Identification of Networked Tunnelled Applications

by

Ghulam Mujtaba

A Doctoral Thesis

Submitted in partial fulfilment of the requirements
for the award of Doctor of Philosophy of Loughborough University

May, 2011

Abstract

In protocol tunnelling, one application protocol is encapsulated within another carrier protocol in an unusual way to circumvent firewall policy. Application-layer tunnels are a significant security and resource abuse threat for networks because those applications which are restricted by firewalls such as high data-rate games, peer-to-peer file sharing, video and audio streaming, and chat are carried through via allowed protocols like HTTP, HTTPS and the firewall security policy is thwarted. Protocols such as HTTP and HTTPS are indispensable today for any network which has to be connected to the Internet; hence these become a high value target for running restricted applications via tunnelling. The identification of the actual application running across a network is important for network management, optimization, security and abuse prevention. The existing techniques for identification of applications running across the network, for example port number based identification, and packet data analysis techniques are not always successful, especially for applications which use encrypted tunnels. This work describes a statistical approach to detect applications which are running using application layer tunnels. Previous work has shown the packet size distribution to be an effective metric for detecting most network applications, both UDP and TCP based applications. In this work it is shown how packet stream statistics including packet size distributions can be used to differentiate and identify networked tunnelled applications successfully. Tunnelled applications are identifiable using the traffic statistical parameters. Traffic trace files of the applications were captured, statistical parameters were derived from the trace files, and then these parameters were used for training machine learning algorithms. The trained machine learning algorithm is then able to classify the other packet trace data as belonging to an application. Five different machine learning algorithms have been applied, and their performance accuracy is discussed. The entropy distance based Nearest Neighbour machine learning algorithm and the Euclidean Distance based Nearest Neighbour classifier had better results than others. This method of identification of tunnelled applications can be complimentary to other network security systems such as firewalls and Intrusion Detection Systems.

Acknowledgements

I am thankful to God, first and foremost for creating me, bestowing me with countless blessings, capabilities and guiding me.

I would express many thanks to my supervisor Professor David J Parish for his guidance, encouragement, time, inspiration and reflection throughout my PhD. The kind and insightful advice from David has always motivated me and helped me achieve the objectives of this work. I am grateful to my mother, brothers, sisters, wife and children for their support, encouragement and sacrifices.

I am thankful to my research colleagues in the High Speed Networks (HSN) group for their support and for sharing their knowledge. These include Xiaoming Wang, Yaqoob Juma Yaqoob Al-Raisi, Konstantinos Kyriakopoulos, Dr. Akhtar H. Khalil, Dr. Li Bo, Dr. Mark Withall, Dr. Marcelline Shirantha de Silva, Dr. John Whitley, Roshdi, Rye, and Jin Fan.

Table of contents

List of Figures

Acronyms

ATM	Asynchronous Transfer Mode
DNS	Domain Name Service
HTTPS	Hypertext Transfer Protocol Secure
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
PDU	Protocol Data Unit
ssh	Secure shell
TLS	Transport Layer Security
SSL	Secure Sockets Layer
MTU	Maximum Transmission Unit
OSI	Open Systems Interconnection
PC	Personal Computer
RFC	Request For Comments
RPC	Remote Procedure Call
RTP	Real Time Transport Protocol
NBayes	Naïve Bayes
SMTP	Simple Mail Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
PSD	Packet Size Distribution
NN or IB1	Nearest Neighbour Euclidean
MLP	Multilayer Perceptron
NN K*, or IBK*	Nearest Neighbour based on Entropy

Chapter 1 Introduction

The Internet is one of the greatest developments of the twentieth century. It has penetrated into almost all aspects of a modern man's life, i.e. entertainment, communications, telephony, video conferencing, commerce, education, research, telemedicine, gaming, etc. Unparalleled advancements in processing powers and the pervasive usage of the computers in homes and offices have led to the development of a large number of new applications, utilizing the enormous computational resources. Internet bandwidth is a finite and expensive resource for an organisation and needs to be protected from spammers, criminals, hackers, time-wasters and employee misuse. It is expensive, limited and a vital business tool [1]. There has been an enormous growth of some new bandwidth-hungry applications, such as networking games, video/audio on demand, voice over IP, video conferencing, peer to peer file sharing and so on, and traditional applications like email and file transfer have also increased in usage [2].

The network administration in an organisation at times does not want a certain class of applications to run on their network. These applications may be conflicting with the acceptable usage policy of the organisation. Almost every organisation, school, university, ISP has nowadays its own acceptable internet usage policy, which needs to be policed [3]. The network administrators want to shape or block some of the traffic, because of the enormous bandwidth consumption of these applications or because of the policies. The misuse of an organisation's internet facilities for non-work related purposes is of greater concern for most employers. Problems are not limited to information security violations alone. Many organizations are losing a lot of money through uncontrolled internet access or network abuse. A survey of nearly 200 international companies carried out by Infosec, Netpartners and Secure Computing Magazine estimated that a typical large company (1,000 employees) could be losing £2.5 million per year through employees' use of Internet for non-business purposes – an average loss of £2,500 per employee [2]. One-third of time

spent online at work is non-work-related [4]. 75% of companies cited employees as a likely source of hacking attacks. There are more than 43 million users of consumer IM (Instant Messengers) at work. 44 per cent of corporate employees actively use streaming media [4]. Game playing on office computers actually costs businesses about $50 billion a year [3]. When these applications are blocked using firewalls or other filtering programs, tunnelling techniques are employed to breach the filtering.

1.1 Networked Tunnelled Applications

Application layer tunnels are employed by malicious network users to access prohibited applications after bypassing the firewall restrictions. Tunnelling is a way of hiding one application's packets as the payload of another protocol's packets which can then be called a carrier. The protocol of the carrier is one which is not restricted by the firewall. In tunnelling, the firewall policy is circumvented and restricted/unwanted applications are run behind a protocol which is allowed over the network, such as HTTP [5]. In other words it is the process of putting one packet inside another [24]. Those applications which employ this technique across the network are Networked Tunnelled Applications.

The identification of the actual application running inside a protocol tunnel across the internet or a local area network (LAN) in an attempt to avoid detection is the objective of this thesis. This identification information is attractive to network administrators, network service providers and security systems. Traditional uses of this information include activities such as network policing and network management [2]. This information can be useful in preventing improper network use which may include illegal activities, consuming a large amount of bandwidth or violation of network usage policies of an organisation. For this purpose an efficient mechanism for identification of the application generating a traffic stream is required for security and network abuse detection, where a network operator may wish to be made aware if certain applications are invoked, or even to stop them from operating. Depending on policies and legal obligations, blocking some applications is also a security issue which becomes more important for schools, colleges, universities and enterprises. Many businesses and government institutions use

filtering programs such as Websense [90], in addition to the firewall to keep employees from visiting certain websites while at work. These programs can either prohibit the use of certain websites or monitor all of the sites that employees do visit while in the office. For example, Websense is designed to block internet traffic in several categories, such as adult material, entertainment, drugs, games, sports, internet communication, peer-to-peer file sharing, gambling, instant messaging, health, shopping, job search, internet telephony, special events, travel, violence, advertisements, freeware and software download, pay-to-surf, malicious websites, and many more [90]. HTTP Tunnelling programs are employed to bypass the firewall and the filtering programs such as Websense. An HTTP Tunnel is often used as a means for communication from network locations with restricted connectivity to access blocked sites applications (games/IM clients/browsers). The examples of such tunnelling software are VPNTunnel, http-tunnel [91].

Today the number of tunnelling tools being used to hide the exact application information is on the rise. There are encrypted tunnels which use HTTPS [6] protocol and hence cannot be detected by even Deep Packet Inspection [7]. This thesis attempts to detect applications running in tunnels both encrypted or unencrypted using the statistical properties of the application trace files.

1.2 Application Identification: An Overview

Traditionally, networking applications were identified through the TCP or UDP port number. The port number is a numerical figure used for identity of the process at the transport layer of the network architecture. This approach worked well for traditional applications because applications generally had uniquely identifiable and well know port numbers that were registered internationally. Hence, knowing the port number held in a packet header of any of the packets from the data stream generated by the application uniquely identified that application. Many such applications use the TCP protocol for their Transport Layer, such as the web protocol HTTP, which is uniquely identified by the port number 80.

However, gradually the number of applications which do not use registered port numbers increased because it was not binding to use the registered port and applications started using unregistered port numbers. Examples of these applications include real time applications, such as network games, and audio/video transfer applications such as video conferencing.

A more reliable method for application detection is that of "Deep Packet Analysis". This method attempts to look at the data held in the packets, and usually it is based on searching certain unique identification patterns or signatures held in the data. Mostly Deep Packet Analysis requires capturing the packets of session establishment and session initiation phase, because these are the packets which are likely to contain the unique signature pertaining to the application protocol. This technique is also used in Intrusion Detection Systems, for example Snort [37] and Bro [38]. The captured packets are searched for unique identifiers and keywords which are kept in a fairly large database. The detector in such a case would fail to detect if it is initiated after the session has been established, or the network is heavily loaded with capturing becoming difficult. Deep Packet Analysis or Deep Packet Inspection requires more processing power too as it has to operate with faster networks and the process itself is computationally quite intensive. Then in case of encryption, this technique cannot work.

In [48], the author presents a different approach to the problem of detecting non-registered UDP based applications, which is based on packet statistics. The author showed empirically that the packet size distribution follows an application specific profile, and this statistic is now considered as a detection metric. Hence the process takes a sample of the data stream generated by an application, and performs detection based on or over that sample, rather than an individual packet. The Packet Size Distribution based detection mechanism can also operate on encrypted applications. The scheme uses significantly less information than the previous Deep packet analysis techniques, reducing the storage and processing requirements. Then, the work of Li Bo et al [49] applied the same principles and used packet size distribution in identification of TCP based applications which form the bulk of traffic over the internet today. Based on this research and other work described in

later chapters, the identification of covert applications inside tunnels is undertaken in this thesis.

1.3 Original Contributions of this research

In this thesis, the objective was to show that the applications running inside encrypted or unencrypted tunnels are differentiable, identifiable based on the statistical properties of their traffic traces.

Traditional application detection techniques heavily relied upon the content inside the traffic. The work by Parish [48] and Li Bo [49] proposed the Packet size distribution as an identification metric for a large number of UDP and TCP based applications. In this thesis the realm of application identification has been extended to the covert channel applications running inside protocol tunnels. Recently protocol tunnels have become popular with network abusers to circumvent security and monitoring policies by hiding the applications inside allowed protocols such as HTTP. Even the use of encrypted tunnels which use secure protocols like SSL, TLS or SSH is observed which further complicates detection. There are some other statistical based identification techniques in the research literature which will be described in later chapters, but the packet size distribution based statistic with a combination of other traffic parameters is investigated in this thesis. The use of packet size distribution, along with a number of other statistical metrics to successfully identify tunnelled applications, is shown.

The other discriminant statistics/metrics are included which have been proposed in several research works and these are used in combination and also in separation with the packet size distribution. These metrics are Data rates (bytes/sec) for the upstream direction, Data rates (bytes/sec) for downstream direction, Data packet ratio, ByteRatio, Ratio of large and small packets, time spent idle downstream, and time spent idle upstream. Some temporal (time related) metrics were also investigated such as maximum Interarrival time, minimum interarrival time etc., but it was shown that these don’t behave in a consistent and reliable manner and should be discarded. The best performance is obtained when the parameters are used in combination with the packet size distribution. Thus the identification of suitable

parameters' combination which can perform the detection of applications running inside a tunnel is another contribution.

A key feature of this work is that this detection mechanism is able to identify the actual application which is being run inside an application layer tunnel. There is other work in which researchers have tried to detect tunnelling activities in a network; however their focus was mostly to identify whenever one protocol is tunnelled inside another protocol. That is not as desirable as the detection of the application, since several times tunnelling traffic is quite legitimate. This will be explained in later chapters in detail. This work aims at the identification of the application which is being tunnelled.

The use of machine learning algorithms for the purpose of application identification with the training parameters containing Packet Size Distribution as the key parameter is also a novel aspect of this work. The work of Li Bo [49] and Parish [48] were based on Packet Size Distribution bins only, and they used simple statistical tests like Chi Squared Test for the purpose of identification of applications. The number of statistical metrics/discriminators was increased in this work as described already; and five different machine learning algorithms have been used for the identification of tunnelled applications and the performance and efficiency of these algorithms is compared.

1.4 List of Applications Identified

The methodology introduced in this work was tested to identify the following applications when running inside protocol tunnelling. More applications can be added following the same procedures.

World of Warcraft (abbreviated WW), VoipRaider (abbreviated VR),iVisit (abbreviated IV), QuakeLive (abbreviated QL), Skype (abbreviated SK), Lord of Ultima (abbreviated LU), Quake3Arena (abbreviated QA), Xlite (abbreviated XL), Medal of Honour Allied Assault (abbreviated MA), Camfrog (abbreviated CF), Zattoo (abbreviated ZT), RealPlayer (abbreviated RP), Remote Desktop

(abbreviated RD), GuildWars (also abbreviated GW), Unreal Tournament (abbreviated UT), Runescape (also abbreviated RE).

1.5 Thesis Overview

The rest of the thesis contents briefly are described below:

Chapter 2 introduces the main concepts related to the work in this thesis starting from firewalls; the purpose and types of firewalls. Then it explains how application layer tunnels are used to sneak through firewalls. An overview of the main tunnel protocols is also given. Then the previous work in the area of tunnelling detection is critically analysed, the traditional and current popular application detection techniques, and their characteristics are briefly discussed.

Chapter 3 describes the background and rationale for the packet size distribution to be used as application identification metric, since it is the most important statistical property used in this work. Here the packet size distribution of tunnelled applications is obtained and analysed using the Chi Squared test and it is established that the packet size distributions of the same application are statistically similar whereas those of different applications are significantly different.

Chapter 4 describes the methodology used for the detection of the tunnelled applications from captured trace files to obtaining the training data and applying the machine learning tools.

Chapter 5 gives the results obtained by applying different machine learning algorithms for the purpose of detection and with various combinations of statistical parameters obtained from the training data. The results are discussed, and an optimal combination of parameters is outlined.

Chapter 6 concludes the work by summarising the key findings, discussing limitations of the approach and recommendations about the further work in the same area.

Chapter 2 Firewalls and Tunnels Review

This chapter introduces the concept of tunnelling and the reasons protocol tunnelling is performed for. First the firewall is explained, including their purpose and types. Then, with the example of HTTP tunnelling, firewall breaching via tunnelling is elaborated. The protocols which are commonly used in tunnelling are described. The motivation behind tunnelling, and some of the popular software tools used for tunnelling, are presented, including the common programs used in tunnelling. In the last section, previous work in the area of tunnelling detection and traffic classification is summarised.

2.1. Firewalls

The internet is a huge resource of information which is made available to the user via a home pc or at work all the time. The access to the internet has become an integral necessity of everyday life. Nevertheless, connecting a computer or a private network to the public internet opens a door of access to critical and confidential data for malicious attackers from anywhere. Therefore it is very important to be aware of these threats and have protection against them. Firewalls are one of several strategies to serve this purpose.

Firewalls are commonly in practice to secure private networks, home or corporate, from malicious intrusions from the Internet. These intrusions can harm in a number of ways causing data loss or denial of service. The firewall can be either a hardware device or software running on a secure computer. The firewall is located at the periphery of the private network, usually at the junction between private and the public network, as depicted in the figure 2-1.

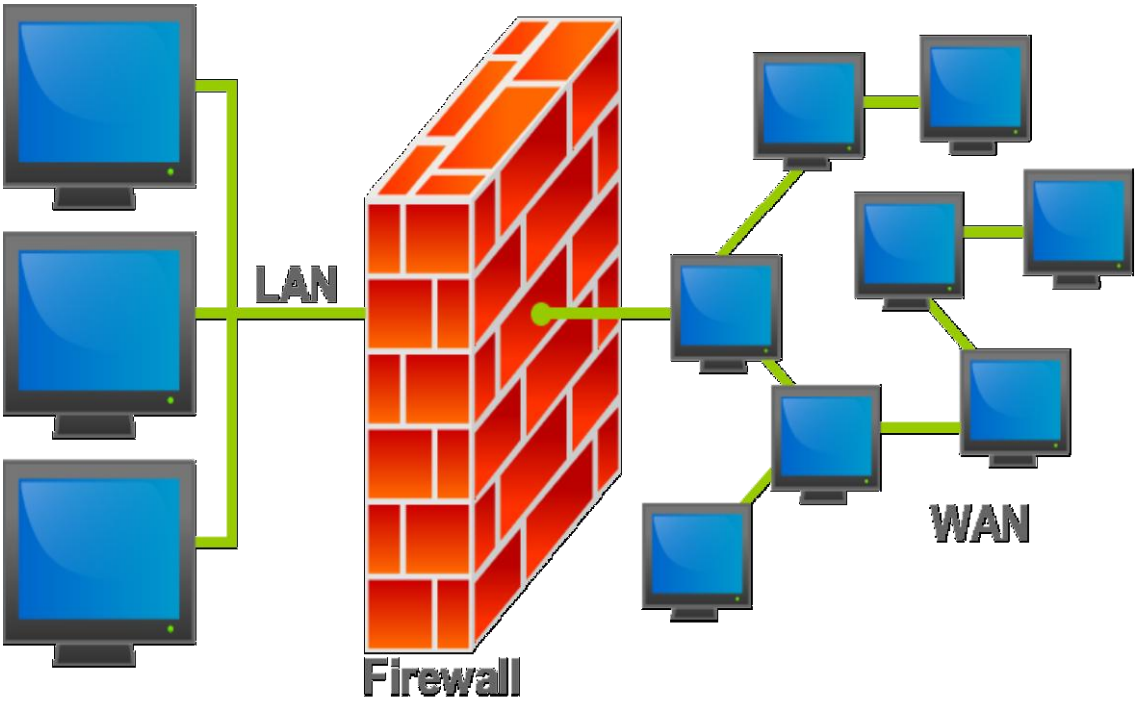


Figure 2-1: A view of the location of firewall in a network, adapted from [21]

The firewall performs the functions of monitoring all incoming and outgoing packets going between the two networks it is connected to. The criteria for monitoring are set by the network administrator, and only traffic conforming to the criteria is delivered, otherwise, it is discarded.

Firewalls can perform packet filtering for example, based on their source and destination addresses and ports. In a TCP/IP environment, usually the corporate firewall is configured to block all incoming and outgoing data, and gives access to the internet only through an HTTP proxy. The proxy usually has filtering capabilities and access to the proxy often requires credentials (login/password). This gives greater control to the network administrator over what and who is going in and out of the network.

Firewalls can be divided into two fundamental types:

1. Packet Filtering firewalls.
2. Application gateway or proxy firewalls.

2.1.1 Packet-filtering Firewalls

The fundamental function of packet filtering firewalls is to selectively discard network traffic packets or allow them through based on the security policy. Packet filters use the network layer header (IP) and the transport layer header (TCP or UDP) in order to filter packets. The information used can be packet type, i.e. protocols and port addresses of sources and/or destinations. Another function usually performed by such firewalls is Network Address Translation (NAT), a method used to hide the network's architecture inside of the firewall, blocked from the outside world, and to conserve routable Internet Protocol (IP) addresses. Different methods of packet filtering have developed over the years. However, 90 per cent or more of today's firewalls are still only performing these two main functions [22].

This kind of firewall provides little or no protection to the application layer. For example, an FTP [18] service can go through HTTP (a web service) with most of today's firewalls, and there is nothing that most packet-filtering firewalls can do about it. In summary, packet-filtering firewalls, while providing basic protection to the internal network, is not fully able to protect internet applications [22].

2.1.2 Application Gateway/Proxy Firewalls

An application gateway is a firewall system that uses the application layer data in order to filter packets. The main idea that stands behind an application gateway is not to allow direct connection between programs running on an external computer to the programs of the internal network. All the packets are intercepted at the firewall or proxy server/gateway. Then packets are inspected for different criteria. Usually the packets are checked for their sources and destination addresses, type of protocols and port numbers, sometimes even the contents of the payloads and commands. If the packets pass the inspection criteria, they are reframed and sent out to their original destinations. Since all original packets are destroyed before being forwarded to their destinations, this type of firewalls prevent attacks based on the weaknesses of the TCP/IP protocols, which were never designed with security in

mind. Moreover, this method allows the firewall to perform deep inspection that the packet-filtering method cannot. [22]

2.1.3 Weaknesses of Firewalls:

Even the second type of firewall is quite vulnerable. Firstly, not all application proxy firewalls implement 100 per cent of the application proxy functionalities as described above for each protocol and application. Even with well-known protocols like HTTP, type two firewalls don't always cover all applications that could tunnel through the HTTP protocol. Mostly this is because many applications are being developed using HTTP. HTTP is now used to transfer much more than the ordinary web site content which includes audio and video as well as VPN tunnels and peer-to-peer file exchange and other applications which could be against the organisation's network usage policy. Another issue is that not all traffic is clear text, on which the firewall can conduct deep inspection. Firewalls can't inspect what is inside encrypted HTTP traffic. Sometimes other services are being tunnelled through the encrypted protocols, such as HTTPS, SSH through these firewalls. [22]

2.2. Tunnelling: Introduction

Application layer tunnels are nowadays getting significant attention by malicious network users trying to access prohibited applications after bypassing the firewall restrictions. Tunnelling is a way of hiding one application's packets as the payload of another carrier protocol. The protocol of the carrier is one which is not restricted by the firewall or application-level gateway. In a firewall, security is implemented by listing the allowed application layer protocols and sometimes by mentioning the allowed destination addresses which can be contacted through these applications. In tunnelling, however, the blocked protocol is wrapped inside the legitimate protocol, such as HTTP [23], and thus the firewall is breached. Those applications which employ this technique across the network are Networked Tunnelled Applications. There are numerous examples where this kind of technique is applied.

Tunnelling is different from the usual layered protocol model such as those of OSI or TCP/IP. Protocols describe the rules that control conversations between processes that run at corresponding layers within the OSI Reference Model or TCP/IP model. The messages used in communicating information between protocols, are called protocol data units (PDUs). Each PDU has a specific format that implements the features and requirements of the protocol. The communication between layers above the physical layer is logical; the only hardware connection is at the physical layer. In order for a protocol to communicate, it passes down its PDU to the next lower layer for transmission. At any particular layer N, a PDU is a complete message that implements the protocol at that layer. However, when this layer N PDU is passed down to layer N-1, it becomes the data that the layer N-1 protocol is supposed to service. Thus, the layer N protocol data unit (PDU) is called the layer N-1 service data unit (SDU). Layer N-1 transports this SDU by placing the layer N SDU into its own PDU format, preceding the SDU with its own headers and appending footers as necessary. This process is called encapsulation, as the entire contents of the higher-layer message are encapsulated as the payload of the message at the lower layer.

The tunnel protocol is usually (but not always) at a higher level than the payload protocol, or at the same level. Protocol encapsulation that is carried out by conventional layered protocols, in accordance with the OSI model or TCP/IP model, for example HTTP over TCP over IP over Ethernet protocols, should not be considered as tunnelling [25].

2.3. HTTP Tunnelling Example

To illustrate http tunnelling, consider an example, as shown in the following figure 2-2:

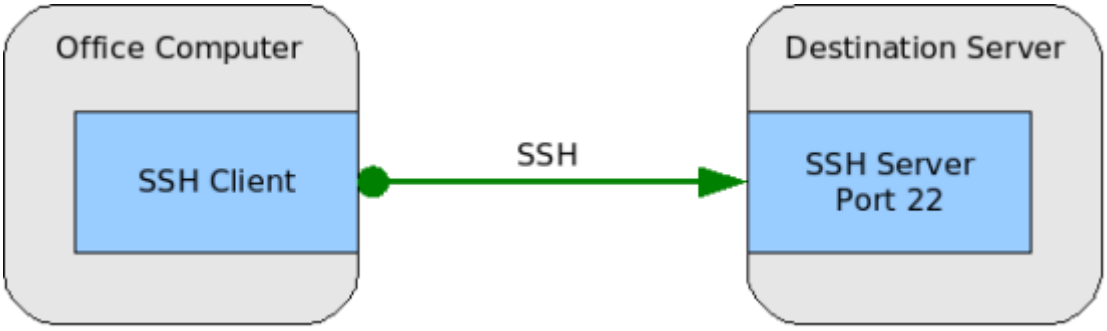


Figure 2-2: Normal SSH client/server [26]

In this figure an SSH client application running on a host is successfully communicating with a remote SSH server located somewhere over the Internet. The SSH protocol runs above the TCP and its purpose is to ensure confidentiality and integrity of data between two systems over a network. This protocol is normally used for command execution through a secure shell and file copy between peers [28]. In another scenario there is a firewall at the client machine and only outgoing HTTP traffic is let through the firewall. This case is shown in Figure 2-3.

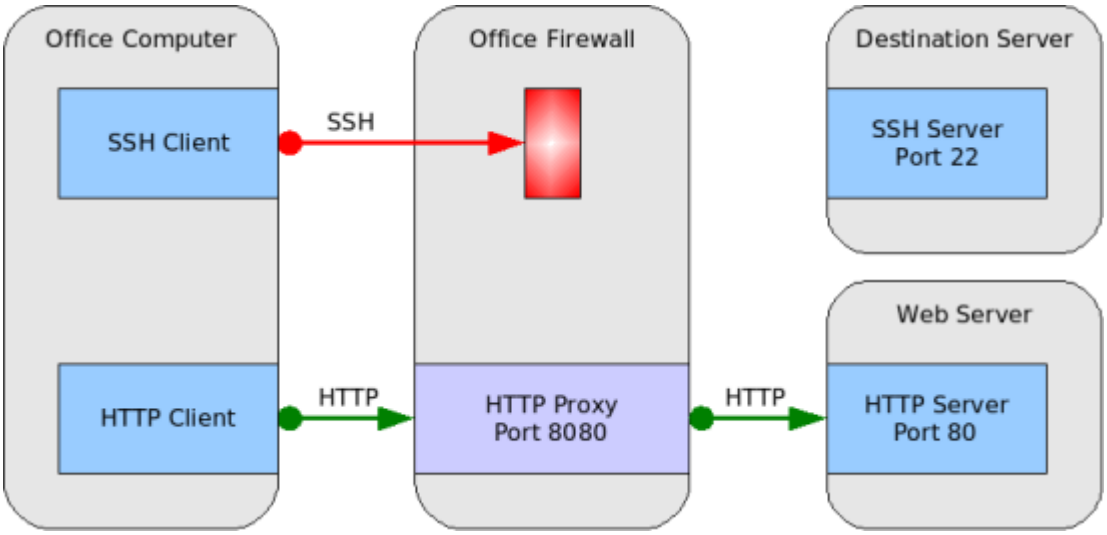


Figure 2-3: Firewall blocking SSH traffic[26]

Here the SSH server is not accessible because the firewall is not allowing its access. The SSH client program is trying to communicate with the SSH server, but this cannot occur. The firewall (possibly in combination with an HTTP proxy) however does allow HTTP traffic through from the client.

The Tunnelling software when employed utilizes the allowance of outgoing HTTP traffic from the firewall, and it creates an HTTP tunnel out to an external Tunnel server (Firedrill, in figure 2-4) over which all normal TCP communications can take place. The Server side of the tunnel software can then unwrap the original client TCP data and forward it on to the ultimate destination [26].

This scenario is shown below in Figure 2-4, where an SSH client makes a tunnel through the firewall:

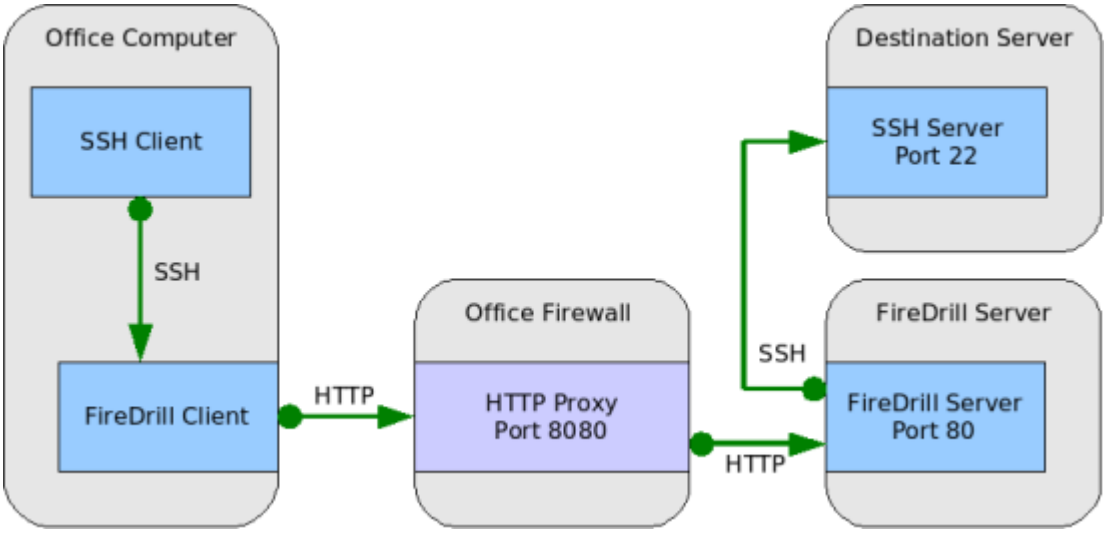


Figure 2-4: Firedrill tunnelling SSH traffic[26]

This tunnelling system is called an HTTP tunnelling proxy. It consists of two components; a client (software) installed on a local PC, and a server part which can be remote over the Internet.

The client software acts as a Socks proxy [27] to any application running on the users local PC that is trying to connect to the internet. It reads data from that application, encapsulates that data in HTTP, and tunnels it out to a corresponding tunnel server [26].

The server takes the HTTP data sent from the client and conversely unwraps the original data, and forwards it on to the actual destination to which communication was intended. Any response from there is also conveyed back to the client wrapped in HTTP which unwraps it and transfers it to the original application.

In this manner, using the HTTP protocol for encapsulating data from desired applications the communication is achieved through restrictive firewalls and web proxies which now allow those connections through which might have been blocked before.

2.4. Various Tunnelling Protocols:

Although theoretically tunnelling can be done using a host of protocols, in practice the following kinds of tunnelling tools are utilized at the application layer:

2.4.1 DNS tunnelling

Application-layer tunnels are possible using the Domain Name Service (DNS) protocol by just utilizing the way regular DNS requests for a given domain are forwarded to its servers. DNS Tunnelling works by abusing DNS records to traffic data in and out of a network. Since the DNS protocol is not usually blocked on the Internet, this technique can be very successful, and people have tried to implement it as in [28] and [29]. "Due to the mechanism's complexity, however, DNS tunnels can rarely achieve throughputs higher than a few kb/s, and are therefore seldom used." [28]. So in this thesis, the focus has been on rather more practical and efficient tunnelling implementations which exploit the common protocols of HTTP and SSL to tunnel TCP and UDP based application data.

2.4.2 HTTP tunnels

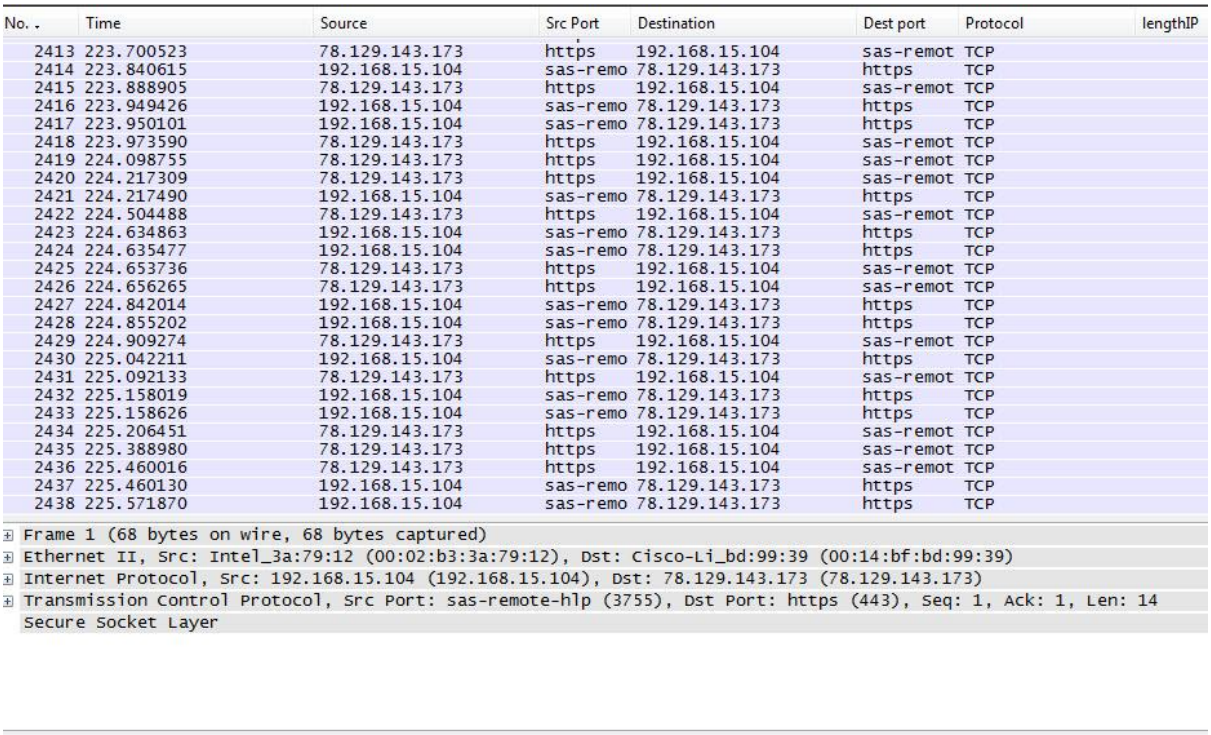
The HTTP (Hypertext Transfer Protocol) is the protocol defining rules for transferring files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. HTTP is an extensively used protocol for internet traffic and thus there is a widespread potential for misuse of HTTP for tunnelling data. In [30] the Internet traffic of a large enterprise was monitored for a one week period and it was found that over 40% of all incoming and over 90% of all outgoing data consisted of HTTP traffic. The HTTP traffic is mostly allowed to pass all network peripheries and application layer proxy firewalls. If internet access is allowed through an HTTP proxy or firewall, it is possible to use HTTP tunnels to connect to a computer outside the firewall. These policies are violated for running restricted applications through the http tunnel as explained in the previous section with the Firedrill tunnelling system example. There are many tools available on the Internet, which for little cost, help users bypass proxy servers and firewalls to run applications that use either TCP or UDP as their communication protocol, for

example, World of Warcraft, X-Lite, Final Fantasy XI, Everquest II, TeamSpeak, Camfrog, MSN Messenger, Yahoo Messenger, ICQ, Trillian, Skype etc. [31].

2.4.3 Encrypted Tunnel

There are tunnels which run through encryption capable protocols such as the Transport Layer Security (TLS) protocol, or its predecessor, the Secure Sockets Layer (SSL) protocol, or even the Secure Shell (SSH) protocol. TLS is an IETF standard protocol described in RFC 5246 based on SSL [32]. These cryptographic protocols provide security and data integrity for data transport over networks by encrypting the network connections at the Transport Layer, end-to-end. The combination of HTTP and TLS/SSL is known as Hyper-Text Transfer Protocol Secure (HTTPS). The HTTP protocol uses port 80, normally whereas HTTPS use port 443 by default. Several versions of the TLS/SSL protocols are used widely in many applications, a frequent example being a secured login webpage of email clients, like Hotmail, Yahoo mail etc. The firewall administrators open outbound HTTPS to selected users so that they can go to secure Web sites. However, using tunnelling tools, the same HTTPS (SSL) is used to run applications which normally would be blocked by the firewall or application layer gateway, and since the application runs in an SSL tunnel, virtually no firewall can inspect the contents of the SSL stream [33]. Secure (HTTPS) tunnelling works in exactly the same way as standard tunnelling, except that all data between the HTTPS tunnel client and the server is encrypted. Since data is encrypted, any firewall security policy that utilizes the Deep Packet Inspection (DPI) is totally circumvented. The SSH protocol runs above the TCP and its purpose is to ensure confidentiality and integrity of data between two systems over a network. This protocol is normally used for command execution through a secure shell and file copy between peers [28]. With the ordinary HTTP tunnels there is some possibility that using advanced Application Layer Gateways based on DPI the exact nature of the tunnelled traffic could be identifiable. However, in the case of an encrypted tunnel using HTTPS or SSH, the deep packet inspection and signature-based identification are thwarted because of cryptographic encryption. Today security and encryption have become essential in many applications, therefore for the administrators blocking all encrypted traffic is

not an option. Thus, any application data can be tunnelled in and out of the network using encrypted tunnels. Figure 2-5 shows the packet capture details of an application running in an HTTPS tunnel with the Secure Sockets Layer protocol in operation.



No.	Time	Source	Src Port	Destination	Dest port	Protocol	lengthIP
2413	223.700523	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2414	223.840615	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2415	223.888905	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2416	223.949426	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2417	223.950101	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2418	223.973590	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2419	224.098755	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2420	224.217309	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2421	224.217490	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2422	224.504488	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2423	224.634863	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2424	224.635477	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2425	224.653736	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2426	224.656265	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2427	224.842014	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2428	224.855202	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2429	224.909274	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2430	225.042211	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2431	225.092133	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2432	225.158019	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2433	225.158626	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2434	225.206451	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2435	225.388980	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2436	225.460016	78.129.143.173	https	192.168.15.104	sas-remot	TCP	
2437	225.460130	192.168.15.104	sas-remo	78.129.143.173	https	TCP	
2438	225.571870	192.168.15.104	sas-remo	78.129.143.173	https	TCP	

Figure 2-5: A snapshot of HTTPS packets captured

2.5. Motivations for Tunnelling:

There are numerous motivations to use tunnelling in a network environment. The technique could be used for mischievous purposes, to bypass a firewall and run restricted applications.

At the network layer, tunnelling provides a solution to use the existing IPv4 routing infrastructure to carry IPv6 traffic, which makes the new IPv6 compatible with the existing IPv4 hosts and routers. This is very helpful in the seamless transitioning to IPv6 of the internet, letting the existing IPv4 to remain functional where it prevails. IPv6 or IPv4 hosts and routers can tunnel IPv6 datagrams over areas of IPv4 routing topology by encapsulating them within IPv4 packets. The entry node of the tunnel

creates an encapsulating IPv4 header and sends the encapsulated packet. The exit node of the tunnel receives the encapsulated packet, removes the IPv4 header, updates the IPv6 header, and processes the received IPv6 packet [34]. This work does not relate to the tunnelling employed at the network layer. The applications which make use of tunnels in malicious ways are at the application layer, and the focus of this work is also application layer tunnels. At the application layer, tunnelling techniques have several motivations which could possibly be illegal or not accord with the network usage policy. These tools can enable a user to perform several of the following functions, depending on the capabilities of the particular tool:

- To bypass the existing firewalls.
- To bypass proxy servers in place for stopping users' access to certain websites.
- To perform anonymous surfing of the Internet (Hide IP while browsing).
- To circumvent traffic shaping that affects users' online gaming experience.
- To prevent snooping on the users Internet activities.
- To chat and use Instant Messengers like MSN, Yahoo messenger etc. in a restrictive environment.
- To communicate using VoIP programs in spite of restrictions.[31]
- To play online games without being known.
- To have unlimited data transfer despite policing.
- To use restricted programs without being monitored by work, school, ISP or government [35].

Certain countries' governments sternly censor the internet, so people are inclined to this kind of firewall/proxy bypassing to have more freedom. Also, some of the tunnelling tool developers claim to believe in the concept of fully open, free and uncensored communications for all [74]. For example, a user who experiences internet applications censorship can create a tunnelled connection to a location which is outside the censorship periphery, and run the applications as if he were

situated in that location. Some services are offered for a monthly fee, others are ad-supported.

Then tunnelling can be used in a positive way as well to provide VPN solutions and secure communications, to provide additional security for financial transactions and to secure internet browsing sessions. For example HTTP-Tunnel Corp, which provide tunnelling services, have agreement to provide Citi Bank with their VCM solution for secure communication between bank employees. Also HTTP-Tunnel Corp. provides Toshiba, Japan a custom secure solution of Remote Anywhere software for communication between the Asian regions to the head office in Minato-ku. [35]

So the usefulness and harmfulness of the tunnelling is relative depending on the context in which it is employed and the environment in which it is used.

2.6. Tunnelling Software:

There are numerous software tools available on the internet which provide tunnelling service for HTTP and SSL or SSH protocols. Usually they allow the client to be freely downloaded, and when installed it can behave as a tunnelling client on the host machine. The tunnelling servers are maintained by the providers, for which a subscription can be purchased. The cost of subscription for a year varies for different types of services and is usually not very high, ranging from $ 60 to a few hundred dollars per user per year. Some providers allow the server software to be downloaded as well, and then the user can have its own server machine to be located outside the intranet. In the list below, some of the tools the author has come across are mentioned. This is by no means an exhaustive list, and there are many other tools available, but mostly they are very similar in their purpose and operation, although the number and type of applications supported and bandwidth provided may vary from one provider to another. Most of these tools also give the user the choice of whether to use HTTP tunnelling or encrypted tunnelling. Some however, like PingFu Iris [31] always provide encrypted tunnelling:

- http-tunnel (www.http-tunnel.com): provides HTTP and HTTPS tunnelling [35].
- Gnu Http tunnel (www.nocrew.org/software/httptunnel.html) provides http tunnelling [19]
- Firedrill (www.fire-drill.com): provides http tunnelling
- Pingfu Iris, Pingfu UDP (www.artofping.com): provides https tunnelling
- Super network tunnel (http://www.networktunnel.net): http tunnelling
- Jhttp tunnel (www.jcraft.com/jhttptunnel): http tunnelling
- Tunnelier (http://www.bitvise.com/tunnelier): ssh tunnelling

- Secure Tunnel (https://secure-tunnel.com/index.cfm): https tunnelling

Figure 2-6: the http-tunnel snapshot [35]

In Figure 2-6 a snapshot of HTTP-tunnel is given which is the popular tunnelling tool.

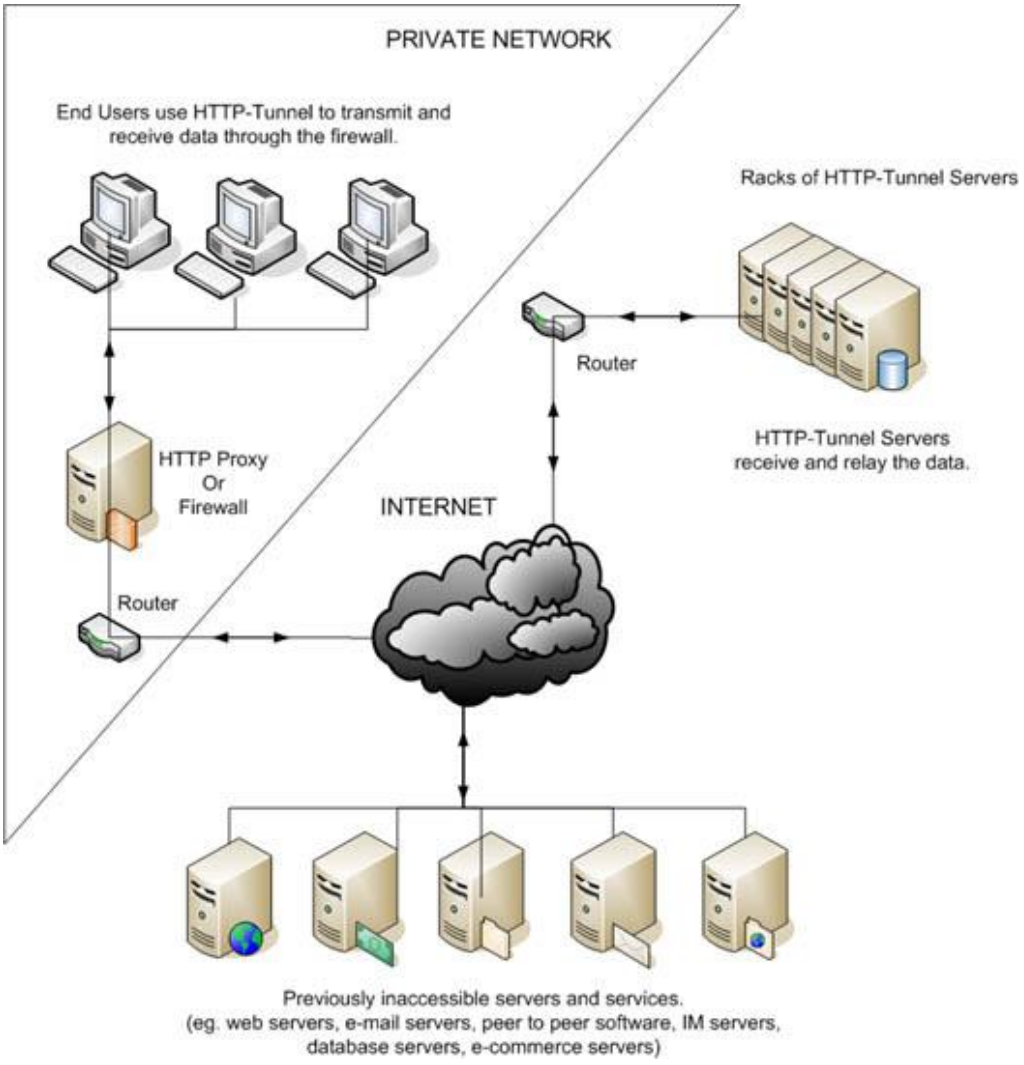


Figure 2-7: The http-tunnel server configuration [35]

Figure 2-7 is a depiction of the configuration used by the http-tunnel team provided by them. They have their own servers which are connected to by the http-tunnel client sitting on host machine. Then the http-tunnel servers communicate with the servers which the host machine wanted to access, but could not do so directly for security or other reasons.

Figure 2-8: Another Tunnelling tool snapshot [36]

The software shown in figure 2-8, i.e. Super Network tunnel, is available in server and client mode as well. The user can start his own server on a machine, and use the client to tunnel data from the host machine.

2.7. Applications being used in Tunnelling:

The following are some of the applications supported by the above mentioned tunneling software. Some of these applications were used in this research as well.

Games

World Of WarCraft, Steam, EveOnline, RuneScape, QuakeLive, Need for Speed online, EverQuest, America's Army, Lord of Ultima etc.

Voip & IM clients

Skype, VoipRaider, Camfrog, Google Talk, MSN messenger, etc.

P2P clients

Azeurus, utorrent, Bitspirit, emule, etc.

2.8. Tunnelled Application Detection: A Review of Previous Work

The most basic approach for application identification is by looking at the port number used by the application packet because some applications use special port numbers e.g. HTTP uses port 80, HTTPS used port 443, FTP uses port 21, 22 is used for SSH etc. The ports 0-1023 are well known ports, and each is used by a known application. However, there are many more applications not using these ports. The registered port numbers could be used for arbitrary applications too. Hence the port number cannot be reliably used for identifying applications, and it is completely misleading in tunnelled applications. When it comes to tunnelled applications, the port number will always show the carrier protocol port number, and not the port number generated by the application inside the carrier. For example if a network game is run inside an HTTP tunnel, all the packets would have the same port number 80 for normal HTTP traffic, and in an HTTPS tunnel all applications would have 443 as the port number. So, the port number would give no clue about the application inside.

Deep Packet Inspection or Deep Packet Analysis is the alternative technique for application identification as explained in the previous chapter. This method attempts to look at the data held in the packets, and usually it is based on searching certain unique identification patterns or signatures held in the data. This technique mostly requires capturing the packets of the session establishment and session initiation phase, because these are the packets which are likely to contain the unique signature pertaining to the application protocol. This technique is also used in Intrusion Detection Systems, for example Snort [37] and Bro [38]. Snort is a free and open source and network intrusion detection (NIDS) which performs intrusion detection by packet logging and real-time traffic analysis on IP networks. Snort performs

protocol analysis, and content searching/matching, it is commonly used to block or detect different types of attacks, such as buffer overflows, stealth port scans, web application attacks, etc. The software is mostly used for intrusion prevention purposes, by dropping attacks as they are taking place. Bro is another open-source, Unix-based network intrusion detection system that monitors network traffic for suspicious activity. The captured packets are searched for unique identifiers and keywords which are kept in a fairly large database. The detector in such a case would fail to detect if it had been initiated after the session has been established, or the network was heavily loaded, with capturing becoming difficult. Deep Packet Analysis or Deep Packet Inspection requires more processing power too as it has to keep up with faster networks and the process itself is very intensive. Then in the case of encryption, this technique cannot work. Hence researchers are more interested in devising methods of application identification which merely rely on the statistical information held about the packets rather than digging the data portion out of the packets. Several attempts have been made using the statistical information of packet flows for detection of applications in general and tunnelled applications specifically too, which are described here briefly.

The work by Daniel J. Pack *et al.* [30] has been one of the foremost in detection of the tunnelling activities. Prior to [30], some of the firewall and Intrusion Detection Systems (IDS) developers were aware of the possibility of illegal HTTP tunnelling, and simple signature matching techniques had been used in some firewalls and IDSes. However these techniques had the same weaknesses as described above already. In [30], a system to detect HTTP tunnelling activities is presented using behaviour profiles based on packet flow directions, packet sizes, large and small packet ratios, average packet size, connection duration, size of data transferred etc. for only three types of sessions, namely interactive tunnelling session, scripted session and stream session behaviour profile. So, this work can't go further than categorizing the session as one of the three sessions. Some of the metrics were found useful by the author, which are also included into this work, along with other metrics. Borders and Prakash *et al.* in [39] have proposed an anomaly detection system that takes advantage of legitimate web request patterns to detect covert communication, backdoors, and spyware activity that is tunnelled through outbound

HTTP connections called “Web taps”. Web Tap uses the features at the HTTP layer, such as HTTP transaction rates, request regularity, bandwidth usage, interrequest delay time, and transaction size, transaction times, etc and can detect spyware and adware programs. However, it can’t deal with encrypted tunnelled traffic.

In [92], Rui Wang *et al.* have proposed an optimized support vector machines model for classification of peer-to-peer traffic at the application level. They have used the transport layer information for four peer to peer applications which are Bittorrent, Kazaa, eDonkey and one peer to peer multimedia application, *pplive*. In [92] it is shown that their proposed method identifies peer-to-peer applications at 75% accuracy at least. However the statistical parameters chosen by Rui Wang *et al.* are only workable for peer-to-peer applications and not general applications. Other than the parameters containing different destinations connected with the same source, bytes/packets is the metric used. This metric alone would not be able to differentiate between other applications, as many applications could possibly have the same average bytes/packets. The case of http or https tunnelling of the peer-to-peer applications is again not considered at all. They have not included udp protocol based peer to peer applications. Although file sharing is not done using udp but multimedia peer-to-peer could use the udp protocol.

The paper [93] describes a performance comparison of five machine learning algorithms for IP traffic flow classification. The algorithms were compared on classification speed rather than classification accuracy. The C4.5 tree algorithm was computationally found to be faster than other four. The feature set chosen in [93] is containing the maximum, minimum, mean and standard deviation of inter-arrival time between packets as well. These features could as well be a function of the congestion state of the network nodes, routers and the distance between the source and destination along with the application producing it; hence they are unreliable in traffic classification.

Moore and Zuev [46] identified 248 flow features and used them in their supervised Naïve Bayes classification algorithm to differentiate between different types of applications. These included packet size, inter- arrival times, some features derived by transforming others, TCP header derived features. Correlation-based feature

analysis was used to find the stronger features which showed that only fewer than 20 features were required for accurate classification.

In [94], Manuel Crotti *et al.* present another mechanism for traffic classification through simple statistical fingerprinting. The features used in their classification scheme are packet size, inter-arrival time and arrival order, as in the beginning stage of each connection; these statistical features depend mostly on the application layer protocol. In that case, it is a requirement to capture the initial few packets of the connection. The features are organized for the protocols in structures called protocol fingerprints and classification is done based on normalized thresholds.

Mena *et al.* [95] was one of the foremost publications in the classification of application-specific traffic and it shows how Real Audio traffic can be detected. Using simple analysis of packet lengths and inter-arrival times their technique is for QoS deployment for audio traffic. Dewes *et al.* [96] uses a similar approach to analyze chat traffic. These works are concerning with a single type of application only.

In [97] Laurent Bernaille *et al.* proposed a simple technique of using the first five packets of a TCP flow to identify the application, and only the size of the first five packets is used as discriminant. Even for the nine application protocols, the results were of 80 % classification accuracy, which is likely to deteriorate more with addition of more applications. Again the tunnelled applications were not included in their experiments, and if used it would only give the protocol of the carrier in classification results. The requirement of capturing exactly the first five packets of a flow is quite stringent as well, and in our detection mechanism any few packets would be sufficient.

Tunnel Hunter [28] is another effort where the authors have used Naïve Bayes classification techniques to identify protocol tunnelling in POP3, CHAT, SMTP protocols mainly using as discriminants packet inter-arrival time and packet size. Their work can tell when these protocols are being used inside tunnels, but gives no idea of what application is run from behind the tunnel. In [40] the same approach as of [28] is applied for encrypted tunnel detection. Using inter-arrival time as a discriminant, as in [28], [41] also looks unconvincing because interarrival time is

very much dependent on the networking infra-structure hardware being used. If routers and other network components have high processing power, the inter-arrival time could be variable. Also network congestion conditions can affect the behaviour of temporal attributes like inter-arrival time. Another issue with these approaches is that they can identify when protocol tunnelling is happening, but not all tunnelling is mischievous. Tunnelled traffic can just as well be benign or legitimate, or it can also be employed to make secure legitimate communications. For example, HTTP-tunnel [35] has deals with Toshiba and Citibank to secure their communications. HTTP tunnelling is also used in network management activities like remote procedure calls encapsulation [42]. So, it is imperative to know what application is being tunnelled as well. HTTP is used in these activities because its packets are not likely to be blocked by a firewall in any network. Thus, these above detection mechanisms would be likely produce a large number of false positives in these legitimate activities, because they are also using tunnelling.

Then there are several studies on analysis of traffic from a statistical point of view as opposed to the techniques used in DPI-based mechanisms. From the studies of Paxson such as [43], researchers have investigated several algorithms based on traditional classification techniques such Neural networks, Nearest Neighbour and Bayesian Machine Learning techniques [44]. In [45] a statistical chi–square-signatures based system is developed for internet traffic classification which is basically targeted at UDP traffic. The work of Andrew Moore is also quite significant for determining discriminants for use in the traffic classification, and some of the discriminants proposed by him have been used in this work as well [46], [51]. In [47] Tom Auld *et al.* have presented their traffic classifier using supervised machine learning based on a Bayesian trained neural network and they have used a large number of features of the flows. The list of statistical approaches to traffic classification is quite long; however it is observed that none of these techniques has yet been tested on tunnelled traffic with the objective of identification of the applications running inside tunnelled data.

The packet size distribution, in most cases can be used as an alternative application signature. The main advantage of this approach over Deep Packet Inspection is that it does not rely on data portion examining of the packets, and it does not require all

packets or some specific packets to be captured. In [48] and [49], the packet size distributions of networked applications, especially with high data rate applications were observed extensively, and they followed signature profile behaviour. Hence this observation led to the detection of applications by storing profiles of known applications in a database, and comparing the captured traffic against these profiles. Simple statistical tests were used to perform the comparison, such as correlation detection; Chi-squared goodness of fit test, or Nearest Neighbour Detection.

In this work the idea of packet size distribution, along with a few other statistical discriminants as an identification signature, is researched for the tunnelled application detection problem. First the usefulness of the packet size distribution for the identification is analysed in Chapter Three. Different trace files of the applications are collected under tunnel operation, and then different trace files are matched statistically for their packet size distributions using Chi squared goodness of fit test. The next chapter explains this in further detail.

Chapter 3 Investigating the Packet Size Distribution for Tunnelled Application Detection

In this chapter the investigation of the packet size distribution for tunnelled application detection and its background is given. Previously extensive research has been made on the use of the packet size distribution for the detection of application using UDP and TCP protocols. Although in this work, the author has not used just the PSD as done by Parish et al. [48] and Li Bo et al. [49], still the packet size distribution is a significant statistical parameter used for the machine learning approach for application detection. Therefore this chapter gives statistical analysis of the suitability of the Packet Size Distribution in application identification using the statistical Chi Squared Test. Several other metrics were used along with the Packet Size Distribution bins which are explained in later chapters.

3.1. Statistical Techniques

Content based detection techniques of Deep Packet Inspection and signature matching clearly have certain advantages but in some instances they will fail in their task as mentioned in the last chapter. In addition to these, statistical methods are available which can prove to be useful. Investigations into statistical techniques have been focussed by several researchers up to now. If statistical properties of the traffic streams are sufficiently unique to a given application, identification can be made with very little processing overhead. One could simply match these parameters (measured for an unknown application) with pre-evaluated samples stored in a database and select the appropriate match. The tunnelled applications would normally run in TCP packets at the transport layer. The statistical parameters can be extracted from the packet stream and saved, and the packets need not be saved [48].

The temporal characteristics that one could extract from the traffic stream are:
Packet inter-arrival time (mean, variance, max, min), packet duration.
Clearly, any temporal characteristic will be affected by the conditions of network congestion. Delay in the network would affect the packet inter-arrival time [52]. Hence such characteristics should be avoided as they would not be consistent for the identification.
The non-temporal statistical characteristic one may extract from the traffic stream are the packet size, ratio of large to small packets, distribution of packet sizes etc. The distribution of packet sizes observed over some time interval could be determined and since it is sufficiently consistent for a given application but different across different applications, one could use this as an application signature [52].

3.1.1 Packet size distribution

The packet size distribution, in most cases can be used as an application signature, and a lot of work has been done regarding this in the last few years [52]. The main advantages of this approach over Deep Packet Inspection is that it does not rely on data portion examining of the packets, and it does not require all packets or some specific packets to be captured. The length of packet header is known in advance and is a standard size. The length of the payload (the message) can be known by checking the transmitted packet sizes or by looking at the packet header field. While network applications are running, the information that they exchange would generally conform to a certain pattern which means the messages or, the length of the packets would also conform to this pattern. It is observed that many networked applications have their own unique packet size distributions. [51] The packet size distributions of most networked applications, especially with high data rate applications were observed extensively, and they followed signature profile behaviour. Hence this observation led to the detection of applications by storing known profiles of known applications in a database, and comparing the captured traffic against these profiles. Simple statistical tests are used to perform the comparison like correlation detection; Chi squared goodness of fit test, or Nearest Neighbour Detection. This method was first used for the detection of UDP based

applications as in [48]. Later Li Bo and David J. Parish et al. worked on utilizing the same technique for TCP based applications as well [49].

In the following sections the suitability of the packet size distribution as an application signature is researched for the tunnelled application detection problem. Different trace files of the applications are collected under tunnel operation, and then different trace files are matched statistically for their packet size distributions using Chi squared goodness of fit test.

3.2 Generating Application Traces

Several network applications' traces were collected for analysis of the Packet size distributions of each application. The applications were running inside tunnelling software. Two tunnelling software applications were used here. An HTTP tunnelling application, called Cubehub HTTP Tunnel [53] and the PingFu tunnelling applications provided by *artofping* [54]. The *artofping* has three tunnelling tools for this purpose. PingFu UDP[54] is Gaming Tunnelling software to bypass Firewalls/Proxy Servers. PingFu UDP has low latency for bandwidth intensive UDP games and voice chat applications. It has support for a plethora of applications, including internet games such as SecondLife, Steam, America's Army, VoIP such as Camfrog, Ventrilo, Skype, MSN/Yahoo/AIM, VoipRaider, LowrateVoIP, etc. Applications can be added to the list by the user as well. The reasons for choosing PingFu tools were their automatic configuration, 32 bit security encryption of all tunneled data and tunnelling of any type of application including those which use TCP & UDP [54]. Pingfu Iris and AutoTunnel GG are the other tools by *artofpoing* [54]. The application to be run through the tunnel is installed on the host machine, and is run from within the tunnelling client, which first connects to the tunnelling servers. The subscription for tunnel servers can be free for limited use or usually purchased for a small fee.

Some popular games and applications were selected for this work in order to demonstrate the efficacy of the Packet Size Distribution as detection metric for these applications. These are World of Warcraft, Runescape, QuakeLive, Camfrog, VOIPRaider, Quake 3 Arena, Lord of Ultima, X-Lite, IVisit, Skype, etc. Some are

these are very popular online games which have a widespread demand to play under restrictive networking environments. For example, RuneScape is a Java-based Massively Multiplayer Online Role-Playing Game (MMORPG) operated by Jagex Ltd. recognized by Guinness World Records as the world's most popular free MMORPG [57]. Similarly having more than 11.5 million monthly subscribers, World of Warcraft is currently the world's largest MMORPG, and holds the Guinness World Record for the most popular MMORPG [58]. In April 2008, World of Warcraft was holding about 62% of the massively multiplayer online game (MMOG) market [59]. Quake Live (formerly known as Quake Zero) is a first-person shooter browser video game developed by id Software, which was first announced in 2007. Other applications include voip client Voipraider [60], and video chat client, Camfrog [61]. Another application Skype [62] is a highly popular software application for making internet telephony, i.e. voice calls using internet protocol. Once a user has got a registered account he can call other account holders free, while calls to the traditional landline telephones and cellular mobile phones can be made for a fee. The volume of international traffic routed via Skype is significant. It has become the largest international voice carrier. [63] Computer-to-computer traffic between Skype users in 2005 was 2.9% of international carrier traffic in 2005 and about 4.4% of the total international traffic of 264 billion minutes in 2006.[64] In 2008, about 8% of cross-border calls were carried by Skype.[63] In 2010, a report by TeleGeography Research stated that Skype-to-Skype calls accounted for 13% of all international call minutes in 2009. Out of the 406 billion international call minutes a total of 54 billion were used by Skype calls.[65] Another application used is X-Lite, a proprietary free VoIP soft phone that uses the Session Initiation Protocol. In 2005, the X-Lite was Internet Telephony magazine's Product of the Year [69].

These applications were run using Cubehub and Pingfu tunnelling software and their packet traces were collected. Several trace files of each application were taken over time. Wireshark [70] was used to capture the packet traces. Wireshark is very similar to *tcpdump*, another famous command line packet capturing utility. The advantage over *tcpdump* is its graphical front-end, and many more information sorting and filtering options. It allows the user to capture all the traffic received by

putting the network interface into promiscuous mode. The format of the Wireshark's network trace file is the same *libpcap* format which is also supported by *libpcap* and *winpcap*, so it can read capture files from applications such as *tcpdump* which also uses the same format, and its captured traffic can be read by applications that use *libpcap* or *winpcap* to read captured files. *Libpcap* is unix's API (application programming interface) for capturing network traffic, and *winpcap* is a port of *libpcap* used by Windows.

The packet size distributions of these traces were generated. It is observed that the packet size distribution profiles of these applications is significantly different from each other and can be used as a discriminant to identify the application. In the figures in the next section, the two graphs give a visual representation of the packet size distribution for the applications whose trace files were collected. While the packet size distribution of the same application is not exactly identical for every new trace file for that application, it is observed that there is similarity within an application, and the graphs are quite different from other applications' graphs, hence the packet size distribution can be used as an identifier statistic.

3.3. Graphs of Packet Size Distributions

Traces were taken from the applications running in protocol tunnels as mentioned earlier. Then the Packet size distributions were obtained from each trace file. Wireshark and Matlab were used for this purpose. Figure 3-1 to Figure 3-11 show the plots of the packet size distributions of the various trace files for the applications chosen. The Resolution i.e. how many bins were used for the graph are mentioned in the figure in a text box, along the total packets count for the graph or else it is mentioned below.

In this work, results are obtained for various resolutions such as 50 bins, 30 bins, 100 bins, 150 bins, etc. for each application. The traces were taken off Ethernet network, and the Maximum Transmission Unit (MTU) for Ethernet is 1500 bytes and Maximum Segment Size, MSS, is 1460 bytes for the Transport Layer. The 40 bytes is the IP header. If number of bins used is 50, then each bin is around 1460/50

or 30 bytes approximately, and similarly for other resolutions. The number of packets falling in a particular bin form the frequency of the that bin. Visually it can be seen that there is a degree of similarity between different tracefiles of the same application. However the height of the histograms may appear different in some cases; which is also dependent on the total number of packets in the tracefile. The total packet count is also mentioned in the graphs.

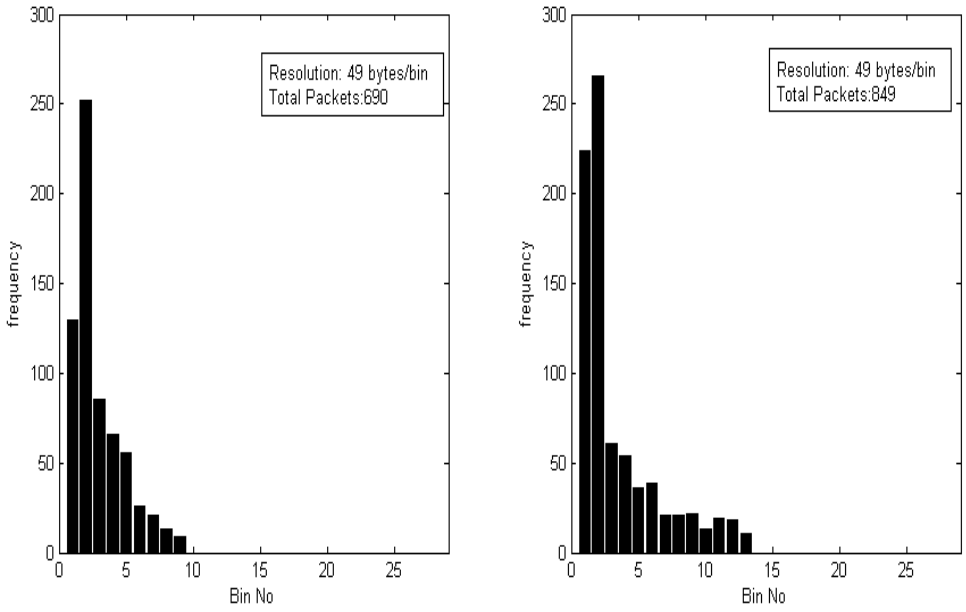


Figure 3-1 Packet size distribution for *world of warcraft* Resolution is 30 bins, bin size = 50 bytes

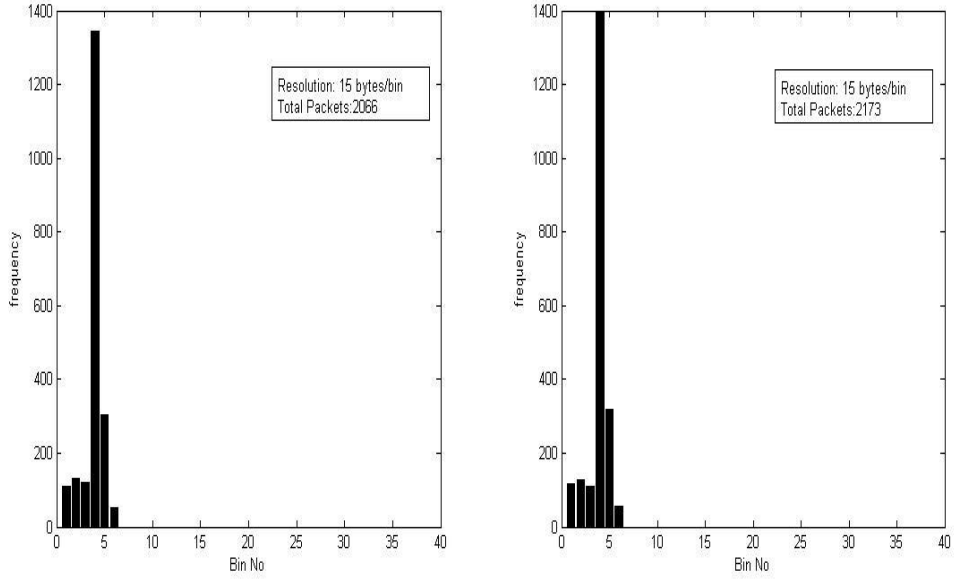


Figure 3-2 Packet Size Distribution for *Runescape*, resolution is 100 bins, bin size is 15 bytes.

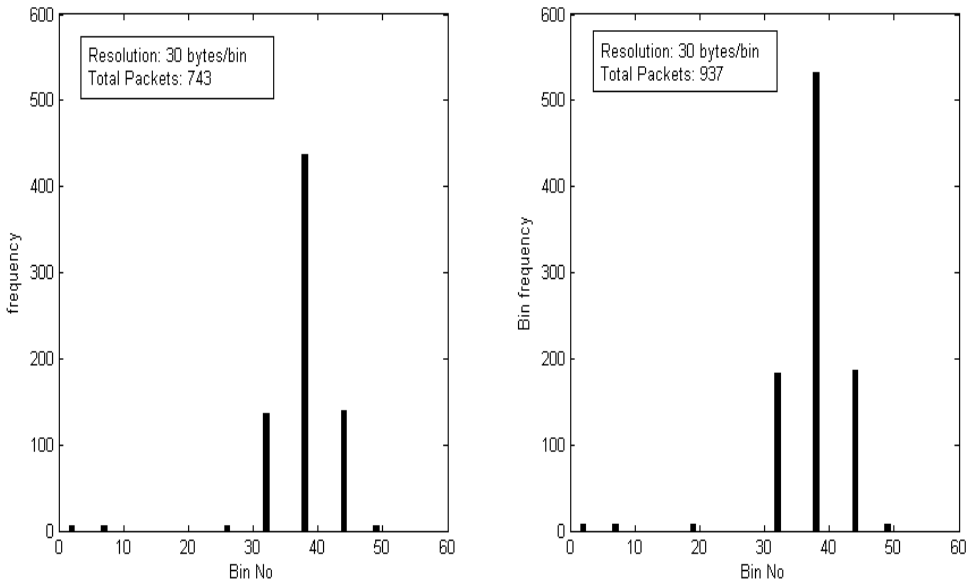


Figure 3-3: Packet size distribution for *Voip Raider*. Resolution is 50 bins, bin size is 30 bytes

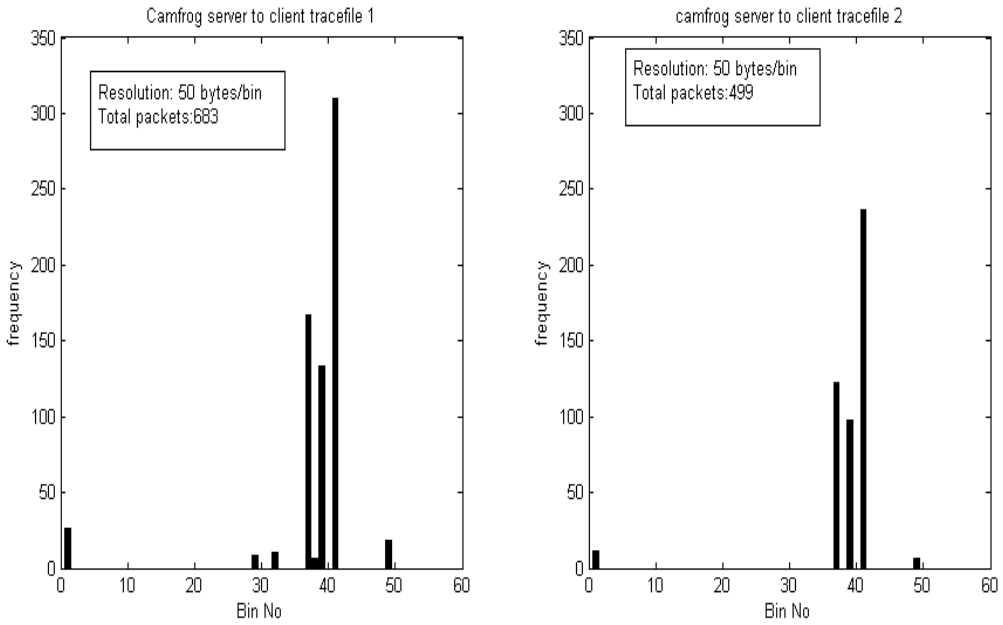


Figure 3-4 Packet Size distribution for *Camfrog*. Resolution is 30 bins, bin size is 50 bytes

Here it is noteworthy that there is visible variation in the amplitudes in some of the lines in the two distributions for some applications. This is mainly because the total number of packets in the sample is different for the two distributions. Otherwise the shapes of the distributions are visibly similar, though they are not exactly identical. If the distribution is normalized, i.e. the each bin frequency is divided by the total number of packets in the sample, and then a better resemblance is obtained in the two distributions. The difference of normalized and not normalized distribution can be detected in the following figures with same samples and resolution for Skype trace files:

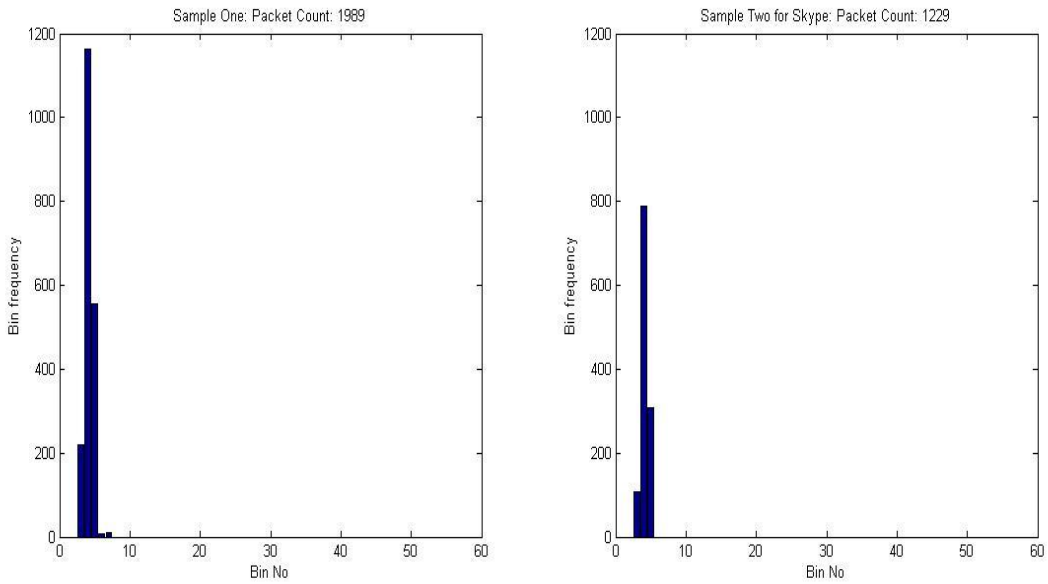


Figure 3-5 *Skype* Packet size distributions without normalization, resolution is 50 bins, bin size is 30 bytes.

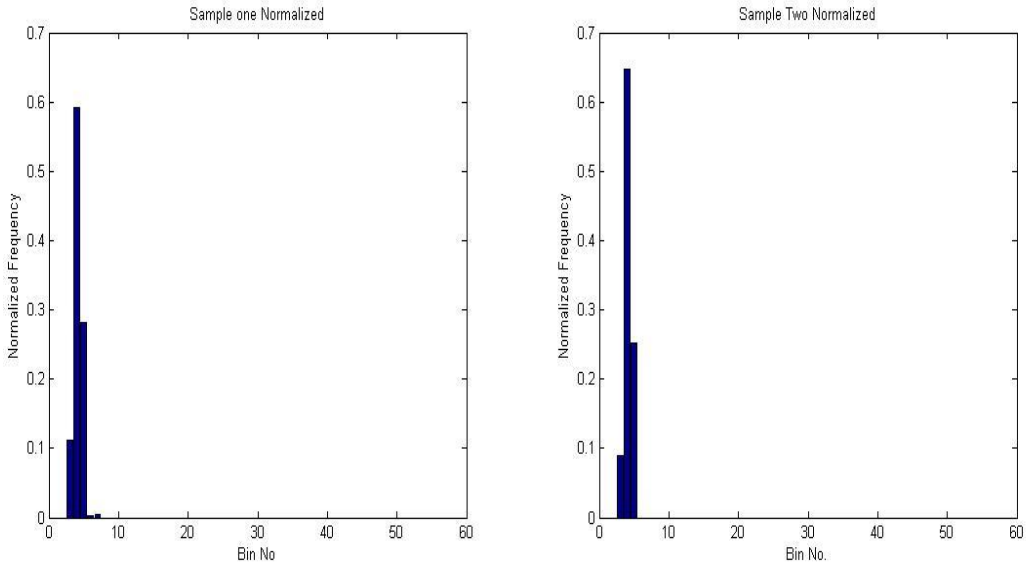


Figure 3-6 *Skype* Packet size Distribution graphs with normalized frequencies, resolution is 50 bins, bin size is 30 bytes

The normalized graphs are better in similarity of the shapes. In the Chi square test for the matching of distributions, the normalized distribution is used because otherwise the distributions cannot be consistent and would depend on the packet count for the sample.

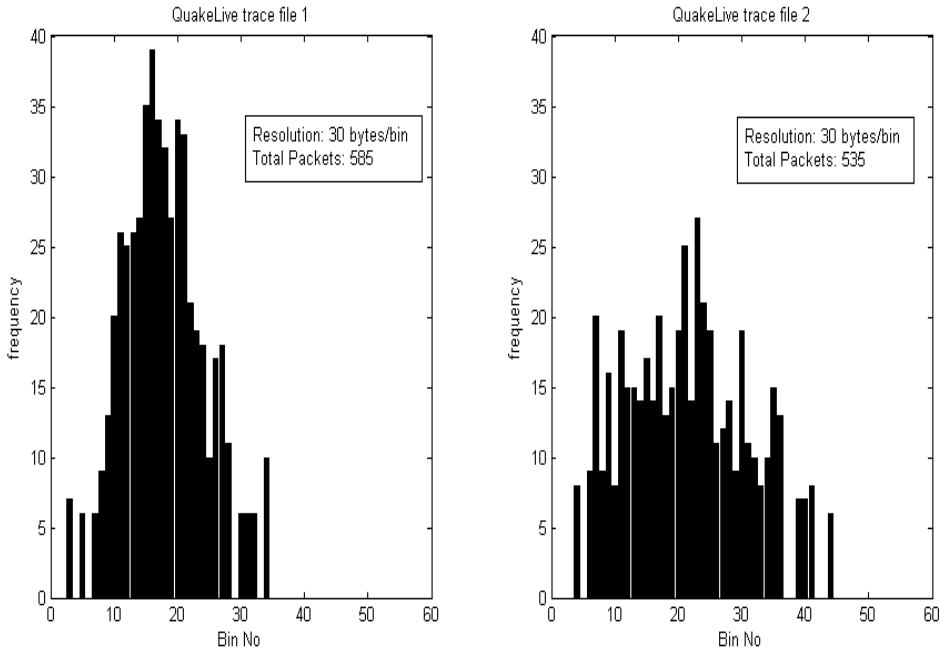


Figure 3-7 Packet Size Distribution for *QuakeLive,* Resolution is 50 bins, bin size is 30 bytes

To notice the effect of varying the resolution for the same application, the following figures 3-8 - 3-11 should be seen which are taken from the same traces of the IVisit videoconferencing software and X-Lite softphone respectively. It is seen that for the same resolution, the Packet size distribution would have similar shapes; however the shape of 50 bins is different from shape of 150 bins resolution. Therefore when matching the trace files statistically, one must be consistent with the resolution of the distribution. If 50 bins is used then it should be 50 bins for all the trace files.

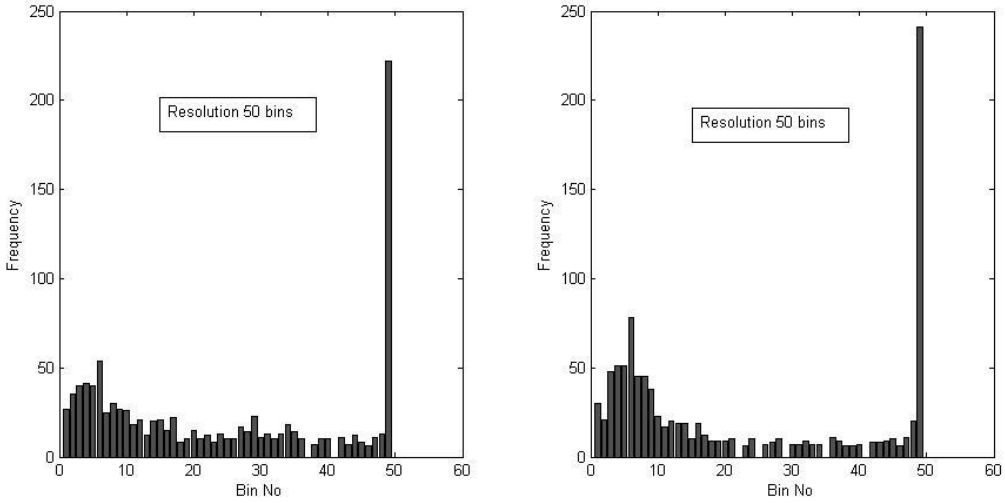


Figure 3-8 *Ivisit* with 50 bins resolution, bin size is 30 bytes

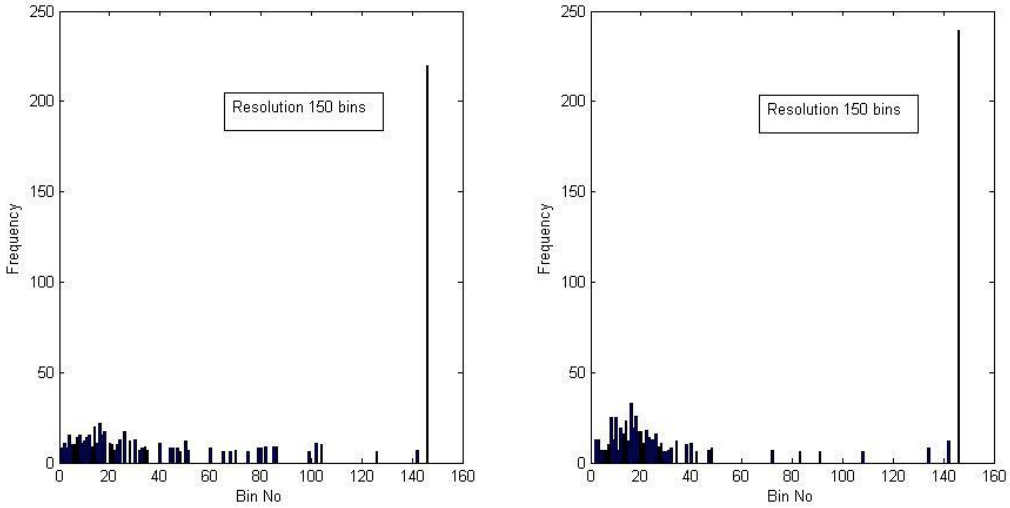


Figure 3-9 *Ivisit* with 150 bins resolution, bin size is 10 bytes

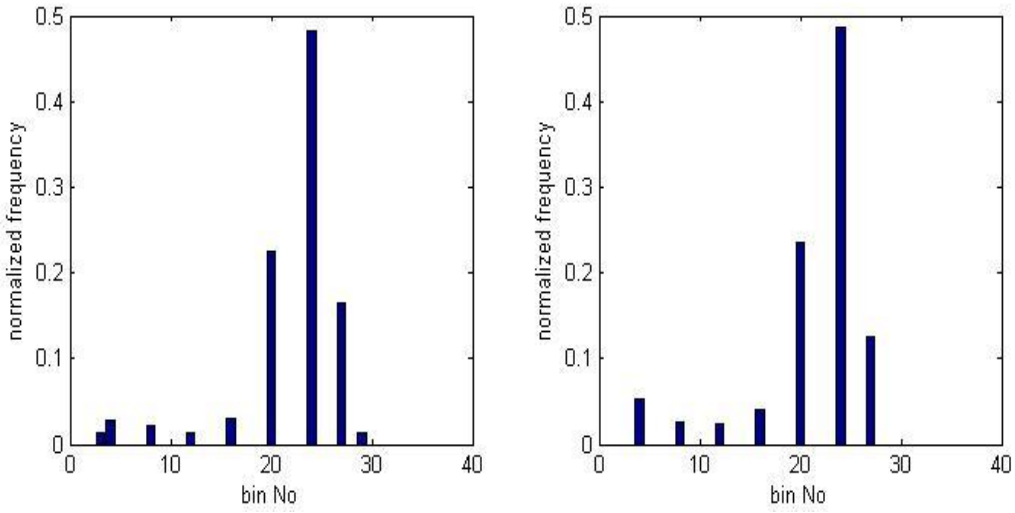


Figure 3-10 *X-Lite* PSD with 30 bins (Normalized Frequency), bin size is 50 bytes

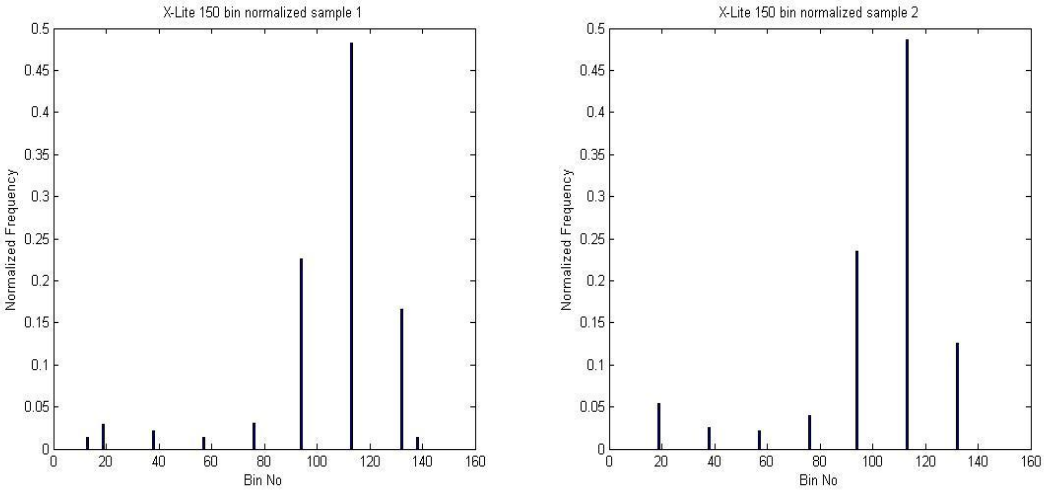


Figure 3-11 *X-Lite* PSD with 150 bins (Normalized Frequency), bin size is 10 bytes

This section has only showed that the shapes of Packet size distributions of different captured trace files of these applications are visually similar within the same application provided the resolution is kept the same for the trace files. At the same time the shapes are very different for the trace files of different applications. To use this fact for the purpose of application identification, a statistical tool is required that can handle this similarity mathematically, which is the statistical Chi squared Test.

3.4. Statistical Chi-Square goodness-of-fit Test

The packet size distribution is a significant metric in this method for tunnelled application identification. First the Packet size distribution metric is investigated using a statistical test, known as Chi square test. Here a comparison is made between packet size distributions of the trace files of tunnelled applications. The statistical Chi square test is employed to effectively measure the goodness of fit of the two distributions. The reason for choosing the Chi-square test over other statistical test methods is that the packet size distribution is a frequency measurement, and the Chi-square test is particularly appropriate with variables expressed as frequencies [71]. In addition, when used for frequency comparisons, the chi-square test is a non-parametric test, since it compares entire distributions rather than parameters (means, variances) of distributions.

The chi-square goodness-of-fit test is defined for the hypothesis:

H0: The observed data follows the expected distribution (obtained from stored trace files of the tunnelled applications)

Ha: The observed data does not follow the distribution specified by the store packet file's distribution.

The data which is in the form of Packet size distribution in this case is divided into k bins and the test statistic is defined as

$$\chi^2 = \sum \frac{(o_j - e_j)^2}{e_j} \qquad (3.1)$$

where o_j is the observed frequency for bin *i* and e_j is the expected frequency for bin j .The chi-square value (χ^2) is an overall measure of discrepancy between the observed frequencies and the expected frequencies under H_0.

This weighted sum of squared differences is equal to 0 if and only if every observed frequency is equal to the corresponding expected frequency under H_0 , that is, if the fit is perfect. If there are large differences between o_j and e_j , then χ^2 will be large, which in turn suggests that the null hypothesis should be rejected [72].

Therefore, the hypothesis that the data are from a population with the specified distribution is rejected if

$$\chi^2 > \chi^2_{(\alpha,k-c)}$$

where $\chi^2_{(\alpha,k-c)}$ is the chi-square percent point function with k - c degrees of freedom and a significance level of α. The $\chi^2_{(\alpha)}$ is the critical value from the chi-square distribution which can be looked up in statistical tables for a given confidence level and degrees of freedom. The chi-square test requires that frequency counts in all categories should be no less than 5 unless the number of categories is very large and only very few categories have frequency counts less than 5. Those categories with small expected frequencies can be combined with neighbour categories to meet this requirement in order to improve the approximation [73].

3.5. Result of Chi Square Tests for PSDs

Next Chi squared analysis of the trace files using Packet size distributions is performed. First the test was performed over trace files of the same application, to find out if they are conforming with good confidence. Then the Chi square test was done for traces of one application with other applications to find out that they are differentiated from one another.

Figure 3-13 is a graph for the observed and critical chi squared values for a bin size of 30 bytes of the trace files of the applications, which means that packet sizes were categorized by 30 bytes intervals. These are the results of the goodness of fit test between two profiles (distributions) of packet sizes of same application but from different trace files. The plot shows that for all the applications under observation, the Chi values were less than the critical values at 95%; hence the applications can

be statistically identified using their packet size distribution profiles using a simple statistic such as chi squared. A similar plot is given for different bin sizes of 50 in Figure 3-14, 100 bins in Figure 3-15, and 150 bins in Figure 3-16. Varying the bin size does not affect the plots too much and the notion holds in all plots that the chi values are well less than critical values. The results corroborate the idea that the packet size distributions of different traces of same application are similar enough that they can be put under same category by the chi squared test.

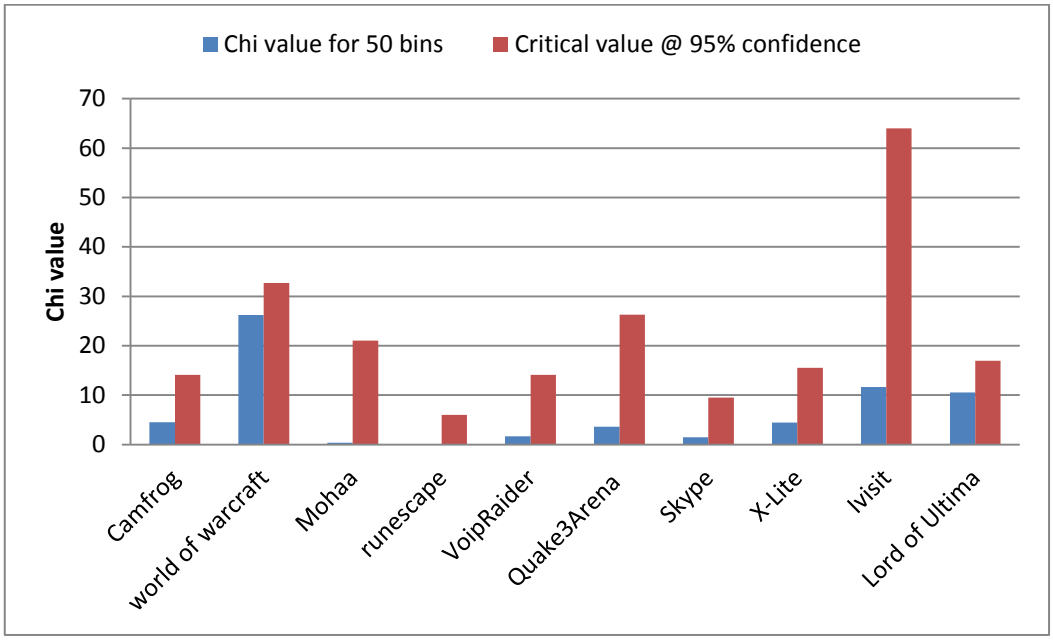


Figure 3-12 Result of chi squared goodness-of-fit (gof) test for traces of same applications with resolution of 50 bins and bin size = 30 bytes

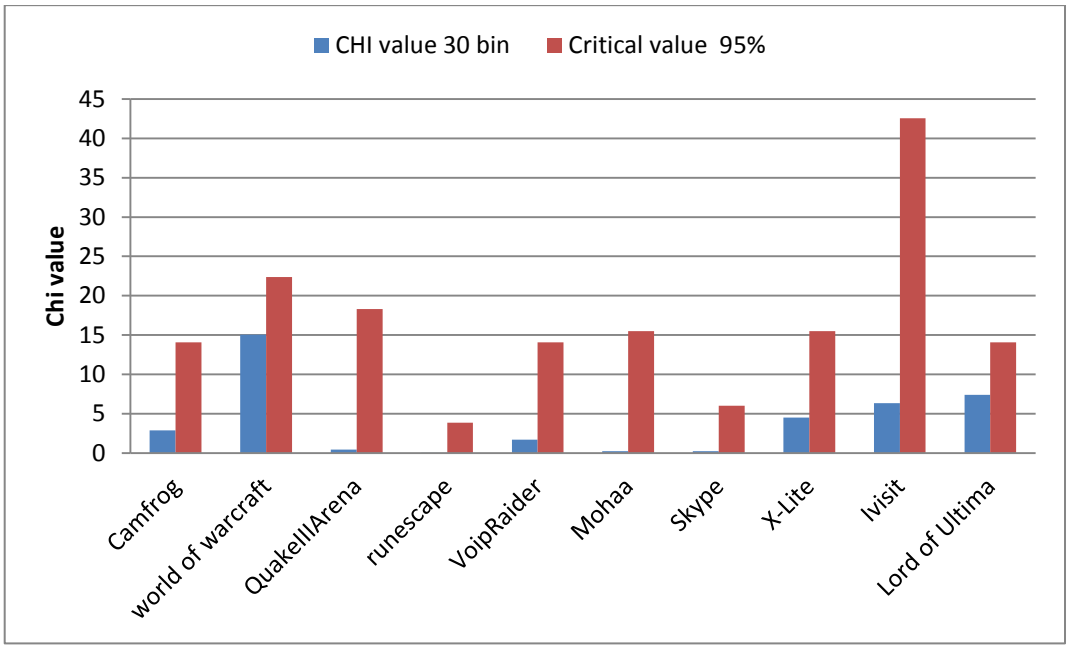


Figure 3-13 Result of chi squared gof test for traces of same applications with resolution of 30bins, and bin size of 50 bytes

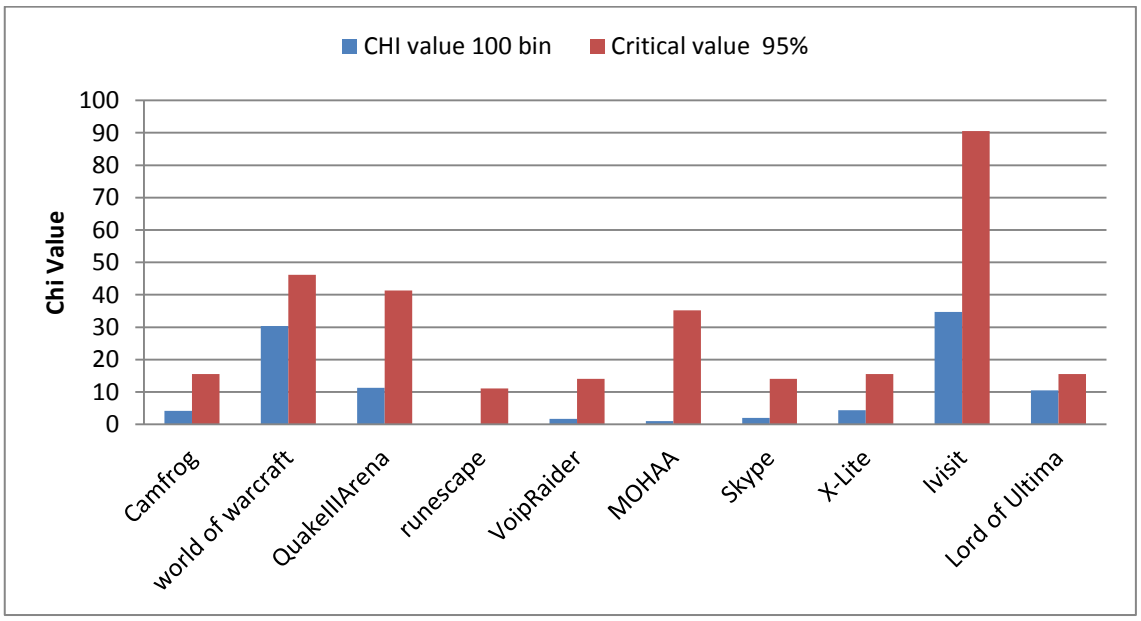


Figure 3-14 Result of chi squared gof test for traces of same applications with bin resolution = 100 bins, and bin size is 15 bytes

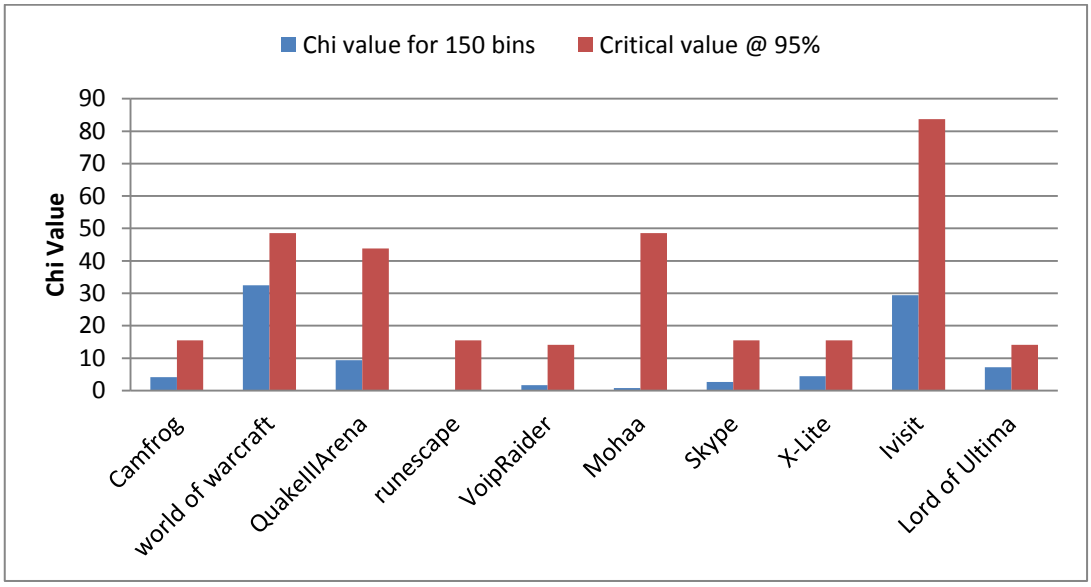


Figure 3-15. Result of chi squared gof test between tracefiles of same applications with bin resolution = 150 bins, and bin size is 10 bytes

Figure 3-17 i, ii, iii are given for comparison of the chi squared results when performed for two different applications' trace files. The names of the applications are abbreviated in the lower part of the figure. Here the results are quite opposite to the previous plots. In these graphs, the observed Chi values are much higher than the critical values. These graphs are given to analyse the possibility of false positives or erroneous application detection. The results show that when performed across different application traces, the chi squared test does not place them in one category and tells them apart, by giving a higher Chi value than critical values. Hence there is not much chance of confusion or mismatch in the tunnel application detection based on packet size distribution. It is observed that the Chi square goodness of fit test does not render the two distributions of different application trace files as one application. These results are summarised in the form of Table 3.1 below:

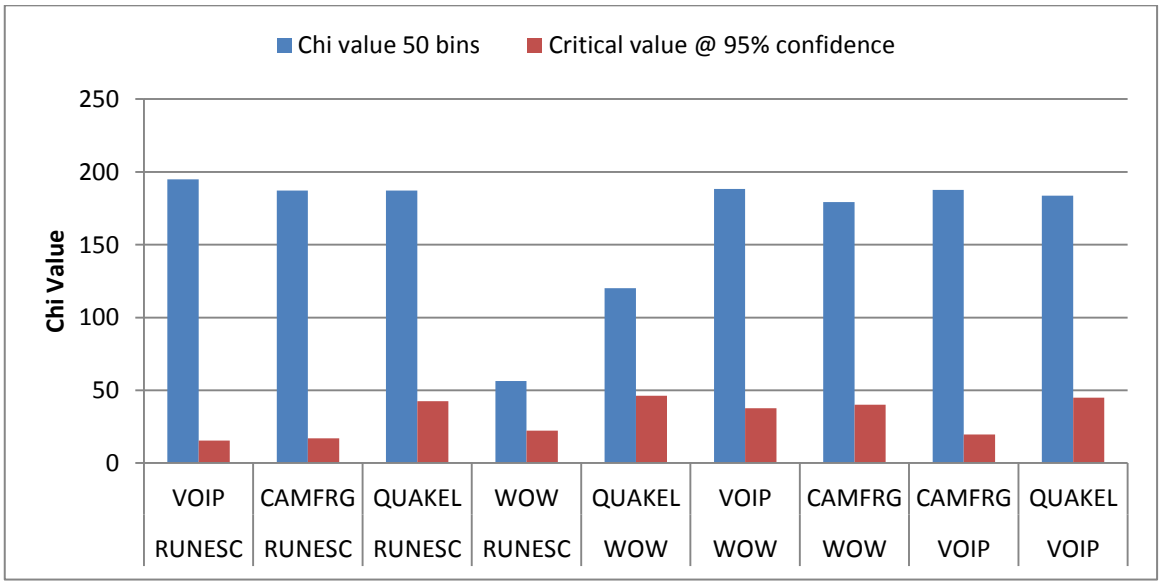


Figure 3-16 i : part 1 of Chi squared gof test values for comparison of traces of two different applications. Resolution is 50 bins, bin size is 30 bytes

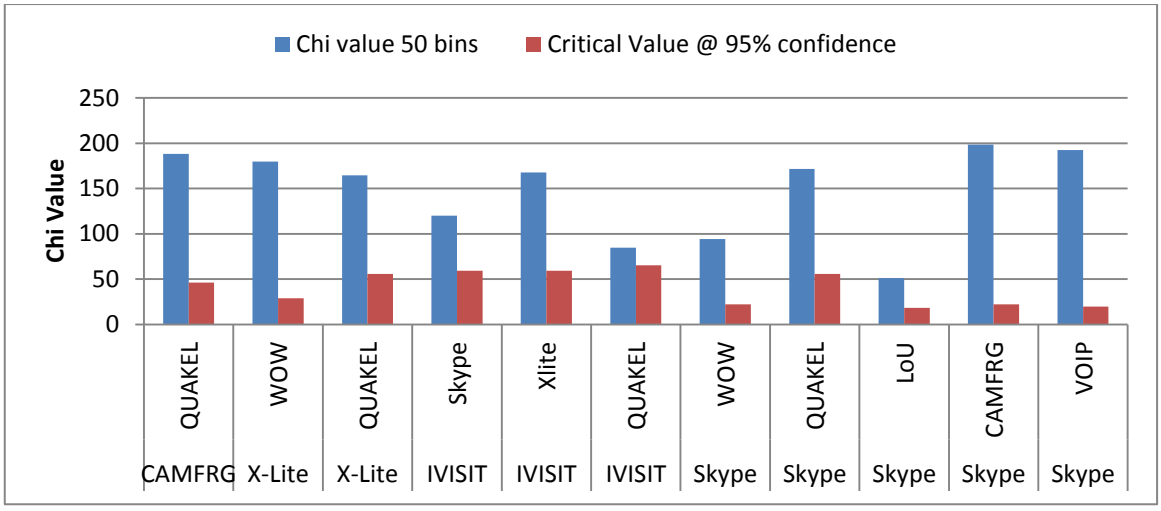


Figure 3-17 ii: part 2 of Chi squared gof test values for comparison of traces of two different applications. Resolution is 50 bins, bin size is 30 bytes

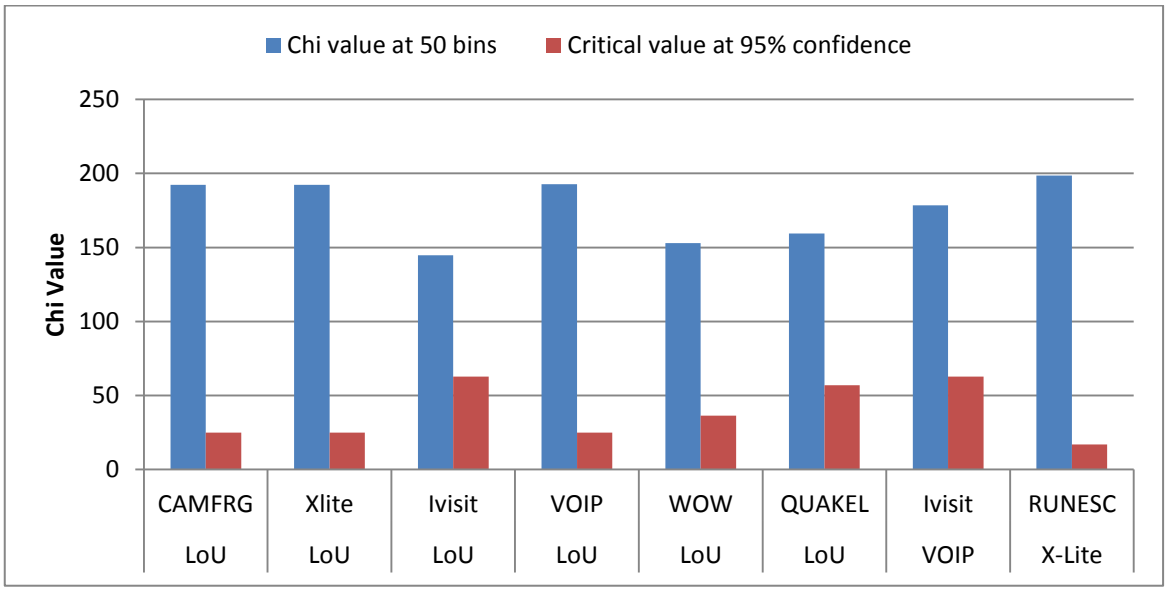


Figure 3-17 iii: part 3 of Chi squared gof test values for comparison of traces of two different applications. Resolution is 50 bins, bin size is 30 bytes

Table 3.1 Results of Chi-squared tests between different application traces

Application	Chi-Value	Degrees of	Critical value	Critical value	Next cloest	Chi-Value	Critical value
Camfrog	4.5	7.0	14.1	6.3	world of warcraft	179.3	40.1
world of warcraft	26.2	21.0	32.7	20.3	Runesc	56.4	22.4
runescape	0.0	2.0	6.0	1.4	WoW	56.4	22.4
VoipRaider	1.7	7.0	14.1	6.3	QuakeL	183.7	45.0
QuakeLive	32.0	37.0	52.2	36.3	IVisit	84.9	65.2
Skype	1.5	4.0	9.5	3.4	LoU	51.1	18.3
X-Lite	4.4	8.0	15.5	7.3	QuakeL	164.5	55.8
Ivisit	11.6	47.0	64.0	46.3	QuakeL	84.9	65.2
Lord of Ultima	10.5	9.0	16.9	8.3	Skype	51.1	18.3

Table 3.1 shows a general Chi-square summary for all application that had been tested. For each application, the Chi-squared value resulting from the computation with the pre-stored trace for that application is shown, along with the corresponding 95% and 50% confidence value (which varied according to number of degrees of freedom). The lowest Chi-squared value resulting from the computation with a different application trace is also given, again with the corresponding 50% confidence value. As the computation ignores packet size probabilities of zero in both data sets, the confidence value varies from application to application.

The table shows that for each of the applications, the Chi squared test resulted in lowest chi-squared value when the test was performed on the traces of the same application. In all of the cases, this Chi-square value is below the critical value for 95% confidence value. In all cases, the next lowest Chi-squared value from a different application is seen to be significantly greater that the first, and much

greater than the associated 50% confidence value for this second choice application. Hence the second choice application is not to be predicted in favour of the first.

Thus the proposed idea of tunnelled application identification works successfully for all practical purposes for the both the encrypted and plain tunnelled applications studied here. Theoretically it cannot be claimed that two applications would not have the same packet size distribution, in tunnel or without tunnel operation. Given the vast number of applications, this claim would be impossible to prove empirically. However for a large number of applications the approach works successfully. Since this is a relatively much simple computation, Packet Size Distribution based detection can be used as an additional security tool alongside the existing tools and practices.

In this chapter, the consistency of the packet size distributions of the tunnelled applications was tested. Different applications from games, real-time audio, video, voice over IP, etc. were selected and it is experimentally established that the packet size distributions of these applications even inside tunnel have a consistently unique form which can be statistically recorded using the Chi square goodness of fit test. This fact will be utilized in the next chapters where the parameters or metrics which are used in the identity of the application are explored. The packet size distribution forms a significant parameter, although some other statistics are used as well to further improve the detection mechanism.

Chapter 4 Methodology for Tunnelled Application Detection

This chapter gives a description of the methodology employed to achieve identification of applications. Application traces were collected using network sniffing tools. The major flows were identified from the network trace file and each flow was treated as a connection. Following the machine learning paradigm, the methodology developed has two stages, a learning phase, and a classification phase. The objective of the learning phase is to find out a relationship in terms of the statistical parameters of the traffic flows/connections, including packet size distributions and other parameters for the applications under consideration, and saving the information in a usable form. The classification phase uses the learned parameters to find the actual application from the statistical parameters of unknown flows. Several machine learning algorithms are considered and their performance is compared in Chapter 5.

The methodology employed in this work for identification of applications running inside protocol tunnels is based on analysis of application trace files from a statistical perspective. From the trace file the individual connections are identified. A connection is a tuple of Source Address, Destination Address, Source Port, and Destination Port and for a particular protocol. Since all application layer tunnelling protocols are TCP based, so the TCP connections are considered only. For each connection, the traffic flow parameters are extracted. There are two key components in the process: a learner and a classifier. The learner arrives at a relationship between the connection parameters and the applications from the training data set. Subsequently this learned mapping is used by the classifier for classification of the connection. This method is also called Supervised Learning because the training data is fully labelled. The steps involved in the process are described below and also summarised in Figure 4-1 in the form of a flowchart.

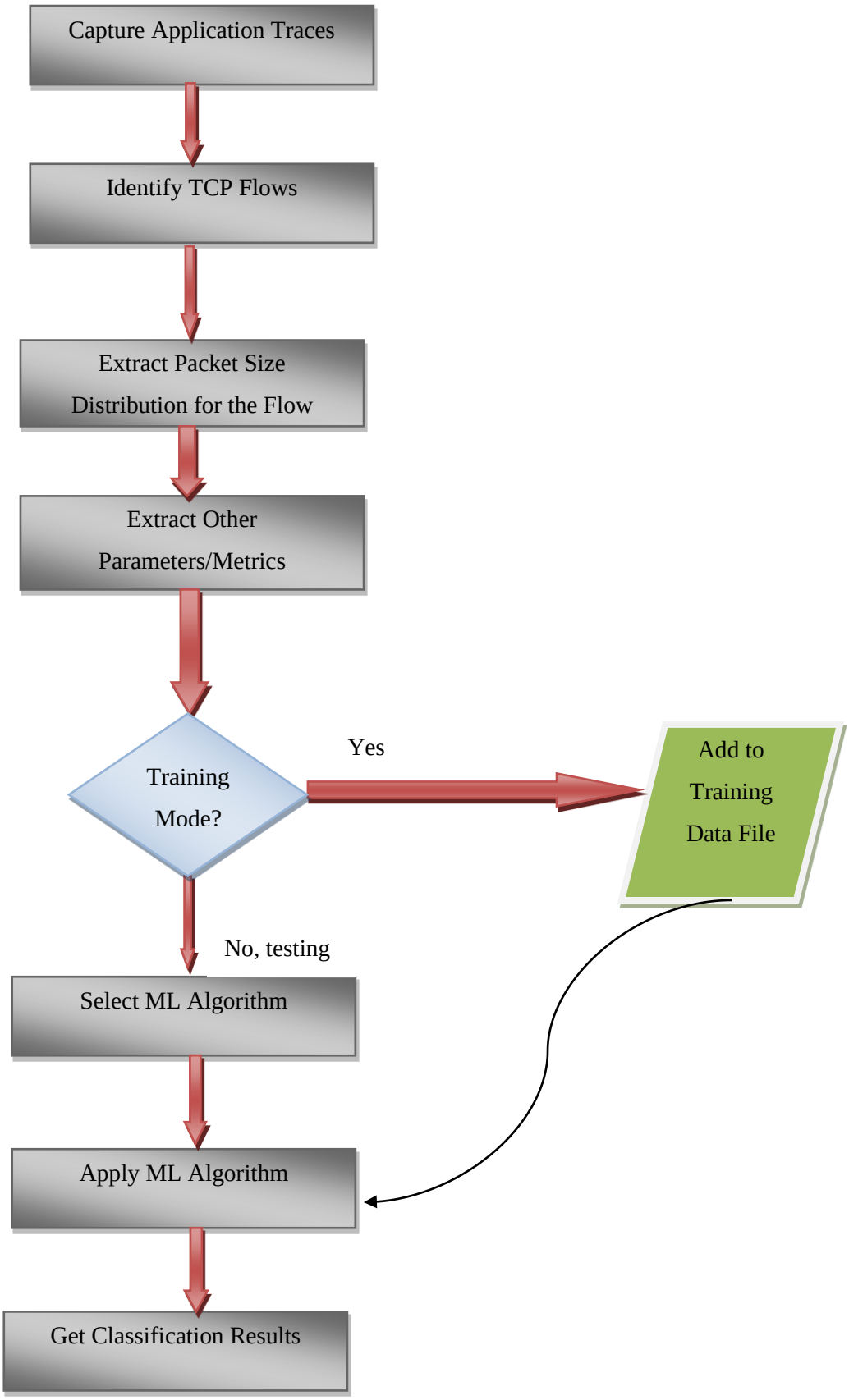


Figure 4-1 Flowchart of the Tunnelled Application Identification methodology

4.1 Capturing Pcap Traces of Tunnelled Applications:

The network applications were run under tunnelling software. HTTP tunnelling software, called Cubehub HTTP Tunnel [53], pingfu UDP [54] (which provided HTTPS encrypted tunnell for UDP based applications like VoipRaider, Skype etc.) and Pingfu Iris [54] which is also HTTPS based tunnelling software for TCP based applications such as World of Warcraft, etc. were used. The packet traces were captured for the applications for several minutes of run time per application. These traces were processed to produce the data sets used for experimentation.

Previous research shows that for approximately 30 seconds of application run, the packets should be captured for each instance so that stable and consistent statistical parameters including Packet size distribution could be obtained [52]. For this purpose, the trace files were captured using the Wireshark network sniffing and analysing tool [76].

The size of each tracefile captured was of the order of a few Megabytes, and it contained ample packets for statistical analysis. Each file was saved in pcap format which is the default format for Wireshark.

4.2 Processing of the CapturedTrace Files:

The captured traffic was analyzed by a command line tool called *tshark* [77]. From the captured file, any connections with very few packets were filtered out and discarded, because packet size distributions using these connections would not be reliable. *TShark* is a network protocol analyzer. It can capture live network traffic as well as read and analyse a stored trace file. *TShark*'s trace files are of *libpcap* format. Tshark works very much like *tcpdump* and unlike Wireshark, does not have a Graphical User Interface. Having captured the application trace files and saving in the pcap format, the next step was to extract the parameters of interest from the file. A Matlab program was written to store the connections of each application and their statistical parameters. The applications running across the network were supervised at the capture time, i.e. only a single known application was run at a time, hence from the processed trace file the large connections were assigned to that application.

Matlab cannot directly read from the *pcap* format, hence *tshark* was used to convert the trace files into a format readable by the analyser program.

The command that was used in this work is given below. This command serves the purpose of reading a libpcap capture file and extract it to a text file with the same name and obtaining the required information from each packet. Each packet forms one line of the text file obtained. For each packet the following information is obtained: Time, Source IP address, Destination IP address, source port number, destination port number, length of the datagram, Protocol of the packet. These are the destination IP and port addresses assigned by the tunnelling software to the connections, not the actual IP addresses the application would be communicating to if it were run without tunnelling. Then this detail is written to a text file for each packet, so that the text file can be read for further processing:

```
tshark -o column.format:"No.","%m","Time","%t","Source","%s",
"Destination","%d","srcport","%uS","dstport","%uD","len","%L",
"Protocol","%p" -r fname > fname.txt
```

The various options are explained below:

-o <preference> :<value>

This option is used to set a custom preference value, instead of the default value or any value read from a preference file. Here a format is overridden which contains the columns of interest. The format used in the above command is displaying for each packet: "No" which is an index number of the packet, "Time", which is the time of packet capture, "Source", is the source IP address, "Destination", the destination IP address, "srcport", is the port number used by source machine, "dstport" is the port number of the destination machine, "len", is the length of the packet in bytes, and "Protocol", is the protocol used.

-r <file>

Read traffic data from *file*, which is a *pcap* file in any of the supported capture file formats. The *fname* in the above command is the name of the *pcap* file which contains the captured data.

>fname.txt

This is a redirection of stdout to a text file so that the output is displayed in a text file. The format of the output is defined previously. That text file is then read into the detection tool to be analyzed, and statistics are extracted from it for each connection.

In the following figure 4-2, a glimpse of a portion of the generated text file is given. Each packet's information forms one line of the file. On the basis of the same protocol, source IP address, destination IP address, source port and destination port the packets are assigned to connections. The entries of the file follow this order: index, time, source IP, destination IP, source port, destination port, size of the packet, protocol.

```
 1   0.000000 158.125.48.160 -> 72.13.82.175 51354 443 906 TCP
 2   0.273670 72.13.82.175 -> 158.125.48.160 443 51354 60 TCP
 3   0.427982 72.13.82.175 -> 158.125.48.160 443 51354 542 SSL
 4   0.626643 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
 5   1.272770 158.125.48.160 -> 72.13.82.175 51354 443 675 TCP
 6   1.505681 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
 7   1.505687 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
 8   1.505691 72.13.82.175 -> 158.125.48.160 443 51354 453 SSL
 9   1.505812 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
10   4.665125 158.125.48.160 -> 72.13.82.175 51354 443 1039 TCP
11   4.898743 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
12   4.898749 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
13   4.898753 72.13.82.175 -> 158.125.48.160 443 51354 1390 SSL
14   4.898880 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
15   4.899112 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
16   4.899118 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
17   4.899121 72.13.82.175 -> 158.125.48.160 443 51354 98 SSL
18   4.899124 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
19   4.899193 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
20   4.899496 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
21   4.899500 72.13.82.175 -> 158.125.48.160 443 51354 289 SSL
22   4.899547 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
23   6.456550 158.125.48.160 -> 72.13.82.175 51354 443 1090 TCP
24   6.690975 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
25   6.690983 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
26   6.691136 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
27   6.691284 72.13.82.175 -> 158.125.48.160 443 51354 1390 SSL
28   6.691292 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
29   6.691299 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
30   6.691392 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
31   6.691739 72.13.82.175 -> 158.125.48.160 443 51354 1390 SSL
32   6.691745 72.13.82.175 -> 158.125.48.160 443 51354 142 SSL
33   6.691804 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
34   6.845735 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
35   6.845741 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
36   6.845830 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
37   6.846086 72.13.82.175 -> 158.125.48.160 443 51354 1390 SSL
38   6.846091 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
39   6.846096 72.13.82.175 -> 158.125.48.160 443 51354 1434 SSL
40   6.846176 158.125.48.160 -> 72.13.82.175 51354 443 54 TCP
41   6.846468 72.13.82.175 -> 158.125.48.160 443 51354 1390 SSL
```

Figure 4-2 the text file containing packet information

4.3 Extracting Metrics from Captured Traffic:

The Matlab program (code is given in the appendix A, "The Matlab code") would read the text file corresponding to each pcap trace file. Every line contained the information from one packet. Depending on the source address, destination address, source port, destination port and protocol, the packets were assigned into connections. A connection is a tuple of these five parameters. For each connection the required metrics or parameters were extracted. The metrics obtained for the connections of each tunnelled application trace file are described below. The Packet Size Distribution bins are derived as described in Chapter 3, based on the work done by Li Bo in [51] about identification of TCP based applications using packet size distribution alone and earlier by Ketan Bharadia in [52]. The other metrics were chosen from [80], [79] where they were used for the classification of internet traffic because of their generality and availability in all packets. Some time based metrics are also included, such as maximum interarrival time in one direction to investigate how they affect. These metrics are described below:

1. **Data rate (bytes/sec) for upstream direction:**

   This metric is a measure of the data transfer rate in the direction from local machine to remote machine. This is calculated by dividing the total bytes transferred by the total time taken during the transmission of first till the last packet in the given TCP connection of the captured file [80].

2. **Data rate (bytes/sec) for downstream direction:**

   The same metric is also obtained for the remote machine to local machine direction. These two parameters would give an insight into the nature of the application from the perspective of data transfer rates. Some applications are quite consistent in these rates while some are not [80].

3. **Data packet ratio: dpktr:**

   This is a ratio of the data packet number downstream to upstream. By data packets are meant the packets which contain some tcp payload data apart from the header. The data packet ratio is calculating by taking the ratio of data carrying packets from the remote to local computer direction and the

data carrying packets in the local computer to remote computer direction [79].

4. **ByteRatio:**
   This is another related metric which is a ratio of the total bytes transferred in the downstream (remote to local) to upstream (local to remote) direction [79].

5. **Ratio of large and small packets:**
   In a given direction, this is a measure of relatively large packets to small packets. The threshold is arbitrarily set to 300 bytes. If a packet is greater in payload size than 300 bytes it would be counted as a large packet, otherwise a small one. Then the ratio is obtained of the two counts [80].

6. **maxIATds:**
   This is the maximum inter-arrival time between two packets in the direction of the remote to local machine.

7. **MinIATds:**
   This is minimum inter-arrival time between two packets in the downstream direction, i.e. remote to local.

8. **maxIATups:**
   The maximum interarrival time in packets from the local machine to remote machine direction.

9. **minIATups:**
   The minimum packet interarrival time recorded for the upstream direction, i.e. local to remote.

10. **time spent idle downstream:**

The idle time is defined as the collection of time periods of 2s or greater duration in which there was no packet sent downstream. It is given as a percent of the total time so that the value is normalized for various length packets.

11. **time spend idle upstream:**

    Similar metric for the upstream direction.

12. **PSD with 15 bins**

    The next metrics are the bins of the packet size distribution. The number of bins is set to 15 for the initial experiments. However this is increased to 30 bins or more as will be seen in later sections of the next chapter. For the sake of initial experiment, this is kept at 15 bins so that the effectiveness of the other metrics along with the packet size distribution can be learned and there is a broad base of statistical metrics to choose from, although for some applications even Packet Size Distribution bins alone are sufficient to detect the application correctly, as is shown in the next chapter.

4.4 Training or Learning Phase:

The training data is saved in the form of an Excel file of comma-separated values (.csv) format. This format was preferred because it is a widely acceptable format for many machine learning applications. Each row of the Excel file corresponds to the data procured from one pcap file for each application. Table 4.1 shows a part of the training data. It contains the columns for all attributes extracted. The attribute name is abbreviated on the top of each column. The table 4.1 doesn't contain all of the attributes, in order to keep it to a manageable size. The Packet Size Distribution bins are b1, b2…. For initial experimental data set there are 15 bins of Packet size Distributions. The last column is the abbreviated name of the application corresponding to the data. If more applications were added, they can be easily appended to the end of the file. The file was automatically generated by the Matlab program used to extract the attributes from the packet captured trace files. Also, for

the 10 applications there are 12 instances of trace files for each, so this gives 12 rows corresponding to each application. Hence there are 120 instances in the database for the experimentation, but more instances can be added whenever more data is available.

Table 4.1 A portion of the training data file

drl2r	drr2l	dpktratio	byteratio	ratio_sl_ds	b1	b2	b3	b4	…	b15	app
59.87	494.72	1.24	8.26	0.1	0.77	0.1	0.03	0.01	…	0.01	WW
102.06	525.15	1.7	5.11	0.03	0.8	0.14	0.02	0	…	0	WW
116.66	526.09	1.47	4.52	0.04	0.8	0.13	0.02	0.01	…	0	WW
115.33	483.71	1.6	4.19	0.03	0.82	0.12	0.03	0.01	…	0	WW
66.21	368.71	1.86	5.57	0.09	0.74	0.12	0.06	0.02	…	0	WW
113.61	314.52	1.09	2.77	0.05	0.81	0.09	0.05	0.02	…	0.01	WW
106.63	38.6	0.87	0.36	0	0.96	0.03	0	0	…	0	WW
69.88	733.3	2.45	10.49	0.1	0.61	0.19	0.1	0.04	…	0.01	WW
108.26	381.16	1.1	3.52	0.11	0.8	0.05	0.04	0.02	…	0.02	WW
84.1	655.95	1.54	7.81	0.21	0.69	0.08	0.06	0.04	…	0.01	WW
108.54	3032.44	1	27.88	7.4	0.02	0.03	0.07	0.13	…	0.01	QL
110.2	3448.79	0.99	31.37	5.05	0.01	0.06	0.08	0.11	…	0.01	QL
89.21	18.27	0.05	0.2	0.01	0.01	0.02	0.01	0	…	0	QL
88.12	2.2	0.02	0.02	0	0.02	0	0	0	…	0	QL
109.31	3235.48	0.99	29.62	6.04	0.01	0.04	0.07	0.12	…	0.01	QL
109.95	3355.86	0.99	30.52	7.06	0	0.04	0.07	0.14	…	0	QL
120.53	1238.47	0.84	10.28	0.91	0.16	0.08	0.12	0.22	…	0	QL
88.85	0.78	0.01	0.01	0	0.01	0	0	0	…	0	QL
88.77	0	0	0	0	0	0	0	0	…	0	QL
88.28	0	0	0	0	0	0	0	0	…	0	QL
107.7	657.43	0.54	6.11	0.52	0.09	0.04	0.07	0.14	…	0	QL
1496.71	8958.23	1.05	5.98	22.04	0.02	0.02	0	0.01	…	0.02	VR
1574.65	9414.7	1.02	5.98	60.92	0.01	0.01	0	0	…	0.01	VR
1570.4	27.16	0.54	0.02	0	1	0	0	0	…	0	VR
1571.51	9412.8	1.02	5.98	65.93	0.01	0.01	0	0	…	0.01	VR
1577.67	26.68	0.53	0.02	0	1	0	0	0	…	0	VR
1577.83	26.97	0.54	0.02	0	1	0	0	0	…	0	VR
1569.74	26.08	0.52	0.02	0	1	0	0	0	…	0	VR
1552.76	26.71	0.54	0.02	0	1	0	0	0	…	0	VR

4.5 Applying Machine Learning Algorithms

Now, the machine learning algorithms can be applied to the data to obtain the results for the prediction accuracy of the algorithms. The data set is used to train the classification algorithm and then the trained algorithm is able to predict the application for a case when the application name is not given. The results of applying these algorithms are discussed in further detail in the next chapter. Some of

the metrics discussed previously might not be as useful as others; hence the performance of various combinations of the metrics was investigated as well.

Chapter 5 Application Predictions using Machine Learning

This chapter contains the results of the experiments performed over the network application trace data captured for tunnelled applications. The previous chapter outlined the process of collecting applications' trace files and converting these into a more useable format and extracting metrics of interest from them. Here the results of Machine Learning experiments performed on different combinations of these metrics are given and a few conclusions are drawn about the most useful metric combinations.

Some of these metrics might not be as useful as others; hence the difference in the performance of the various combinations of the metrics was investigated.

From the last chapter, a database was obtained which contained the metrics for each application extracted from its trace file captured in tunnelling mode. In this chapter, it is empirically demonstrated that this data can be utilized for the identification of the same applications using machine learning classification algorithms or classifiers. Various classifiers were used at this stage and their performance was analysed. WEKA (Waikato Environment for Knowledge Analysis) [81] machine learning software is used because it provides many different algorithms for data mining and machine learning which are easily useable by people who are not data mining specialists, besides being an open source and freely available software [82]. The validation of WEKA results is demonstrated in section 5.8. The classifiers used in the experiments are: naiveBayes, Nearest Neigbour (IB1) using Euclidean distance, C4.5 decision Tree, Neural Network (Multilayer Perceptron), K* Nearest Neighbour, OneR classifier. These classifiers were taken simply to be representative of each group/class of supervised classifiers in the WEKA software, because WEKA includes more than 50 different classifiers grouped into 6 classes of supervised classifiers. Since their performance proved satisfactory for the requirements of this work, investigating the use of other classifiers was left for future work. Also different combinations of the 27 metrics were tested, so that the

relative importance of various attributes, such as packet size distribution, can be determined.

The Weka machine learning software was applied on various data sets, and the results are given in this chapter for those sets. The data sets were obtained from the procedure explained in Chapter 4. The test sets are a section of the complete data sets with all instances labelled. In Machine Learning literature a labelled instance is one whose class is known. Hence when Weka's machine learning algorithm classifies an instance as a particular application, that instance can be matched against its actual class label to see if the result of weka is valid or not. The Weka software calculates the prediction accuracy of each algorithm by comparing the results for the test instances by Weka's algorithms with the actual class labels of the same instances. In section 5.8, the results of the Weka software are validated using two means: First by comparing the classes identified by the Weka with the actual class of the same test instance, and secondly by comparing the Weka results of a machine learning algorithm with the results of a similar algorithm implemented in Matlab. This comparison has been performed for one of the experiments in this chapter, described in section 5.8.

5.1 Using 27 metrics for Identification

In this case, all the 27 statistical metrics (which were described in Chapter 4) extracted from the tracefiles were used in the detection process. The dataset consists of 120 instances. The detection predictions are also referred to as classifications hereafter. The performance of the classifier is naturally measured in terms of its error rate. The classifier predicts the class of each instance, which is actually the name of an application from the training data: if it is correctly predicted, this is a success; otherwise an error. The error rate is the ratio of errors made in a whole set of instances, and it measures the performance of the classifier. However, the desirable feature is the performance on fresh data, not on the already seen data. The class of the instances in the training set is already known, so the real performance is that on test data. In the last section, the results from fresh data are also given. Three different testing schemes are used for each algorithm.

1. **Testing on the training set:** In this mode, the classifier model is built from the training data set, and the same training data is used to test the performance of the classifier. This evaluation is very optimistic and will certainly not be the same for unseen data. However, it still is useful in a sense that it generally gives an upper bound to the classifier's performance on fresh instances of data.

2. **Testing on split data (1/3 for test, 2/3 for training):** In this mode, 66% of the data is used to train the classifier and build the model, whereas one third (33%) is held for testing of the classifier. The test data is not used in the training, so it is fresh or unseen data with respect to the classifier. The disadvantage is that this reduces the amount of data to be used for training as ideally it is desired that in a limited dataset most of it to be used for training. Also if one is unlucky, the sample used for training or testing might not be representative. So there is a dilemma here, to find a good classifier, it is desirable to use as much of training data as possible; and to have a good error rate estimate of the classifier, as much of data as possible is required for testing.

3. **10 fold stratified cross validation:** A more general and better way to mitigate any bias caused by the particular sample (from training data) chosen for holdout is to repeat the whole process, training and testing, several times with different random samples from the data set. According to Witten and Frank in [83], the standard way of predicting the error rate of a learning technique given a single, fixed sample of data is to use stratified 10-fold cross-validation. In this method, the available data set is divided randomly into 10 sections such that the application classes have similar representation in each section as in full dataset. In first run, one of the 10 parts is used for testing, and the remaining 9 parts are used for training. Then similarly in 10 runs, all 10 parts are respectively used for testing set and other 9 used for training, and its error rate is calculated on the test set. Thus the learning procedure is executed a total of 10 times on different training sets (each of which have a lot in common). Finally, the 10 error estimates are averaged to

give an overall error estimate. This procedure is called a stratified 10 fold cross-validation, which has become the standard method in practical terms. [83]

The detailed results for each of these methods for the algorithms of NaiveBayes, C4.5, Multilayer Perceptron, IB1, IB K*, OneR are presented here.

5.1.1 Classification with Naïve Bayes Algorithm:

NaiveBayes in Weka implements the probabilistic Naïve Bayes classifier. A Bayes classifier combines prior knowledge with observed data to assign a posterior probability to a class based on its prior probability and its likelihood given in the training data. A Naive Bayes classifier assumes conditional independence between attributes and assigns the maximum aposterior probability (MAP) class to new instances.

Results: (a) Testing on training Data: The output from running the naiveBayes classifier for detection of the applications is given below. Here all 120 instances of data were used for training the naiveBayes classifier to build a model, and the same dataset was used for testing it. The results are given:

=== Summary ===

Correctly Classified Instances	115	95.8333 %
Incorrectly Classified Instances	5	4.1667 %
Total Number of Instances	120	

Table 5.1 Confusion Matrix for NaiveBayes classifier on training data

a	b	c	d	e	f	g	h	i	j	classified as
12	0	0	0	0	0	0	0	0	0	a = WW
0	11	0	0	0	1	0	0	0	0	b = QL
0	0	12	0	0	0	0	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
4	0	0	0	0	8	0	0	0	0	f = QA
0	0	0	0	0	0	12	0	0	0	g = XL
0	0	0	0	0	0	0	12	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	0	0	0	0	12	j = Sk

The confusion matrix is a simple way of summarizing and displaying the results of the experiment.

The columns of the matrix represent the predictions made by the algorithm, and the rows represent the actual class of those instances. Here from the first row it can be observed that 12 instances were correctly predicted as WW (World of Warcraft). Such cases are also called “True Positives”. The matrix also shows that 11 instances were correctly predicted as QL (QuakeLive). The correctly predicted results will always show on the diagonal of the matrix.

On the other hand, in row 6 of QA (Quake3Arena) it shows that 4 instances were predicted as WW (World of Warcraft) when they were in fact QA. These cases are also referred to as “False Positives” or incorrectly classified instances. The prediction errors can also be visualized in the following error plot:

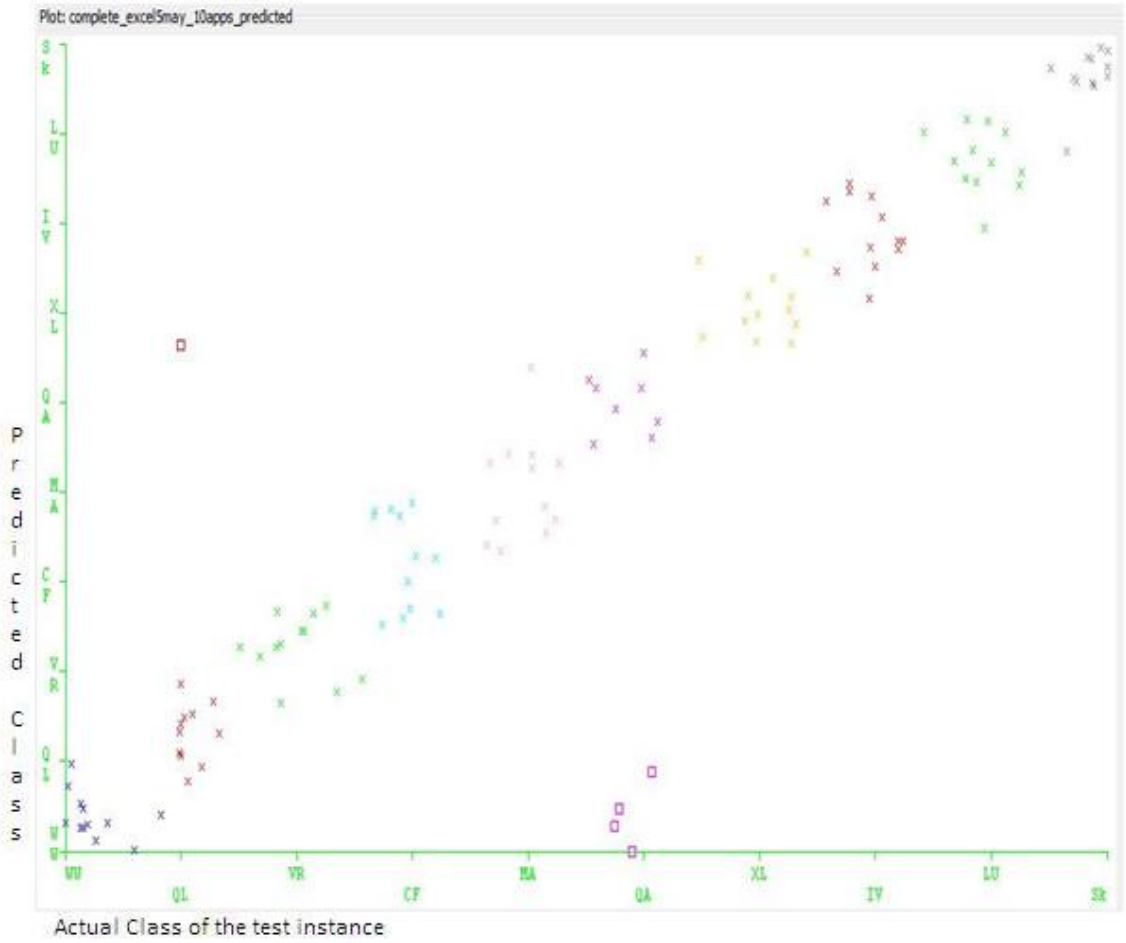


Figure 5-1Plot visualizing erroneous predictions as boxes ‘□’ and correct ones as ‘×’ for the Naïve Bayes testing on training data.

Results (b): Using 10 fold cross validation.

=== Summary ===

Correctly Classified Instances 111 92.5 %

Incorrectly Classified Instances 9 7.5 %

Total Number of Instances 120

Table 5.2 Confusion Matrix for naiveBayes on 10 fold cross validation

a	b	c	d	e	f	g	h	i	j	classified as
11	0	0	0	0	1	0	0	0	0	a = WW
1	10	0	0	0	1	0	0	0	0	b = QL
0	0	11	0	0	0	1	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
3	0	0	0	0	8	0	0	0	1	f = QA
0	0	0	0	0	0	12	0	0	0	g = XL
0	0	0	0	0	0	0	12	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	1	0	0	0	11	j = Sk

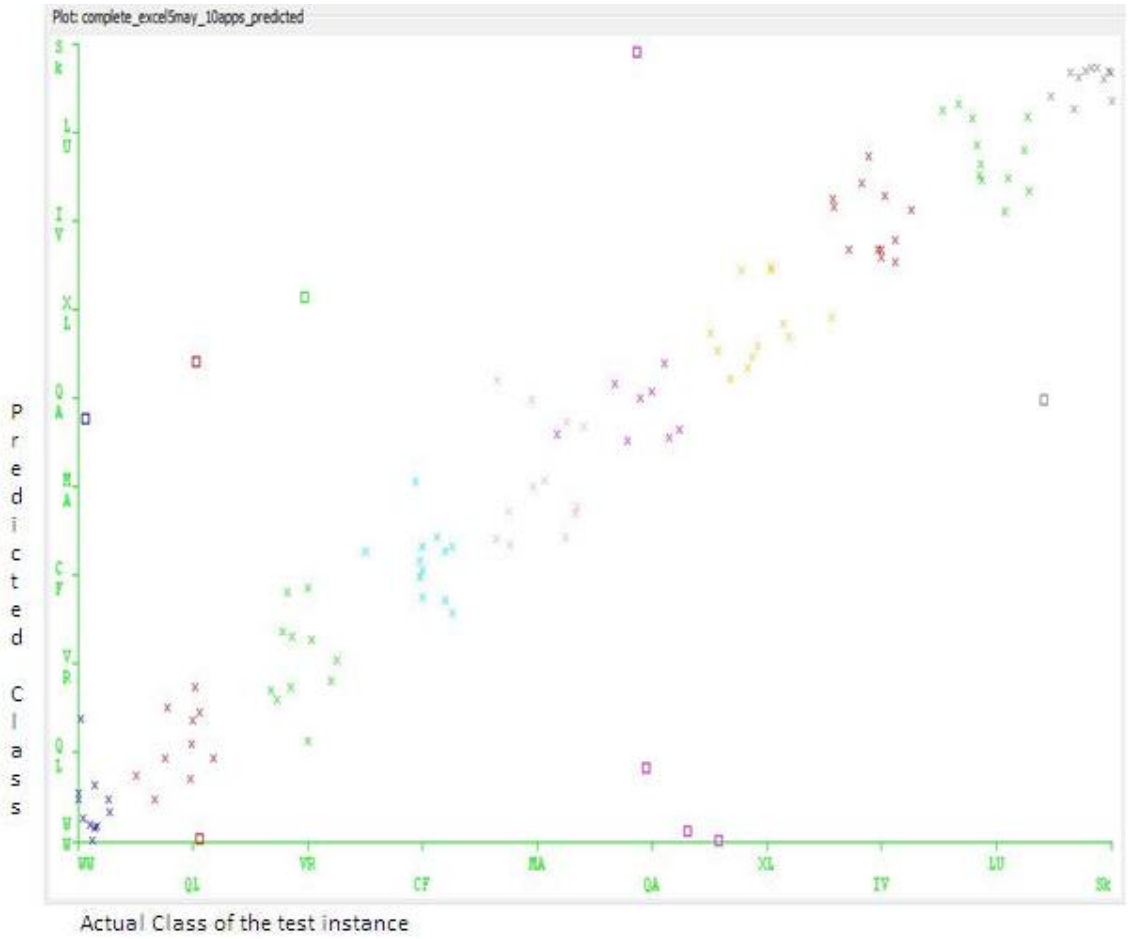


Figure 5-2 Plot visualizing erroneous predictions as boxes '□' and correct ones as '×' for the Naïve Bayes testing on 10 fold cross validation.

Results C : 66 % split training set:

=== Summary ===

Correctly Classified Instances	36	87.8049 %
Incorrectly Classified Instances	5	12.1951 %
Total Number of Instances	41	

Table 5.3 Confusion Matrix for naiveBayes on 33% split test set

a	b	c	d	e	f	g	h	I	j	classified as
5	0	0	0	0	0	0	0	0	0	a = WW
0	1	0	0	0	0	0	0	0	0	b = QL
0	0	3	0	0	0	0	0	0	0	c = VR
0	0	0	8	0	0	0	0	0	0	d = CF
0	0	0	0	4	0	0	0	0	0	e = MA
4	0	0	0	0	4	0	0	0	0	f = QA
0	0	0	0	0	0	2	0	0	0	g = XL
0	0	0	0	0	0	0	3	0	0	h = IV
0	0	0	0	0	0	0	0	5	0	i = LU
0	0	0	0	0	1	0	0	0	1	j = Sk

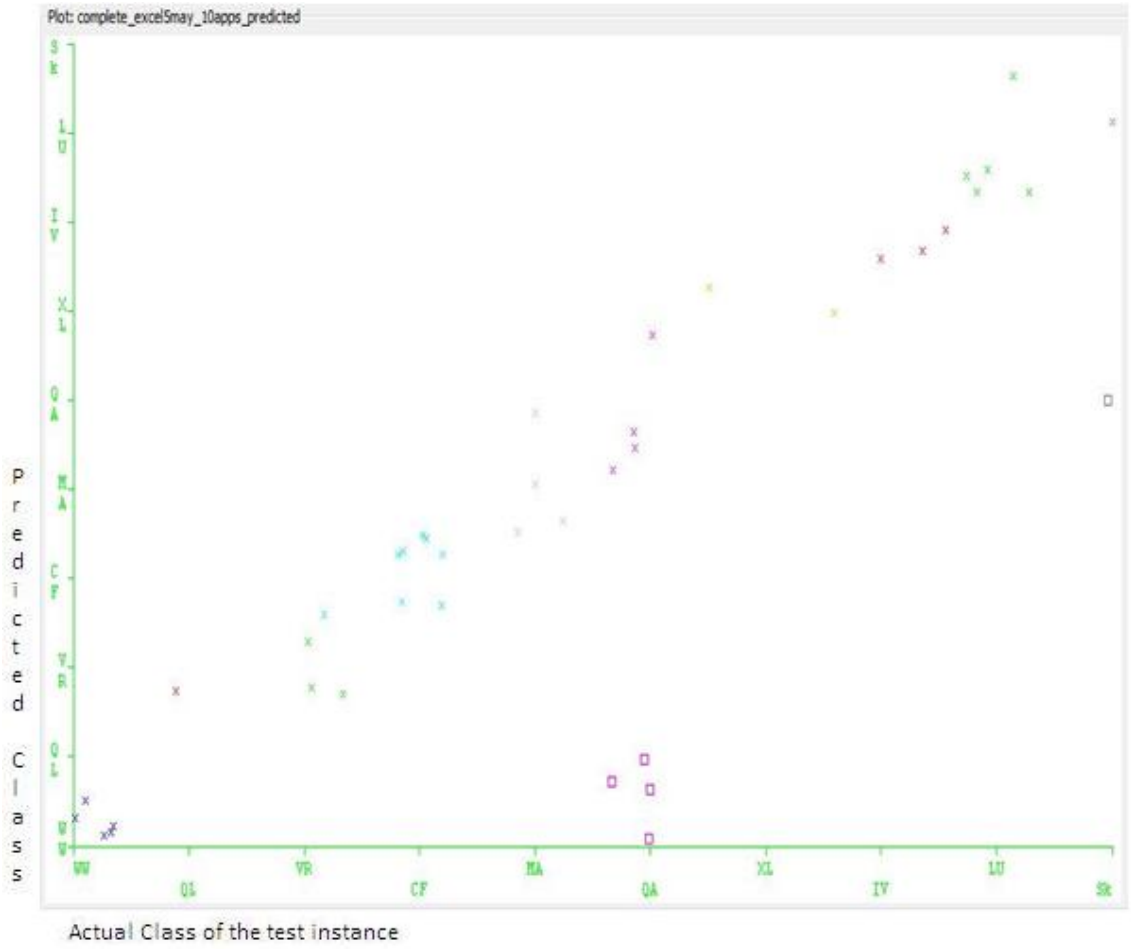


Figure 5-3 Plot visualizing erroneous predictions as boxes '□' and correct ones as '×'using Naïve Bayes with 66% training, 33% test data.

The summary of the Naïve Bayes classifications for the three test cases can be given as:

Table 5.4 summary of the Naïve Bayes classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Naïve Bayes	Training data	120	120	0.01s	115 95.8%)	5 (4.2%)
Naïve Bayes	10 fold cross	120	120	0.01s	111(92.5 %)	9 (7.5%)
Naïve Bayes	66 % split	41	79	0.01s	36 (87.8%)	5 (12.2%)

From Table 5.4, it is observed that reducing the training instances in the 66% split has affected the performance of the classifier in terms of its accuracy. The upper

bound for this classifier has an accuracy of 96%, whereas the 10 fold cross validation estimate, which is the most realistic one, has an accuracy of 92.5%. Now, the results of a few other classification schemes are presented in the following sections.

For the next classifier schemes, i.e. C4.5 Decision tree, Multilayer Perceptron, Nearest Neighbour only the 10 fold cross-validation results will be presented here. The other two cases will be only summarised in the table of results.

5.1.2 Results of C 4.5 Decision Tree Classifier:

The C4.5 classifier is under the trees category of the WEKA classifiers. The C 4.5 Decision Tree was developed by Ross Quinlan [84].

This is a divide and conquer algorithm like the other tree based algorithms. The data is partitioned recursively until every leaf has only the instances of one class or until further partitioning is impossible, when two cases have same values for each attribute but their class is different. Hence, if there are no conflicting cases, the decision tree will be able to classify every training instance correctly, which is "over-fitting", which generally leads to loss of prediction accuracy in most applications. [85] The over fitting problem is overcome usually by removing some of the structure of the decision tree after it has been produced, also known as pruning or sometimes by a stopping condition that prevents some cases from being subdivided. "After a decision tree is produced by the divide and conquer algorithm, C4.5 prunes it in a single bottom-up pass." [86] Here the Weka implementation of C4.5 algorithm, also known as J48 decision tree is used for this data set. The result is given below:

Results: Running J48 on 10 fold cross validation:

=== Summary ===

Correctly Classified Instances 109 90.8333 %

Incorrectly Classified Instances 11 9.1667 %

Table 5.5 Confusion Matrix for J48 classifier on 10 fold cross validation test

a	b	c	D	e	f	g	h	I	j	classified as
11	0	0	0	0	1	0	0	0	0	a = WW
1	10	0	0	0	1	0	0	0	0	b = QL
0	0	11	0	0	0	1	0	0	0	c = VR
0	0	0	11	0	0	0	1	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
0	1	0	0	0	9	0	0	0	2	f = QA
0	0	0	0	1	0	10	0	1	0	g = XL
0	0	0	1	0	0	0	11	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	0	0	0	0	12	j = Sk

The summary of the j48 classifications for the three test cases can be given as:

Table 5.6 Summary of j48 Decision Tree classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Decision tree	Training data	120	120	0.03s	116(96.6 %)	4(3.3 %)
Decision tree	10 fold cross	120	120	0.15s	109(90.8 %)	11(9.2%)
Decision tree	66 % split	41	79	0.02s	38(92.6 %)	3(7.3 %)

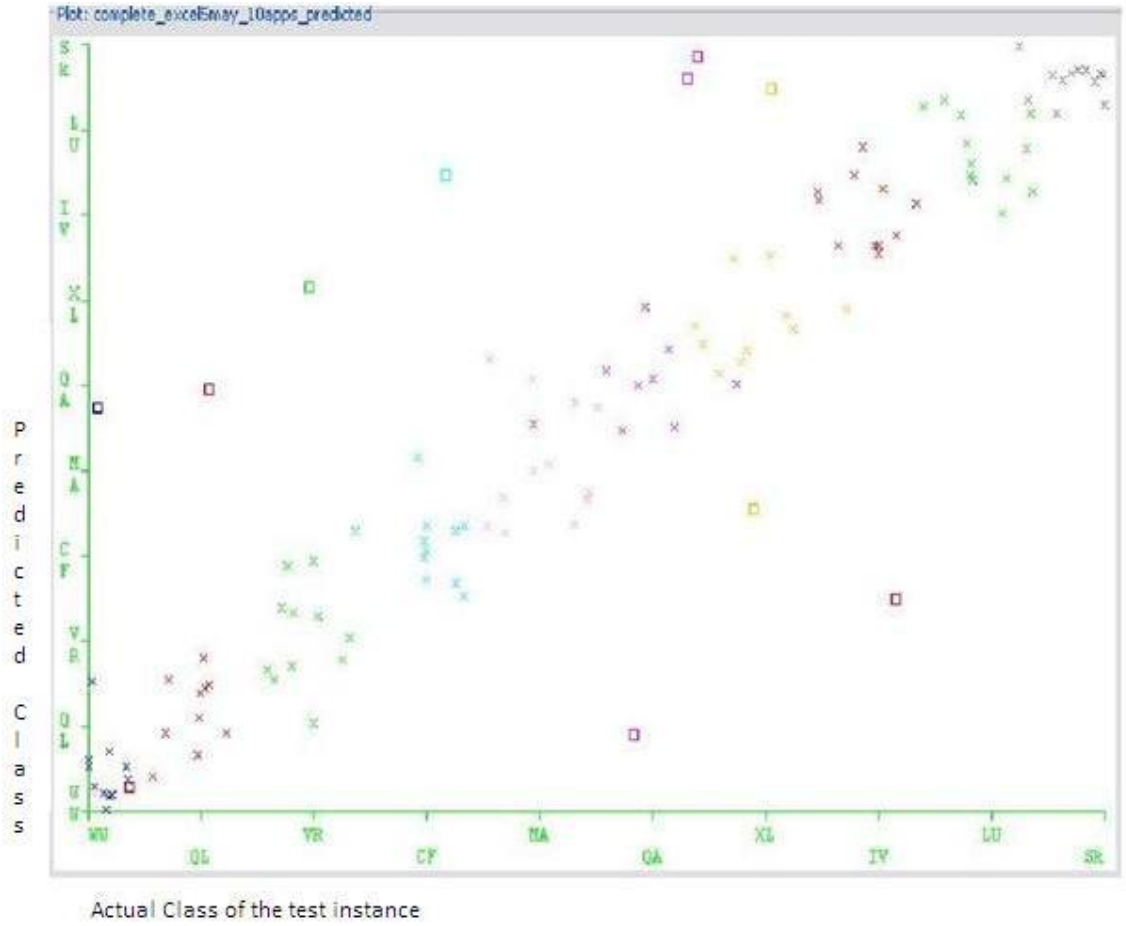


Figure 5-4 Plot visualizing erroneous predictions as boxes '□' and correct ones as '×' for the J48 Decision Tree testing on 10 fold cross validation.

5.1.3 Results of Neural Network classifier (Multilayer Perceptron):

The Multilayer Perceptron is a neural network architecture that trains using backpropagation and classifies instances. In [87], it is stated that " A multilayer perceptron is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. It is a modification of the standard linear perceptron in that it uses three or more layers of neurons (nodes) with nonlinear activation functions, and is more powerful than the perceptron in that it can distinguish data that is not linearly separable." Feedforward means that data flows in one direction from input to output layer through one or more hidden layers (forward). This type of network is trained with the backpropagation learning algorithm. MLPs are extensively employed for pattern classification, prediction and approximation. The network used here consists of three layers, an input layer which

has 26 input nodes corresponding to the 26 attributes of the training data. The 27th ‘app’ or application name attribute is the class or output attribute. The network has one hidden layer and an output layer with 10 nodes. In Weka, the default value for hidden layers is 'a', which means one hidden layer, with the number of nodes being the sum of input nodes and output nodes divided by 2. So in this experiment there are eighteen (18) nodes in the hidden layer, because the number of input nodes is 26 and the number of output nodes is 10 whose sum is 36. Some other default parameters for this network are:

learningRate -- The amount the weights are updated is 0.3

momentum -- Momentum applied to the weights during updating is 0.2

No. Of epochs or passes through training data is set to 500 by default.

The nodes in this network are all sigmoid. The details of these configuration parameters can be found in [83] by Witten and Frank.

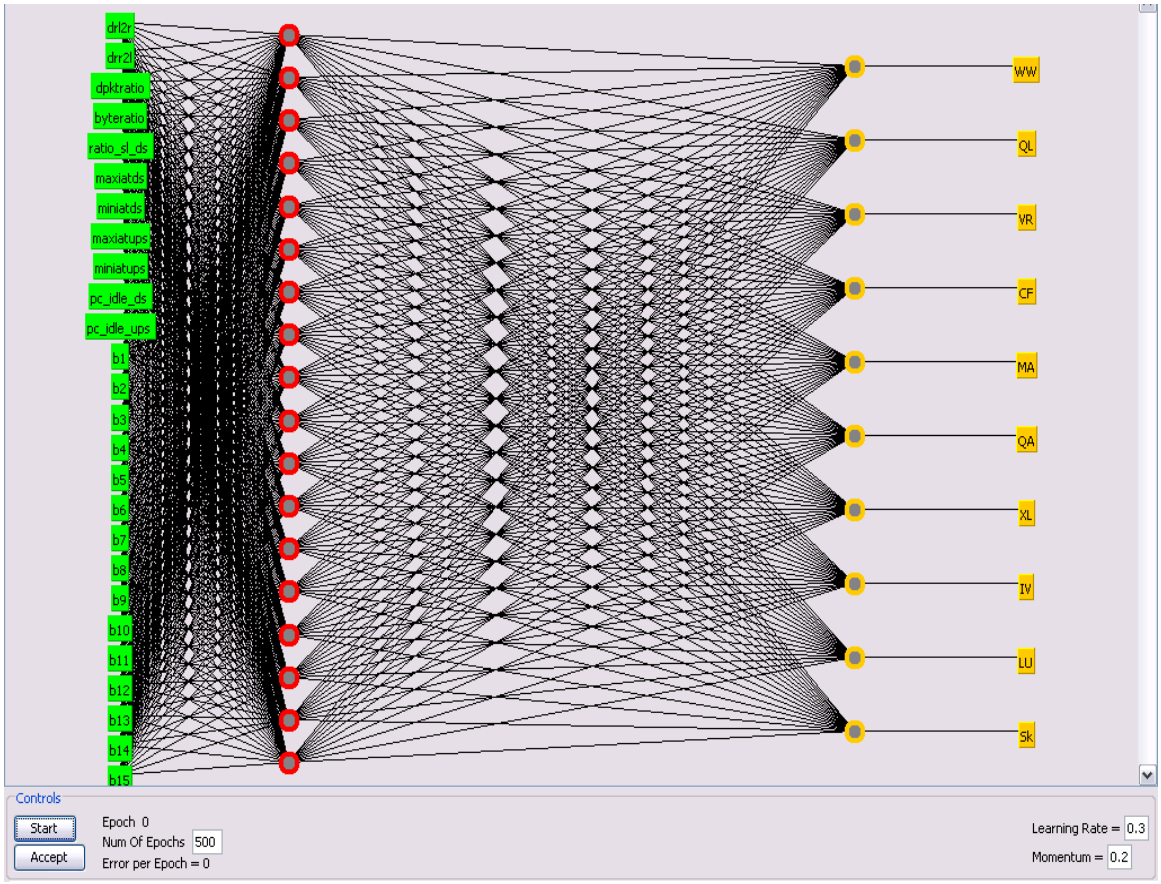


Figure 5-5 The neural network with one hidden layer, one input and one output layer.

Results: Running Multilayer Perceptron using 10 fold cross validation:

=== Summary ===

Correctly Classified Instances 115 95.8333 %

Incorrectly Classified Instances 5 4.1667 %

Total Number of Instances 120

Table 5.7 Confusion Matrix for MLP on 10 fold cross validation test

a	b	c	d	e	f	g	h	i	j	classified as
11	0	0	0	0	1	0	0	0	0	a = WW
0	12	0	0	0	0	0	0	0	0	b = QL
0	0	11	0	0	0	1	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
0	0	0	0	0	10	0	0	0	2	f = QA
0	0	0	0	1	0	11	0	0	0	g = XL
0	0	0	0	0	0	0	12	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	0	0	0	0	12	j = Sk

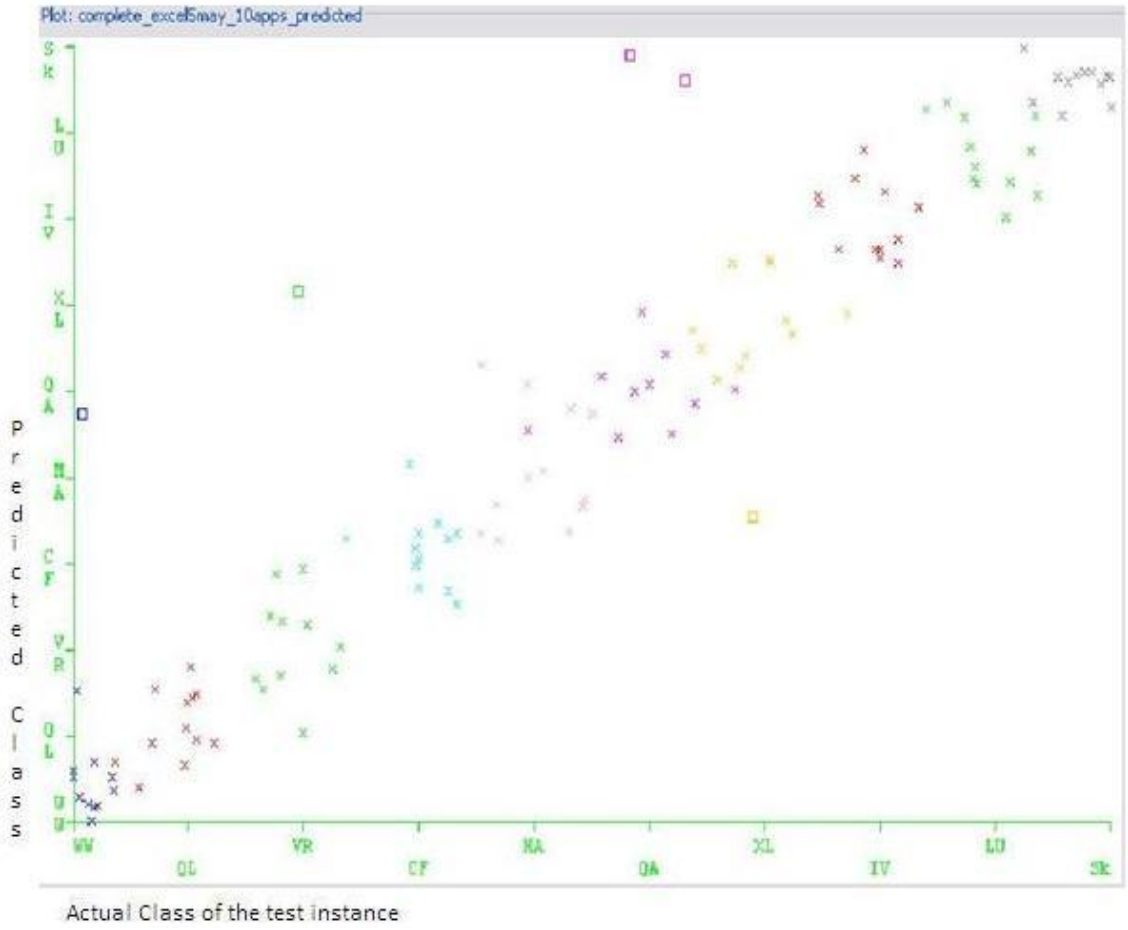


Figure 5-6 Plot visualizing erroneous predictions as boxes '□' and correct ones as 'x' for the Multilayer Perceptron tested on 10 fold cross validation.

The result of Multilayer Perceptron neural network for other schemes of test data is summarised in the table below:

Table 5.8 Summary of the Multilayer Perceptron Classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Multilayer perceptron	Training data	120	120	8.63 s	117(97.5 %)	3(2.5 %)
Multilayer perceptron	10 fold cross	120	120	9.04s	115(95.8 %)	5(4.2 %)
Multilayer perceptron	66 % split	41	79	8.46s	40(97.5 %)	1(2.5 %)

5.1.4 Results of Nearest Neighbour, IB1 Classifier:

The nearest neighbour algorithm is in the category of lazy learners in Weka software. Lazy learners mean that these algorithms merely save the training instances and don't do any real processing until the classification stage, or otherwise they could be quite fast in building a classifier. In Weka software, 'IB1' is a basic instance-based learner (also known as nearest neighbour classifier) which finds the training instance closest in Euclidean distance to the given test instance and predicts the same class as this training instance. If several instances have same Euclidean distance, the first one found is used as the predicted class. [83]

Here the results of applying the IB1 nearest neighbour classification on the dataset are given:

=== Summary ===

Correctly Classified Instances	116	96.6667 %
Incorrectly Classified Instances	4	3.3333 %
Total Number of Instances	120	

Table 5.9 Confusion Matrix for the IB1 classifier on 10 fold cross validation test

a	b	c	d	e	f	g	h	i	j	classified as
11	0	0	0	0	1	0	0	0	0	a = WW
0	12	0	0	0	0	0	0	0	0	b = QL
0	0	11	0	0	0	1	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
0	0	0	0	0	12	0	0	0	0	f = QA
0	0	0	0	0	0	11	0	0	1	g = XL
0	0	0	0	0	0	0	12	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	1	0	0	0	11	j = Sk

The result of IB1 Nearest neighbour for other schemes of test data is summarised in the table below:

Table 5.10 Summary of IB1 classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Nearest Neighbour	Training data	120	120	<0 .01s	120(100 %)	0 (0 %)
Nearest Neighbour	10 fold cross	120	120	<0 .01s	116(96.7 %)	4(3.3 %)
Nearest Neighbour	66 % split	41	79	<0 .01s	41(100 %)	0(0 %)

For the case of running the classifier model on the same training set, the correct prediction percentage is 100%. This is because of the operational nature of the lazy instance base classifier Nearest Neighbour. Since it finds the training instance closest in Euclidean distance to the given test instance and predicts the same class as this training instance, so in the case when training instance is the same as test instances it is able to successfully classify all the test instances, because they are the same as it was trained on.

5.1.5 Results of K* Nearest Neighbour, IBK*:

IBK* is another instance-based classifier. The predicted class of a test instance is based upon the class of those training instances nearest to it, as determined by some distance function. The distance function used here is unlike that of IB1 (the Euclidean nearest neighbour), and is an entropy-based distance function.

Fundamentally the distance between instances can be defined as the complexity of transforming one instance into another. The calculation of the complexity can be performed in two parts. First a finite set of transformations which map instances to instances is defined. A "program" to transform one instance (*a*) to another (*b*) is a finite sequence of transformations beginning at *a* and ending at *b*. The complexity of a program is the length of the shortest string representing the program, and a Kolmogorov distance between two instances is defined to be the length of the shortest string connecting the two instances. The result is a distance measure from which K* distance is calculated by summing over all possible transformations between two instances [88].

The results of running IBK* algorithm with the dataset in 10 fold cross validation are given below:

=== Summary ===

Correctly Classified Instances 115 95.8333 %

Incorrectly Classified Instances 5 4.1667 %

Total Number of Instances 120

Table 5.11 Confusion Matrix for IB K* on the 10 fold cross validation test

a	b	c	d	e	f	g	h	i	j	classified as
10	0	0	0	0	2	0	0	0	0	a = WW
0	12	0	0	0	0	0	0	0	0	b = QL
0	0	11	0	0	0	1	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	e = MA
0	0	0	0	0	11	0	0	0	1	f = QA
0	0	0	0	1	0	11	0	0	0	g = XL
0	0	0	0	0	0	0	12	0	0	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	0	0	0	0	0	0	0	12	j = Sk

The confusion matrix for K* shows the instances that were misclassified.

The result of the K* Nearest Neighbour for other schemes of test data is summarised in the table below:

Table 5.12 Summary of IBK* classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
K* Nearest	Training data	120	120	0.01 s	120(100 %)	0(0 %)
K*Nearest Neighbour	10 fold cross	120	120	<0 .01s	115(95.8 %)	5(4.2 %)
K*Nearest Neighbour	66 % split	41	79	<0 .01s	40(97.5 %)	1(2.4%)

5.1.6 Results of OneR classifier:

The 1R classifier uses the minimum-error attribute for prediction, discretizing numeric attributes. 1R is from one rule and this simple classifier generates a one-level decision tree expressed in the form of a set of rules that all test one particular attribute. 1R is a simple and quick method that often produces useful rules for characterizing the structure in data. Simple rules frequently achieve surprisingly high accuracy [83] hence it was applied on this dataset and the results are summarised below:

=== Summary ===

Correctly Classified Instances	67	55.8333 %
Incorrectly Classified Instances	53	44.1667 %
Total Number of Instances	120	

Table 5.13 Confusion Matrix for the 1 R classifier on the 10 fold cross validation

a	b	c	d	e	f	g	h	i	j	classified as
5	2	0	0	0	2	0	0	3	0	a = WW
3	4	1	0	1	2	0	0	1	0	b = QL
0	0	6	0	3	0	3	0	0	0	c = VR
0	0	0	12	0	0	0	0	0	0	d = CF
0	0	0	0	7	1	4	0	0	0	e = MA
4	3	0	0	0	3	0	1	1	0	f = QA
0	0	1	0	3	0	5	1	2	0	g = XL
0	0	0	0	1	1	0	8	0	2	h = IV
0	0	0	0	0	0	0	0	12	0	i = LU
0	0	1	0	0	1	1	4	0	5	j = Sk

The matrix shows that 1R classifier didn't achieve very accurate results here.

The result of 1R classifier for other schemes of test data is summarised in the table below:

Table 5.14 Summary of the 1R classifications

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
1R	Training data	120	120	<0 .01s	83(69.2%)	37(30.8%)
1R	10 fold cross	120	120	<0 .01s	67(55.8 %)	53(44.2%)
1R	66 % split	41	79	<0 .01s	13(31.7%)	28(68.2%)

From the Table 5.15,it is observed that except for the 1R classifier which did not perform well on this dataset in terms of the accuracy of predictions , the other classifiers have fairly good prediction accuracy, with 10 fold cross-validation experiment accuracy ranging from 97.5% for K* nearest neighbour (IBK*), to 90.8% for the C4.5 Decision Tree classifier. The other classifiers also lie in this general range with the Neural Network at 95.8% and IB1 at 96.5% being the two next best. However, the differences in the performances are somewhat insignificant in terms of only one, two, or up to five instances classified incorrectly out of 120, and it can be stated that their performance is statistically fairly similar.

The training time for all the classifiers has been of the order of 0.01 seconds, except for the neural network, which took much more time for training i.e. 8 -9s.

The metrics chosen for the detection of tunnelled application have proved to be sufficiently discriminatory so as to classify the instances to correct classes.

Summing up the results of all the above experiments in one table:

Table 5.15 Summary of the 27 attributes classifications

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	0.01 s	120(100 %)	0(0%)
K*Nearest Neighbour	10 fold cross	120	120	<0 .01s	115(95.8%)	5(4.2) %
K*Nearest Neighbour	66 % split	41	79	<0 .01s	40(97.5%)	1(2.4 %)
Nearest Neighbour	Training data	120	120	<0 .01s	120(100 %)	0(0%)
Nearest Neighbour	10 fold cross	120	120	<0 .01s	116(96 7%)	4(3.3%)
Nearest Neighbour	66 % split	41	79	<0 .01s	41(100%)	0(0%)
Multilayer perceptron	Training data	120	120	8.63 s	117(97.5%)	3(2.5%)
Multilayer perceptron	10 fold cross	120	120	9.04s	115(95.8%)	5(4.2%)
Multilayer perceptron	66 % split	41	79	8.46s	40(97.5%)	1(2.4 %)
Decision tree	Training data	120	120	0.03s	116(96 7%)	4(3.3%)
Decision tree	10 fold cross	120	120	0.15s	109(90.8%)	11(9.2%)
Decision tree	66 % split	41	79	0.02s	38(92.7%)	3(7.3%)
Naïve Bayes	Training data	120	120	0.01s	115(95.8%)	5(4.2%)
Naïve Bayes	10 fold cross	120	120	0.01s	111(92.5%)	9(7.5%)
Naïve Bayes	66 % split	41	79	0.01s	36 (87.8%)	5 (12.2%)
1R	Training data	120	120	<0 .01s	83(69.2%)	37(30.8%)
1R	10 fold cross	120	120	<0 .01s	67(55.8%)	53(44.2%)
1R	66 % split	41	79	<0 .01s	13(31.7%)	28(68.3%)

5.2 Using Packet Size Distributions alone for Identification

In the previous section, additional discriminating metrics were added to the packet size distribution metric, which was previously used alone for the detection of network applications, UDP based applications [48], and also for TCP-based applications [49]. Next the data set is stripped of the newly added metrics, and only the bins of the packet size distribution are used so that one can observe how well this metric performs in the case of tunnelled application detection with the machine learning algorithms:

5.2.1 Using IB1 nearest Neighbour Detection: with PSD alone

The results for the three cases with IB1 classifier are summarised in table 5.16

Table 5.16 Summary of IB1 classifications with only PSD bins as attributes

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Nearest Neighbour	Training data	120	120	<0 .01s	106(88.3 %)	14(11.7%)
Nearest Neighbour	10 fold cross	120	120	<0 .01s	95(79.2%)	25(20.8%)
Nearest Neighbour	66 % split	41	79	<0 .01s	35(85.3 %)	6(14.6%)

5.2.2 Using K* Nearest neighbour Detection: with PSD alone

The results for the three cases with IB K* classifier are summarised in table 5.17

Table 5.17 Summary of IB K* classifications with only PSD bins as attributes

Algorithm	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
K*Nearest Neighbour	Training data	120	120	<0 .01s	106(88.3 %)	14(11.7%)
K*Nearest Neighbour	10 fold cross	120	120	<0 .01s	100(83.3 %)	20(16.7%)
K*Nearest Neighbour	66 % split	41	79	<0 .01s	35(85.4%)	6(14.6%)

5.2.3 Using Naïve Bayes with PSD alone

The results for three cases with NaïveBayes classifier are summarised in table 5.18

Table 5.18 Summary of Naïve Bayes classifications with only PSD bins

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Naïve Bayes	Training data	120	120	<0 .01s	92(76.7%)	28(23.3%)
Naïve Bayes	10 fold cross	120	120	<0 .01s	87(72.5%)	33(27.5%)
Naïve Bayes	66 % split	41	79	<0 .01s	33(80.5%)	8(19.5%)

5.2.4 Using C4.5 Decision Tree with PSD alone

The results for the three cases with C4.5 Decision Tree classifier are summarised in table 5.19

Table 5.19 Summary of C4.5 Decision Tree classifications with only PSD bins

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Decision tree	Training data	120	120	0.01s	101(84.2 %)	19(15.8%)
Decision tree	10 fold cross	120	120	0.01s	90(75%)	30(25%))
Decision tree	66 % split	41	79	0.01s	30(73.2%)	11(26.8%)

5.2.5 Using Neural Network with PSD alone

The results for three cases with Neural Network classifier are given in table 5.20

Table 5.20 Summary of Neural Network classifications with only PSD bins

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Prediction	Incorrect Prediction
Neural Network	Training data	120	120	5.13	96(80%)	24(20%)
Neural Network	10 fold cross	120	120	5.18 s	91(75.8%)	29(24.2%)
Neural Network	66 % split	41	79	5.48 s	32(78%)	9(22%)

From the above tables if one compares with the tables of the previous dataset, in which all the metrics were used, the performance has degraded considerably. The same comparison can be seen in the line chart below between the two sets of metrics used. The lines represent the accuracy of predictions for various classifier schemes for the both combinations of attributes or metrics.

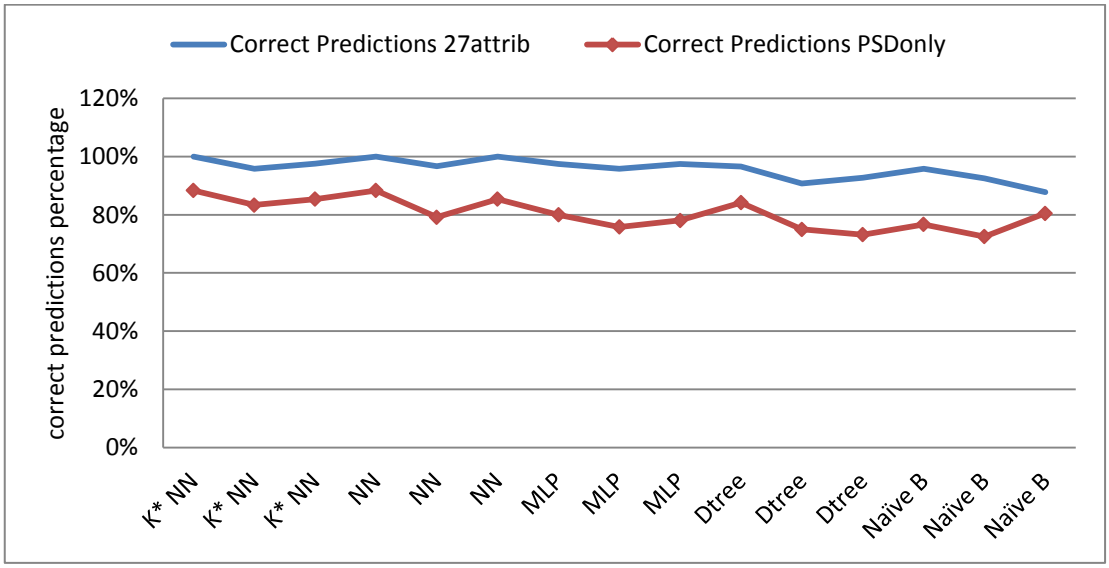


Figure 5-7 comparing the % accuracy using 27 attributes and 15 attributes (psd only)

The lower line represents the accuracy of predictions for the classifiers using packet size distribution alone, and is significantly lower than the classifiers using 27 attributes including packet size distribution. The 27 attributes line is above 90% accuracy whereas the other hovers around or below 80%. The reason for the poor performance when using packet size distribution alone is possibly that the 15 bins resolution used for representing the distribution is less than optimal [52]. Increasing the number of bins might have better effect on the performance. This aspect is explored later when packet size distribution resolution will be increased.

5.3 Using 12 metrics excluding Packet Size Distribution for Identification

Now it is known that only packet size distribution alone is not as good as when used in combination with the other attributes/metrics, it is worth examining how the classifiers would fare if the packet size distribution is completely removed from the

set, and only the 12 other attributes are used in classification. Table 5.21 gives a summary of this for the 5 classifier schemes:

Table 5.21 Summary of classifications with 12 attributes excluding PSD bins

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	0.01 s	120(100 %)	0(0%)
K*Nearest Neighbour	10 fold cross	120	120	<0.01 s	111(92.5%)	9(7.5%)
K*Nearest Neighbour	66 % split	41	79	<0.01 s	40(97.5%)	1(2.5%)
Nearest Neighbour	Training data	120	120	<0.01 s	120(100 %)	0(0%)
Nearest Neighbour	10 fold cross	120	120	<0.01 s	104(86.7%)	16(13.3%)
Nearest Neighbour	66 % split	41	79	<0.01 s	35(85.4%)	6(14.6%)
Multilayer perceptron	Training data	120	120	3.8 s	110(91.7%)	10(8.3%)
Multilayer perceptron	10 fold cross	120	120	3.85 s	103(85.8%)	17(14.2%)
Multilayer perceptron	66 % split	41	79	3.82 s	29(70.7%))	12(29.3%)
Decision tree	Training data	120	120	0.02s	117(97.5%)	3(2.5%)
Decision tree	10 fold cross	120	120	0.01s	105(87.5%)	15(12.5%)
Decision tree	66 % split	41	79	0.01s	37(90.2%)	4(9.8%)
Naïve Bayes	Training data	120	120	0.01s	110(91.7%)	10(8.3%)
Naïve Bayes	10 fold cross	120	120	0.01s	105(87.5%)	15(12.5%)
Naïve Bayes	66 % split	41	79	0.01s	37(90.2%)	4(9.8%)

In the table 5.21, when the time taken to build a trained model is less than 0.01s then it is given as 0 seconds. To make the comparison clearer, the following plot shows the percentages of correct predictions for the 12 attribute data sets excluding the

packet size distributions, and the 27 attribute one. It is again overall lower than for 27 attributes:

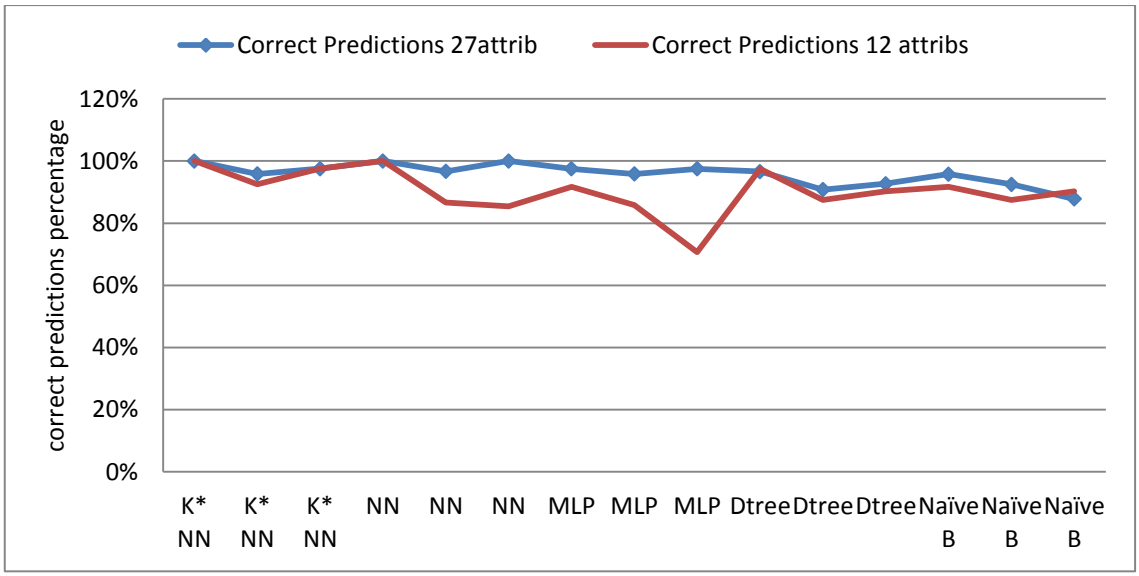


Figure 5-8 Comparing the accuracy using 27 attributes and 12 attributes (excluding psd)

However, the difference is not as huge as the difference for the packet size distribution only case. The cases of coincidence can be explained easily for the Nearest Neighbour and K* Nearest Neighbour in the "use training set mode" and in "split training set" mode because of the lazy nature of these classifiers.

This can be seen in the Figure 5-9 which shows the three cases together. So using the 27 attributes, including packet size distribution bins is the appropriate combination of the three rather than packet size distribution alone or excluding packet size distribution. The case of increasing the packet size distribution resolution needs to be explored, however.

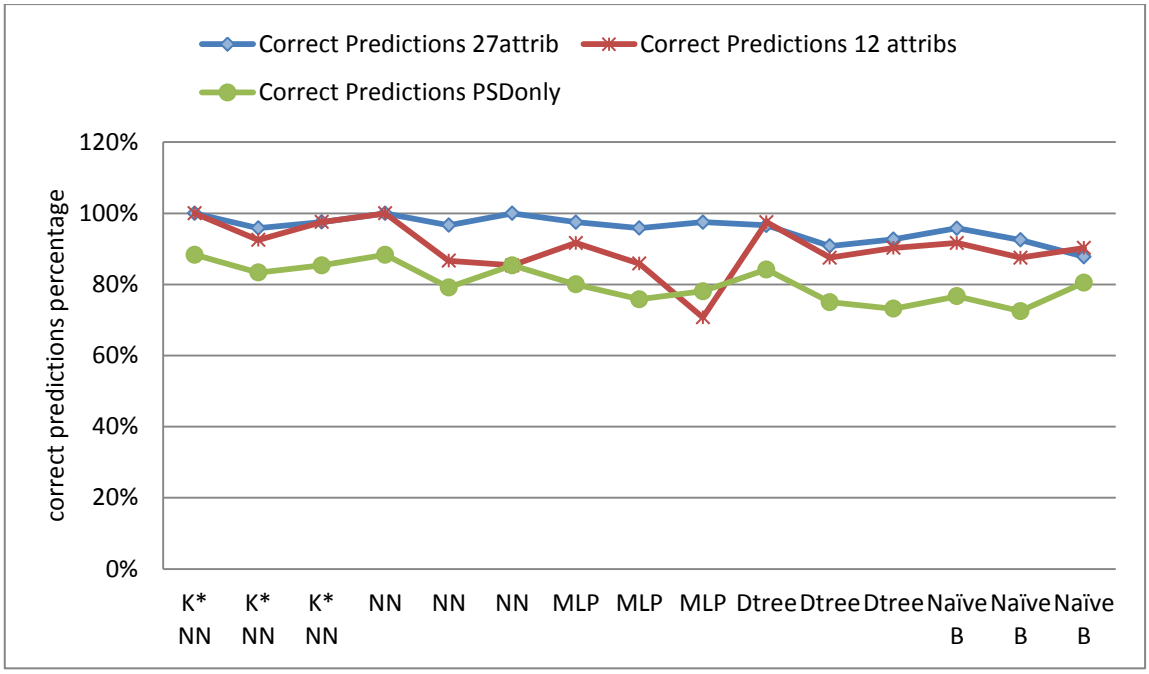


Figure 5-9 Comparing the % accuracy using 27, 15 (PSD only) and 12 attributes.

5.4 Excluding Temporal Attributes for Identification

It is questionable if the attributes which involve packet interarrival times (and there are 4 of them in the 27 attribute set), are consistent over varying network conditions, and would change according to the congestion state of the network. So these 4 attributes are simply discarded, and then the effect on the results is observed. Here the experiment is repeated with 23 attributes including the packet size distributions and excluding the interarrival time ones.

Table 5.22 Summary of classifications with exclusion of interarrival time attributes

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	<0.01 s	120(100 %)	0(0%)
K*Nearest Neighbour	10 fold cross	120	120	<0.01 s	118(98.8%)	2(1.2%)
K*Nearest Neighbour	66 % split	41	79	<0.01 s	41(100 %)	0(0%)
Nearest Neighbour	Training data	120	120	<0.01 s	120(100 %)	0(0%)
Nearest Neighbour	10 fold cross	120	120	<0.01 s	116(96.7%)	4(3.3%)
Nearest Neighbour	66 % split	41	79	<0.01 s	41(100 %)	0(0%)
Multilayer perceptron	Training data	120	120	7.96 s	113(94.2%)	7(5.8%)
Multilayer perceptron	10 fold cross	120	120	7.58	112(93.3%)	8(6.7%)
Multilayer perceptron	66 % split	41	79	7.1 s	39(95.1%)	2(4.9%)
Decision tree	Training data	120	120	0.02s	116(96.7%)	4(3.3%)
Decision tree	10 fold cross	120	120	0.01s	112(93.3%)	8(6.7%)
Decision tree	66 % split	41	79	0.02s	38(92.7%)	3(7.3%)
Naïve Bayes	Training data	120	120	0.01s	114(95%)	6(5%)
Naïve Bayes	10 fold cross	120	120	0.01s	106(88.3%)	14(11.7%)
Naïve Bayes	66 % split	41	79	<0.01 s	36(87.8%)	5(12.2%)

For comparing these results with those obtained from using 27 attributes/metrics, let us look at the following plot:

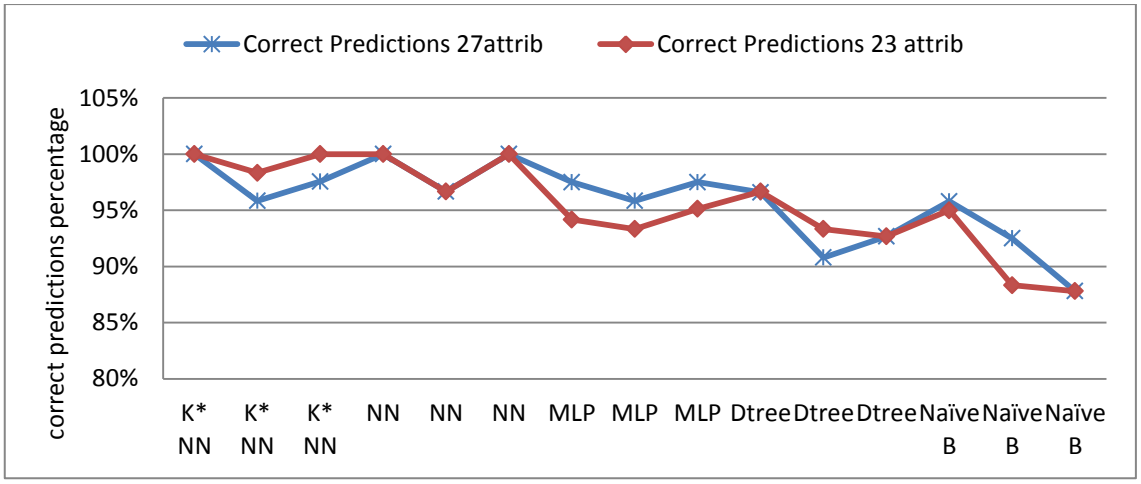


Figure 5-10 Comparing the % accuracy using 27 attributes and 23 attributes (excluding IAT attributes)

Looking at the resolution of the plot, the accuracy doesn't show much variation, and also the two lines cross each other at several places. This fact suggests that the time related attributes are not reliable and don't have a consistent effect on the classification predictions. Hence in further experiments the interarrival time attributes will not be included.

5.5 Using 30 bins of Packet Size Distribution for Identification

The resolution of the packet size distribution has now been increased to 30 bins; in addition to that there are 8 other attributes, because the interarrival time based attributes have been discarded. Now the following experiments are performed using the 38 attribute data set:

1. Using PSD alone
2. Using all 38 bins

First the results of using 30 bins Packet Size Distribution alone for the detection are taken and compared with 15 bins Packet Size Distribution:

Table 5.23 Summary of classifications with 30 bins PSD alone

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	<0.01 s	113(94.2%)	7(5.8%)
K*Nearest Neighbour	10 fold cross	120	120	<0.01 s	108(90%))	12(10%)
K*Nearest Neighbour	66 % split	41	79	<0.01 s	36(87.8%)	5(12.2%)
Nearest Neighbour	Training data	120	120	0.01 s	113(94.2%)	7(5.8%)
Nearest Neighbour	10 fold cross	120	120	0.01 s	105(87.5%)	15(12.5%)
Nearest Neighbour	66 % split	41	79	<0.01 s	40(97.6%)	1(2.4%)
Multilayer perceptron	Training data	120	120	10.31 s	110(91.7%)	10(8.3%)
Multilayer perceptron	10 fold cross	120	120	10.07 s	105(87.5%)	15(12.5%)
Multilayer perceptron	66 % split	41	79	10.31 s	40(97.6%)	1(2.4%)
Decision tree	Training data	120	120	0.04 s	109(90.8%)	11(9.2%)
Decision tree	10 fold cross	120	120	0.02 s	96(80%)	24(20%)
Decision tree	66 % split	41	79	0.03 s	32(78%)	9(22%)
Naïve Bayes	Training data	120	120	0.02s	88(73.3%)	32(26.7%)
Naïve Bayes	10 fold cross	120	120	0.02 s	87(72.5%)	33(27.5%)
Naïve Bayes	66 % split	41	79	0.02 s	31(75.6%)	10(24.4%)

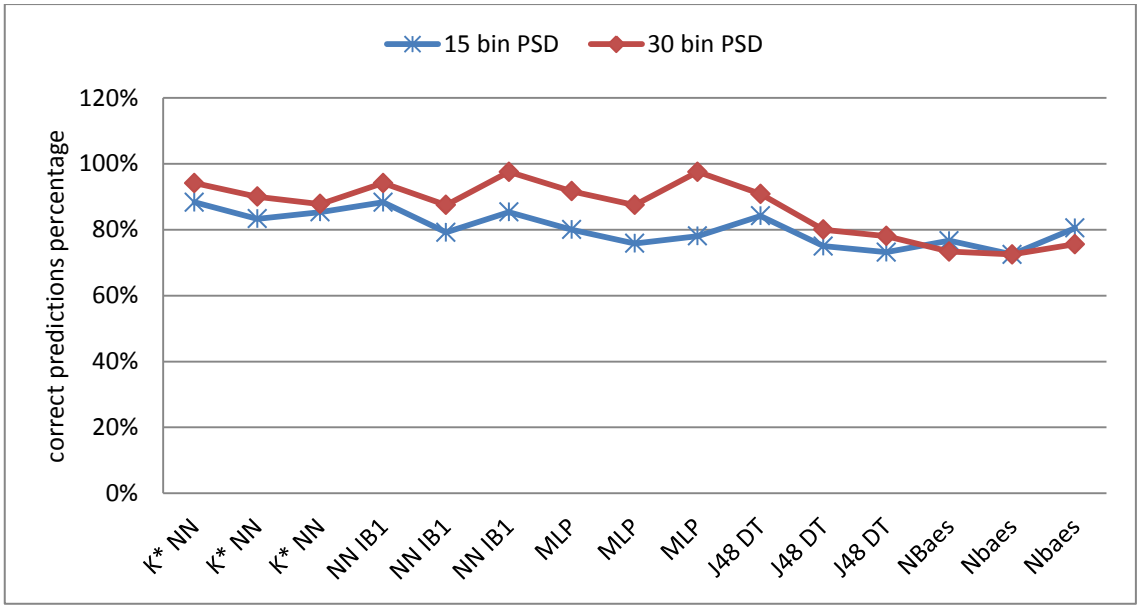


Figure 5-11 Performance comparison of the ML classifiers for the 30 bin PSD and 15 bin PSD

From Figure 5-11, the 30 bin resolution is more accurate for all the classifiers except the Naïve Bayes. The Naïve Bayes could not make any better results by increasing Packet size Distribution resolution. For other classifiers the difference is significant and varies from 5% to 20% improvement in performance.

This was the comparison for the packet size distributions only. What about the results for the overall classification which involves all the 38 attributes including Packet size distribution? The following table summarises the results for the classifiers using all 38 attribute/metrics altogether.

The performance of the classifiers using the 38 attributes including 30 PSD bins versus that of using 23 attributes with 15 PSD bins can be observed from the line graph in Figure 5-12. For the Nearest Neighbour classifier and K* Nearest Neighbour classifier, and the J48 Decision tree the performance of the classifiers in both cases is almost exactly the same. The neural network classifier (Multilayer Perceptron) and Naïve Bayes classifier have shown significant improvement in their performance with 38 bins case.

Table 5.24 Summary of classifications with 38 attributes including 30 PSD bins

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	<0.01 s	120(100 %)	0(0%)
K*Nearest Neighbour	10 fold cross	120	120	<0.01 s	117(97.5%)	3(2.5%)
K*Nearest Neighbour	66 % split	41	79	<0.01 s	41(100 %)	0(0%)
Nearest Neighbour	Training data	120	120	0.01 s	120(100 %)	0(0%)
Nearest Neighbour	10 fold cross	120	120	0.01 s	116(96.7%)	4(3.3%)
Nearest Neighbour	66 % split	41	79	<0.01 s	41(100 %)	0(0%)
Multilayer perceptron	Training data	120	120	19.28 s	118(98.8%)	2(1.2%)
Multilayer perceptron	10 fold cross	120	120	13.42 s	116(96.7%)	4(3.3%)
Multilayer perceptron	66 % split	41	79	12.98 s	41(100 %)	0(0%)
Decision tree	Training data	120	120	0.1 s	119(99.2%)	1(0.8%)
Decision tree	10 fold cross	120	120	0.05 s	111(92.5%)	9(7.5%)
Decision tree	66 % split	41	79	0.03 s	38(92.7%)	3(7.3%)
Naïve Bayes	Training data	120	120	0.01s	114(95%)	6(5%)
Naïve Bayes	10 fold cross	120	120	0.01 s	112(93.3%)	8(6.7%)
Naïve Bayes	66 % split	41	79	0.01 s	39(95.1%)	2(4.9%)

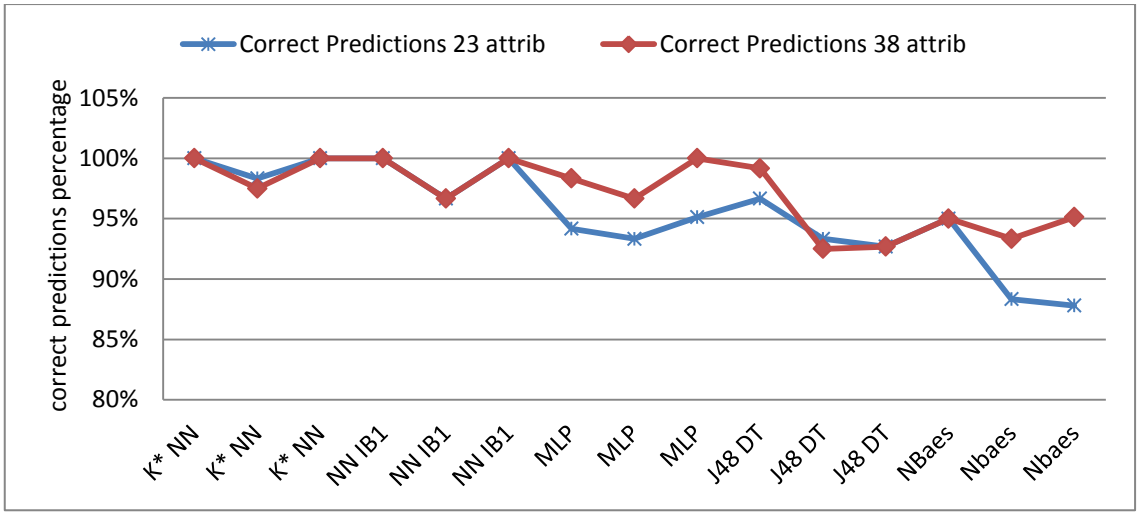


Figure 5-12 comparison of performance of ML algorithms with 23 attributes and 38 attributes

For further clarity, the 10 fold cross validation results are depicted in the histogram in Figure 5-13. The first two classifiers namely K* Nearest Neighbour and Nearest Neighbour, IB1 have the same performance because their accuracy is already very close to 100% in both the cases. So there wasn't much room for improvement. The difference for the J48 Decision Tree is also not statistically significant. The difference for the neural network and Naïve Bayes classifiers shows significant improvement, thus overall the 38 bins has resulted in a better performance.

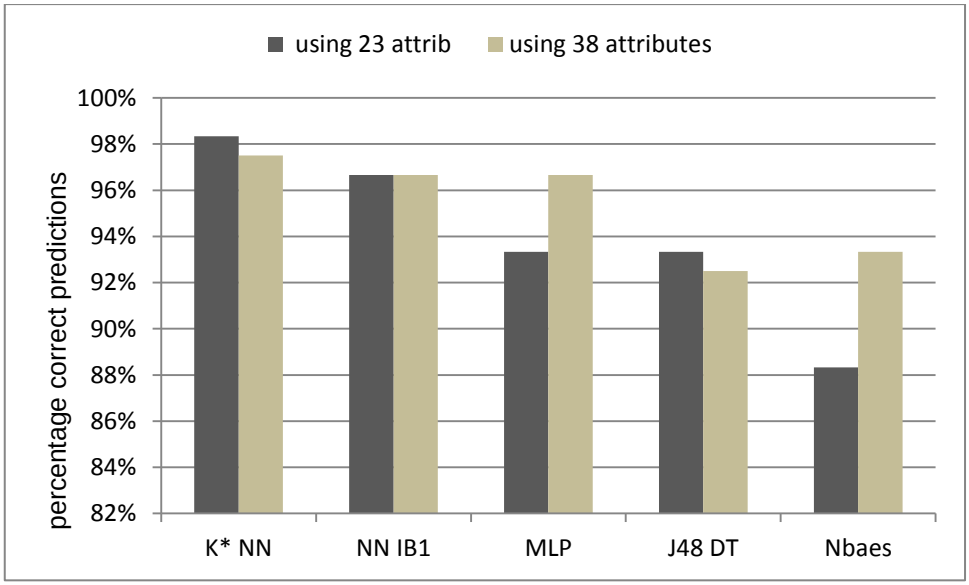


Figure 5-13 comparison in 10 fold cross validation case only

Now the comparison of the 38 attributes classification and the 30 bins PSD only classifications can be made for the classifiers in Figure 5.14:

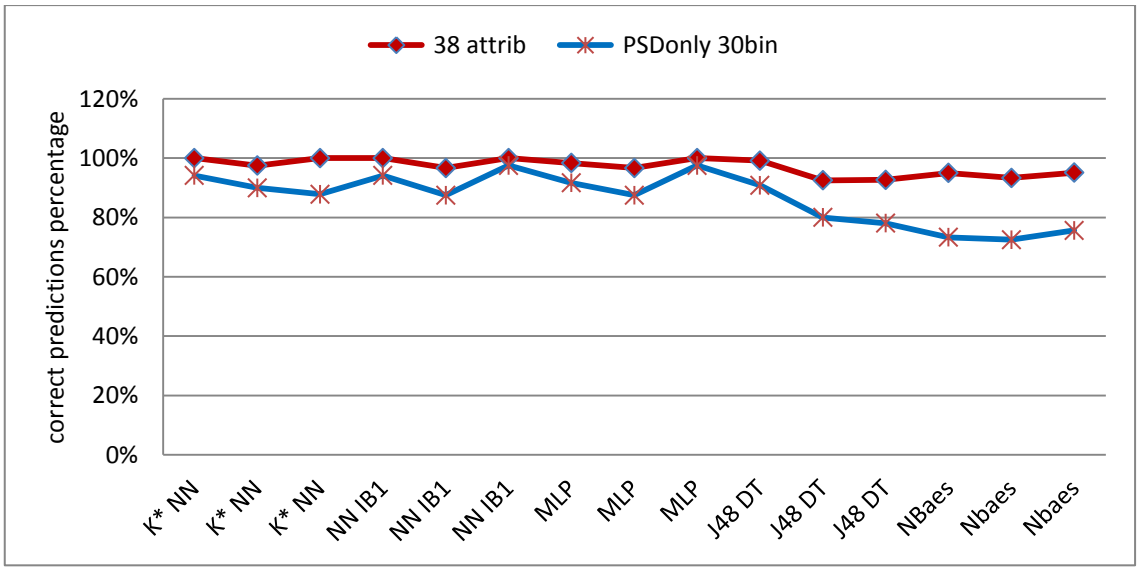


Figure 5-14 Comparing 38 attributes classifications and the 30 bins PSD only

The performance for all the classifiers has increased in percentage accuracy using the 38 attributes rather than only using packet size distributions with 30 bin resolution.

From the previous few sections, it was seen that the 30 bin resolution has resulted in better performance for the case of application identification by machine learning algorithms especially when used in combination with the 8 other traffic attributes. Next the case of 50 bin resolution is taken to see if the improvement trend still exists in increasing the resolution from 30 bins to 50 bins.

5.6 Using 50 bins of Packet Size Distribution for Identification

The resolution of the packet size distribution has been increased to 50 bins, which means that each bin is of 30 bytes in size. Now in all there are 58 attributes with 50 being PSD bins, and 8 other attributes, because the inter-arrival time based attributes have been discarded. Here it needs to be investigated whether further increasing the packet size distribution bins also increases the performance in terms of correct predictions of the machine learning algorithms. Now the following experiments using the 58 attribute data set is given:

Table 5.25 Summary of classifications with 58 attributes

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Training data	120	120	<0.01 s	120(100 %)	0(0%)
K*Nearest Neighbour	10 fold cross	120	120	<0.01 s	117(97.5%)	3(2.5%)
K*Nearest Neighbour	66 % split	41	79	0 .01 s	41(100 %)	0(0%)
Nearest Neighbour	Training data	120	120	0.01 s	120(100 %)	0(0%)
Nearest Neighbour	10 fold cross	120	120	0.01 s	117(97.5%)	3(2.5%)
Nearest Neighbour	66 % split	41	79	<0.01 s	41(100 %)	0(0%)
Multilayer perceptron	Training data	120	120	24.86 s	120(100 %)	0(0%)
Multilayer perceptron	10 fold cross	120	120	25.6 s	115(95.8%)	5(4.2%)
Multilayer perceptron	66 % split	41	79	25.27 s	41(100 %)	0(0%)
Decision tree	Training data	120	120	0.1 s	119(99.2%)	1(0.8%)
Decision tree	10 fold cross	120	120	0.06 s	110(91.7%)	10(8.3%)
Decision tree	66 % split	41	79	0.03 s	36(87.8%)	5(12.2%)
Naïve Bayes	Training data	120	120	0.06s	115(95.8%)	5(4.2%)
Naïve Bayes	10 fold cross	120	120	0.02 s	113(94.2%)	7(5.8%)
Naïve Bayes	66 % split	41	79	0.02 s	38(92.7%)	3(7.3%)

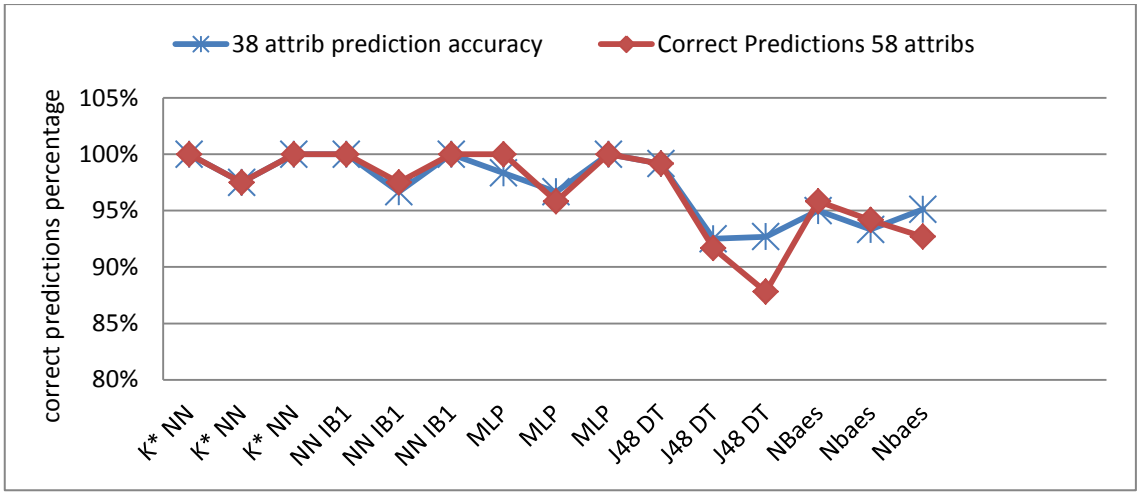


Figure 5-15 Classifications using 38 attributes and 58 attributes

From the table 5.25 and Figure 5-15, it is clear that increasing the resolution of the Packet size distribution to 50 bins from 30 bins has resulted in no further improvement in the classifier's performance in terms of the percentage of correct predictions being made. The two lines are very much coincident, and the difference is not significant. Hence it can be inferred that although increasing the packet size distribution resolution from 15 bins to 30 bins resulted in better performance of the classifiers in general. Further increasing the packet size distribution resolution to higher levels like 50 bins or 100 bins would not improve the results of the classifiers. However, this would add significantly to the processing overhead, because the total number of attributes increasing from 38 to 58 means more processing requirements. For example, the Multilayer Perceptron has increased the training time from 13.42 seconds to 25.6 seconds in the stratified cross validation case only.

Thus this provides evidence that going to higher resolutions for the Packet size distributions is unnecessary, because the performance does not improve. Hence it can be concluded that 15 bins is a bit too small a resolution for use, and 50 bins would be unnecessarily large, so the 30 bins of Packet size distributions are near optimal for use in the machine learning algorithms, along with the other 8 attributes.

5.7 Testing on Previously Unseen Data

In the previous experiments, the testing of machine learning algorithms was performed using the same training data, although with different variants such as splitting 66% into training and 34% into test parts or by 10 fold cross-validation, or even on exactly the same training set. Statistically, it seems that these results would be a good estimate of the actual performance over real test data. Now these training data are used with unseen data for testing, i.e. not from the same dataset. This test data is collected from new pcap trace files. Hence, it was never used in the training. This would give further insight into the robustness of the methodology.

The dataset prepared with the new tracefiles contains 33 instances to be tested from the same applications used in previous sections. The following table summarises the results of the 6 different classifier algorithms tested using the separately applied test data. The new trace files also have 30 bins of packet size distribution in their corresponding attributes files, because it has been previously shown that 30 bins is optimal resolution for the Packet size distributions.

Table 5.26 Summary of classifications of fresh data using 38 attributes

Classifier	Test Mode	Total test instances	Total training	Time taken to	Correct Predictions	Incorrect Predictions
K* Nearest	Separate test set	33	120	<0.01 s	33(100 %)	0(0%)
Multilayer perceptron	Separate test set	33	120	17.24 s	26(78.8%)	7(21.2%)
Decision tree	Separate test set	33	120	0.08 s	30(90.9%)	3(9.1%)
Nearest Neighbour	Separate test set	33	120	<0.01 s	33(100 %)	0(0%)
Naïve Bayes	Separate test set	33	120	0.13s	32(97%)	1(3%)
OneR	Separate test set	33	120	0.02 s	21(63.6%)	12(36.4%)

In the Figure 5-16, the performance on the fresh test data is compared with two cases of the previous sections, i.e. the 10 fold cross validation case in figure 5-16 and ‘use training data for testing’ case in figure 5-17.

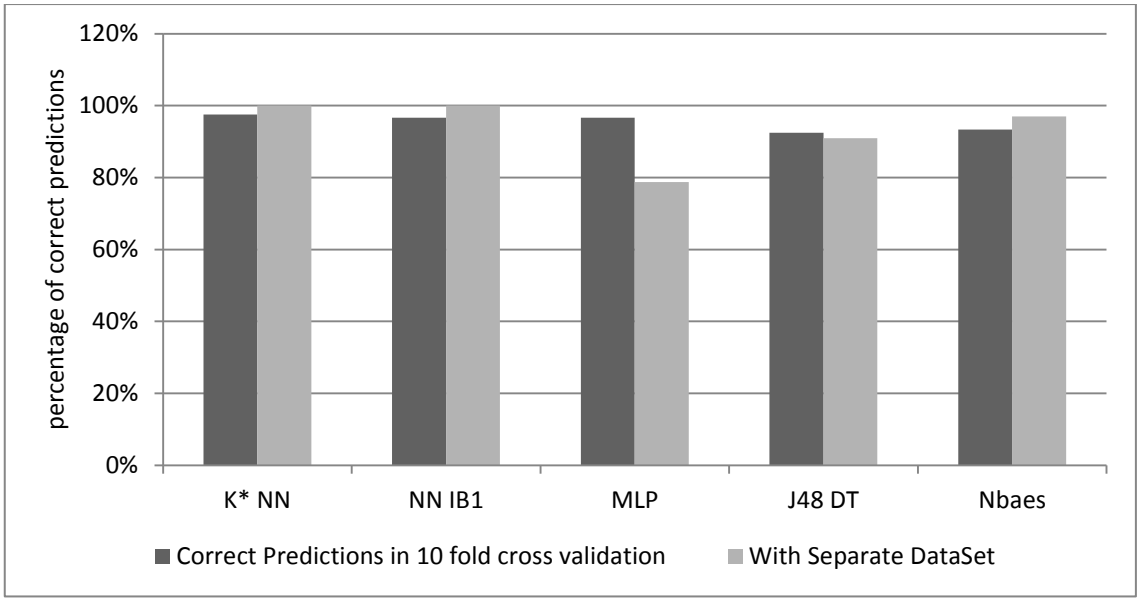


Figure 5-16 Predictions in fresh data and in 10 fold cross validation

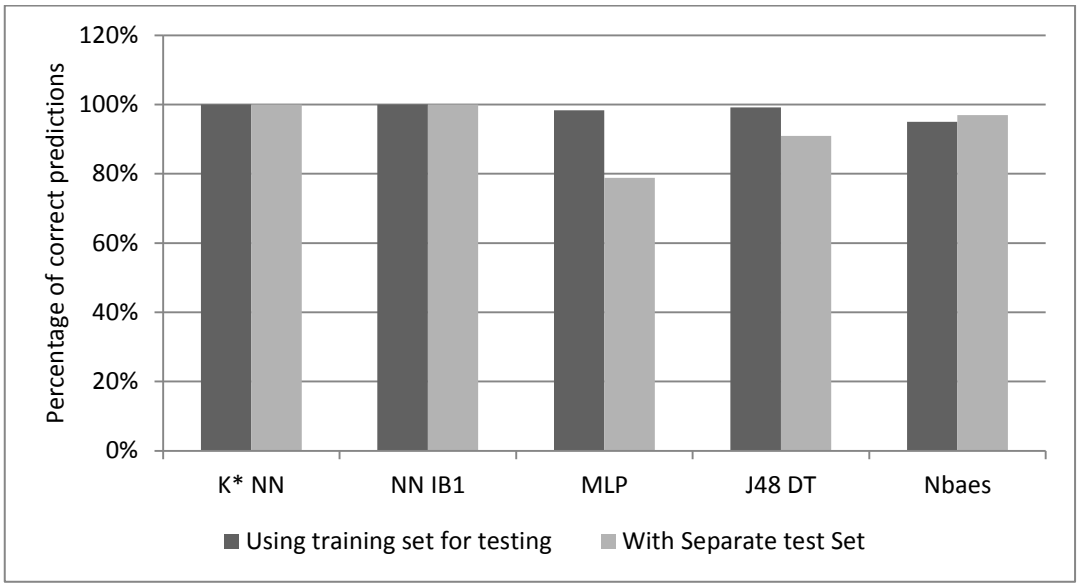


Figure 5-17 Predictions on fresh data and predictions on the training set

The overall results do not show much difference between the fresh data or the 10 fold cross validation case, although the 10 fold cross validation was a little better for the Multilayer Perceptron, so it can be observed that Multilayer Perceptron performance deteriorated slightly when it was tested on unseen data. The other algorithms had excellent accuracy results even for unseen data. The best were the distance based classifiers Euclidean distance based Nearest Neighbour, or its sister classifier K* Nearest neighbour. With the 'use training set for testing' comparison,

it is just slightly higher in accuracy for MLP and Decision Tree, and for others it is the same, because it was already stated that this mode is an optimistic mode to test the algorithm. So, the key finding here is that these machine learning algorithms worked well, not only when tested on the training data in various combinations, but also on fresh test data.

5.8 Validation of the Weka Results:

The Weka machine learning software was used for classification of the application's distributions. The test data used in the classification experiments was already supervised, i.e. the classes of the test instances were known beforehand. The purpose of the experiments was to investigate the performance of the algorithms for the identification/classification of applications. In this section the results of the Weka software are validated using two means:

1. By comparing the classes identified by the Weka with the actual class of the same test instance.
2. By comparing the Weka results of an algorithm with the results of a similar algorithm in Matlab.

This validation has been performed for one experimental case only, since it would be redundant to repeat it for all the other cases in this chapter. The algorithm used in this validation is the k-nearest neighbour based classification algorithm using Euclidean Distance. The data set for this validation case is the dataset used in section 5.5 and table 5.23 which contains the packet size distribution profiles of the 10 applications. The 120 instances of the applications data are divided into two groups: 80 instances form the training set, and the remaining 40 instances are used as test set. The test sets are defined beforehand so that the results of the classification are compared with the actual application classes later.

Results produced by Weka:

Correctly Classified Instances	32	80	%
Incorrectly Classified Instances	8	20	%
Total Number of Instances	40		

The confusion matrix in table 5.27 gives in detail the classification results produced by the Weka software. The erroneous classifications are: 3 instances of Voipraider are classified as X-Lite, 3 instances of Skype are classified as Voipraider and 2 instances of Quake3 Arena classified as Skype. The rest of classifications are correct.

Table 5.27 Confusion Matrix for Weka results for Euclidean Nearest Neighbor algorithm

a	b	c	d	e	f	g	h	i	j	classified as
1	0	3	0	0	0	0	0	0	0	a = VR
3	0	0	0	0	0	0	0	0	0	b = Sk
0	0	5	0	0	0	0	0	0	0	c = XL
0	0	0	3	0	0	0	0	0	0	d = CF
0	0	0	0	5	0	0	0	0	0	e = MA
0	2	0	0	0	3	0	0	0	0	f = QA
0	0	0	0	0	0	3	0	0	0	g = LoU
0	0	0	0	0	0	0	4	0	0	h = IV
0	0	0	0	0	0	0	0	4	0	i = WW
0	0	0	0	0	0	0	0	0	4	j = QL

1. Comparing with the Actual Classes:

The 40 instances of application data packet size distribution are given in Appendix C, “The test set for Validation of Weka Results”. By comparing the class of the test set with the class produced by the weka using the Euclidean nearest neighbor algorithm, the results are exact and accurate. When the “visualize classification errors” option of the Weka is selected a new window pops up, which graphically shows each instance classified correctly or incorrectly in a scatter-plot. When the

instance is clicked, its data is popped up, for example the following information is shown for the first instance:

Instance: 4

b1 : 0.01	b2 : 0.0	b3 : 0.0	b4 : 0.01	b5 : 0.0	b6 : 0.0
b7 : 0.0	b8 : 0.0	b9 : 0.0	b10 : 0.0	b11 : 0.0	b12 : 0.01
b13 : 0.0	b14 : 0.0	b15 : 0.0	b16 : 0.01	b17 : 0.0	b18 : 0.0
b19 : 0.0	b20 : 0.2	b21 : 0.0	b22 : 0.0	b23 : 0.0	b24 : 0.57
b25 : 0.0	b26 : 0.0	b27 : 0.2	b28 : 0.0	b29 : 0.0	b30 : 0.01

predictedapp : XLite, app : VoipRaider

This instance was matched against the 4th instance of the test data set as shown in the appendix B and its class was actually found to be VoipRaider. Similarly the rest of the instances classified correctly were compared with the actual class and was found to be the same. The instances classified incorrectly were also checked in the test data set to be rightfully done by Weka.

2. Comparing with similar Algorithm in Matlab:

The same training data set and test data set were used in Matlab with similar function for the classification using the Euclidean k nearest neighbor algorithm. The function was knnclassify from the bioinformatics toolbox: `Class = knnclassify(Sample, Training, Group)` classifies the rows of the data matrix `Sample` into groups, based on the grouping of the rows of `Training`. Care needs to be taken in the dimensions of the three data sets.

The summary of the results produced by this function of Matlab are:

Correctly Classified Instances	31	78%
Incorrectly Classified Instances	9	22%
Total Number of Instances	40	

Table 5.28 Confusion Matrix for Matlab results for Euclidean Nearest Neighbor algorithm

a	b	c	d	e	f	g	h	i	j	classified as
1	0	3	0	0	0	0	0	0	0	a = VR
3	0	0	0	0	0	0	0	0	0	b = Sk
0	0	5	0	0	0	0	0	0	0	c = XL
0	0	0	3	0	0	0	0	0	0	d = CF
0	0	0	0	5	0	0	0	0	0	e = MA
0	2	0	0	0	3	0	0	0	0	f = QA
0	0	0	0	0	0	3	0	0	0	g = LoU
0	0	0	0	0	0	0	4	0	0	h = IV
0	1	0	0	0	0	0	0	3	0	i = WW
0	0	0	0	0	0	0	0	0	4	j = QL

The results are very similar to the Weka results and the same instances are misclassified by Matlab as are misclassified by Weka, except for one instance of World or Warcraft which is misclassified as Skype in table 5.28, which possibly arose from difference in the implementation of the algorithms in the two software packages. Weka produced 80% classification accuracy, and the similar algorithm from Matlab produced similar accuracy of 78% with only slight difference of misclassification of one instance. In this thesis, the actual class of the test data fed into Weka was already known (i.e. supervised testing), hence the Weka outputs could always be compared with the actual classes of the instances.

5.9 Scalability

The previous results have shown that the machine learning schemes are able to successfully identify the applications in the data set based on the statistical parameters of the application trace file. The data set included 10 applications. The issue of scalability is that would the method perform similarly if the number of applications is increased further. The definite answer is not possible to prove, because this work is based on empirical results of the applications' data. Here the

number of applications is increased from 10 to 15 applications and it is investigated whether the method still is able to differentiate between them.

Here the machine learning algorithm of "IB K*" i.e. the Nearest Neighbour learner based on entropy distance is used only. The distance is not Euclidean but entropy based. This has performed better in accuracy than other algorithms. The five new applications used are: Unreal Tournament, Zattoo which is an IP TV application, Real Player, Remote Desktop Application and Guild Wars, a role playing game. The packet size distribution resolution of the trace files is 30 bins. The test mode selected is 10 fold cross-validation which is explained in section 5.1. According to Witten and Frank in [83], the standard way of predicting the error rate of a learning technique given a single, fixed sample of data is to use stratified 10-fold cross-validation. The results of the experiment on using 15 applications are shown:

=== Summary ===

Correctly Classified Instances	171	96.0674 %
Incorrectly Classified Instances	7	3.9326 %
Total Number of Instances	178	

The incorrect classifications are also shown in the figure 5.18

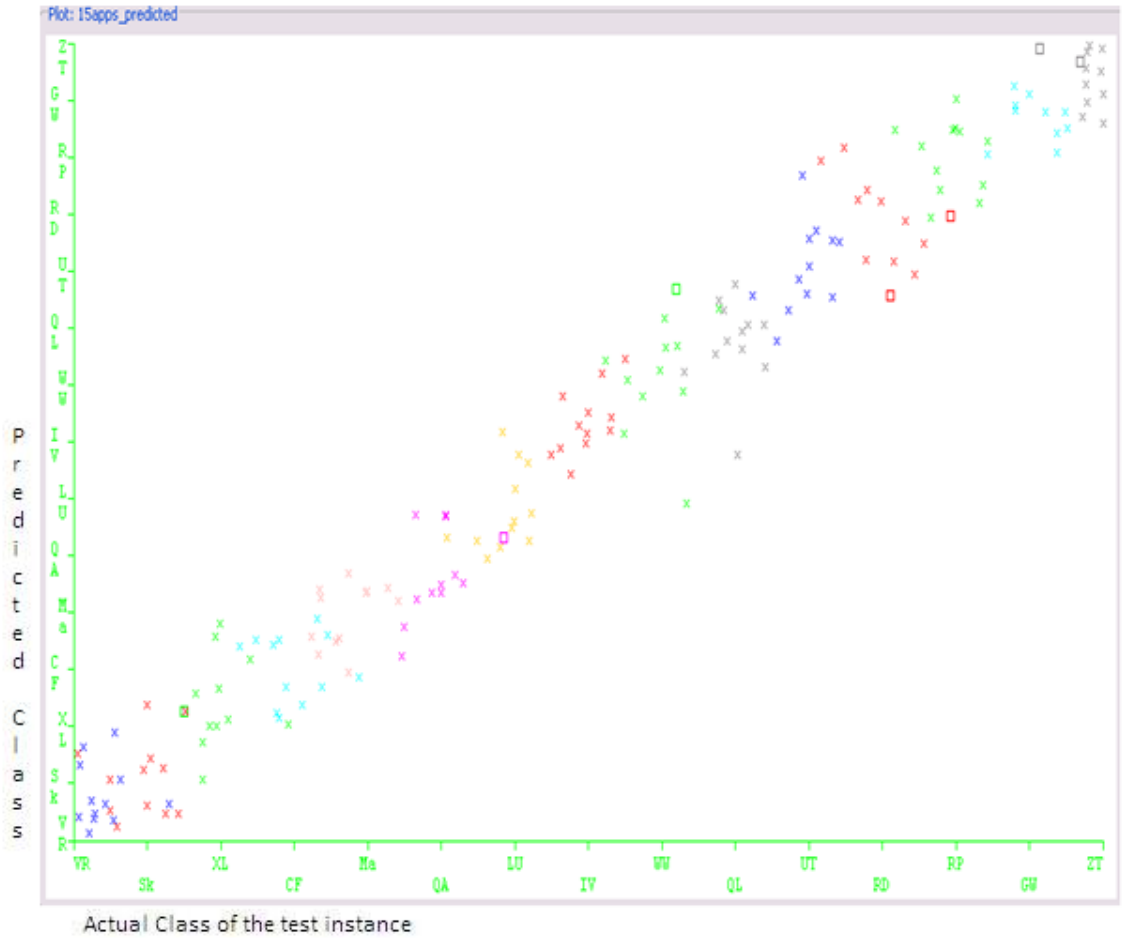


Figure 5-18 Plot visualizing erroneous predictions as boxes '□' and correct ones as '×' for the IB K* algorithm for 15 applications on 10 fold cross validation.

Table 5.29 Confusion Matrix for IBK* classifier on 15 applications data

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	classified as
12	0	0	0	0	1	0	0	0	0	0	0	0	0	0	a = VR
0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	b = SK
1	0	11	0	0	0	0	0	0	0	0	0	0	0	0	c = XL
0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	d = CF
0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	e = MA
0	1	0	0	0	11	0	0	0	0	0	0	0	0	0	f = QL
0	0	0	0	0	0	12	0	0	0	0	0	0	0	0	g = LU
0	0	0	0	0	0	0	12	0	0	0	0	0	0	0	h = IV
0	0	0	0	0	0	0	0	11	1	0	0	0	0	0	i = WW
0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	j = QL
0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	k = UT
0	0	0	0	0	0	0	0	0	1	0	10	0	1	0	l = RD
0	0	0	0	0	0	0	0	0	0	0	0	12	0	0	m = RP
0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	n =GW
1	0	0	1	0	0	0	0	0	0	0	0	0	0	10	o = ZT

These results suggest that addition of new applications would not affect the classification accuracy very much at least, as the 15 applications which have been tested here show an accuracy of 96.1 %. The same case of the IB K* algorithm with 10 applications tested with 10 fold cross validation had an accuracy of 96.67% as given in Table 5.24. Therefore these figures suggest that the method would hold its utility for a considerable number of applications. However it has not been proven that it would work given an extremely large database of applications. The exact limiting number of applications is not possible to show using the empirical methods in this work.

Chapter 6 Conclusions and Future Work

In this chapter, concluding remarks about this work and recommendations for future work are given. The main contributions of the thesis are highlighted below for a quick overview:

- This work presents a statistical approach for tunnelled application identification combining PSD with other parameters.
- The identification can be performed for the actual application being run inside the application layer tunnel.
- The optimal parameters combination found out is to use 30 bins of PSD with 8 other parameters.
- Among the supervised machine learning algorithms, Nearest Neighbor (IB1 and IBK*) are best in accuracy of classification.
- Works for unencrypted HTTP as well as encrypted HTTPS tunnelled applications.
- Scalability: Increasing from 10 applications to 15 applications, the performance did not degrade significantly.

6.1 Conclusions

Application-layer tunnels are a significant security threat for networks because those applications which are restricted by firewalls like bandwidth consuming network games, peer-to-peer file sharing, video and audio streaming, and chat are carried through via the allowed protocols like HTTP, HTTPS and the firewall security policy is thwarted. The identification of the actual application running across the network is important for network management, security and abuse prevention. The existing techniques for identification of applications running across the network, for example port number based identification, packet data analysis techniques are not always successful, especially for encrypted packets. This work has described a statistical approach to detect applications which are running using application level tunnels. Previous work has shown the packet size distribution to be an effective metric for detecting most network applications, both UDP and TCP based applications. In this work packet stream statistical parameters with packet size distribution being an important parameter are shown to be able to identify the network applications successfully. The tunnelled applications are differentiable and identifiable based on the parameter combinations used.

The applications running within protocol tunnels can be identified from their traffic statistical parameters using machine learning algorithms. The significant metrics or parameters for this have been found to be data packet ratio, ratio of large to small packets, byte ratio, percentage of time spent idle and the packet size distribution bins. The temporal attributes of Interarrival Times were found to be not very helpful in classification because of their inconsistency and the fact that they are dependent on the congestion state of the underlying network. The approach doesn't require all packets of a traffic stream to be saved, however there need to be enough packets to form a consistent packet size distribution.

Secondly, the packet size distribution was found to be a significant component of the attribute set. A 30 bin resolution is the optimal one considering the accuracy of classification and processing requirements. The performance of the machine learning algorithms was compared in cases of resolution at 15 bins, 30 bins and 50 bins. While the performance improved in the transition from 15 bin resolution to the

30 bin resolution, the same did not happen for the next transition from 30 bins to 50 bins. Rather the performance for the most cases was similar or even worse in a few cases. It should also be considered that this increase from 30 bin resolution to 50 bin resolution meant adding 20 parameters to every instance of the data set for training or testing. This increases the training time for the classifiers as well, as can be seen from table 5.25 for 58 attributes and table 5.24 for 38 attributes classifications. This fact would be more significant when the data set size becomes much larger to include thousands of instances and when the analysis has to be done in real time. Hence it is concluded that 30 bin resolution should be used for the packet size distribution. The effect of smaller changes in the resolution such as changes to 35 bins or 25 bins or 40 bins has not been considered in this work. It is assumed that the difference introduced by such smaller changes would still be smaller than the difference represented by the 30 bin to 50 bin resolution change. It was also shown that in the case of the Packet size distribution not being available, or being manipulated, discarded, one can still have an identification of the application based on the other statistics as in Figure 5-9, although the confidence of prediction is reduced.

The use of Machine Learning algorithms for the purpose of application identification with the training parameters containing Packet Size Distribution as the key parameter is a novel aspect in this work. The work of Li Bo [49] and Parish [48] were based on Packet Size Distribution bins only and they used simple statistical tests like Chi Squared Test for the purpose of identification. Machine learning algorithms used in this research with the exception of the OneR algorithm have produced fairly accurate results, and any one would be a suitable candidate if a single algorithm detector is desired to be built. The neural network based detector would be a little exacting on time though compared with simpler ones. The distance based lazy learning algorithms , namely, the Nearest Neighbour based on Euclidean distance (IB1) and the Nearest Neighbour based on entropy measured distance (IBK*) have performed better than the rest in all cases and combinations of the input attributes. These two algorithms have the least time for training of the classifier as well. Hence it is recommended that if only one algorithm is to be used in the identification process, it should be either of these two. However it still would

be best to use a few other algorithms in addition, so that the results can be compared. Possibly if the training data set size becomes very large due to inclusion of a large number of applications, the relative accuracy of the algorithms might change, hence a supervised learning test for the algorithms would be advisable after inclusion of a few tens of applications in the training data.

The training data preparation should be taken with care. It is important that the size of the connection to be put for investigation be large enough to produce statistically accurate results, for this it is recommended that the connection should have at least 800 packets.

The issue of scalability is that the method should work for large number of applications. This method was tried using 10 applications with various machine learning algorithms. The results of the nearest neighbour algorithm based on entropy distance, known as IB K* in the WEKA software was the best performing algorithm. In the 10 fold cross validation case of the 10 applications' data, the algorithms showed a classification accuracy of 96.67%as shown in Table 5.24 in the case of IB K* with 10 fold cross validation. Then to have an estimate of the scalability, five new applications were added to the applications' dataset. The same algorithm was tested again and it showed the classification accuracy of 96.1% which is very close to the 10 applications case. Although the dataset had increased by 50%, still the classification results were almost of the same accuracy. Only the results of the 10 fold cross validation were taken because this is a more general and better way to mitigate any bias caused by the particular sample (from training data) chosen for holdout than the other methods described in section 5.1. In 10 fold cross validation, the whole process of training and testing is repeated several times with different random samples from the data set. According to Witten and Frank in [83], this is the standard way of predicting the error rate of a learning technique. The results of the 10 fold cross validation for the 15 application case suggests that the method would be scalable for addition of new applications. However since these results are based on empirical evidence only, there is not a definite way to prove that this statistical approach would work for an arbitrary increase in applications. At least the 15 applications tested are proven to work with empirical evidence.

This method does not require that a specific portion of application packets stream be captured by the detection mechanism, unlike other approaches in traffic classification. Some approaches require that the beginning portion of the TCP connection be captured so that signature mapping can be performed over it, while others require the whole length of the connection be saved. This can be difficult at times for the reason that it requires large amounts of data to be captured and saved. In this technique, however the trace file could be taken from any time of the operation of the application as long as the size of the trace is sufficient to build the statistical packet size distribution and other parameters.

An important feature of this work is that this detection mechanism is able to identify the actual application which is being run inside an application layer tunnel. There is other work in which researchers have tried to detect tunnelling activities in a network; however their focus was mostly to identify whenever one protocol is tunnelled inside another protocol. In other words most tunnelling detection work has tried to identify whether there is tunnelling activity going on or not. That is not as desirable as the approach in this work, as in some cases tunnelling traffic is quite legitimate. For example, in HTTPS tunnelling, several legitimate web pages use HTTPS encryption to communicate their login details or other secure transactions or VPNs.

6.2 Future Work

This method of identification of tunnelled applications requires that the tunnelling application packet traces should be saved first and then trained or tested using the captured pcap trace files in an offline manner. Several different tools are employed to perform different parts of the process including Wireshark and its command line counterpart ‘tshark’ performing the packet capture and filtering work. Then a Matlab program is used for the extraction of the attributes for the connections, and saving the training data file. Machine learning is then performed from the saved files using the WEKA software. The next enhancement to the work which would require a lot of programming effort is to perform these tasks in a single process, so that the results can be obtained in real time while the applications are running.

Although the IB K* algorithm is computationally efficient in addition to having excellent prediction accuracy, it would be useful to investigate the machine learning algorithms and categorize them for computational performance as well and not just for classification accuracy alone. In this work the classification accuracy only was considered when choosing the best performing machine learning algorithm. The reasons why this is acceptable for this work are that firstly the methodology adopted is an offline analysis of packet trace files, and secondly the applications' dataset were not very large in size. However for real-time detection operation, and with a large number of applications in the dataset, the computational performance of the machine learning algorithm is also worth considering.

In this work only the five machine learning algorithms were considered from the WEKA (Waikato Environment for Knowledge Analysis) software. These are:

Naïve Bayes

J48 Decision Tree

Multilayer Perceptron

Nearest Neighbour Euclidean IB1

Nearest Neighbour Entropy based, IB K*

The IB K* was the best algorithm in accuracy and performance of the computations. These algorithms gave acceptable results for the purpose of this study, however in future the usage of a plethora of other machine learning algorithms can also be undertaken to compare the performance and accuracy of the results.

Publications based on this work:

Mujtaba, G. and Parish, D.J., "Detection of Applications Within Encrypted Tunnels Using Packet Size Distributions", ICITST 09, London, November 2009, ISSN 978-1-4244-5647-5.

Mujtaba, G. and Parish, D.J., "Detection of Tunnelled Applications using Packet Size Distributions", PGNet 2009, Liverpool JMU, Liverpool JMU, June 2009, pp 53-57, ISBN 978-1-902560-22-9.

References

[1] “Badwidth Bandits”, white paper, http://www.computerweekly.com/Articles/2010/07/09/241896/White-paper-Bandwidth-bandits-Are-you-keeping-them-in.htm, last accessed on 14th December, 2010.

[2] Bo Li, “Identification of TCP Applications”, Doctoral Thesis, Loughborough University, 2006.

[3] Siau, K., Nah, F., and Teng, J. Internet abuse and acceptable Internet use policy, Commun. ACM 45, 1 (Jan. 2002), 75-79.

[4] Internet use statistics, http://www.connections-usa.com/employee-internet-usage.html accessed last on 27th July, 2010

[5] J. Hill, Bypassing Firewalls: Tools and Techniques, in: 12th Annual FIRST Conference, Chicago, IL, USA, 2000.

[6] HTTPS, “Secure Hypertext Transfer Protocol”, rfc 2660, http://tools.ietf.org/html/rfc2660, last accessed on 14th December, 2010

[7] Ido Dubrawsky (2003-07-29). "Firewall Evolution - Deep Packet Inspection". Security Focus. http://www.securityfocus.com/infocus/1716. Retrieved 1st Oct., 2010.

[8] M. Dusi, M. Crotti, F. Gringoli, L. Salgarelli, Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting, Computer Networks, Volume 53, Issue 1, 16 January 2009, Pages 81-97, ISSN 1389-1286

[9] OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection, Hubert Zimmermann, IEEE Transactions on Communications, vol. 28, no. 4, April 1980, pp. 425 – 432

[10] H. Clare, “Internet and e-mail: use and abuse”, Institute of Personnel and Development, 2000.

[11] Ec. Comer Computer Networks and Internets, prentice-Hall, 1999

[12] F. Kurose W. Rose, Computer Networking, Pearson Education, 2003

[13] RFC 1122, Requirements for Internet Hosts -- Communication Layers, IETF, R. Braden, October 1989

[14] W. Richard Stevens, TCP/IP Illustrated Volume 1, Addison-Wesley, 1994.

[15] Z. Wang and J. Crowcroft, A new congestion control scheme: slow start and search, Computer Communications Rev, 1991

[16] http://www.ietf.org/rfc/rfc0793.txt, last accessed on 14th December, 2010.

[17] Tanenabaum, Andrew .S (2003), Computer Network, fourth edition, New Jersey.

[18] RFC 959 – (Standard) File Transfer Protocol (FTP). J. Postel, J. Reynolds. October 1985.

[19] Gnu Http tunnel, www.nocrew.org/software/httptunnel.html, last accessed on 14th December, 2010.

[20] Wireshark, the world's foremost network protocol analyzer, www.wireshark.org, accessed last on 28th July, 2010.

[21] http://commons.wikimedia.org/wiki/File:Firewall.png, last accessed on 14th December, 2010.

[22] Introduction to firewalls, http://www.tns.com/firewalls.asp, last accessed on 14th December, 2010.

[23] J. Hill, Bypassing Firewalls: Tools and Techniques, in: 12th Annual FIRST Conference, Chicago, IL, USA, 2000.

[24] 'Tunneling for Dollars: Comparing IPSec and PPTP for Extranet Security' By Julie Bort http://www.intranetjournal.com/foundation/tunneling.shtml, last accessed on 14th December, 2010

[25] C. Perkins **"**IP Encapsulation within IP", RFC2003, October 1996

[26] http://www.fire-drill.com/about.html, last accessed on 14th December, 2010

[27] "Socks Protocol Version 5" http://www.ietf.org/rfc/rfc1928.txt, last accessed on 14th December, 2010

[28] M. Dusi, M. Crotti, F. Gringoli, L. Salgarelli, Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting, Computer Networks, Volume 53, Issue 1, 16 January 2009, Pages 81-97, ISSN 1389-1286

[29] http://www.dnstunnel.de last accessed: 25 July, 2010.

[30] Proceedings Of The 2002 IEEE Workshop On Information Assurance "Detecting HTTP Tunnelling Activities" Daniel J. Pack, , William Streilein, Seth Webster, and Robert Cunningham

[31] http://www.artofping.com last accessed: 25th July, 2010

[32] The SSL Protocol: Version 3.0 Netscape's final SSL 3.0 draft.http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt (November 18, 1996)

[33] "The evils of SSL tunneling", http://msmvps.com/blogs/shinder/pages/12268.aspx, last accessed on 14th December, 2010

[34] Youngsong Mun, Hyewon K. Lee, Understanding IPv6 Springer 2005, ISBN: 0387254293

[35] http://www.http-tunnel.com last accessed : 25th July, 2010

[36] http://www.networktunnel.net last accessed : 25th July, 2010

[37] http://www.snort.org last accessed : 25th July, 2010

[38] http://www.bro-ids.org last accessed : 25th July, 2010

[39] K. Borders, A. Prakash, Web tap: detecting covert web traffic, in: CCS'04: Proceedings of the 11th ACM conference on Computer and Communications Security, Washington DC, USA, 2004, pp. 110–120.

[40] M. Dusi, M. Crotti, F. Gringoli, L. Salgarelli, Detection of encrypted tunnels across network boundaries, in: Proceedings of the 43rd IEEE International Conference on Communications (ICC 2008), Beijing, China, 2008.

[41] M. Crotti, M. Dusi, F. Gringoli, L. Salgarelli, Detecting HTTP tunnels with statistical mechanisms**,** in: Proceedings of the 42nd IEEE International Conference on Communications (ICC 2007), Glasgow, Scotland, 2007, pp. 6162–6168.

[42] Using HTTP as an RPC transport," http://msdn.microsoft.com/library/psdk/rpc/pv-http 7h4k.htm.

[43] V. Paxson, Empirically derived analytic models of wide-area TCP connections, IEEE/ACM Transactions on Networking 2 (4) (1994) 316–336.

[44] A.W. Moore, D. Zuev, Internet traffic classification using Bayesian analysis techniques, in: SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Banff, Alberta, Canada, 2005, pp. 50–60.

[45] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection," in Traffic Measurement and Analysis (TMA), Springer-Verlag LNCS 5537, May 2009.

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

[47] AULD, T., MOORE, A.W. and GULL, S.F., 2007. Bayesian neural networks for internet traffic classification. *IEEE Transaction on Neural Networks*, 18, 223-239.

[48] D.J. Parish, K. Bharadia, A. Larkum, I. W. Phillips and M. Oliver "Using Packet Size Distributions to Identify Real-Time Networked Applications" Communications, IEE Proceedings, vol. 150, Aug 2003, pp. 221-227

[49] Bo, L., Parish, D.J., Sandford, J.M. and Sandford, P., "Using TCP Packet Size Distributions for Application Detection", PGNet 2006 Proceedings, Merabiti, Pereira and Abuelma'atti, Liverpool John Moores University, PGNet, Liverpool John Moores University, June 2006, pp 184-189

[50] IANA Port Assignments, "Current list of well-known and registered port assignments, 2003", http://www.isi.edu/in-notes/iana/assignments/port-numbers, Last accessed on 14/12/2010

[51] A. W. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt. Architecture of a Network Monitor, In Passive & Active Measurement Workshop, 2003, (PAM2003), La Jolla, CA, April 2003.

[52] Ketan Bharadia, "Network Application Detection Techniques", Doctoral Thesis, Loughborough University, June 2001

[53] Cubehub HTTP Tunnel, http://twock.com/java/cubehub. Last accessed on 10/12/2010

[54] Pingfu HTTP tunnel solutions, http://www.artofping.com/ Last accessed on 10/12/2010

[55] L. Brinkhoff, GNU httptunnel, <http://www.nocrew.org/software/ httptunnel.html Last accessed on 10/12/2010

[56] R. Mills, The Linux Academy HTTP Tunnel, http://the-linuxacademy.co.uk /downloads.htm Last accessed on 10/12/2010.

[57] RuneScape in Guinness World Records!". Jagex. http://news.runescape.com/newsitem.ws?id=1386. Last accessed on 2008-08-22.

[58] Guinness World Records Gamer's Edition - Records - PC Gaming

[59] MMOG Subscriptions Market Share April 2008". mmogchart.com, Bruce Sterling Woodcock. http://www.mmogchart.com/Chart7.html. Retrieved on 2008-09-24.

[60] www.voipraider.com, Last accessed on 14th December, 2010

[61] www.camfrog.com, Last accessed on 14th December, 2010

[62] Skype: LLC Books,, General Books LLC, 2010. ISBN1155277457, 9781155277455.

[63] http://www.networkworld.com/news/2009/032509-skype-is-largest-international-voice.html?tc=vc=html, Last accessed on 14th December, 2010

[64] International carriers' traffic grows despite Skype popularity". TeleGeography Report and Database. http://www.telegeography.com/cu/article.php?article_id=15656&email=html. Retrieved 2006-12-07.

[65] "Skype Commands 13 Percent of International Phone Calls." The Inquisitr. 3 May 2010. Web. 4 May 2010. <http://www.inquisitr.com/71802/skype-commands-13-percent-of-international-calls/>.

[66] http://secondlife.com/ Last accessed on 14th December, 2010.

[67] "Emmy Online". Emmyonline.tv., http://www.emmyonline.tv/mediacenter/tech_2k7_winners.html. Retrieved 2010-02-19.

[68]"Current user metrics for Second Life". Secondlife.com. http://secondlife.com/xmlhttp/secondlife.php. Retrieved 2010-02-19.

[69] “CounterPath(TM) Wins INTERNET TELEPHONY(R) Magazine's "Product of the Year" Award for 2005” http://goliath.ecnext.com/coms2/gi_0199-5136616/CounterPath-TM-Wins-INTERNET-TELEPHONY.html , last accessed on 14th December, 2010.

[70] Angela Orebaugh, Gilbert Ramirez, Josh Burke, “Wireshark and Ethereal network protocol analyzer toolkit”, Syngress, 2007, ISBN1597490733, 9781597490733

[71] W. Schefler, Statistics Concepts and Applications, The Benjamin/Cummings Publishing Company, 1988.

[72] J. Kitchens, Exploring Statistics: A Modern Introduction to Data Analysis and Inference, 2nd Edition Brooks/Cole Publishing Campany, 1996

[73] Snedecor, George W. and Cochran, William G. (1989), Statistical Methods, Eighth Edition, Iowa State University Press.

[74] How to bypass Internet censorship, http://www.zensur.freerk.com accessed last on 02 September, 2010.

[75] J. Hill, Bypassing Firewalls: Tools and Techniques, in: 12th Annual FIRST Conference, Chicago, IL, USA, 2000.

[76] Network Sniffer and Analyzer, www.wireshark.org, last accessed on 14th December, 2010

[77] tshark - Dump and analyze network traffic, http://www.wireshark.org/docs/man-pages/tshark.html, last accessed on 14th December, 2010

[78] Gilat, Amos MATLAB : an introduction with applications / Hoboken, N.J. : Wiley ; Chichester : John Wiley, 2008.

[79]. Li, Z.; Yuan, R.; Guan, X., “Accurate classification of the internet traffic based on the SVM method ”, *IEEE ICC 2007*, pp. 1373-1378.

[80]. PACK, D.J., STREILEIN, W., WEBSTER, S. and CUNNINGHAM, R., 2002. Detecting HTTP Tunneling Activities, in 2002 IEEE, Workshop on Information Assurance,. 2002. United States Military Academy, West Point, NY: IEEE, 2002.

[81] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.* 11, 1 (November 2009), 10-18. DOI=10.1145/1656274.1656278

[82] http://www.cs.ccsu.edu/~markov/weka-tutorial.pdf, last accessed on 14th December, 2010

[83] Ian H. Witten, Eibe Frank. Data Mining: Practical Machine Learning Tools and Techniques (Second Edition) June 2005

[84] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[85] Quinlan, J. R. 1986. Induction of decision trees. Machine Learning, 1, 81-106.

[86] Patil, Purushottam R., Revankar, Pravin and Joshi, Prashant. The Application of Data Mining for Direct Marketing 2009

[87] Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4), 303–314.

[88] John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. : 12th International Conference on Machine Learning, 108-114, 1995.

[89] AULD, T., MOORE, A.W. and GULL, S.F., 2007. Bayesian neural networks for internet traffic classification. *IEEE Transaction on Neural Networks*, 18, 223-239.

[90] "Websense, Essential Information Protection", http://www.websense.com/content/home.aspx retrieved on 11th February, 2011.

[91] "How to bypass Websense", http://www.thepicky.com/internet/how-do-i-bypass-websense , retrieved on 4th February, 2011.

[92] Rui Wang, Yang Liu, Yuexiang Yang, Xiaoyong Zhou, "Solving the App-Level Classification Problem of P2P Traffic Via Optimized Support Vector Machines," Intelligent Systems Design and Applications, International Conference on, pp. 534-539, Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06) Volume 2, 2006

[93] N. Williams, S. Zander, and G. Armitage, “A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification,” Special Interest Group on Data Communication (SIGCOMM) Computer Communication Review, vol. 36, no. 5, pp. 5– 16, 2006.

[94] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, “Traffic classification through simple statistical fingerprinting,” SIGCOMM Computer Communications, Rev., vol. 37, no. 1, pp. 5–16, 2007.

[95] A. Mena and J. Heidemann. An Empirical Study of Real Audio Traffic. In Proceedings of the IEEE Infocom, pages 101–110, Tel-Aviv, Israel, March 2000.

[96] C. Dewes, A. Wichmann, and A. Feldmann. An analysis of Internet chat systems. In IMC ’03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement, pages 51–64, Miami Beach, FL, USA, October 2003.

[97] Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., Salamatian, K.: Traffic classification on the fly. SIGCOMM Comput. Commun. Rev. 36 (2006) 23–26

Appendix A: The Matlab Code

Matlab Code: m files

Here the matlab code used in this work is attached. There are 4 files:

Chi2gui.m: the file used for chi square analysis of the PSDs from tracefiles.

Modegui.m: this file is used to input the pcap wireshark file

Dostsh.m: the code to convert the pcap to text file

Onecon.m: the code to obtain traffic parameters from the TCP conections

1. Chi2gui.m

```
clc
close all
clear all;
display('observed excel file ?');


 [fname, dirpath] = uigetfile('*.xls')


     ob_xl_fp = xlsread([dirpath fname]); % Observed excel file

      [fname2,dirpath2] = uigetfile('*.xls')


     exp_xl_fp = xlsread([dirpath2 fname2]); % expected excel
  file


     %removing zero packet sizes
     k = 1;
     for n = 1:max(ob_xl_fp(:,1))

         if(ob_xl_fp(n,2) ~= 0.0)
           newobfile(k,2) = ob_xl_fp(n,2);
           newobfile(k,1) = k;
           k = k +1;

         end

     end

     k = 1;
     for n = 1:max(exp_xl_fp(:,1))

```

```
35.          if(exp_xl_fp(n,2) ~= 0.0)
36.            newexfile(k,2) = exp_xl_fp(n,2);
37.            newexfile(k,1) = k;
38.            k = k +1;
39.
40.          end
41.
42.      end
43.
44.      %  for a = 1:4
45.
46.
47.      no_of_bins = input('enter the no of bins for the
   distribution   '); % no. of Bin
48.      bin_size = ceil(1460/no_of_bins)
49.
50.
51.      % initializing arrays for observed and expected
   distributions
52.      % clear o_distribution;
53.      % clear e_distribution;
54.      o_distribution(no_of_bins) = 0.0;
55.      e_distribution(no_of_bins) = 0.0;
56.
57.      % Processing to bin allocations for observed file
58.      for i = 1:max(newobfile(:,1))
59.          binpos = ceil(newobfile(i,2)/bin_size);
60.          o_distribution(binpos) = o_distribution(binpos)+1;
61.      end
62.
63.      % removing the smaller frequencies
64.      for m = 1:no_of_bins
65.           if((o_distribution(m))<=5)
66.               o_distribution(m)=0;
67.            end
68.      end
69.
70.      % clear o_dist_nor;
71.      o_dist_nor = o_distribution/max(newobfile(:,1)); %
   Normalization process
72.
73.
74.
75.      % Processing to bin allocations for expected file
76.      for i = 1:max(newexfile(:,1))
77.          binpos = ceil(newexfile(i,2)/bin_size);
78.          e_distribution(binpos) = e_distribution(binpos)+1;
79.      end
80.
81.      % removing the smaller frequencies
82.
83.      for m = 1:no_of_bins
84.          if((e_distribution(m))<=5)
```

```
85.             e_distribution(m) = 0;
86.         end
87.
88.     end
89.
90.     % clear e_dist_nor;
91.     e_dist_nor = e_distribution/max(newexfile(:,1)); %
   Normalization process
92.
93.     % clear theory_dist;
94.     theory_dist = ((o_dist_nor*200)+(e_dist_nor*200))*0.5;
95.     %
96.     % clear DOF;
97.     DOF  = -1;
98.     % clear chi_val;
99.     chi_val = 0;
100.    for i = 1:no_of_bins
101.        if (theory_dist(i) ~= 0.0)
102.            DOF = DOF+1;
103.            chi_val = chi_val+(((o_dist_nor(i)*200)-
   (theory_dist(i)))^2)/(theory_dist(i));
104.        end
105.    end
106.    display('the chi value is :')
107.    display(chi_val)
108.    display('for the degree of freedom = ')
109.    display(DOF)
110.
```

2 Modegui.m : this file is used to input the pcap wireshark file

```
3  function varargout = modegui(varargin)
4  % MODEGUI M-file for modegui.fig
5  %      MODEGUI, by itself, creates a new MODEGUI or raises the
   existing
6  %      singleton*.
7  %
8  %      H = MODEGUI returns the handle to a new MODEGUI or the
   handle to
9  %      the existing singleton*.
10 %
11 %      MODEGUI('CALLBACK',hObject,eventData,handles,...) calls
   the local
12 %      function named CALLBACK in MODEGUI.M with the given input
   arguments.
13
14 %      instance to run (singleton)".
15 %
16 % See also: GUIDE, GUIDATA, GUIHANDLES
17
18 % Edit the above text to modify the response to help modegui
```

```
19
20 % Last Modified by GUIDE v2.5 08-Mar-2010 13:59:55
21
22 % Begin initialization code - DO NOT EDIT
23 gui_Singleton = 1;
24 gui_State = struct('gui_Name',       mfilename, ...
25                    'gui_Singleton',  gui_Singleton, ...
26                    'gui_OpeningFcn', @modegui_OpeningFcn, ...
27                    'gui_OutputFcn',  @modegui_OutputFcn, ...
28                    'gui_LayoutFcn',  [] , ...
29                    'gui_Callback',   []);
30 if nargin && ischar(varargin{1})
31     gui_State.gui_Callback = str2func(varargin{1});
32 end
33
34 if nargout
35     [varargout{1:nargout}] = gui_mainfcn(gui_State,
   varargin{:});
36 else
37     gui_mainfcn(gui_State, varargin{:});
38 end
39 % End initialization code - DO NOT EDIT
40
41
42 % --- Executes just before modegui is made visible.
43 function modegui_OpeningFcn(hObject, eventdata, handles,
   varargin)
44 % This function has no output args, see OutputFcn.
45 % hObject    handle to figure
46 % eventdata  reserved - to be defined in a future version of
   MATLAB
47 % handles    structure with handles and user data (see GUIDATA)
48 % varargin   command line arguments to modegui (see VARARGIN)
49
50 % Choose default command line output for modegui
51 handles.output = hObject;
52
53 % Update handles structure
54 guidata(hObject, handles);
55
56 % UIWAIT makes modegui wait for user response (see UIRESUME)
57 % uiwait(handles.figure1);
58
59
60 % --- Outputs from this function are returned to the command
   line.
61 function varargout = modegui_OutputFcn(hObject, eventdata,
   handles)
62 % varargout  cell array for returning output args (see
   VARARGOUT);
63 % hObject    handle to figure
64 % eventdata  reserved - to be defined in a future version of
   MATLAB
65 % handles    structure with handles and user data (see GUIDATA)
```

```
66
67 % Get default command line output from handles structure
68 varargout{1} = handles.output;
69
70
71 % --- Executes on button press in pushbutton1.
72 function pushbutton1_Callback(hObject, eventdata, handles)
73 % hObject    handle to pushbutton1 (see GCBO)
74 % eventdata  reserved - to be defined in a future version of
   MATLAB
75 % handles    structure with handles and user data (see GUIDATA)
76 %getpcapfile;
77 [fname, dirpath] = uigetfile('*.pcap')
78 dostsh(fname,dirpath);
79 % --- Executes on button press in pushbutton2.
80 function pushbutton2_Callback(hObject, eventdata, handles)
81 % hObject    handle to pushbutton2 (see GCBO)
82 % eventdata  reserved - to be defined in a future version of
   MATLAB
83 % handles    structure with handles and user data (see GUIDATA)
84
85
86 % --- Executes on button press in pushbutton3.
87 function pushbutton3_Callback(hObject, eventdata, handles)
88 % hObject    handle to pushbutton3 (see GCBO)
89 % eventdata  reserved - to be defined in a future version of
   MATLAB
90 % handles    structure with handles and user data (see GUIDATA)
91
92 exit;
93 % --- If Enable == 'on', executes on mouse press in 5 pixel
   border.
94 % --- Otherwise, executes on mouse press in 5 pixel border or
   over pushbutton3.
95 function pushbutton3_ButtonDownFcn(hObject, eventdata, handles)
96 % hObject    handle to pushbutton3 (see GCBO)
97 % eventdata  reserved - to be defined in a future version of
   MATLAB
98 % handles    structure with handles and user data (see GUIDATA)
99
```

3 Dostsh.m: the code to convert the pcap to text file

```
function dostsh(fname,dirpath)
%fullname = [dirpath fname]

tshcmd1                  =                  ('tshark                  -o
column.format:"No.","%m","Time","%t","Source","%s","Destination","%d",
"srcport","%uS","dstport","%uD","len","%L","Protocol","%p" -r ');
fulln = [tshcmd1 fname '>' fname '.txt'];
```

```
%this makes the whole command complete with variable input file name.
the resulting
%text file will be named as fname.txt e.g voiptunnel1.pcap.txt

dos(fulln)
onecon([fname '.txt']);
end
```

1. onecon.m: the code to obtain traffic parameters from the TCP conections

```
function  g = onecon(fname)

   fid = fopen(fname);
[C] = textscan(fid, '%u %f %s %s %s %f %f %f %s');
fclose(fid);
index  = C{1,1};
time  = C{1,2};
srcaddr  = C{1,3};
data4  = C{1,4};
dstaddr  = C{1,5};
srcport  = C{1,6};
dstport  = C{1,7};
size  = C{1,8}; % this is ip.size, not tcp.size
prot  = C{1,9};
j = 0;
k = 0;
size = size -54; %making it tcp.size
for i = 1:max(index)
    if
(strcmp(srcaddr(i),'192.168.15.104')||strcmp(srcaddr(i),'192.168.4.2')
||strcmp(srcaddr(i), ...

'192.168.15.101')||strcmp(srcaddr(i),'192.168.1.5')||strcmp(srcaddr(i)
,'192.168.15.102') ...

||strcmp(srcaddr(i),'192.168.1.2')||strcmp(srcaddr(i),'158.125.48.160'
) ...

||strcmp(srcaddr(i),'192.168.1.6')||strcmp(srcaddr(i),'192.168.1.10'))
      j = j+1;
        connl2r(j).index = j;
        connl2r(j).time = time(i);
        connl2r(j).srcaddr = srcaddr(i);
        connl2r(j).dstaddr = dstaddr(i);
        connl2r(j).srcport = srcport(i);
        connl2r(j).dstport = dstport(i);
        connl2r(j).size = size(i);
        connl2r(j).prot = prot(i);

    else

        k = k+1;
```

```
        connr2l(k).index = k;
        connr2l(k).time = time(i);
        connr2l(k).srcaddr = srcaddr(i);
        connr2l(k).dstaddr = dstaddr(i);
        connr2l(k).srcport = srcport(i);
        connr2l(k).dstport = dstport(i);
        connr2l(k).size = size(i);
        connr2l(k).prot = prot(i);
end
end

% other metrics for this connection

%1. Data rates(bytes/sec for l2r and r2l: drl2r, drr2l
%1.a drl2r:
timel2r = connl2r(length(connl2r)).time - connl2r(1).time
bytesl2r = 0;
for m = 1:length(connl2r)
    bytesl2r  = bytesl2r + connl2r(m).size;
end

drl2r = bytesl2r/timel2r

%2 drr2l:
timer2l = connr2l(length(connr2l)).time - connr2l(1).time
bytesr2l = 0;
for m = 1:length(connr2l)
    bytesr2l  = bytesr2l + connr2l(m).size;
end

drr2l = bytesr2l/timer2l

%3. data packet ratio: dpktr: ratio of data packets downstream and
upstream
%dpktr = dnzctr/upnzctr(downstream nonzerocounter/upstreamnonzeroctr)

upnzctr = 0;
dnzctr = 0;
for m = 1:length(connl2r)
    if (connl2r(m).size)
        upnzctr  = upnzctr + 1;
    end
end
for m = 1:length(connr2l)
    if (connr2l(m).size)
        dnzctr  = dnzctr + 1;
    end
end
dpktratio = dnzctr/upnzctr

%4. ByteRatio: ratio of bytes downstreamto upstream
byteratio = bytesr2l/bytesl2r

%5.Ratio of large and small packets:
%large packets are data>300bytes
%small packets are data<300bytes
ctrlarge = 0;
ctrsmall = 0;
for m = 1:length(connr2l)
```

```
    if connr2l(m).size > 300
        ctrlarge = ctrlarge+1;
    else
        ctrsmall = ctrsmall+1;
    end
end
    if ctrsmall == 0
        ratio_sl = ctrlarge;
    else
    ratio_sl_ds = ctrlarge/ctrsmall;
    end


  %6.maxIAT for a data packet downstream
  % or here just maximum iat downstream.
  for m = 2:length(connr2l)
      iatds(m-1) = connr2l(m).time - connr2l(m-1).time;
  end
  maxiatds = max(iatds);
  %8. minatds
  miniatds = min(iatds);
  %iatds
  %9. max////// upstream
  for m = 2:length(connl2r)
      iatups(m-1) = connl2r(m).time - connl2r(m-1).time;
  end
  maxiatups = max(iatups);
  %10. minatups
  miniatups = min(iatups)


  %10. % time spent idle downstream: idle time is 2s or more
  %. timer2l is total time in downstream
  sum_idle_ds = 0;
  for m = 1: length(iatds)
  if iatds(m)>= 2.00
      sum_idle_ds = sum_idle_ds + iatds(m);
  end
  end
 pc_idle_ds = sum_idle_ds/timer2l*100;


  % 11 %time spend idle upSstream
  sum_idle_ups = 0;
  for m = 1: length(iatups)
  if iatups(m)>= 2.00
      sum_idle_ups = sum_idle_ups + iatups(m);
  end
  end
 pc_idle_ups = sum_idle_ups/timel2r*100;
  % PSD with 15 bins
  %12.psd Remote to local:
  for m = 1:length(connr2l)
      sizesr2l(m) = connr2l(m).size;
  end
  %removing zero sizes
  k = 1;
  for g = 1: length(sizesr2l)
      if sizesr2l(g) ~= 0
          size_nz(k) = sizesr2l(g);
```

```
            k = k+1;
        end

    end
    no_of_bins = 30;
    bin_size = ceil(1460/no_of_bins);

    distribution(no_of_bins) = 0.0;
    % Processing to bin allocations for observed file
try
    for i = 1:length(size_nz)
    binpos = ceil(size_nz(i)/bin_size);
    distribution(binpos) = distribution(binpos)+1;
    end
catch %means there were no nonzero sizes
    % don't need to do anything, distribution(nobins)=0 holds
end
%normalization:
dist_nor_r2l(no_of_bins) = 0.0;
for m = 1:no_of_bins
    dist_nor_r2l(m) = distribution(m)/length(sizesr2l);
end

   appname = input('enter application name', 's');
   appncell = {appname}

%load var_saved;% MAKE SURE IT EXISTs BEFORE
%nrows = length(all_metnumeric(:,1));


all_metnumeric = [drl2r drr2l dpktratio byteratio ratio_sl_ds
pc_idle_ds pc_idle_ups dist_nor_r2l]
%save ('var_saved','all_metnumeric');
% Convert numeric data, T, to a cell array of doubles
allmCell= num2cell(all_metnumeric);
% Concatenate dataCell and Tinto one cell array
try
    load full_var;
    nrows = length(outCell(:,1));

    outCell(nrows+1,:) = [allmCell appncell];


    save('full_var','outCell');
catch %means that the full_var containging outcell doesn't exist yet
    outCell = [allmCell appncell];
    save('full_var','outCell');
end

xlswrite('file2.xls',outCell);
```

Appendix B: The OSI and TCP/IP Internet Models

To understand the tunnelling activity, it is useful to have a little reminder of the TCP/IP model of the internet, its different layers and related protocols. Hence a brief description of TCP/IP follows:

TCP/IP Reference Model

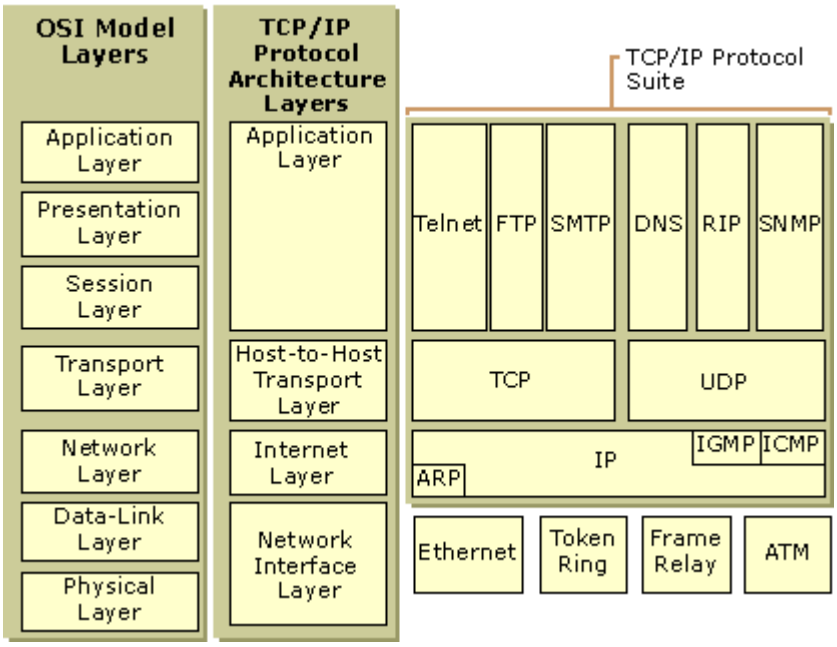


Figure B-1 OSI and TCP/IP Models and associated protocols

Figure B-1 is the hypothetical seven-layer Open Systems Interconnection OSI [9] model which is the foundation of the TCP/IP reference model described in the Internet Engineering Task Force (IETF) RFC 1122 [13], used in practice. Each layer has got its specific functionality and communicates directly with the layer immediately above and below it. The upper three layers of OSI i.e Application Layer, Presentation Layer and Session Layer are concerned with user services, applications, and activities whereas the lower four are concerned more with the actual transmission of information.

The TCP/IP reference model is implemented in the worldwide Internet, which is the largest communication network in existence. The second column in the above figure is showing the correspondence between the TCP/IP Reference Model and OSI Model. The TCP/IP model has done away with the session or presentation layers and the necessary functionality is implemented in the application layer itself. TCP and UDP protocols have been defined at the Transport Layer, while the Internet Layer defines the packet format and protocol called the Internet Protocol. The TCP/IP model does not specify details about the layers below the Internet Layer, but points out that the host has to connect to the network using some protocols over which IP packets can be sent/received.

Application Layer

A survey of nearly 200 international companies carried out by Infosec, Netpartners and Secure Computing Magazine estimated that a typical large company (1,000 employees) could be losing £2.5 million per year through employees' use of Internet for non-business purposes – an average loss of £2,500 per employee [10]. The Application layer of the TCP/IP model gives networking applications the ability to use the services of the other layers and defines the protocols that applications need to use for data communication. There are many Application layer protocols and new protocols are always being developed.

Some of the widely used application layer protocols are:

The Hypertext Transfer Protocol (HTTP) is used to transfer files in the World wide web and the files constitute the web pages.

The File Transfer Protocol (FTP) is the protocol which is used for interactive file transfer.

The Simple Mail Transfer Protocol (SMTP) is the application layer protocol whose purpose is the transfer of e-mails.

Telnet, ssh, HTTPS etc are some other application layer protocols

There are other application layer protocols that help facilitate the use and management of TCP/IP networks:

The Domain Name System (DNS) is used translate a host name to an IP address and vice versa also called host resolution.

The Routing Information Protocol (RIP) is a routing protocol for the purpose of communicating routing information on an IP internetwork.

The Simple Network Management Protocol (SNMP) is used between a network management console and network devices (routers, bridges, intelligent hubs) to collect and exchange network management information.

Transport Layer and TCP/UDP Protocols

The Transport layer (also known as the Host-to-Host Transport layer) is responsible for providing the Application layer with session and datagram communication services. The major transport layer protocols are TCP and UDP.

UDP

The User Datagram Protocol (UDP), defined by IETF RFC 768, provides a simple, one-to-one or one-to-many, connectionless, unreliable communications service. The UDP header has a source port identifier and destination port identifier, which are used to identify the application running on the host. By connectionless means that a datagram can be sent at any moment without prior negotiation or connection establishment [11]. The datagram is just sent and it is hoped that the receiver is able to handle it. There is no guarantee that the datagram will be delivered to the destination host. Not only could the datagram be undelivered, but it could be also delivered in an incorrect order. It means a packet can be received before another one, even if the second has been sent before the first one received. [12]

UDP datagram has a header and a data portion. The header consists of four fields, of which are source port, destination port, length and checksum. The 16 bit field specifies the length in bytes of the entire datagram both header and data. The minimum length is 8 bytes since that’s the length of the header. The field size sets a theoretical limit of 65,527 bytes for the data carried by a single UDP datagram. Checksum consists of 16 bits used for error-checking of the header and data.

UDP can transport many of the Internet's popular application protocols. UDP applications do not require reliability mechanisms and may even be hindered by them, including the real time applications such as audio and video.

	First 16 bits	**16 – 31 bits**
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

Figure B-2 UDP Packet structure.

TCP:

The TCP (Transmission Control Protocol), is described in RFC-793[16]. TCP is a connection-oriented protocol that is responsible for reliable communication between two end processes [14]. A three way handshake process of connection establishment must be completed before actual data exchange starts. The data transfer operates in both directions i.e. send and receive in a single session. TCP guarantees that all data sent will be received without any error (reliable) and in the correct order. Should any error occur, it will automatically be corrected (retransmitted as needed) or the error will be notified if it cannot be corrected [15].

The TCP Protocol Data Unit PDU also has a header part and data part as shown in figure B-3. There are 11 fields making the header, and 10 are required for every pdu.

Source port and Destination port are 16 bit fields used to identify the sending port and receiving port of the communicating machines respectively. Sequence Number

and Acknowledgement number are for ordering of packets. Data offset is a 4-bit field which specifies the size of the TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes. This field gets its name from the fact that it is also the offset from the start of the TCP packet to the data [16]. Then the 4 bits of reserved field are for future. The 8 bit flags are to identify different flags as:

URG : Urgent

ACK: Acknowledgement

PSH: Push.

RST: Reset the connection.

SYN: Synchronize sequence numbers.

FIN: No more data from sender.

The window field signifies the number of bytes the sender is expecting to receive starting from the acknowledgement field value. The 16-bit checksum field is used for error-checking of the header and data.

TCP is the transport protocol for many of the Internet's most popular application protocols including the World Wide Web, e-mail, File Transfer Protocol and Secure Shell.

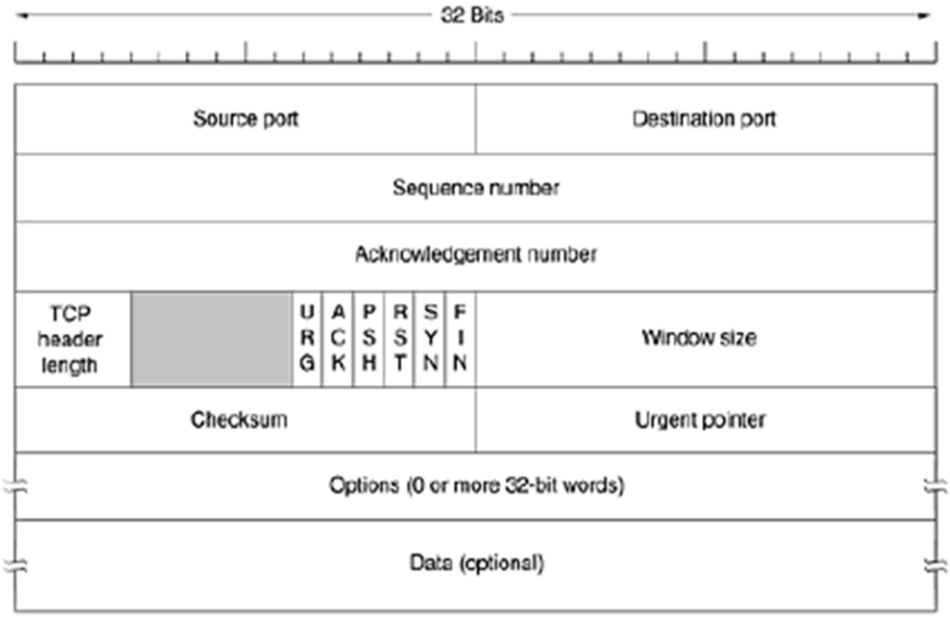


Figure B-3 TCP Segment Structure [17]

Internet Layer

The Internet layer is responsible for addressing, packaging, and routing functions. The core protocols of the Internet layer are IP, ARP, ICMP, and IGMP. The internet layer of TCP/IP is similar to the Network Layer of OSI model.

The Internet Protocol (IP) is a protocol which performs the functions of IP addressing, routing of IP packets and also the fragmentation and reassembly of packets.

The Address Resolution Protocol (ARP) performs the address translation between the IP address and the hardware address on the network.

The Internet Control Message Protocol (ICMP) and Internet Group Management Protocol IGMP protocols for diagnostic and management purposes.

Appendix C: The test Data set used for Validation of Weka Results

b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15	…
0.02	0	0	0.02	0	0	0	0	0	0	0	0.01	0	0	0	…
0.01	0	0	0.01	0	0	0	0	0	0	0	0.01	0	0	0	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.01	0	0	0.01	0	0	0	0	0	0	0	0.01	0	0	0	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.03	0	0.01	0.03	0	0	0	0.02	0	0	0	0.01	0	0	0	…
0.16	0	0	0.05	0	0	0	0.02	0	0	0	0.02	0	0	0	…
0.33	0	0	0.08	0	0	0	0.01	0	0	0	0	0	0	0	…
0.22	0	0.01	0.06	0	0	0	0.02	0	0	0	0.02	0	0	0	…
0.3	0	0	0.12	0	0	0	0.01	0	0	0	0.01	0	0	0	…
0.02	0.03	0.04	0.03	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	…
0.02	0.05	0.03	0.02	0.03	0.01	0.01	0.01	0.01	0	0.01	0.01	0.01	0.01	0.01	…
0.03	0.03	0.04	0.03	0.02	0.02	0.01	0.01	0.01	0.01	0	0.01	0.01	0.01	0.01	…
0.77	0.19	0.01	0.01	0	0	0.02	0	0	0	0	0	0	0	0	…
0.93	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.69	0.3	0.01	0	0	0	0	0	0	0	0	0	0	0	0	…
0.8	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.77	0.2	0	0.01	0	0	0.02	0	0	0	0	0	0	0	0	…
0.56	0.44	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.55	0.44	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.56	0.44	0	0	0	0	0	0	0	0	0	0	0	0	0	…
0.04	0.07	0.14	0.24	0.28	0.12	0.04	0.02	0.01	0.01	0.01	0	0	0	0	…
0	0.04	0.11	0.24	0.42	0.09	0.03	0.02	0.02	0.01	0	0	0	0	0	…
0.14	0.01	0.5	0.03	0.05	0.03	0.01	0.01	0.01	0	0	0	0	0	0	…
0.09	0.01	0.53	0.03	0.09	0.02	0.01	0.01	0.01	0	0	0	0	0	0	…
0.14	0.01	0.5	0.03	0.05	0.03	0.01	0.01	0.01	0	0	0	0	0	0	…
0.19	0.04	0.07	0.07	0.03	0.03	0.02	0.02	0.01	0.01	0.02	0.02	0.01	0.02	0.01	…
0.34	0.04	0.04	0.05	0.03	0.02	0.02	0.01	0.02	0.02	0.01	0.01	0.01	0.01	0.01	…
0.18	0.06	0.06	0.07	0.04	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.01	0.02	0.02	…
0.33	0.04	0.07	0.07	0.03	0.03	0.03	0.01	0.02	0.01	0.02	0.01	0.01	0.01	0.02	…
0.49	0.2	0.04	0.04	0.03	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.01	0	0	…
0.73	0.07	0.02	0.03	0.01	0.03	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0	0	…
0.59	0.23	0.05	0.04	0.03	0.02	0.01	0.01	0.01	0	0	0	0	0	0	…
0.43	0.32	0.06	0.06	0.04	0.02	0.01	0.02	0.01	0.01	0.01	0	0	0	0.01	…
0	0.01	0.02	0.01	0.02	0.04	0.06	0.08	0.08	0.1	0.1	0.08	0.09	0.05	0.05	…
0.01	0.01	0.02	0.04	0.04	0.04	0.06	0.05	0.05	0.05	0.05	0.05	0.06	0.06	0.07	…
0.01	0	0	0.02	0.01	0	0	0	0	0	0	0	0	0	0	…
0.01	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	…

b16	b17	b18	b19	b20	b21	b22	b23	b24	b25	b26	b27	b28	b29	b30	app
0.01	0	0	0	0.13	0	0	0	0.66	0	0	0.11	0	0	0.02	VoipRaider
0.01	0	0	0	0.18	0	0	0	0.59	0	0	0.19	0	0	0.01	VoipRaider
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VoipRaider
0.01	0	0	0	0.2	0	0	0	0.57	0	0	0.2	0	0	0.01	VoipRaider
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Skype
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Skype
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Skype
0.03	0	0	0	0.22	0	0	0	0.47	0	0	0.16	0	0.01	0	XLite
0.03	0	0	0	0.2	0	0	0	0.41	0	0	0.11	0	0	0	XLite
0.02	0	0	0	0.16	0	0	0	0.3	0	0	0.09	0	0	0	XLite
0.03	0	0	0	0.19	0	0	0	0.36	0	0	0.08	0	0.01	0	XLite
0.02	0	0	0	0.13	0	0	0	0.33	0	0	0.09	0	0	0	XLite
0.02	0.02	0.02	0.03	0.03	0.03	0.05	0.02	0.03	0.03	0.15	0	0.01	0.01	0.29	Camfrog
0.01	0.01	0.02	0.03	0.03	0.04	0.05	0.01	0.04	0.04	0.14	0.01	0.01	0.02	0.29	Camfrog
0.02	0.01	0.02	0.02	0.03	0.03	0.06	0.02	0.03	0.03	0.14	0.01	0.01	0.01	0.3	Camfrog
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MOHA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MOHA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MOHA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MOHA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MOHA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Quake3A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Quake3A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Quake3A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Quake3A
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Quake3A
0	0	0	0	0	0.01	0.01	0	0	0	0	0	0.05	0.13	0	LoU
0	0.01	0	0	0	0	0.01	0.01	0	0	0	0	0.03	0.11	0	LoU
0	0	0	0	0	0.01	0.01	0	0	0	0	0	0.05	0.13	0	LoU
0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.24	IVisit
0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0	0	0.01	0.01	0.01	0.01	0.2	IVisit
0.02	0.02	0.02	0.01	0.02	0.02	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.2	IVisit
0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.12	IVisit
0.01	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0.01	WOW
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.02	WOW
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	WOW
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WOW
0.04	0.04	0.02	0.02	0.01	0.02	0.01	0.01	0.01	0	0.01	0	0	0	0.01	QuakeLive
0.03	0.04	0.04	0.04	0.03	0.04	0.04	0.01	0.02	0.02	0.01	0.01	0.01	0	0.01	QuakeLive
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	QuakeLive
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	QuakeLive