to using optimised parametric and non-parametric classifiers. Because the maximum detection window is the number of packets required for making decisions by all detectors based on the 10 classes of network traffic flows, the reduction of maximum detection window can help the classifier to detect network traffic flows faster, as computational complexity and memory consumption can be reduced. The reduction of the average detection window is also an advantage, which indicates that fewer packets might be required when new traffic classes are taken into consideration. In conclusion, despite only little accuracy decrease, the overall optimised classifier can classify network traffic flows much faster than if considering only parametric algorithms or only non-parametric algorithm.

6.3 Summary

In this chapter, the method of selecting optimised parameters for the detectors from the results of validating is covered, and the selected optimised parameters for parametric algorithms and non-parametric algorithm are listed. In addition, by considering both the optimised parameters of parametric and non-parametric algorithms, the overall optimised configuration for detectors has been obtained.

The performance of the optimised classifiers was obtained by testing, which shows that using only optimised parametric classification has the highest accuracy and using only optimised non-parametric gives lower accuracy but shorter detection windows. The overall optimised classifier gives the shortest detection windows without losing too much accuracy. In conclusion, by combining parametric classification and non-parametric classification, better performance can be obtained in terms of classification speed, and the classification accuracy remains relatively high. In the next chapter, the performance comparison of optimised classifier and unoptimised classifiers proposed by other researchers will be considered.

CHAPTER 7

Performance Comparison

The performance comparison of the proposed optimised classifier and other unoptimised classifiers proposed by other researchers will be demonstrated in this chapter. At the beginning of this chapter, performing the controlled experiments is discussed in order to ensure the same datasets are used, and the proper methodology is applied, which means that the results produced are comparable with our proposed system. Then, the classification results obtained from proposed classifier and the control classifiers are compared and analysed.

7.1 Controlled Experiments

In this section, two kinds of classifiers other than the proposed one will be considered for the purpose of performance comparison. One is a single algorithm classifier with full traffic flows, the other one is single algorithm classifier with a optimised detection window. Before discussing these classifiers, the datasets used for controlled experiments will be covered.

7.1.1 Datasets for Controlled Experiments

The datasets used for the controlled experiments should be similar to the ones used for our proposed system in terms of the number of instances and the percentages used for training, validating and testing.

There are three datasets that have been used in our proposed system: the training dataset, the validating dataset and the testing dataset. Referring to Figure 5.4, in the validating phase, the training dataset is used for training the detectors and the validating dataset is used for testing different possible parameters. In the testing phase, detectors are retrained with the training dataset and the validating dataset, and the detectors are tested with the testing dataset. In summary, the training dataset and the validating dataset are used for building our proposed classifier. The instances and their classes in these two datasets are known by the classifier. The testing dataset is totally unseen by the proposed classifier before evaluating.

In order to keep the same percentage of seen and unseen instances, the same datasets can be used for the controlled experiments. In summary, the training and the validating datasets used before are used again as seen data for building the control classifiers, and the testing dataset used before is used again as unseen data for evaluating the performance of the control classifiers. Since the control classifiers are built and tested with the exact same data as our proposed classifier, the performance of the classification is comparable.

7.1.2 Single Algorithm Classifiers with Full Traffic Flows

These kind of control classifiers use only one statistical algorithm to classify all kinds of traffic as proposed by other researchers [SSM07] [WY08] [HCL+09] [LM07] [TC97] [NSV06] [PBL+03] [Li07] [FZS08]. The parametric algorithms including k-NN [HCL+09], decision tree [WY08] [LM07] and ANN [NSV06] [TC97] are considered, and a non-parametric algorithm, which is K-S [FZS08], is also considered. The attributes used for parametric algorithms are the same as for our proposed system; these are shown in Table 3.1, and the way of constructing the distributions for K-S classification is also the same as for our proposed system, which is described in Section 3.3.

Because the algorithms are predetermined and full traffic flows are used for cal-

culating the discriminators, the validating phase is not needed when building the classifiers. The process is simple, the classifiers are trained and tested with different datasets. As stated in the previous subsection, we used both the training dataset and the validating dataset for building our proposed system, and retrained our proposed system with both training and validating datasets. Therefore, both training and validating datasets should be used for training these kinds of control classifiers. Consequently, the same testing dataset can be applied for obtaining the performance.

Because optimised acceptance thresholds are not used in this kind of classifiers when applying the K-S algorithm, the incoming testing instances are simply classified as the same classes as the matched instances in the database with the smallest distances.

The performance of these classifiers in terms of classification accuracies is listed in Table 7.1.

Table 7.1: Performance of Single Algorithm Classifiers (Full Flows)

Algorithm	Classification Accuracy
k-NN	94.00%
Decision Tree	95.25%
ANN	91.75%
K-S	90.50%

As stated earlier, when an unprofiled flow is input into this kind of K-S classifier, it may be classified as a false positive as there is no maximum acceptance distance employed. This can be discovered by investigating the confusion matrix, which is shown in Table 7.2.

Table 7.2: Confusion Matrix for K-S Single Algorithm Classifiers (Full Flows)

Classified as→	a	b	c	d	e	f	g	h	i	j	k
a=MS-RDP	7	0	1	0	0	0	0	1	0	0	1
b=RTSP	0	9	0	0	0	0	0	0	0	0	1
c=POP3	0	1	7	1	0	0	0	0	0	0	1
d=SMTP	0	0	1	9	0	0	0	0	0	0	0
e=SSH	1	0	0	0	9	0	0	0	0	0	0
f=FTP	0	0	0	0	0	10	0	0	0	0	0
g=IMAPS	1	0	1	0	0	0	8	0	0	0	0
h=IRC	1	0	0	0	0	0	1	7	0	0	1
i=Telnet	0	0	1	0	0	0	0	0	9	0	0
j=FTP-data	0	0	0	0	0	0	0	0	0	5	5
k=Others	6	3	2	3	1	0	1	2	0	0	282

There are 18 instances from the class 'Others', which are misclassified as other profiled classes. This number is more than our optimised K-S classifiers using the optimised acceptance thresholds, which is shown in Table 6.5, where only 2 instances from 'Others' are misclassified.

7.1.3 Single Algorithm Classifiers with Optimised Detection Windows

Additional controlled experiments have been performed by optimising the detection windows as described in [WY08]. Because only one algorithm is used in each classifier, there is only a single global detection window applied in each classifier for classifying all traffic flows. In order to optimise the global detection window, the same validating phase as we used in our proposed system is involved. We trained the classifiers with the training dataset, and tested them with the validating dataset. Then, the optimised detection windows can be obtained. Next, the classifiers were retained with both the training dataset and the validating dataset, and tested with the unseen testing dataset. The classification performance of the classifiers are shown in Table 7.3, where the optimised detection windows are also listed.

Table 7.3: Performance of Single Algorithm Classifiers (Optimised Windows)

Algorithm	Optimised Window	Classification Accuracy
k-NN	14	97.00%
Decision Tree	186	93.25%
ANN	31	89.50%
K-S	6	95.50%

Comparing the results with the classifiers using full traffic flows discussed in the previous subsection, the accuracies still remain high when only several packets are taken into consideration, which makes the classification process faster. However, the decision tree and ANN still need much more packets than our proposed system. When using K-S or k-NN algorithms, relatively higher accuracies can be obtained, and a small amount of packets are required.

7.2 Overall Performance Comparison

The overall performance comparison of our proposed classifier and the control classifiers can be obtained, which is listed in Table 7.4,

Table 7.4: Performance Comparison

Algorithm	Detection Window	Classification Accuracy
Proposed Optimised	2–12	97.50%
k-NN	Full Flow	94.00%
Decision Tree	Full Flow	95.25%
ANN	Full Flow	91.75%
K-S	Full Flow	90.50%
k-NN	14	97.00%
Decision Tree	186	93.25%
ANN	31	89.50%
K-S	6	95.50%

From the table, our proposed classifier has the highest accuracy among all the other control classifiers. Instead of using the statistical characteristics of full traffic flows, using several packets at the beginning can increase the classification speed without losing much accuracy. In our proposed classifier, by employing different algorithms and different parameters, the accuracy goes higher [WP10b]. Although the k-NN and K-S classifiers with optimised detection windows give high levels of accuracy and require less packets, our proposed system still has the following advantages.

Firstly, the localised detection windows in our system can produce some output very quickly. Referring to Table 6.3, only 10 and 12 packets are needed when detecting the POP3 and RTSP traffic respectively, because of the high confidence of the detectors, the rest of the traffic classes can be determined using 6 packets. Secondly, because the detectors are independent to each other, further algorithms or detectors can be added easily, which gives more flexibility when deploying the system in different practical environments. Finally, by using independent detectors a rule-based decision making mechanism could output all possible classes for ambiguous input instances, which gives users more information for taking further action. In addition, the proposed classifier can benefit from parallel computing as each detector can be trained, validated and deployed independently.

7.3 Summary

In this chapter, some control classifiers have been built for performance comparison, including single algorithm classifiers with full traffic flows and single algorithm classifiers with optimised detection windows. The performance comparison between these classifiers and our proposed optimised classifier has been listed at the end of this chapter, which proves our system has the highest accuracy and produces results quicker.

CHAPTER 8

Conclusions & Future Work

In this chapter, the conclusions of this work will be presented, and the results produced by the experiments will be summarised. Finally, the recommendations for future research related to this work will be discussed.

8.1 Conclusions

To understand what kinds of traffic are in operation across packet-switching communication networks is always attractive to network administrators. This information can contribute to network performance management, security enhancement, authorisation and accounting, etc. Although some current network protocols have the ability of providing differential service levels, however, flow-level QoS cannot be provided unless types of traffic are correctly classified. For applying different policies to different flows, the traffic type of each flow should be identified. Therefore, traffic classification plays an important role in network performance management. Network security can also be enhanced with advanced traffic classification, because illegal traffic can be identified and filtered preciously at an early stage. A finer authorisation and accounting can be provided with the help of

flow-level traffic classification.

Traditional network traffic classification techniques are based on information contained in the layer 3 and layer 4 packet headers. For widely deployed TCP/IP protocol stack, port numbers and IP addresses are normally used. Although some applications have designated port numbers, many applications use random port numbers or the port numbers configured by users. Furthermore, the employments of port forwarding and proxy change the port numbers when packets are transmitted across networks. Therefore, classifying traffic based on port numbers is not reliable, especially for intentional masked traffic, where the designated port numbers are normally not used. Although, using layer 3 address together with port numbers could increase classification accuracy by providing additional information about hosts, the classification is still unreliable due to the utilisation of address forging, proxy, etc. The recently developed technique — DPI, has a very high classification accuracy by matching the contents of packets or flows with a database. However, it has a slow processing speed, and encrypted traffic cannot be identified. Because the information of the application layer has been examined, the deployment of the DPI may breach the privacy of network users.

Statistical approaches are newly proposed for network traffic classification, which use statistical characteristics of flows to identify traffic types. Currently, two kinds of statistical discriminators and various algorithms have been proposed by researchers. One is using absolute statistical values to identify traffic types, which is named parametric classifications in this work. The other one is using distributions, which is named non-parametric classifications. In terms of algorithms, parametric classifications use machine learning or artificial intelligence algorithms, and non-parametric classifications use statistical tests. However, parametric classifications and non-parametric classifications have pros and cons for detecting different traffic types. Different algorithms also show different performances for detecting traffic types.

In this work, the main objective is optimising the statistical traffic classifiers. Because the TCP flow controls, which includes Nagel's algorithm, alter the statistical characteristics of traffic flows in some degrees, classifying TCP traffic is more difficult than classifying UDP traffic with statistical approach. Therefore, the optimisation focuses on TCP traffic classification.

Data used for experiments had been collected and some preliminary experiments had been preformed at the beginning of this work. As expected, classification

techniques and algorithms gave different classification performance for different traffic classes. The classification performance also changed with different detection windows. For non-parametric classification, introduced acceptance thresholds affected classification performance as well. Therefore, the optimisation process included searching the best classification techniques, algorithms, detection windows and acceptance thresholds for detecting different traffic types.

A hierarchical architecture had been proposed in this thesis, which employs a series of binary detectors for detecting different traffic types and a mechanism for making final decisions. Because each detector is only responsible for detecting one kind of traffic, different configurations can be applied to the detectors. The final decision making mechanism produces final outputs when controversial results are obtained from the binary detectors. In our experiments, the mechanism only outputs the results from the detectors with the highest confidence.

In order to optimise the configurations of the detectors and obtain the confidence of the optimised detectors, a validating phase had been performed before testing the proposed system. An exhaustive search was undertaken for searching these parameters. During the searching, both parametric and non-parametric techniques were considered. In terms of algorithms, k-NN, decision tree and ANN for parametric classification, and K-S for non-parametric classification were taking into consideration. For each algorithm, the detection windows ranging from 1 to 500 packets were tried. For K-S, the acceptance thresholds ranging from 0.1 to 0.9 with 0.1 intervals were tried. All the combinations had been tested with the validating dataset for obtaining the best configurations for the detectors. Then, the confidence of the detectors could be obtained for the decision making mechanism.

After all the optimised parameters had been found, the optimised detectors could be constructed. Finally, a testing phase had been performed followed by a performance comparison. Two kinds of control classifiers had been considered for performance comparison. One is using full TCP flows for calculating the discriminators and using single classification algorithm. The other one is using an optimised global detection window for calculating the discriminators and using single classification algorithm. Because of the small amount of data, the accuracies from all the classifiers are very high. However, the results still shows that our proposed system has the highest accuracy. The classification speed is another indicator of classification performance. Since the the binary detectors treat traffic classes individually, the detection speed varies. Although, final decisions should not be made before all the results are obtained from the detectors, some decisions can still be

made if the output detectors have high confidence. In some situations, positives are preferred, such as detecting illegal traffic. Therefore, possible outcomes can be output as soon as positives are obtained from the detectors. Referring to Table 7.4, the proposed system also has the advantage in term of classification speed among all control classifiers.

Compared to the control classifiers, the proposed system has flexible architecture. Due to the distributed binary detectors, traffic classes and algorithms can be added easily. By employing parallel computing, detectors can be distributed for faster processing speed. Although the system was optimised based on overall accuracy in this work, optimising the system based on precisions and recall rates for detecting different traffic classes can be achieved with the same process. By slightly changing the decision making mechanism, all the possible classes for incoming flows can be output in the order of confidence. In summary, traffic policies can be applied in a more flexible way if integrated with the proposed hierarchical traffic classifier.

8.2 Future Work

In future work, a greater variety of traffic classes and more samples can be added for testing the performance of the proposed system. In the performance comparison part of this work, the classification performance is quite high and similar among all the considered classifiers. This is because the number of samples and traffic classes used in the experiments is small. By increasing the size of the dataset, more realistic tests can be performed in order to evaluate the performance of the proposed system in a more practical environment.

More classification algorithms can be considered for both parametric and non-parametric classifications. As proposed by other researchers, besides k-NN, decision tree and ANN, SVM can be used for parametric classification. For non-parametric classification, Chi-square test and correlation coefficient can be taken into consideration in the future.

In this work, the exhaustive search was employed for the purpose of optimising the parameters for the detectors. Although the size of the datasets is small, it is still a time consuming job. After adding more algorithms and increasing the size of the datasets, it is probably impossible for optimising by exhaustive search. Therefore, greedy search or genetic algorithms are recommended for employment

in future work.

The aim of the optimisation was based on the overall accuracy in this work. Because of the hierarchical architecture of the proposed system, the optimisation can be based on the precisions or recall rates of the detectors. For example, for strictly detecting illegal FTP traffic inside an internal network, the FTP detector may be optimised based on recall rates, in which situation, outputting positives are preferred. On the other hand, the detectors can be optimised based on precisions if the bias of outputting negatives is required.

The final decision making mechanism can be improved by advanced algorithms, such as machine learning algorithms or statistical probabilities. This requires more training samples for building the model. In practice, instead of only outputting the results with the highest confidence, multiple possible results can be output together with their confidence. This gives more flexibility to other systems connected to the classifier. In summary, the interface between the proposed classifier and traffic shapers, traffic accountants, security systems, etc. can be investigated further.

References

[ARZ09] T. A. Al-Radaei and Z. A. Zukarnain. Comparison Study of Transmission Control Protocol and User Datagram Protocol Behavior over Multi-Protocol Label Switching Networks in Case of Failures. *Journal of Computer Science*, 5(12):1042–1047, 2009.

[AT] S. Aperghis-Tramoni. Net::Pcap Module for Perl. `http://search.cpan.org/~saper/Net-Pcap-0.16/Pcap.pm`. Retrieved 14-Feb-2010.

[BBCD98] S. Blake, D. L. Black, M. A. Carlson, and E. Davies. An Architecture for Differentiated Services, 12 1998. RFC 2475.

[BBM03] G. E. A. P. A. Batista, A. L. C. Bazzan, and M. C. Monard. Balancing Training Data for Automated Annotation of Keywords: a Case Study. In *WOB*, pages 10–18, 2003.

[BCS94] B. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview, 6 1994. RFC 1633.

[Ben09] R. Bendrath. Global technology trends and national regulation: Explaining Variation in the Governance of Deep Packet Inspection. `http://userpage.fu-berlin.de/~bendrath/Paper_Ralf-Bendrath_DPI_v1-5.pdf`, 3 2009. Retrieved 4-Feb-2010.

[Bla98] U. Black. *ATM: Internetworking with ATM*. Prentice Hall, 1998.

[Bra89] R. Braden. Requirements for Internet Hosts – Communication Layers, 10 1989. RFC 1122.

[Bra08] D. Bradley. Throttled by Your ISP. `http://www.sciencetext.com/throttled-by-your-isp.html`, 5 2008. Retrieved 27-Jan-2010.

[CDGS07a] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Detecting HTTP Tunnels with Statistical Mechanisms. In *ICC*, pages 6162–6168. IEEE, 2007.

[CDGS07b] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *Computer Communication Review*, 37(1):5–16, 2007.

[Cis02] Cisco Systems. Evolution of the Firewall Industry. `http://www.cisco.com/univercd/cc/td/doc/product/iaabu/centri4/user/scf4ch3.htm`, 9 2002. Retrieved 4-Feb-2010.

[Cla98] D. Clark. Are ATM, Gigabit Ethernet ready for prime time. *Computer*, 31(5):11–13, May 1998.

[CLR67] I. M. Chakravarti, R. G. Laha, and J. Roy. *Handbook of Methods of Applied Statistics*, volume 1, pages 392–394. John Wiley and Sons, 1967.

[CLW03] M. Claypool, D. LaPoint, and J. Winslow. Network analysis of Counter-strike and Starcraft. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 261–268, April 2003.

[Den87] D. E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[DHS01] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*, pages 394–413. John Wiley & Sons, second edition, 2001.

[DKSL04] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro*, 24(1):52–61, 2004.

[DO01] T. Dunigan and G. Ostrouchov. FLOW CHARACTERIZATION FOR INTRUSION DETECTION. Technical report, Oak Ridge National Laboratory, 7 2001.

[FH98] P. Ferguson and G. Huston. *Quality of Service: Delivering QoS on the Internet and in Corporate Networks*. Wiley & Sons, New York, USA, January 1998.

[Fyo98] Fyodor. Remote OS detection via TCP/IP Stack FingerPrinting. `http://nmap.org/nmap-fingerprinting-article.txt`, 10 1998. Retrieved 2-Feb-2010.

[FZS08] E.P. Freire, A. Ziviani, and R.M. Salles. Detecting VoIP calls hidden in web traffic. *Network and Service Management, IEEE Transactions on*, 5(4):204–214, December 2008.

[GM01] P. Gupta and N. McKeown. Algorithms for packet classification. *Network, IEEE*, 15(2):24–32, Mar/Apr 2001.

[GY08] F. Gont and A. Yourtchenko. On the implementation of TCP urgent data. `http://www.gont.com.ar/talks/IETF73/ietf73-tcpm-urgent-data.ppt`, 11 2008. Retrieved 2-Feb-2010.

[HCL+09] S. Huang, K. Chen, C. Liu, A. Liang, and H. Guan. A statistical-feature-based approach to internet traffic classification using Machine Learning. In *Ultra Modern Telecommunications & Workshops, 2009. ICUMT '09. International Conference on*, pages 1–6, October 2009.

[HP08] Y.-T. Han and H.-S. Park. UDP based P2P game traffic classification with transport layer behaviors. In *Communications, 2008. APCC 2008. 14th Asia-Pacific Conference on*, pages 1–5, Oct. 2008.

[IAN10] IANA. PORT NUMBERS. `http://www.iana.org/assignments/port-numbers`, 2 2010. Retrieved 3-Feb-2010.

[IBM] IBM. Allow TCP push (PSH) flag to be set in last packet. `http://www-01.ibm.com/support/docview.wss?uid=swg1PJ29832`. Retrieved 2-Feb-2010.

[ID06] ITU-D. Teletraffic Engineering Handbook. `http://www.com.dtu.dk/teletraffic/handbook/telenook.pdf`, 7 2006. Retrieved 24-Jan-2010.

[IT04] ITU-T. ITU-T I.371 : Traffic control and congestion control in B-ISDN Section. `http://www.itu.int/rec/T-REC-I.371-200403-I/en`, 3 2004. Retrieved 25-Jan-2010.

[ITU94] ITU. X.200 : Information technology - Open Systems Interconnection - Basic Reference Model: The basic model. http://www.itu.int/rec/T-REC-X.200-199407-I/en, 7 1994. Retrieved 25-Jan-2010.

[JC98] H. Julkunen and C. E. Chow. Enhance Network Security with Dynamic Packet Filter. In *ICCCN*, pages 268–275. IEEE, 1998.

[Kat03] F. Katayama. Hacker accesses 5.6 million credit cards. http://www.cnn.com/2003/TECH/02/17/creditcard.hack/, 2 2003. Retrieved 27-Jan-2010.

[KBB+04] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *Proceedings of the GLOBECOM 2004 Conference*, Dallas, Texas, November 2004. IEEE Computer Society Press.

[l7h09] L7-filter – Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/, 1 2009. Retrieved 4-Feb-2010.

[l7p09] L7-filter – Supported Protocols. http://l7-filter.sourceforge.net/protocols, 1 2009. Retrieved 4-Feb-2010.

[LGW99] A. Leon-Garcia and I. Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures*. McGraw-Hill, New York, NY, December 1999.

[Li07] B. Li. *Detect TCP-Based Applications Using Packet Size Distributions*. PhD thesis, Loughborough University, 8 2007.

[LM07] W. Li and Andrew W. Moore. A Machine Learning Approach for Efficient Traffic Classification. In *MASCOTS*, pages 310–317. IEEE Computer Society, 2007.

[MF01] C. Macian and R. Finthammer. An evaluation of the key design criteria to achieve high update rates in packet classifiers. *Network, IEEE*, 15(6):24–29, Nov/Dec 2001.

[Mic] Microsoft. How the TCP Push Bit Was Changed for Windows NT 3.5. http://support.microsoft.com/kb/123749. Retrieved 2-Feb-2010.

[MIN09] MINTS. Minnesota Internet Traffic Studies. http://www.dtc.umn.edu/mints/home.php, 11 2009. Retrieved 23-Jan-2010.

[Mog89] J Mogul. Simple and Flexible Daragram Access Controls for Unix based Gateways. *DEC WRL*, Res Rep 89/4, 1989.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[MP88] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition, 1988.

[MZ05a] A. Moore and D. Zuev. Discriminators for use in flow-based classification. Technical report, Intel Research and Cambridge, 2005.

[MZ05b] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In Derek L. Eager, Carey L. Williamson, Sem C. Borst, and John C. S. Lui, editors, *SIGMETRICS*, pages 50–60. ACM, 2005.

[Nag84] J. Nagle. Congestion Control in IP/TCP Internetworks, 1 1984. RFC 896.

[Nie98] J. Nielsen. Nielsen's Law of Internet Bandwidth. `http://www.useit.com/alertbox/980405.html`, 4 1998. Retrieved 23-Jan-2010.

[NSV06] A. Nogueira, P. Salvador, and R. Valadas. Detecting Internet Applications using Neural Networks. In *ICNS*, page 95. IEEE Computer Society, 2006.

[Pax99] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[PBL+03] D.J. Parish, K. Bharadia, A. Larkum, I.W. Phillips, and M.A. Oliver. Using packet size distributions to identify real-time networked applications. *Communications, IEE Proceedings-*, 150(4):221–227, Aug. 2003.

[PL06] P. Piyachon and Y. Luo. Efficient memory utilization on network processors for deep packet inspection. In Laxmi N. Bhuyan, Michel Dubois, and Will Eatherton, editors, *ANCS*, pages 71–80. ACM, 2006.

[PM04] J. J. Prichard and L. E. MacDonald. Cyber Terrorism: A Study of the Extent of Coverage in Computer Science Textbooks. *JITE*, 3:279–289, 2004.

[PWC] T. Potter, S. Wehner, and Y. Champoux. NetPacket Module for Perl. `http://search.cpan.org/~yanick/NetPacket-0.41.1/lib/NetPacket.pm`. Retrieved 14-Feb-2010.

[Qui92] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1st edition, October 1992.

[Raj05] S. Raja. Why "Always On" Stateful Inspection and Deep Packet Analysis are Essential to Deliver Non-Stop Protection. Technical report, Top Layer Networks, Inc., 2005.

[Ram98] L. Raman. OSI systems and network management. *Communications Magazine, IEEE*, 36(3):46–53, Mar 1998.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and PDP Research Group, editors, *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1: foundations, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[RN95] S. J. Russell and P. Norving. *Artificial Intelligence: A Modern Approach*, pages 570–571. Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[Roe99] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *LISA*, pages 229–238. USENIX, 1999.

[RP94] J. Reynolds and J. Postel. Assigned Numbers, 10 1994. RFC 1700.

[RSS01] P. Rodriguez, S. Sibal, and O. Spatscheck. TPOT: translucent proxying of TCP. *Computer Communications*, 24(2):249–255, 2001.

[SGO+09] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan. Evaluating GPUs for network packet signature matching. In *ISPASS*, pages 175–184. IEEE, 2009.

[SHHP00] O. Spatscheck, J. S. Hansen, J. H. Hartman, and L. L. Peterson. Optimizing TCP forwarder performance. *IEEE/ACM Trans. Netw.*, 8(2):146–157, 2000.

[SM07] K. Scarfone and P. Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. National Institute of Standards and Technology, 2 2007. Special Publication 800-94.

[Sou] Sourcefire. Snort – the de facto standard for intrusion detection and prevention. `http://www.snort.org`. Retrieved 28-Jan-2010.

[SSB05] D. Steinkrau, P. Y. Simard, and I. Buck. Using GPUs for Machine Learning Algorithms. In *ICDAR*, pages 1115–1119. IEEE Computer Society, 2005.

[SSM07] M. Shevertalov, E. Stehle, and S. Mancoridis. A genetic algorithm for solving the binning problem in networked applications detection. In *IEEE Congress on Evolutionary Computation*, pages 713–720. IEEE, 2007.

[Ste97] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, 1 1997. RFC 2001.

[Tan03] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4 edition, 3 2003.

[TC97] K. M. C. Tan and B. R. Collie. Detection and Classification of TCP/IP Network Services. In *ACSAC*, pages 99–107. IEEE Computer Society, 1997.

[tcp09] tcpdump workers. tcpdump/libpcap. `http://www.tcpdump.org/`, 1 2009. Retrieved 13-Feb-2010.

[TTNC02] C. Trivedi, H. J. Trussel, A. Nilsson, and M-Y. Chow. Implicit Traffic Classification for Service Differentiation. Technical report, ITC Specialist Seminar, Wurzburg, Germany, 7 2002.

[wek] Weka 3: Data Mining Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`. Retrieved 15-Feb-2010.

[WF05a] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*, page 97. Morgan Kaufmann, San Fransisco, CA, USA, 2nd edition, 2005.

[WF05b] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*, pages 192–196. Morgan Kaufmann, San Fransisco, CA, USA, 2nd edition, 2005.

[wir] Wireshark. `http://www.wireshark.org/`. Retrieved 17-Apr-2010.

[WP10a] X. Wang and D. J. Parish. Optimised Multi-stage TCP Traffic Classifier Based on Packet Size Distributions. In *2010 Third International Conference on Communication Theory, Reliability, and Quality of Service*, pages 98–103, June 2010.

[WP10b] X. Wang and D. J. Parish. Optimised TCP Traffic Classification with Multiple Statistical Algorithms. In *Information Networking and Automation (ICINA), 2010 International Conference on*, pages 261–265, October 2010.

[WY08] Y. Wang and S.-Z. Yu. Move Statistics-Based Traffic Classifiers Online. In *CSSE (4)*, pages 721–725. IEEE Computer Society, 2008.

[Zho08] X. Zhou. A P2P Traffic Classification Method Based on SVM. In *ISCSCT (2)*, pages 53–57. IEEE Computer Society, 2008.

[ZLZ04] X. Zhang, C. Li, and W. Zheng. Intrusion Prevention System Design. In *CIT*, pages 386–390. IEEE Computer Society, 2004.

APPENDIX A

Classification Recall Rates for Different Detection Windows Using Single Classifier

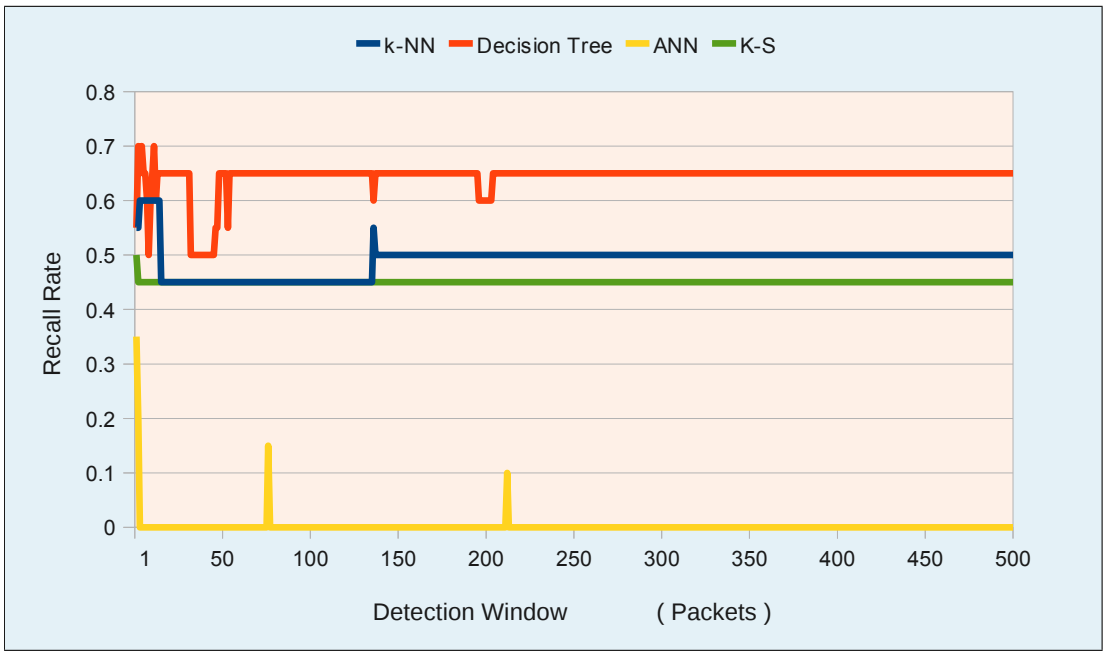

Figure A.1: Classification Recall Rates for FTP-DATA vs Detection Windows Using Single Classifier

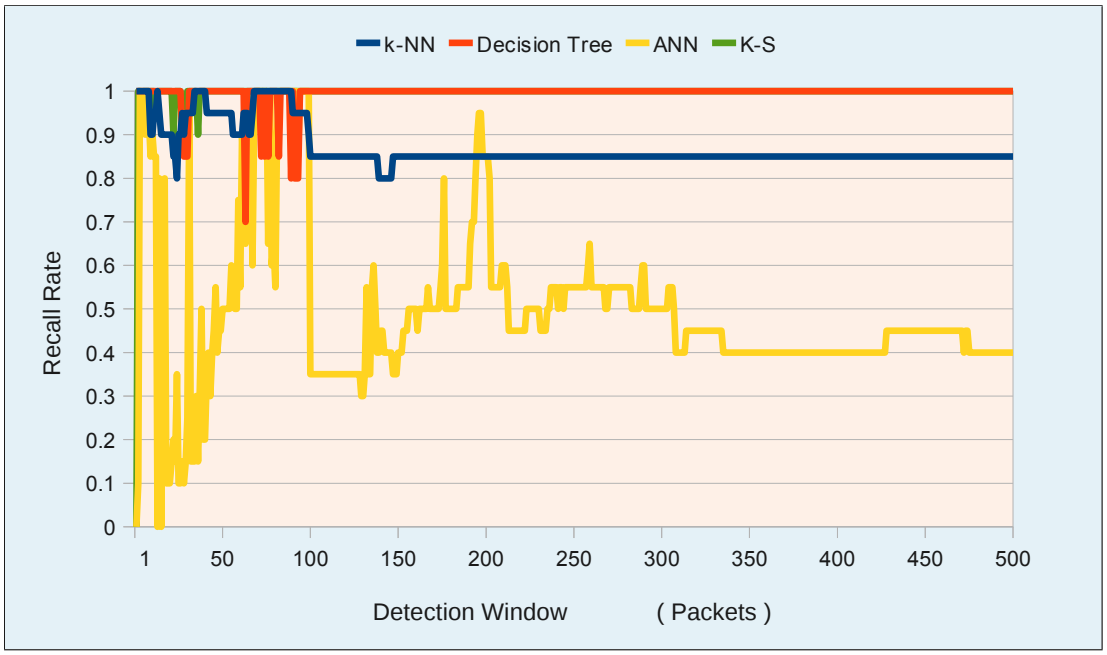

Figure A.2: Classification Recall Rates for FTP vs Detection Windows Using Single Classifier

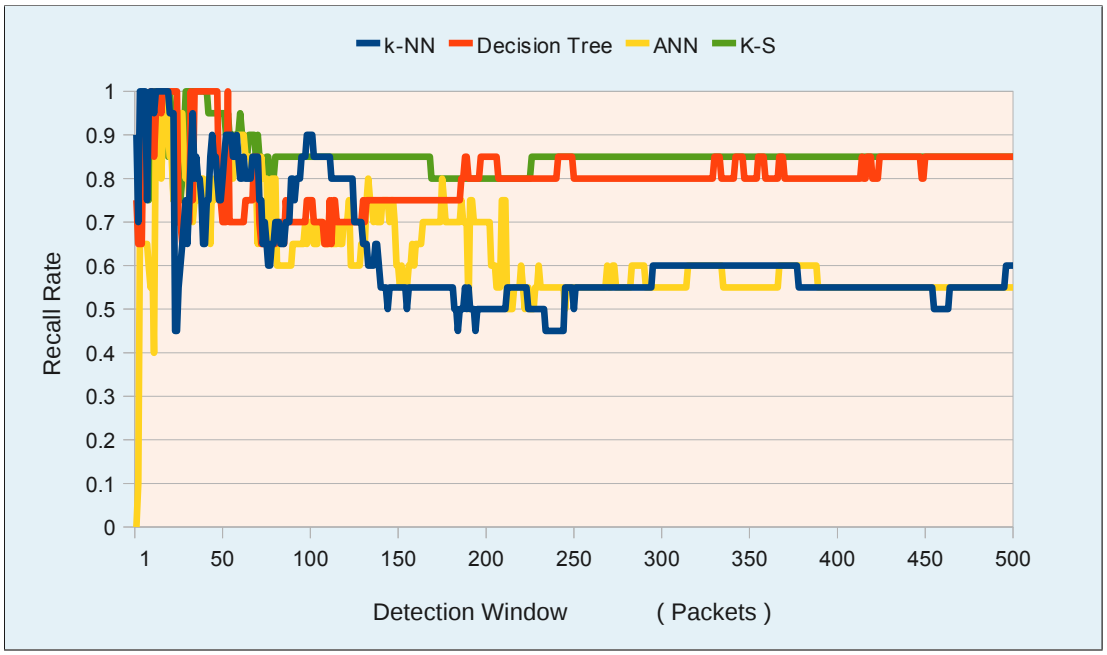

Figure A.3: Classification Recall Rates for IMAPS vs Detection Windows Using Single Classifier

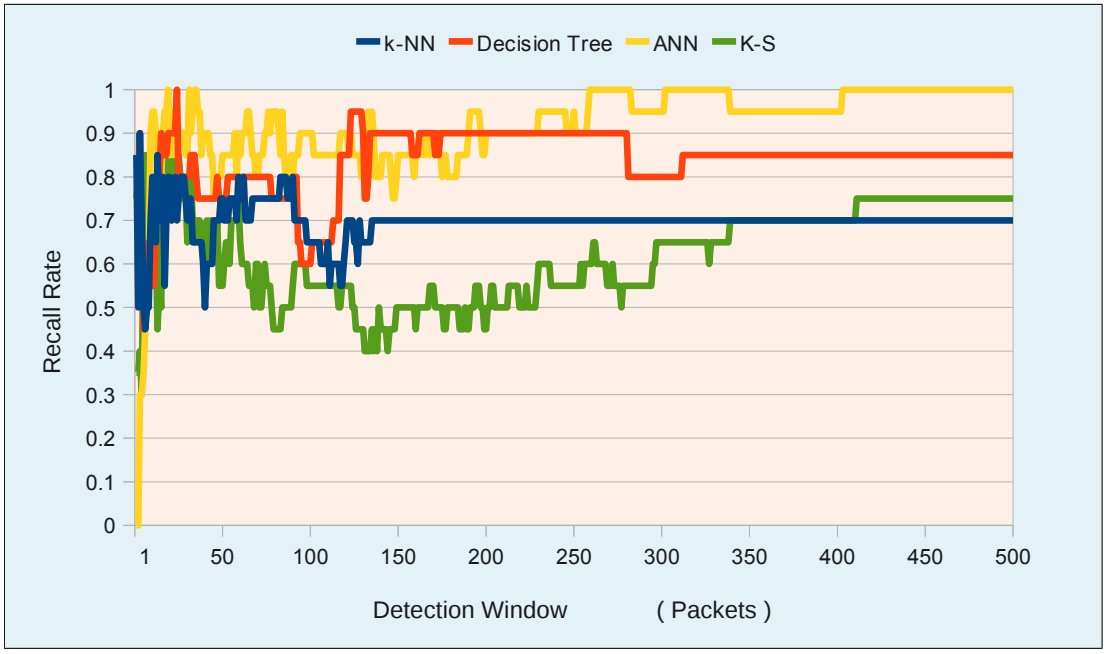

Figure A.4: Classification Recall Rates for IRC vs Detection Windows Using Single Classifier

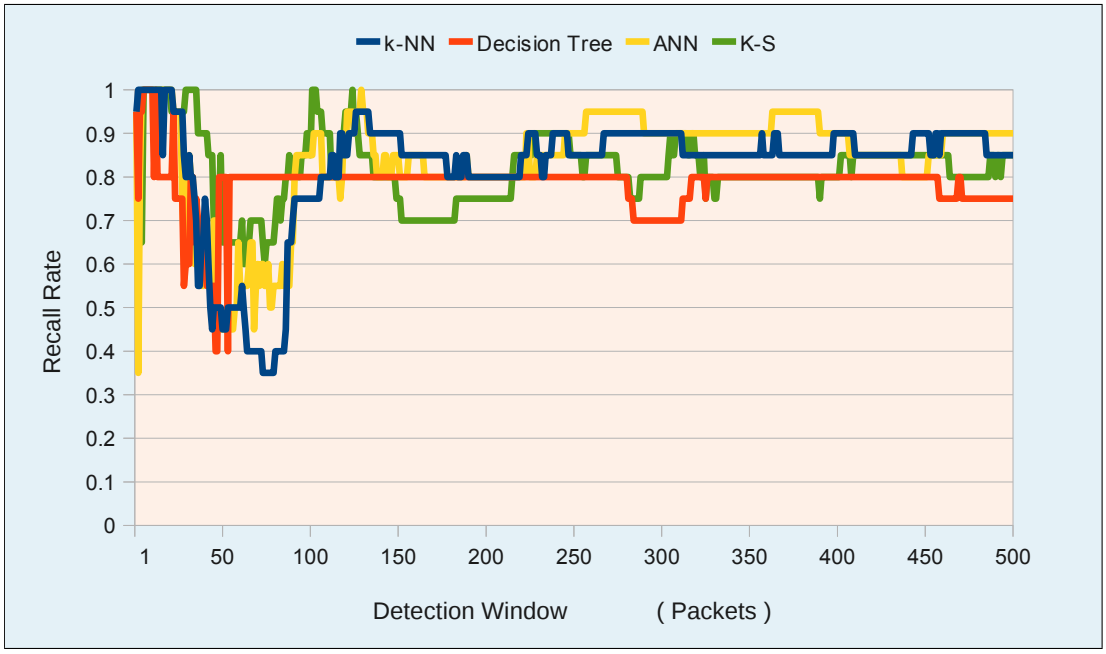

Figure A.5: Classification Recall Rates for MS-RDP vs Detection Windows Using Single Classifier

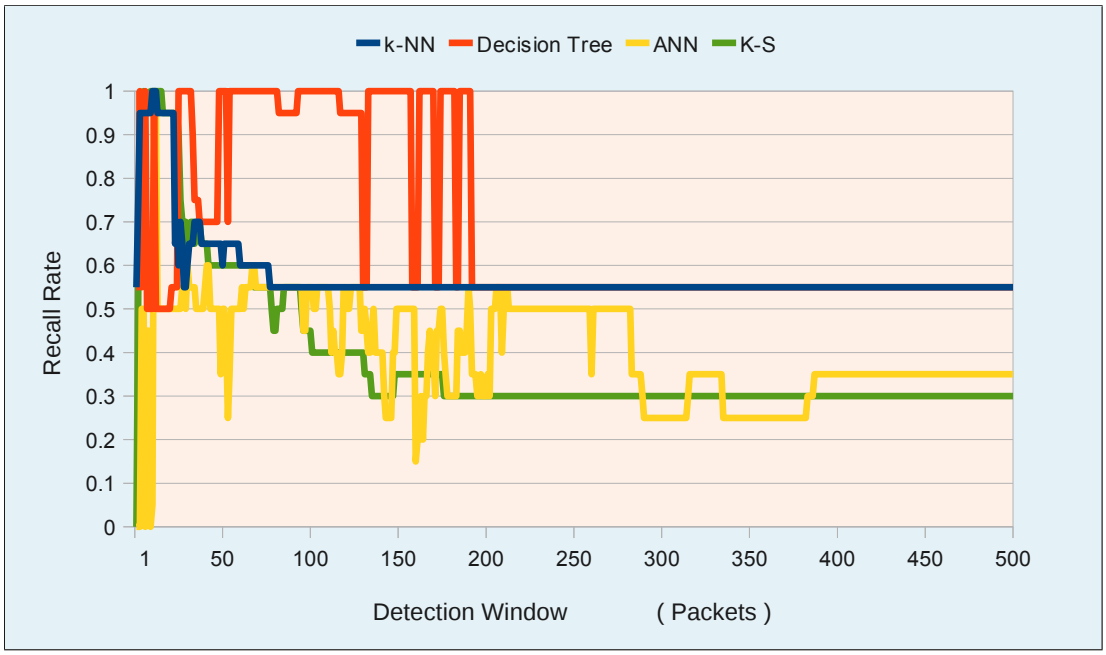

Figure A.6: Classification Recall Rates for POP3 vs Detection Windows Using Single Classifier

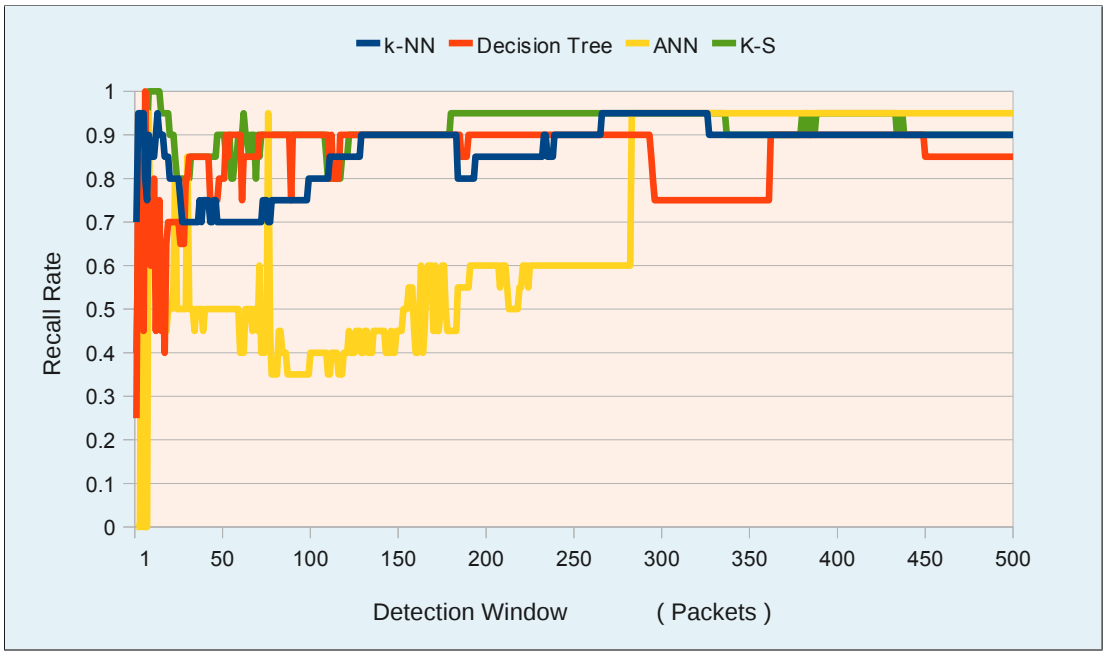

Figure A.7: Classification Recall Rates for RTSP vs Detection Windows Using Single Classifier

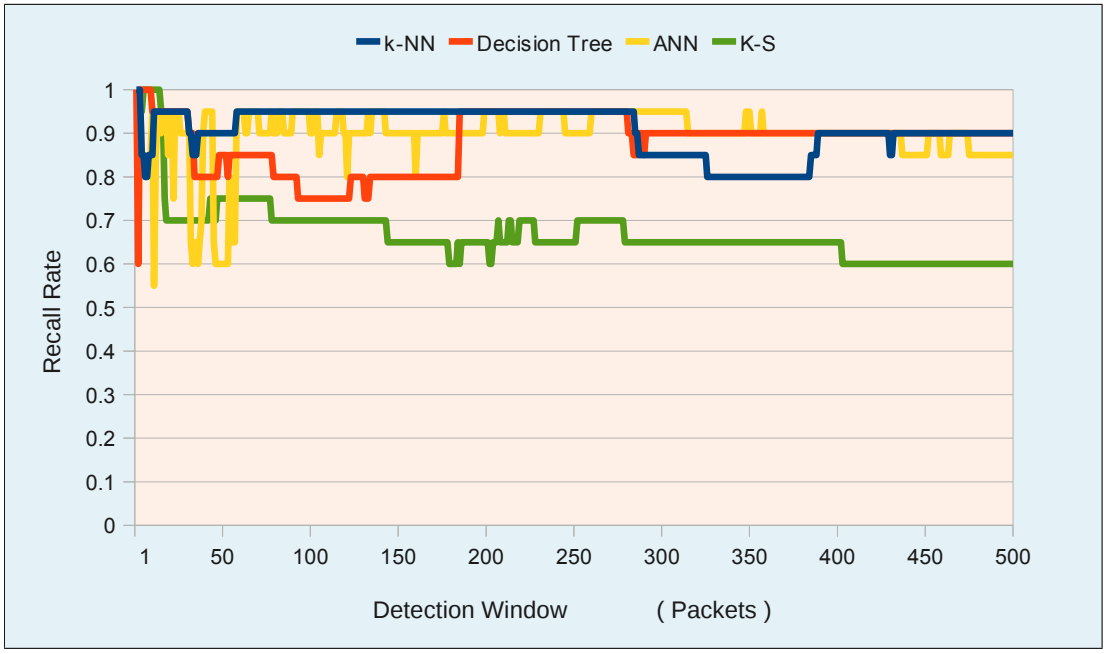

Figure A.8: Classification Recall Rates for SMTP vs Detection Windows Using Single Classifier

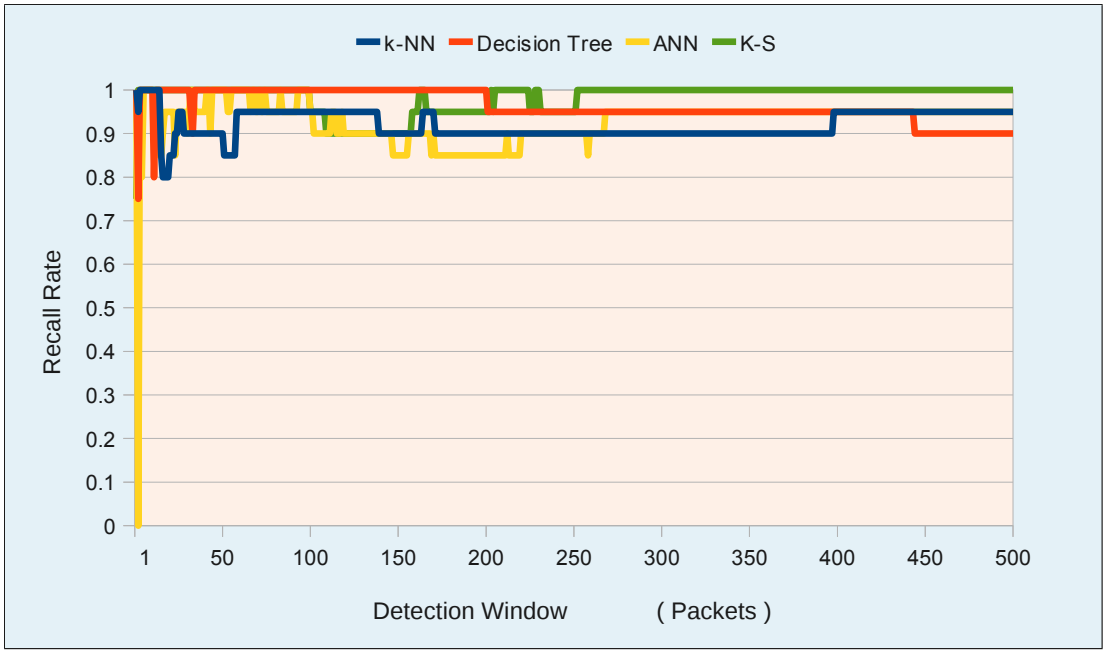

Figure A.9: Classification Recall Rates for SSH vs Detection Windows Using Single Classifier

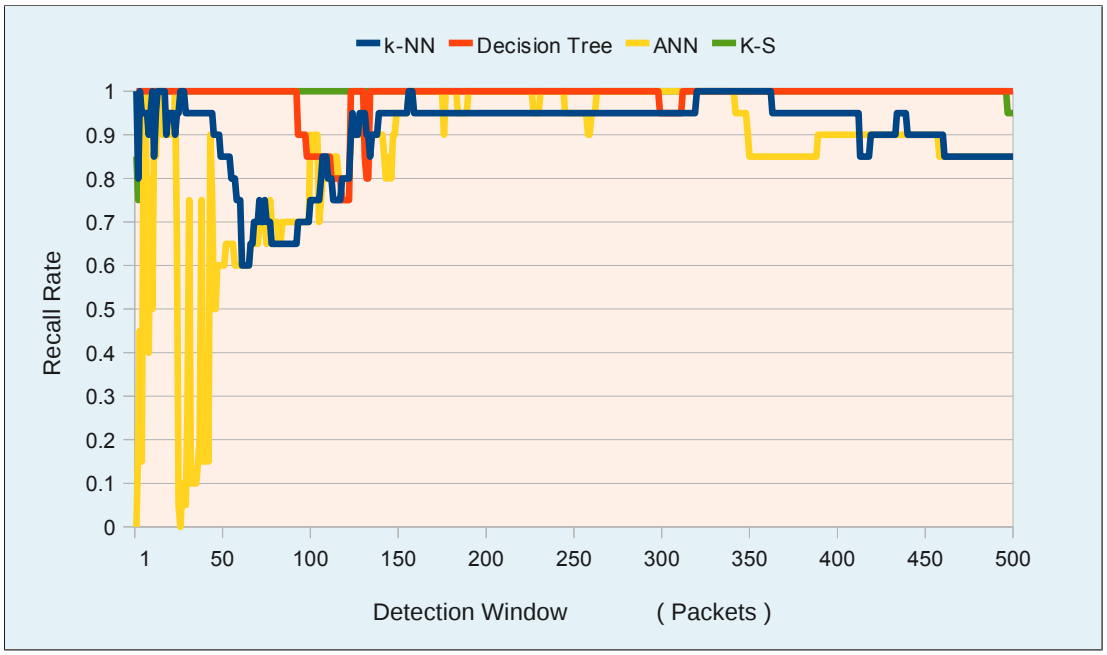

Figure A.10: Classification Recall Rates for Telnet vs Detection Windows Using Single Classifier

APPENDIX B

Classification Precision for Different Detection Windows Using Single Classifier

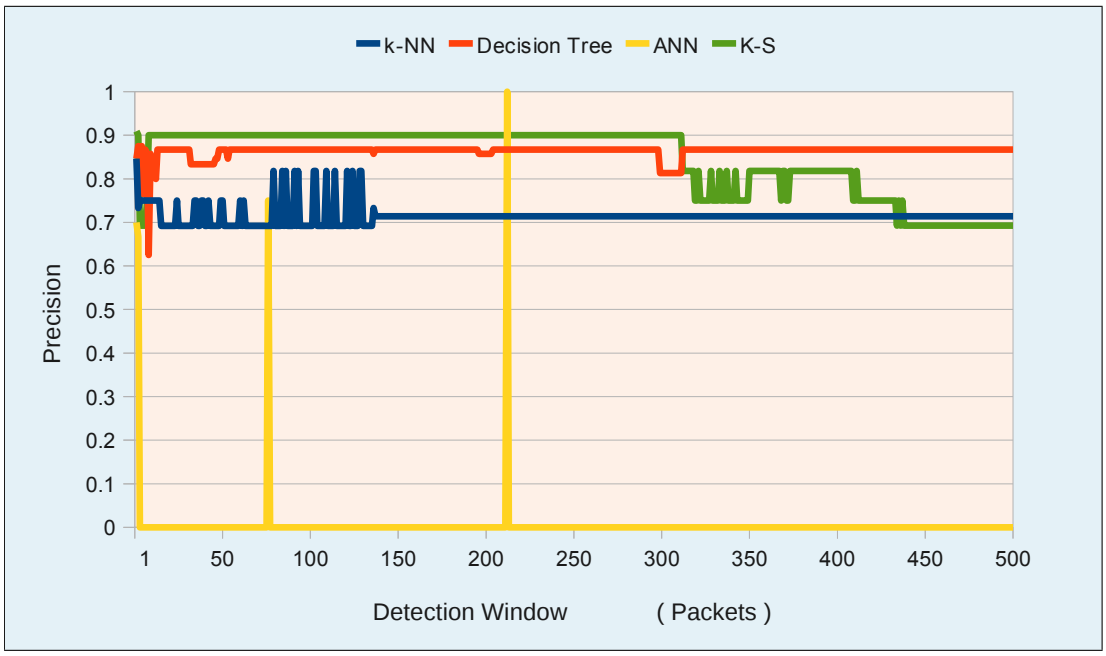

Figure B.1: Classification Precisions for FTP-DATA vs Detection Windows Using Single Classifier

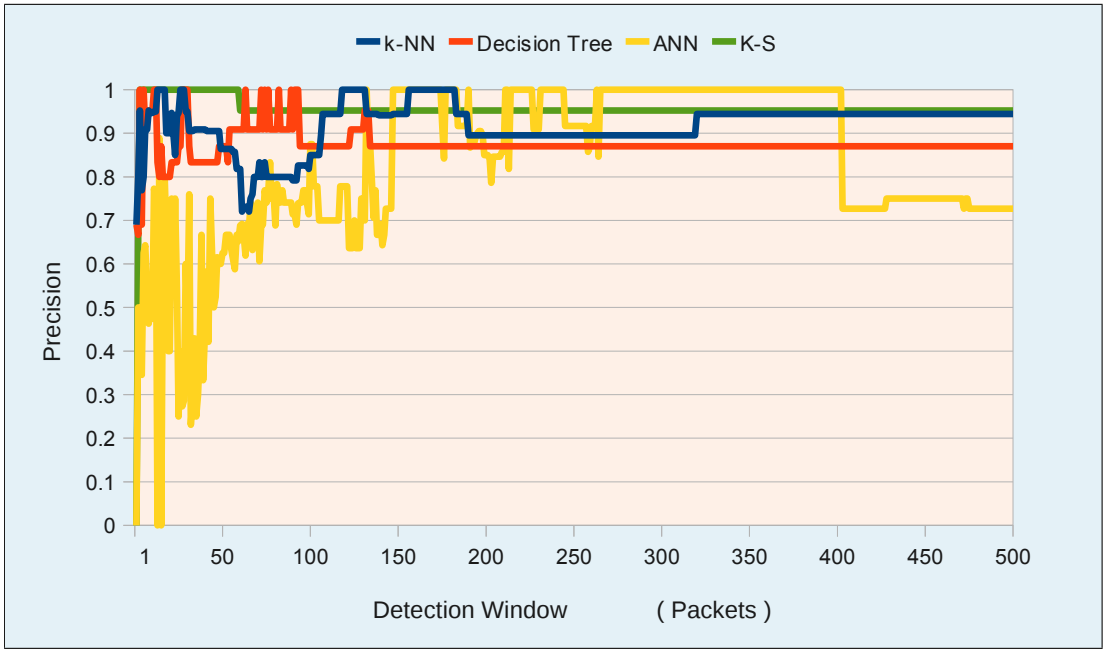

Figure B.2: Classification Precisions for FTP vs Detection Windows Using Single Classifier

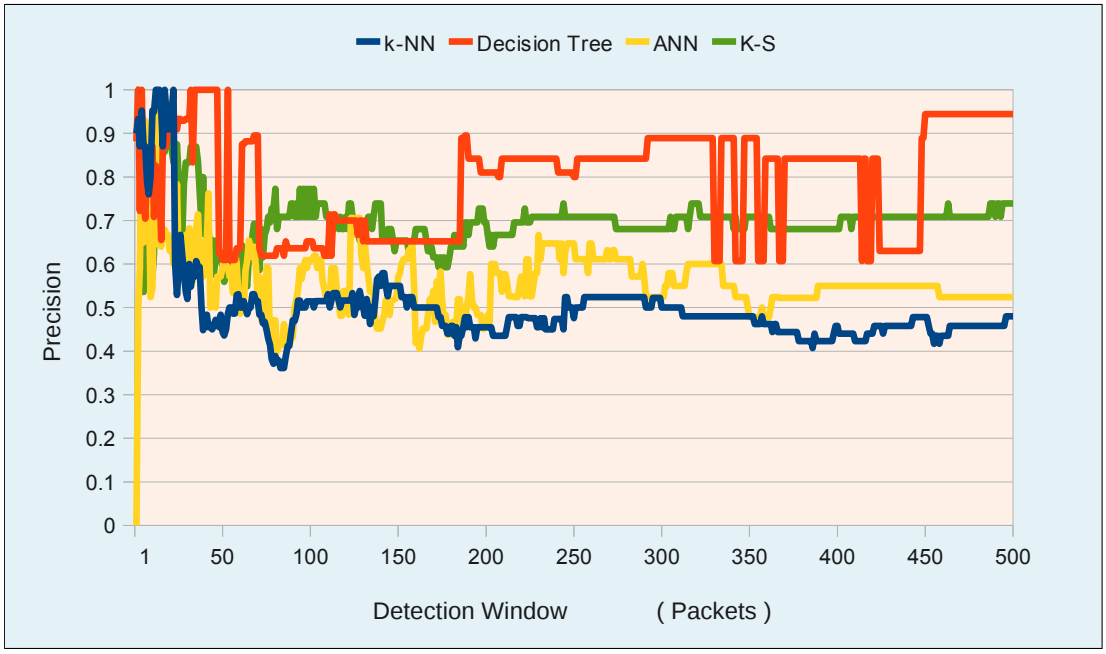

Figure B.3: Classification Precisions for IMAPS vs Detection Windows Using Single Classifier

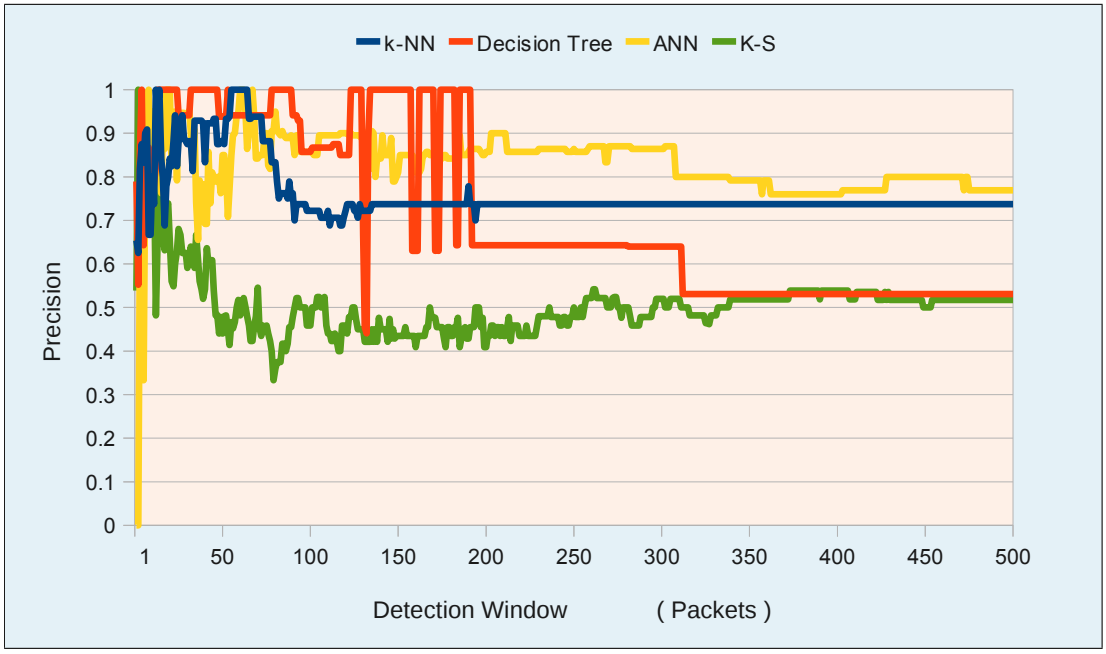

Figure B.4: Classification Precisions for IRC vs Detection Windows Using Single Classifier

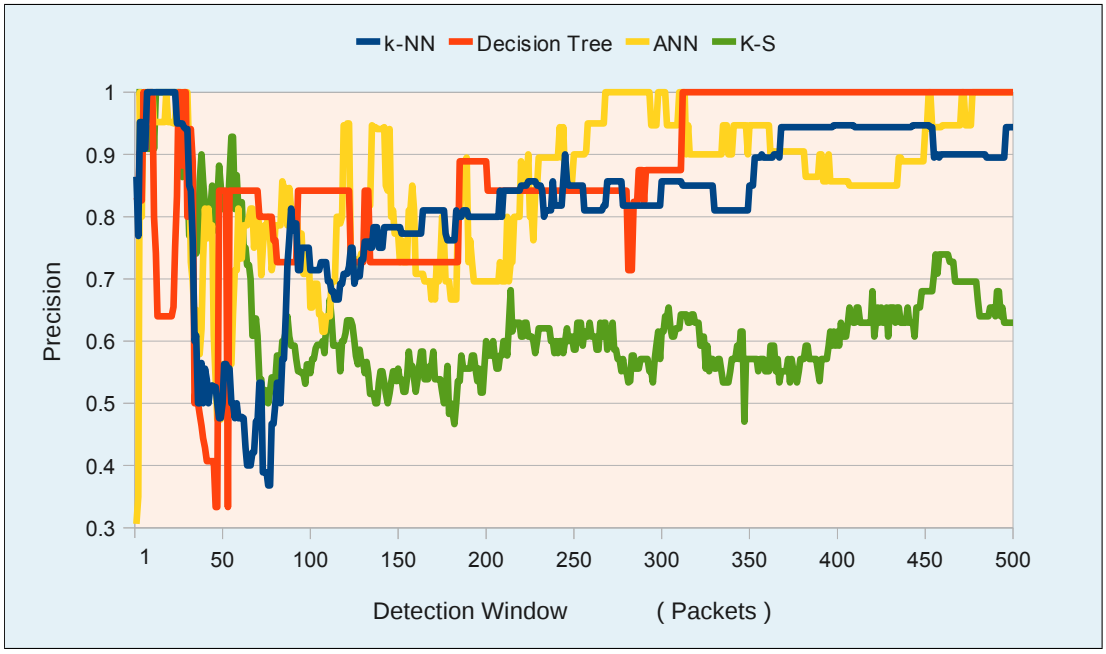

Figure B.5: Classification Precisions for MS-RDP vs Detection Windows Using Single Classifier

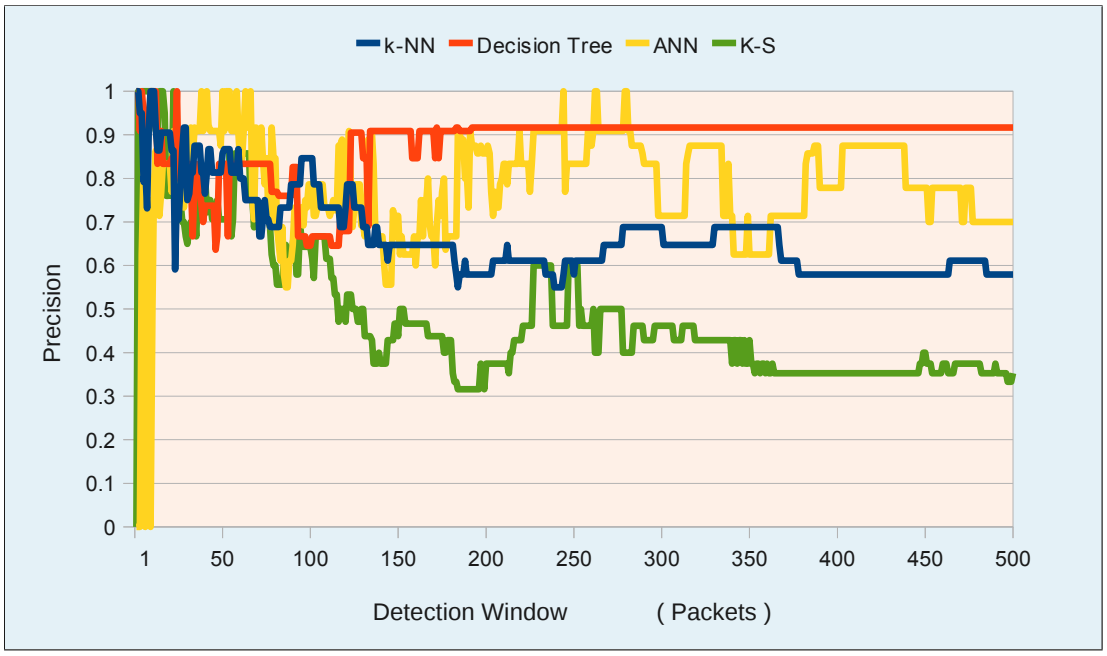

Figure B.6: Classification Precisions for POP3 vs Detection Windows Using Single Classifier

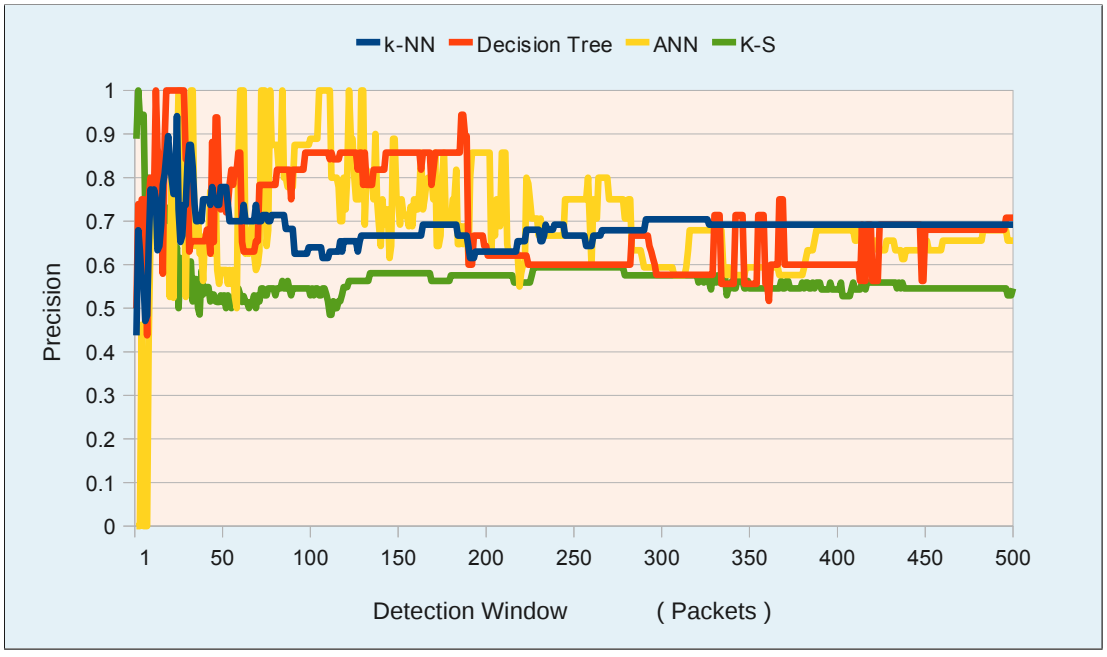

Figure B.7: Classification Precisions for RTSP vs Detection Windows Using Single Classifier

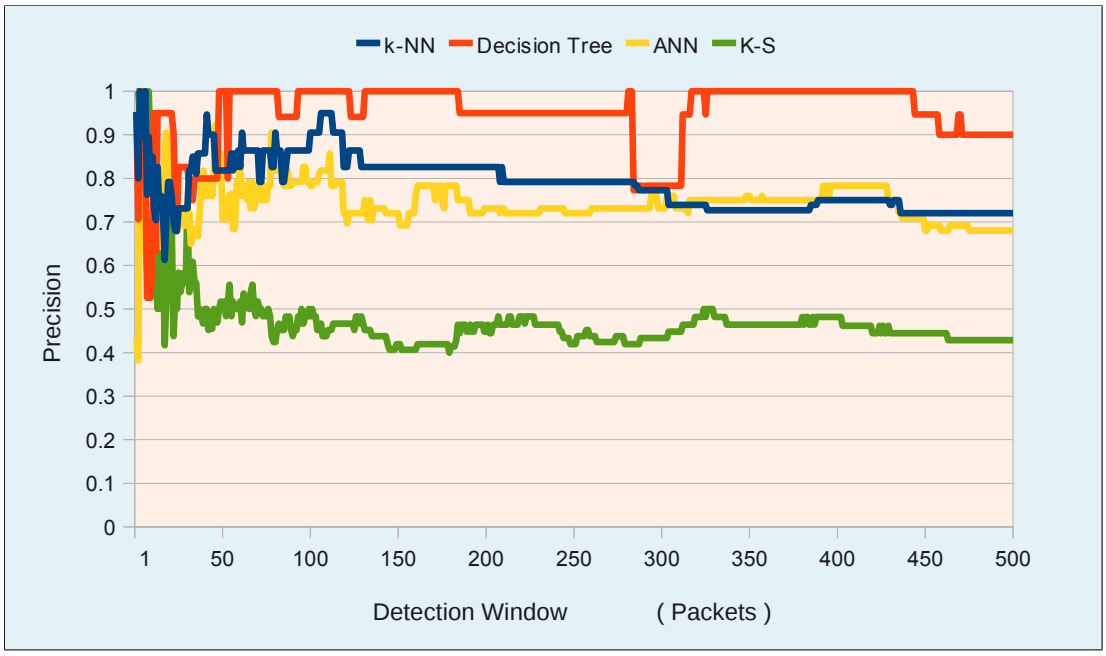

Figure B.8: Classification Precisions for SMTP vs Detection Windows Using Single Classifier

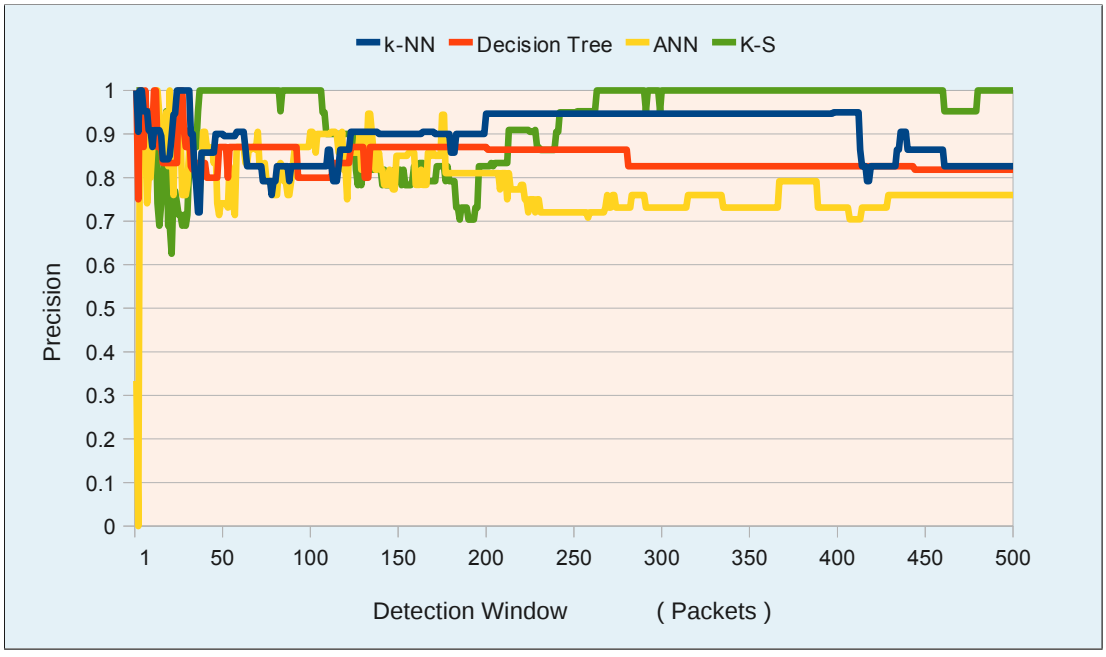

Figure B.9: Classification Precisions for SSH vs Detection Windows Using Single Classifier

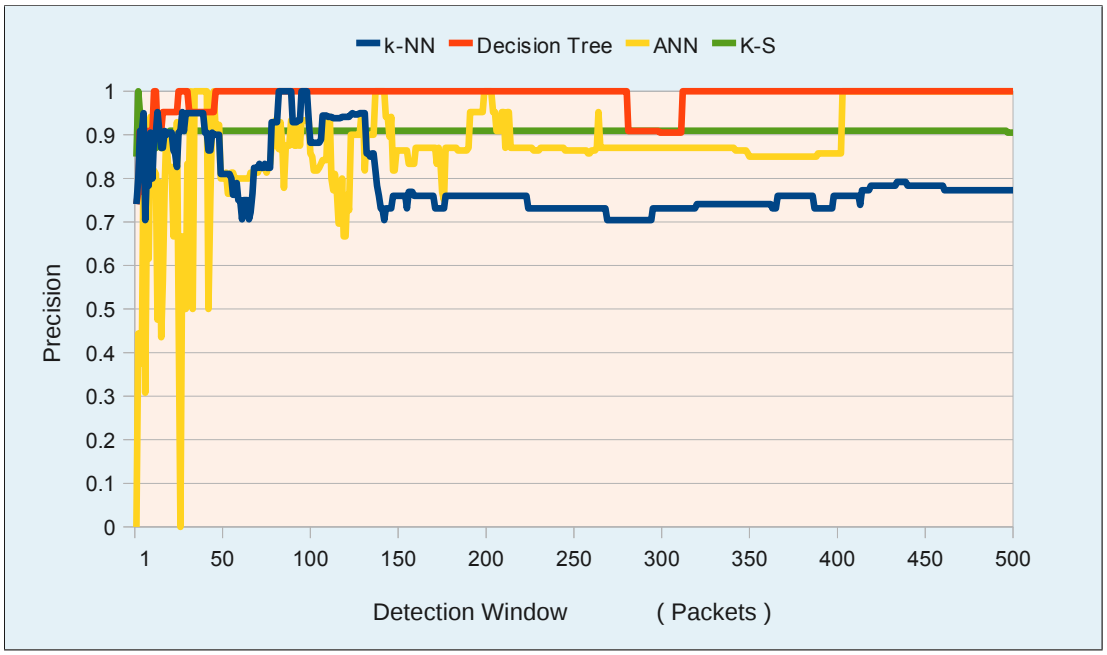

Figure B.10: Classification Precisions for Telnet vs Detection Windows Using Single Classifier

APPENDIX C

Key Source Code

C.1 build_connection.pl

```
#! /usr/bin/perl -w
# Reconstruct packets into flows
# Written by Xiaoming Wang
# ARGV[0]=input pcap file
# ARGV[1]=output file
# ARGV[2]=max_conn (max connection should be printed)
# ARGV[3]=print class (which service should be print based on port)

use Cwd;
use IO::Handle;
use Net::Pcap;
use NetPacket::Ethernet;
use NetPacket::IP;
use NetPacket::IP qw(IP_PROTO_TCP);
use NetPacket::TCP;
use strict;

my $MAX_IDLE_TIMEOUT=600;
my $MAX_TIMEOUT=600;
```

```
my $err;
my ($user_data, $header, $packet);
my $ether_data;
my $ip;
my $tcp;
my $time_sec;
my $time_usec;
my $src_ip;
my $src_port;
my $dest_ip;
my $dest_port;
my $ip_len;
my $ip_hlen;
my $tcp_hlen;
my $payload_len;
my $flag_U;
my $flag_A;
my $flag_P;
my $flag_R;
my $flag_S;
my $flag_F;
my %connection_filename;
my %connection_inittime_sec;
my %connection_inittime_usec;
my %connection_bytes;
my %connection_lasttime_sec;
my %connection_lasttime_usec;
my %connection_pkts;
my %connection_class;
my $packetNo=0;
my $connectionNo=0;
my $inputfile=$ARGV[0];
my $outputfile = $ARGV[1];
my $max_connection=$ARGV[2];
my $print_class=$ARGV[3];
my $datapath = './'.$inputfile.'_tmp/';

# Make working dir
mkdir('./'.$inputfile.'_tmp',0777) || die "Cannot make data dir";

# Create packet capture object on device
my $object;
$object = Net::Pcap::open_offline($inputfile, \$err);
unless (defined $object) {
    die 'Unable to create packet capture on file ', $inputfile, ' - '
        , $err;
}
```

```
# Set up connection summary file;
open SUMMARY, ">>$outputfile".'.csv';

# Set current dir to tmp working path
chdir $datapath || die "Cannot change dir";

# Set callback function and initiate packet capture loop
Net::Pcap::loop($object, -1, \&packets_processing, '');

# Close Object
Net::Pcap::close($object);
&clean_end();

# Callback function
sub packets_processing {
    $packetNo++;
    ($user_data, $header, $packet) = @_;
    # Strip ethernet encapsulation of captured packet
    $ether_data = NetPacket::Ethernet::strip($packet);
    # Decode contents of TCP/IP packet contained within captured
        ethernet packet
    $ip = NetPacket::IP->decode($ether_data);
    if($ip->{proto} == IP_PROTO_TCP){
        $tcp = NetPacket::TCP->decode($ip->{'data'});
        $time_sec=$header->{'tv_sec'};
        $time_usec=$header->{'tv_usec'};
        $src_ip=$ip->{'src_ip'};
        $src_port=$tcp->{'src_port'};
        $dest_ip=$ip->{'dest_ip'};
        $dest_port=$tcp->{'dest_port'};
        $ip_len=$ip->{'len'};
        $ip_hlen=$ip->{'hlen'}*4;
        $tcp_hlen=$tcp->{'hlen'}*4;
        $payload_len=$ip_len-$ip_hlen-$tcp_hlen;
        $flag_U=($tcp->{'flags'} & 32) >>5;
        $flag_A=($tcp->{'flags'} & 16) >>4;
        $flag_P=($tcp->{'flags'} & 8) >>3;
        $flag_R=($tcp->{'flags'} & 4) >>2;
        $flag_S=($tcp->{'flags'} & 2) >>1;
        $flag_F=$tcp->{'flags'} & 1;
        &add_packet();
    }
}

sub add_packet{
    my $connection=$src_ip.','.$src_port.','.$dest_ip.','.$dest_port;
```

```
my $connection_r=$dest_ip.','.$dest_port.','.$src_ip.','.
    $src_port;
my $relativetime;
# New connection
if((!(exists $connection_filename{$connection}))&&(!(exists
    $connection_filename{$connection_r}))&&$flag_S==1){
     $connection_filename{$connection}=$connection.'@'.$time_sec.'
         .'.$time_usec.'.csv';
     $connection_inittime_sec{$connection}=$time_sec;
     $connection_inittime_usec{$connection}=$time_usec;
     $connection_bytes{$connection}=0;
     $connection_lasttime_sec{$connection}=$time_sec;
     $connection_lasttime_usec{$connection}=$time_usec;
     $connection_pkts{$connection}=0;
     $connection_class{$connection}=&port_class($dest_port,
         $src_port);
}
# Exsiting connection same dir
if(exists $connection_filename{$connection}){
    # finish connection by Fin or Reset
    if($flag_F==1||$flag_R==1){
        $connection_lasttime_sec{$connection}=$time_sec;
        $connection_lasttime_usec{$connection}=$time_usec;
        &del_connection($connection);
    }
    # Normal pkt
    if($payload_len!=0 && $flag_F==0 && $flag_R==0){
        $relativetime=$time_sec-$connection_inittime_sec{
            $connection}+($time_usec-$connection_inittime_usec{
            $connection})*0.000001;
        $connection_bytes{$connection}+=$payload_len;
        $connection_lasttime_sec{$connection}=$time_sec;
        $connection_lasttime_usec{$connection}=$time_usec;
        $connection_pkts{$connection}++;
        # Write to tmp conn. file
        open TOFILE, ">>$connection_filename{$connection}";
        printf TOFILE ("%.6f",$relativetime);
        print TOFILE ",C-S,$payload_len\n";
        close TOFILE;
    }
}
# Exsiting connection reverse dir
if(exists $connection_filename{$connection_r}){
    if($flag_F==1||$flag_R==1){
        $connection_lasttime_sec{$connection_r}=$time_sec;
        $connection_lasttime_usec{$connection_r}=$time_usec;
        &del_connection($connection_r);
```

```
        }
        # Normal pkt
        if($payload_len!=0 && $flag_F==0 && $flag_R==0){
            $relativetime=$time_sec-$connection_inittime_sec{
                $connection_r}+($time_usec-$connection_inittime_usec{
                $connection_r})*0.000001;
            $connection_bytes{$connection_r}+=$payload_len;
            $connection_lasttime_sec{$connection_r}=$time_sec;
            $connection_lasttime_usec{$connection_r}=$time_usec;
            $connection_pkts{$connection_r}++;
            # Write to tmp conn. file
            open TOFILE, ">>$connection_filename{$connection_r}";
            printf TOFILE ("%.6f",$relativetime);
            print TOFILE ",S-C,$payload_len\n";
            close TOFILE;
        }
    }
    &check_timeout;
}

# Return the classes based on the dst port
sub port_class{
        return "ftp-data"       if $_[0]==20 || $_[1]==20;
        return "ftp"            if $_[0]==21;
        return "ssh"            if $_[0]==22;
        return "telnet"         if $_[0]==23;
        return "smtp"           if $_[0]==25;
        return "http"           if $_[0]==80;
        return "pop3"           if $_[0]==110;
        return "imap"           if $_[0]==143;
        return "imaps"          if $_[0]==993;
        return "rtsp"           if $_[0]==554;
        return "ms-rdp"         if $_[0]==3389;
        return "irc"            if $_[0]==6667;
        return "others";
}

# Check time out;
sub check_timeout{
   my $key;
   my $idletime;
   my $duration;
   foreach $key (keys %connection_filename){
        $idletime=($time_sec-$connection_lasttime_sec{$key})+(
            $time_usec-$connection_lasttime_usec{$key})*0.000001;
        $duration=($time_sec-$connection_inittime_sec{$key})+(
            $time_usec-$connection_inittime_usec{$key})*0.000001;
```

```
        if($idletime>$MAX_IDLE_TIMEOUT||$duration>$MAX_TIMEOUT){
                &del_connection($key);
        }
    }
}

# Del conn. from temp & write to file
sub del_connection{
    # Compute whole duration for conn.
    my $duration=($connection_lasttime_sec{$_[0]}-
        $connection_inittime_sec{$_[0]})+($connection_lasttime_usec{$_
        [0]}-$connection_inittime_usec{$_[0]})*0.000001;
    # Print valid connections only
    if($connection_bytes{$_[0]}>0 && $connection_class{$_[0]} eq
        $print_class){
        # Restrict print no.
        if($connectionNo>$max_connection){
                &clean_end();
                exit 0;
        }
        # Print summary line
        print SUMMARY '@,'."$connection_inittime_sec{$_[0]},
            $connection_inittime_usec{$_[0]},$_[0],$connection_bytes{
            $_[0]},$connection_pkts{$_[0]},$duration,$connection_class
            {$_[0]}\n";
        # Read from temp file and write to summary file
        open SINGLE, "<$connection_filename{$_[0]}";
        while(<SINGLE>){
                print SUMMARY $_;
        }
        $connectionNo++;
    }
    close SINGLE;
    system ("rm -f $connection_filename{$_[0]}");
    delete $connection_filename{$_[0]};
    delete $connection_inittime_sec{$_[0]};
    delete $connection_inittime_usec{$_[0]};
    delete $connection_bytes{$_[0]};
    delete $connection_lasttime_sec{$_[0]};
    delete $connection_lasttime_usec{$_[0]};
    delete $connection_pkts{$_[0]};
    delete $connection_class{$_[0]};
}

# Clean temp
sub clean_end{
    close SUMMARY;
```

```
    chdir "../";
    system ("rm -rf $inputfile.'_tmp');
    print STDOUT "\n",getcwd(),"/",$inputfile,"\n";
    print STDOUT $packetNo," Packets Processed.\n";
    print STDOUT $connectionNo," Connections Processed.\n";
}
```

C.2 matrix.pl

```
#! /usr/bin/perl -w
# Build feature matrix
# Written by Xiaoming Wang
# ARGV[0]=input filename
# ARGV[1]=output filename
# ARGV[2]=format (csv or arff)
# ARGV[3..]=periods (time)

use IO::Handle;
use Statistics::Basic qw(:all);
use strict;

my $inputfilename = shift @ARGV;
my $outputfilename = shift @ARGV;
my $format = shift @ARGV;
my @periods = @ARGV;

# Set for cul. periods
for (my $i=1;$i<=$#periods;$i++){
        $periods[$i]=$periods[$i]+$periods[$i-1];
}
open OUTPUT, ">$outputfilename";
open SUMMARY, "<$inputfilename";

# Print format header
if($format eq "arff"){
        print OUTPUT '@relation '.$outputfilename."\n\n";
        for (my $i=0;$i<=$#periods;$i++){
                print OUTPUT '@attribute '."PS_MAX_$i"." numeric\n";
                print OUTPUT '@attribute '."PS_MIN_$i"." numeric\n";
                print OUTPUT '@attribute '."PS_AVE_$i"." numeric\n";
                print OUTPUT '@attribute '."PS_VAR_$i"." numeric\n";
                print OUTPUT '@attribute '."IT_MAX_$i"." numeric\n";
                print OUTPUT '@attribute '."IT_MIN_$i"." numeric\n";
                print OUTPUT '@attribute '."IT_AVE_$i"." numeric\n";
                print OUTPUT '@attribute '."IT_VAR_$i"." numeric\n";
```

```
                print OUTPUT '@attribute '."PS_MAX_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_MIN_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_AVE_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_VAR_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_MAX_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_MIN_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_AVE_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_VAR_CS_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_MAX_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_MIN_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_AVE_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PS_VAR_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_MAX_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_MIN_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_AVE_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."IT_VAR_SC_$i"." numeric\n
                    ";
                print OUTPUT '@attribute '."PC_Ratio_$i"." numeric\n"
                    ;
                print OUTPUT '@attribute '."BC_Ratio_$i"." numeric\n"
                    ;
        }
        print OUTPUT '@attribute CLASS {ftp,ftp-data,http,imaps,irc,
            ms-rdp,pop3,rtsp,smtp,ssh,telnet}';
        print OUTPUT "\n\n";
        print OUTPUT '@data';
        print OUTPUT "\n";
}
elsif($format eq "csv"){
        for (my $i=0;$i<=$#periods;$i++){
                print OUTPUT "PS_MAX_$i,PS_MIN_$i,PS_AVE_$i,PS_VAR_$i
                    ,";
```

```
                print OUTPUT "IT_MAX_$i,IT_MIN_$i,IT_AVE_$i,IT_VAR_$i
                    ,";
                print OUTPUT "PS_MAX_CS_$i,PS_MIN_CS_$i,PS_AVE_CS_$i,
                    PS_VAR_CS_$i,";
                print OUTPUT "IT_MAX_CS_$i,IT_MIN_CS_$i,IT_AVE_CS_$i,
                    IT_VAR_CS_$i,";
                print OUTPUT "PS_MAX_SC_$i,PS_MIN_SC_$i,PS_AVE_SC_$i,
                    PS_VAR_SC_$i,";
                print OUTPUT "IT_MAX_SC_$i,IT_MIN_SC_$i,IT_AVE_SC_$i,
                    IT_VAR_SC_$i,";
                print OUTPUT "PC_Ratio_$i,BC_Ratio_$i,";
        }
        print OUTPUT "CLASS";
        print OUTPUT "\n";
}

# Main loop
my $summaryLine;
my @summaryLine;
while(defined($summaryLine=<SUMMARY>)){
        chomp($summaryLine);
        @summaryLine=split /,/, $summaryLine;
        if($summaryLine[0] eq '@'){
                &write_connection($summaryLine[8],$summaryLine[10]);
        }
}
close SUMMARY;
close OUTPUT;

# Function for each conn. (packetNo,class)
sub write_connection{
        my $packetLine;
        my @packetLine;
        my @currentPeriods=@periods;
        my $periodsNo=$#periods+1;
        my $calculatedPeriods=0;
        my $it_current;
        my $last_time=0;
        my $last_CS_time=0;
        my $last_SC_time=0;
        my @ps_both=();
        my @ps_CS=();
        my @ps_SC=();
        my @it_both=();
        my @it_CS=();
        my @it_SC=();
        my @tempPrint=();
```

```
my $pc_both;
my $bc_both;
my $pc_CS;
my $bc_CS;
my $pc_SC;
my $bc_SC;
for(my $i=0;$i<$_[0];$i++){
        $packetLine=<SUMMARY>;
        chomp($packetLine);
        @packetLine=split /,/, $packetLine;

        # Print period stat & reset array
        if($packetLine[0]>=$currentPeriods[0]){
                @tempPrint=&states_ps(@ps_both);
                $pc_both=shift @tempPrint;
                $bc_both=shift @tempPrint;
                printf OUTPUT ("%d,%d,%.6f,%.6f,",@tempPrint)
                    ;
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &
                    states_it(@it_both));
                @tempPrint=&states_ps(@ps_CS);
                $pc_CS=shift @tempPrint;
                $bc_CS=shift @tempPrint;
                printf OUTPUT ("%d,%d,%.6f,%.6f,", @tempPrint
                    );
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &
                    states_it(@it_CS));
                @tempPrint=&states_ps(@ps_SC);
                $pc_SC=shift @tempPrint;
                $bc_SC=shift @tempPrint;
                printf OUTPUT ("%d,%d,%.6f,%.6f,", @tempPrint
                    );
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &
                    states_it(@it_SC));
                printf OUTPUT ("%.6f,%.6f,", ($pc_CS+1)/(
                    $pc_SC+1),($bc_CS+1)/($bc_SC+1));
                $calculatedPeriods++;
                @ps_both=();
                @ps_CS=();
                @ps_SC=();
                @it_both=();
                @it_CS=();
                @it_SC=();
                shift @currentPeriods;

                # Print blank period gap
                while(defined $currentPeriods[0] &&
```

```
                $packetLine[0]>=$currentPeriods[0]){
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,0,0,");
                    printf OUTPUT ("0,0,");
                    $calculatedPeriods++;
                    shift @currentPeriods;
                }
                last if (!defined $currentPeriods[0]);
            }
            if($packetLine[0]<$currentPeriods[0]){
                push (@ps_both, $packetLine[2]);
                $it_current=0;
                $it_current=abs($packetLine[0]-$last_time) if
                    $last_time>0;
                push (@it_both, $it_current) if $it_current
                    >0;
                $last_time=$packetLine[0];
                if($packetLine[1] eq "C-S"){
                    push (@ps_CS, $packetLine[2]);
                    $it_current=0;
                    $it_current=abs($packetLine[0]-
                        $last_CS_time) if $last_CS_time>0;
                    push (@it_CS, $it_current) if
                        $it_current>0;
                    $last_CS_time=$packetLine[0];
                }
                if($packetLine[1] eq "S-C"){
                    push (@ps_SC, $packetLine[2]);
                    $it_current=0;
                    $it_current=abs($packetLine[0]-
                        $last_SC_time) if $last_SC_time>0;
                    push (@it_SC, $it_current) if
                        $it_current>0;
                    $last_SC_time=$packetLine[0];
                }
            }
        }
        # Print last possible period
        if(defined $ps_both[0]){
            @tempPrint=&states_ps(@ps_both);
            $pc_both=shift @tempPrint;
            $bc_both=shift @tempPrint;
            printf OUTPUT ("%d,%d,%.6f,%.6f,",@tempPrint);
```

```
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &states_it(
                    @it_both));
                @tempPrint=&states_ps(@ps_CS);
                $pc_CS=shift @tempPrint;
                $bc_CS=shift @tempPrint;
                printf OUTPUT ("%d,%d,%.6f,%.6f,", @tempPrint);
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &states_it(
                    @it_CS));
                @tempPrint=&states_ps(@ps_SC);
                $pc_SC=shift @tempPrint;
                $bc_SC=shift @tempPrint;
                printf OUTPUT ("%d,%d,%.6f,%.6f,", @tempPrint);
                printf OUTPUT ("%.6f,%.6f,%.6f,%.6f,", &states_it(
                    @it_SC));
                printf OUTPUT ("%.6f,%.6f,", ($pc_CS+1)/($pc_SC+1),(
                    $bc_CS+1)/($bc_SC+1));
                $calculatedPeriods++;
        }
        # Print remining blank periods
        for(my $i=$calculatedPeriods;$i<$periodsNo;$i++){
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,0,0,");
                printf OUTPUT ("0,0,");
        }

        # Print class
        print OUTPUT "$_[1]";
        print OUTPUT "\n";
}

# Function for packets size stats
sub states_ps{
        my $pc=0;
        my $bc=0;
        my $ps_max=0;
        my $ps_min=0;
        my $ps_ave=0;
        my $ps_var=0;
        my $temp=0;
        $pc=$#_+1 if defined $#_;
        foreach $temp (@_){
                $bc+=$temp;
                $ps_max=$temp if $ps_max==0 || $temp>$ps_max;
```

```
                $ps_min=$temp if $ps_min==0 || $temp<$ps_min;
        }
        $ps_ave=$bc/$pc if $pc>0;
        $ps_var=variance(@_) if $pc>0;
        return ($pc,$bc,$ps_max,$ps_min,$ps_ave,$ps_var);
}

# Function for packets interval stats
sub states_it{
        my $it_max=0;
        my $it_min=0;
        my $it_ave=0;
        my $it_var=0;
        my $it_no=0;
        my $temp=0;
        my $temp_sum=0;
        $it_no=$#_+1 if defined $#_;
        foreach $temp (@_){
                $temp_sum+=$temp;
                $it_max=$temp if $it_max==0 || $temp>$it_max;
                $it_min=$temp if $it_min==0 || $temp<$it_min;
        }
        $it_ave=$temp_sum/$it_no if $it_no>0;
        $it_var=variance(@_) if $it_no>0;
        return ($it_max,$it_min,$it_ave,$it_var);
}
```

C.3 len_dist.pl

```
#! /usr/bin/perl -w
# Build packets lens distributions
# Written by Xiaoming Wang
# ARGV[0]=input filename
# ARGV[1]=output filename
# ARGV[3]=max packet No.
# ARGV[4]=incl. class

use IO::Handle;
use strict;

my $inputfilename = shift @ARGV;
my $outputfilename = shift @ARGV;
my $maxpackets = shift @ARGV;
my @includeClass=@ARGV;
```

```
open OUTPUT, ">$outputfilename";
open SUMMARY, "<$inputfilename";


# Main loop
my $summaryLine;
my @summaryLine;
print OUTPUT "Class,Total_packets,Counted_packets,Distribution...\n";
while(defined($summaryLine=<SUMMARY>)){
        chomp($summaryLine);
        @summaryLine=split /,/, $summaryLine;
        if($summaryLine[0] eq '@' && &exists_array($summaryLine[10],
            @includeClass)==1){
                &write_connection($summaryLine[8],$summaryLine[10]);
        }
}
close SUMMARY;
close OUTPUT;

# Function for single conn. (packetNo,class)
sub write_connection{
        my @dist;
        my @packetLine;
        my $packetLine;
        my $counted=0;
        # Init distribution array
        for(my $i=0;$i<=1600;$i++){
                $dist[$i]=0;
        }
        for(my $i=1;$i<=$_[0];$i++){
                $packetLine=<SUMMARY>;
                chomp($packetLine);
                @packetLine=split /,/, $packetLine;
                if($i<=$maxpackets){
                        $dist[$packetLine[2]]++;
                        $counted++;
                }
        }
        print OUTPUT "$_[1],$_[0],$counted";
        # Compute accumulated distribution
        for(my $i=1;$i<=1600;$i++){
                $dist[$i]=$dist[$i]+$dist[$i-1];
        }
        # Normalise & print the distribution
        for(my $i=1;$i<=1600;$i++){
                $dist[$i]=$dist[$i]/$counted;
                print OUTPUT ",$dist[$i]";
```

```
        }
        print OUTPUT "\n";
}

# Sub for check whether a scalar in a array (scalar, array)
sub exists_array{
    my @array=@_;
    my $scalar=shift(@array);
    foreach (@array){
        return 1 if($scalar eq $_);
    }
    return 0;
}
```

C.4 change_class.pl

```
#! /usr/bin/perl -w
# change the class name of dataset
# Written by Xiaoming Wang
# ARGV[0]=input filename
# ARGV[1]=output filename
# ARGV[2..]= names change to others

use IO::Handle;
use strict;

my $inputfilename = shift @ARGV;
my $outputfilename = shift @ARGV;
my @others = @ARGV;

open OUTPUT, ">$outputfilename";
open INPUT, "<$inputfilename";

my $Line;
while(defined($Line=<INPUT>)){
        for(my $i=0;$i<=$#others;$i++){
                $Line=~s/,$others[$i]\n/,others\n/;
        }
        print OUTPUT ($Line);
}
close INPUT;
close OUTPUT;
```

C.5 classify_ks.pl

```
#! /usr/bin/perl -w
# Classify the distribution with ks
# Written by Xiaoming Wang
# ARGV[0]=train filename
# ARGV[1]=test filename
# ARGV[2]=threshold
# ARGV[3]=classifyClass
# ARGV[4]=error output

use IO::Handle;
use strict;

my $trainfilename = shift @ARGV;
my $testfilename = shift @ARGV;
my $threshold=shift @ARGV;
my $classifyClass=shift @ARGV;
my $errorfilename = shift @ARGV;

open TEST, "<$testfilename";
open ERROR, ">$errorfilename";

# Main loop
my $testLine;
my @testLine;
my $result;
my $caseNo=0;
my $correct=0;
print ERROR "Class,Classified_as","\n";
$testLine=<TEST>;
while(defined($testLine=<TEST>)){
        $caseNo++;
        chomp($testLine);
        @testLine=split /,/, $testLine;
        $result=&classify(@testLine);
        print ERROR $testLine[0],',',$result,"\n";
}
close TEST;
close ERROR;

# Sub for classify each testing case
sub classify{
        my $trainLine;
        my @trainLine;
        my $tempD;
        my $minD=2;      # Set default min D larger than 1
```

```
        my $class="none";
        open TRAIN, "<$trainfilename";
        $trainLine=<TRAIN>;
        while(defined($trainLine=<TRAIN>)){
                chomp($trainLine);
                @trainLine=split /,/, $trainLine;
                $tempD=0;
                # For each trainning case find biggest D
                for(my $i=1;$i<=1600;$i++){
                        if(abs($trainLine[$i+2]-$_[$i+2])>=$tempD){
                                $tempD=abs($trainLine[$i+2]-$_[$i+2])
                                    ;
                        }
                }
                # Find the smallest D among all cases
                if ($tempD<=$minD){
                        $minD=$tempD;
                        $class=$trainLine[0];
                }
        }
        # Reject the result if do not reach the threashold
        if($minD>$threshold){
                $class="others";
                $class="rejected_by_$classifyClass";
        }
        close TRAIN;
        return $class;
}
```

C.6 stats_para.pl

```
#! /usr/bin/perl -w
# Statistics for para
# Written by Xiaoming Wang
# ARGV[0]=input filename
# ARGV[1]=output filename

use IO::Handle;
use strict;

my $inputfilename = shift @ARGV;
my $outputfilename = shift @ARGV;

open INPUT, "<$inputfilename";
open OUTPUT, ">$outputfilename";
```

```
my $tmp;
my @tmp;
my $tmp2;
my @tmp2;
my $sum_a;
my $sum_p;
my $correct;
my $sum;
my @a;
my @p;
# For matrix_p csv output
my @className=("ms-rdp","rtsp","pop3","smtp","ssh","ftp","imaps","irc
    ","telnet","ftp-data","others");
print OUTPUT "correct,all,percent";
for (my $i=0;$i<=$#className;$i++){
        print OUTPUT ",",$className[$i],"_a";

}
print OUTPUT ",weighted_ave_a,ave_a";
for (my $i=0;$i<=$#className;$i++){
        print OUTPUT ",",$className[$i],"_p";

}
print OUTPUT ",weighted_ave_p,ave_p,\n";

# Get data from weka output
while(defined($tmp=<INPUT>)){
        chomp($tmp);
        if($tmp=~/Error on test data/){
                $tmp2=<INPUT>;
                $tmp2=<INPUT>;
                chomp($tmp2);
                @tmp2=split /\s+/, $tmp2;
                $correct=$tmp2[3];
                $sum=$tmp2[3];
                $tmp2=<INPUT>;
                chomp($tmp2);
                @tmp2=split /\s+/, $tmp2;
                $sum+=$tmp2[3];
                print OUTPUT $correct,",",$sum,",",$correct/$sum,",";
        }
        if($tmp=~/Detailed Accuracy By Class/){
                $tmp=<INPUT>;
                $tmp=<INPUT>;
                $sum_a=0;
                $sum_p=0;
```

```
                @a=();
                @p=();
                for(my $i=0;$i<11;$i++){
                        $tmp=<INPUT>;
                        chomp($tmp);
                        @tmp=split /\s+/, $tmp;
                        $sum_a+=$tmp[1];
                        $sum_p+=$tmp[3];
                        push @a,$tmp[1];
                        push @p,$tmp[3];
                }
                $tmp=<INPUT>;
                chomp($tmp);
                @tmp=split /\s+/, $tmp;
                foreach (@a){print OUTPUT ($_,",");}
                print OUTPUT ($tmp[2],",",$sum_a/($#className+1),",")
                    ;
                foreach (@p){print OUTPUT ($_,",");}
                print OUTPUT ($tmp[4],",",$sum_p/($#className+1),"\n"
                    );
        }
}

close INPUT;
close OUTPUT;
```

C.7 stats_ks.pl

```
#! /usr/bin/perl -w
# Statistics for ks
# Written by Xiaoming Wang
# ARGV[0]=input filename
# ARGV[1]=output filename

use IO::Handle;
use strict;

my $inputfilename = shift @ARGV;
my $outputfilename = shift @ARGV;
# The input should be in (class,classified as) csv format
open INPUT, "<$inputfilename";

my $tmp;
my @tmp;
my @className=("ftp","ftp-data","others","imaps","irc","ms-rdp","pop3
```

```
    ","rtsp","smtp","ssh","telnet");
# no. of classes
my $validClass=0;
# no. of classifed as
my $validClassified=0;
my %class;
my %classified;
# TP
my %correct;
# Init for hash
for (my $i=0;$i<=$#className;$i++){
        $class{$className[$i]}=0;
        $classified{$className[$i]}=0;
        $correct{$className[$i]}=0;
}
my @a;
my @p;
my $a_w=0;
my $p_w=0;
my $a_a;
my $p_a;
my $correct=0;
my $sum=0;

# Input the data
$tmp=<INPUT>;
while(defined($tmp=<INPUT>)){
        chomp($tmp);
        @tmp=split /,/, $tmp;
        $class{$tmp[0]}++;
        $classified{$tmp[1]}++;
        $sum++;
        if($tmp[0] eq $tmp[1]){
                $correct{$tmp[0]}++;
                $correct++;
        }
        if($tmp[1]=~/rejected/ && $tmp[1]!~/$tmp[0]/){
                $correct++;
        }
}
close INPUT;

# Print the file head if output file do not exists
if (!-e $outputfilename){
        open OUTPUT, ">>$outputfilename";
        print OUTPUT "correct,all,percent";
        for (my $i=0;$i<=$#className;$i++){
```

```
                print OUTPUT ",",$className[$i],"_a";

        }
        print OUTPUT ",weighted_ave_a,ave_a";
                for (my $i=0;$i<=$#className;$i++){
                print OUTPUT ",",$className[$i],"_p";

        }
        print OUTPUT ",weighted_ave_p,ave_p\n";
}else{
        open OUTPUT, ">>$outputfilename";
}

# Print overall
print OUTPUT $correct,",",$sum,",",$correct/$sum;
# Print the accuracy
for (my $i=0;$i<=$#className;$i++){
        if($class{$className[$i]}==0){
                print OUTPUT ",","0";
                push @a,"0";
        }else{
                print OUTPUT ",",$correct{$className[$i]}/$class{
                    $className[$i]};
                push @a,$correct{$className[$i]}/$class{$className[$i
                    ]};
                $validClass++;
        }

}
print OUTPUT ",";

# Print weighted and average accuracy
for (my $i=0;$i<=$#className;$i++){
        $tmp=shift @a;
        $a_w+=$tmp*$class{$className[$i]}/$sum;
        $a_a+=$tmp;
}
print OUTPUT $a_w,",",$a_a/$validClass;

# Print the precesion
for (my $i=0;$i<=$#className;$i++){
        if($classified{$className[$i]}==0){
                print OUTPUT ",","0";
                push @p,"0";
        }else{
                print OUTPUT ",",$correct{$className[$i]}/$classified
                    {$className[$i]};
```

```
                push @p, $correct{$className[$i]}/$classified{
                    $className[$i]};
                $validClassified++;
        }
}

# Print weighted and average precision
for (my $i=0;$i<=$#className;$i++){
        $tmp=shift @p;
        $p_w+=$tmp*$class{$className[$i]}/$sum;
        $p_a+=$tmp;
}
if ($validClassified!=0){
        print OUTPUT ",",$p_w,",",$p_a/$validClassified;
}
else{
        print OUTPUT ",",$p_w,",","0";
}
print OUTPUT "\n";

close OUTPUT;
```

APPENDIX D

Publications

Xiaoming Wang and David J. Parish, "Optimised TCP Traffic Classification with Multiple Statistical Algorithms," in *Proceedings of International Conference on Information, Networking and Automation, IEEE*, 2010, vol. 1, pp. 261–265.

Xiaoming Wang and David J. Parish, "Optimised Multi-stage TCP Traffic Classifier Based on Packet Size Distributions," in *Proceedings of the Third International Conference on Communication Theory, Reliability, and Quality of Service, IEEE*, 2010, pp. 98–103.

Xiaoming Wang, Martin D. Sykora, Robert Archer, David Parish and Helmut E. Bez, "Case Based Reasoning Approach for Transaction Outcomes Prediction on Currency Markets," in *Proceedings of the Third International Workshop on Soft Computing Applications, IEEE*, 2009, pp. 91–96.

X. Wang, M. S. De-Silva and D. J. Parish, "Analyzing the Behavior of Network Attacks at a High Level using Case-based Reasoning," in *Proceedings of the Eighth Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting*, 2007, pp. 397–400.