Performance Evaluation of Re-Organised Access Lists in Packet Filters for Communication Devices

Faheem Bukhatwa

Department of Computer Science
National University of Ireland, Dublin
Belfield, Dublin 4, Ireland

A thesis submitted to the National University of Ireland, Dublin
for the degree of Doctor of Philosophy in the Faculty of Science

July, 2004

Supervisor: Dr. Joseph Carthy
Nominator: Prof. Mark Keane

Contents

List of Figures

[1] All figures in this thesis are the work of the author unless otherwise stated.

List of Tables

Abstract

Firewalls are hardware and software systems that protect a network from attacks coming from the Internet. Most packet filtering operations use packet classifiers known as access lists. Packet filtering firewalls are efficient, fast and provide a good level of security. Packet filtering based firewalls provide protection through granting or denying access to passing packets. Each individual incoming or outgoing packet is inspected against a list of rules in an access list. The result of this inspection determines the decision to be made.

The use of packet filtering based on access lists is no longer limited to firewalls and routers; many other devices and functions in communication systems are becoming dependent on packet filtering. In the future, the great expansion in communication and the increased number of services and users on the Internet is expected to lead to more packets being exchanged. It is also expected that higher levels of security will be needed requiring more rules in access lists. All this places more pressure on packet filtering devices to provide greater security at higher performance levels without causing a communication bottleneck. Many approaches have been suggested to improve the performance of firewalls with varying degrees of success. In this research, a new approach to improving the performance of access list-based packet filtering is presented.

The proposed approach suggests that rearranging rules in access lists can provide better performance in packet filtering for a particular pattern of packet stream. This research aims at investigating this claim and providing definitive evidence for or against such a claim. The approach is based on observing packet streams arriving into a network device, being able to recognise packet patterns in the past and predicting future patterns. It is also based on dividing the rules in an access list into classes and rearranging these classes of rules in the list to suit a particular packet pattern. As a consequence of this work two novel algorithms PRCW (Packet-Rule Cost Weighing algorithm) and HCE (High Cost Elimination

algorithm) were developed, implemented and evaluated. Those algorithms make it possible to predict the best arrangement for an access list for a particular packet pattern which will provide the best performance. This thesis provides a detailed analysis of the proposed scheme. This is achieved through simulating packet filtering operations and providing comparisons of the performance of thousands of simulations.

Acknowledgements

Many thanks to my supervisor Dr. Joseph Carthy for his assistance, advice and most important the unlimited support one can count on. I also wish to thank the following members of the teaching staff at the Department of Computer Science for their support: Mr. Gregory O'Hare, Mr. Henry McLoughlin, Dr. Tahar Kechadi, Dr. Damian Dalton, Dr. Nicholas Kushmerick and Mr. John Dunnion.

I also wish to thank all the postgraduate students in Room A007, at the Department of Computer Science, namely: Nikita Shmidt, Mikhail Sogrin, Thomas Phelan, Buyrhan Hyusein, Kan Wang and from room A005 Georgios Tagalakis. Thanks also to other individuals I was involved with during the course of my research at varying degrees, they are: Séamus Ó Ciardhuáin, Brian McLernon, Tadhg O'Meara, Rinat Khoussainov, Alex Ufimtsev, Pavel Gladyshev, Omar Ashagi and Yuri Ivanov. Special thanks to the Statistician Mr. Adel El-Sherkasi for his help and consultation.

My thanks also to the technical support staff Anthony O'Gara, Paul Martin, and Brian Redmond for their technical support. The extra help from Séamus Ó Ciardhuáin, Anthony O'Gara and Georgios Tagalakis was highly appreciated. Most importantly, I would like to state my appreciation to the secretariat staff Clare Comerford, Angela Logue and Patricia Geoghegan.

Finally, special thanks to my children and my wife.

Chapter 1

Introduction

This thesis describes the design and implementation of a system for the simulation of access list rules, filtering packets arriving into a communication device. The following chapters investigate existing packet filtering and firewalling schemes. The proposed scheme is discussed in detail. The method of implementing the new approach is described in detail outlining the proposed model and algorithms. The overall objective of this work is to investigate a possible improvement in performance of access list based classifiers. In other words, whether access list performance can be improved through the reorganising of these lists. The main aim is to establish better performance, at a low cost and using an easily regenerated solution which can be customised and implemented for access list filtering.

This chapter explains the motivation for the proposed approach and the implementation of the reorganised access lists in packet filters used in communication devices. It outlines the work carried out and describes the results that were obtained.

1.1 Motivation

Filtering access to and from the Internet has largely been implemented by firewalls. Firewalls are the most common, often the only form of protection and usually the first form of protection against the dangers of the Internet. Packet filtering is also referred to as packet classification. The ability to classify packets

is becoming necessary for many other areas of communication systems for a variety of reasons. For example different services can be offered to different classes of packets. Virtual Private Networks (VPN) only permit packets from predefined sources. Quality of Service (QoS) makes some network connections dedicated to some particular services. Also, packet classification is largely used as a mechanism for the implementation of a wide variety of network security policies such as the prevention of unauthorized access to a subnet, as in firewalls, or the prevention of some Internet attacks such as Denial of Service (DoS) attacks. Routers use packet classification for route determination and functions such as traffic shaping and load balancing. The idea is that a packet is classified into a particular class known as a flow. Based on that classification, a particular event occurs with regard to the packet. The purpose of this classification can be one of many such as protection, flow control, routing, load balancing or tunnelling.

The growth of network and Internet communication has placed greater emphasis on two important issues, namely security and performance. As the volume of communication increases and the requirement of availability becomes more important, the demand for better performance and higher levels of security will become even more important(Hazelhurst *et al.*, 1998).

As the number of hosts has increased many fold over the years, so have communication activities. The Internet has grown to be the major medium for many organisations' and corporations' activities, like banking, insurance, retail and many more. Unfortunately, that has meant an overall decrease in security and an increased number of different types of unwanted access or attacks.

On one hand, the number of packets flowing through networks is expected to increase due to the increased number of users and increased level of communication utilisation. On the other hand, higher levels of security will require more rules on access lists, as each individual incoming or outgoing packet is inspected against each rule in the list of rules. Firewalls are likely to become a bottleneck in communication systems unless improved performance can be achieved. This necessitates a search for better or more efficient methods of implementing firewalls. Packet filtering firewalls are efficient, fast and provide a good level of security (CERT, 1996) and have stood the test of time. Finding ways to improve the performance of packet filtering will no doubt help improve performance in all communication areas dependent on packet filtering. Improving the performance of packet filtering is of great importance for the continued development of the safe communication on the Internet.

1.2 Problem Definition

Very briefly, access list-based packet filtering works in the following way. The filtering device will have a list of rules. Each rule contains two parts, a decision and an inspection. The second part shows the inspection to be carried out on the packet. The first part is the decision to accept or reject the packet, only to be applied if the packet conforms with the inspection. This is like the logic of an "IF" statement in programming: $if < condition > then < action >$. In most implementations of access list filtering, as soon as a rule is found in the list to which the packet conforms (i.e. the result of applying the condition to the packet is *true*), then the inspection stops, and the packet is accepted or rejected according to the decision in the rule. There is a small time cost for each rule in the list used to inspect the packet. The total time for a packet to be accepted or rejected will depend on the number of rules used to inspect the packet. The maximum time for an individual packet will be the time needed for all the rules in the list to inspect the packet. The maximum total time cost for a large number of packets will depend on the number of packets and the number of rules in the list.

Traditionally packet filtering was used in firewalls for providing security. Packet filtering has also been applied to provide many other functionalities in communication systems. Packet filtering is popular due to the fact that it is hidden from the user, and in many cases requires little or no maintenance for years. In general, packet filtering is also fast, effective and efficient. Access list-based packet filtering seems to be excellent as long as it does not negatively affect the communication system it is meant to protect.

A network device with packets arriving at the rate of 20 packets per second and an access list of 20 rules is very different to a device with packets arriving at the rate of 2,000,000 packets per second and an access list of 2000 rules. In the latter case, inspecting all the packets can cause large delays which can be well below the acceptable requirements for many services. In such cases, packet filtering can be the bottleneck for communication.

Consider the following question: is it reasonable to expect such high numbers of packets in communication, and large numbers of rules in access lists?

In recent years advances in computing and telecommunication technologies have

expanded computer system capabilities and also greatly expanded user requirements and available tools. As a consequence, users and their organizations are becoming more dependent on the services provided by their systems and computer networks. This trend increased the volume of communication on the Internet particularly as many users were browsing the Web. The number of users, servers and services offered is on the increase. More services are switching to using the Internet instead of mail, TV and other broadcasting services. This answers the first part of the question about the higher number of packets expected to be exchanged. Indeed, the quantity of communication on the Internet has been steadily increasing over the past few years, and is expected to continue for the next decade (Landwehr *et al.*, 1997).

The unfortunate part of this increase of communication on the Internet is the increase in security threats. Security attacks are on the increase, yet experts believe that the number of actual attacks are much higher than is officially reported (Howard, 1998). Many companies and organisations no longer think, for different reasons, that reporting security breaches of their systems is a good idea, especially if the nature of their business is fully or partly, the provision of a secure and safe system. The fact that security attacks are on the increase requires more forms and higher levels of protection. This reflects on access list-based packet filtering by requiring more inspection rules to be added to access lists. This answers the second part of the question about the need for large numbers of rules in access lists in the future. In fact, it is believed that the need for larger access lists is already materialising in many organisations. This places more pressure on firewalls to provide greater security at higher performance levels.

There are two reasons why longer access lists are not at present being used on a large scale despite the need for them. The first reason is the difficulty in actually stating the rules that accurately reflect the security policy required. Access lists are an efficient way of implementing first level security at relatively low cost. As long as the list of rules is short, it will be fast, easy to edit, and easy to append more rules without great difficulty. The difficulty becomes apparent in large and complicated security policies with less experienced users. As lists become larger, editing such lists becomes even more difficult and at best often produces access lists with many duplicated and useless rules. At worst they may not precisely reflect the required security policy.

The second reason for not using long access lists at this point is the performance issue. The more rules in an access list which need to be used to inspect a packet

the longer the processing time required for the particular packet. Clearly, it is desirable to keep this processing time to a minimum. Many approaches have been investigated and many algorithms were suggested. Some are more promising than others and more effective in one respect or another. It is still the case that longer access lists will cause a relatively large amount of delay to the point where some security inspections may be sacrificed for the sake of improved performance.

1.3 Solution Approach

One of the aims of our research is to investigate performance improvements of access list-based packet filtering devices. Many approaches are implemented or proposed which reduce the processing time of the list for a packet in one form or another, e.g. by reducing the size of the list, or using a binary decision diagram implementation of access lists as a compact way of representing and manipulating Boolean expressions. But ultimately, all approaches end up with a list of rules that is applied to inspect incoming or outgoing packets.

It is believed that the most commonly used implementations of packet filtering incur a lookup latency linear in the number of rules in the access list. A linked list structure is typically used to store the access list rules. Search is performed sequentially through the list. This has the advantage of using a small amount of memory for storing the list, but the lookup time is linear with the size of the list.

The way access list rules operate is that a packet is inspected against each rule in the access list until a clear accept or reject rule is met which applies to the packet. Otherwise, all the rules in the list are checked. From a performance point of view, it is desirable that each packet meets that critical (accept or refuse) rule at or near the start of the list. This would reduce the time required for the packet to be inspected and consequently improve performance. If the critical rule for a packet is met at or near the end of the list, or the rule is never met, the processing time for the packet will be very high and performance will degrade.

It is feasible to classify access list rules into different classes based on a number of factors. For example, the rules can be divided into those that inspect and filter a packet based only on the source address or only on the target address or on them both. Also, rules can inspect and filter a packet based on the port number in the packet or the protocol used. So, it is proposed to classify the rules in a list then

reorganise the rules in the list according to their class. It is intended to test the different organisations of an access list from the performance point using the same set of packets. This, of course, is a very simplistic way to visualise the approach, and a number of assumptions are made here. Assume that rules are organised in a particular way such that all rules of some class "x" are made at the top of the access list. When packets start arriving, and if all arriving packets or most of them require the critical "x" rule for acceptance or refusal, then our organisation of the rules list was a successful choice and performance is expected to improve. The reason is that for all or most of the packets, only a few rules will be checked.

An assumption was made here that it was known in advance that arriving packets required a specific type of a filtering rule to determine their acceptance or refusal. Ideally, for each packet arriving, if its filtering rule requirement is known, then those rules are used first to inspect the packet. For example, if a packet required class "x" rules then it is inefficient to inspect the packet using rules belonging to classes "a", "b", "c"... etc. It is best to have class "x" rules immediately inspect the packet.

It is suggested that a profile of packets arriving into a network device can be developed to determine the pattern of such packets. Packet classification of arriving packets in the past can be used to predict types and numbers of packets arriving in the future. This pattern can periodically be inspected for changes in pattern, and consequently to determine the most effective reorganisation of the rules in the access list.

This thesis suggests that different organisations of the rules in an access list will give better performance in filtering a particular pattern of packets. The rest of this thesis investigates whether ordering the rules in an access can make any improvements when processing a flow of packets with some particular characteristics. In particular two innovative algorithms PRCW (Packet-Rule Cost Weighing algorithm) and HCE (High Cost Elimination algorithm) were developed. Numerous experiements were carried out to evaluate the effectiveness of those algorithms.

1.4 Research Objectives

The main objectives can be summarised in three hypotheses in the form of statements. A number of sub-objectives have been identified which are listed later.

Hypothesis One:
"Performance of packet filtering using access lists is improved by the ordering or by the re-arrangement of the rules in an access list based on their classes."

Hypothesis Two:
"When filtering a specific stream of packets, if performance of an access list can be improved by class reordering of the list, then a method can be devised which will find the best order of the classes in an access list to give the best performance possible."

Hypothesis Three:
"If a method can be devised which will find the order of the classes in an access to give the best performance when filtering a packet stream, then an efficient way must exist in finding that order."

In order to achieve those objectives, by finding answers to those hypotheses, the following sub-objectives were identified:

1. Full investigation and understanding of the topics of network security and network communication including Internet protocols.

2. Investigation of existing methods and future expectations of packet filtering in network devices such as firewalls, routers, bridges and gateways.

3. Design and develop a model of the operation of a network device that uses access lists to filter a stream of packets.

4. Perform detailed simulation experiments using different combinations of packets and filtering rules to perform analyses of the experimental results and draw the appropriate conclusions and recommendations.

5. Produce conclusions about the effect of reordering of access lists on the performance of packet filtering operations.

1.5 Relevant Publications

The work in this research was discussed and described in a number of papers. The papers were a representation of different stages of the research and different parts of the thesis. The following is a list of relevant papers which were submitted and published or are awaiting publication:

1. Bukhatwa, F., "*High Cost Elimination Method for best class permutation in Access Lists*", Submitted to the IADIS International Conference - WWW/Internet 2004 Conference, 6-9 October 2004, Madrid, Spain. (http://www.iadis.org/icwi2004/)

2. Bukhatwa, F., "*Packet-Rule Cost Weighting Method for Best Organisation of Access Lists in Packet Filtering.*", Accepted at the 2004 International Conference on Computers, Communication and Control Technologies CCCT'04, August 14-17, 2004 - Austin, Texas, USA.

3. Bukhatwa, F., "*Class Ordering of Access lists in Packet Filtering*", Submitted to the "Journal of Computer Security" (JCS), March 2004.

4. Bukhatwa, F. and Patel, A., 2003. "*Effects of Ordered Access Lists in Firewalls*", In Proceedings of IADIS International Conference - WWW/Internet 2003, Algarve, Portugal, 5-8 November 2003, (http://www.iadis.org/icwi2003/).

1.6 Outline of The Thesis

This thesis presents a study performed to analyse the effects that reordering the access lists has on the performance of communication devices for the purpose of packet filtering.

The thesis starts by investigating packet classification and access lists. A model was developed of the packet classification process. The processes of packet streams arriving to a network device and the filtering of these packets was simulated. The model was first verified and then large numbers of simulations carried out and the results observed. Different size access lists, different orderings and different classes of rules were tested and performance observed. Performance is determined by the measurement of the average processing time per packet.

Reorganising the access list was found not to be the only factor for improving performance. It was found that a close relation between access list organisation and the pattern of arriving packet stream together could improve performance. The thesis describes these findings in more detail. The logical progress through the research is described to the point where the best organisation for an access list

can be determined for a particular packet pattern. The potential benefits, possible implementations, problems and disadvantages are discussed and evaluated.

Finally, conclusions based on the work are presented and some ideas for future research are discussed. There are a total of eight chapters in this thesis:

- ***chapter 2: Network Security and Firewalls.*** This chapter gives some background information about network security in relation to communication. It also provides a detailed introduction to packet filtering and firewalls.
- ***chapter 3: Performance Evaluation.*** This chapter looks at simulation as a tool for developing and testing communication systems. It investigates how simulation can be applied and how models can be developed.
- ***chapter 4: Requirements Analysis.*** This chapter formulates the requirements related to this research for packet filtering using access lists. It identifies the problems and suggests solutions.
- ***chapter 5: Design.*** This chapter discusses the design of the models to be developed based on data previously compiled. It also specifies the testing and verification procedures to be applied.
- ***chapter 6: Implementation.*** This chapter presents the implementation phase of these simulation model and the experiments. It describes the stages of development.
- ***chapter 6: Evaluation.*** This chapter presents the evaluation performed on the model performance and on the simulation results. It describes the performance of the different algorithms used.
- ***chapter 7: Conclusion.*** This Chapter presents the overall conclusions of the thesis and identifies directions for future work.

Chapter 2

Network Security and Firewalls

2.1 Introduction

This chapter discusses issues relating to computer security leading to network and communication security. It outlines the threats, attacks and security problems relating to computer systems and networks. Next, precautions and measures to reduce the effects of computer crime are discussed. This chapter also discusses firewalls as a security tool, how suitable and effective they are, their advantages and disadvantages. Packet filtering is investigated as a method of implementing firewalls via the use of access lists. Access lists are analysed and described in more detail.

2.2 Computer and Network Security

The technology of present day networks offers users increasing transmission capacity (Marzo *et al.*, 2003). As a consequence, users and their organisations are becoming more and more dependent on the services provided by their systems and computer networks (Muftic, 1994). Data, programs and information critical to the functioning of an organisation are kept on computer systems and exchanged over telecommunication facilities. This trend raises the need for secure systems for processing and exchanging information. In broad terms, security is the restriction of actions within a system with the aim of protecting the operation of the

system. Safety and security are closely related, and are often supported by the same mechanisms within a system. For our purposes, safety is distinguished from security using the rule of thumb that safety is to protect a system from itself, while security protects a system from others (Alexander *et al.*, 2001). Highland (1990) from the State University of New York defines security: "Computer security is the protection of a company's assets by ensuring the safe, uninterrupted operation of the system and the safeguarding of its computer, programs and data files". One form of protecting a system is by restricting access to the system and also restricting the flow of information from the system. One level of providing this restriction is through packet filtering. One form of implementation of packet filtering is through firewalls.

2.2.1 Principal Reasons for Security

Computer networks are becoming very convenient targets for attacks and illegal operations. The security of a system is addressed to prevent, detect and correct different forms of attacks. The required levels of security for networks and organisations differ but generally the principal reasons for security are confidentiality, integrity, availability and accountability.

1. **Confidentiality**: refers to the prevention of unauthorised information disclosure. In other words, the data access must be restricted to authorised entities on a legitimate "need to know" basis.

2. **Integrity:** refers to systems and data, and it reflects that users must have confidence that information can be retrieved from the system and, internal system processes work as expected or claimed.

3. **Availability**: refers to the property of a system being accessible and usable on demand by an authorised entity. This encompasses the prevention of unauthorised entities withholding of information or resources. In a system, availability means that the requested services are provided to clients at the desired moments. In other words, availability is the minimum level of service delivery to the users (Wulf, 1997). The minimum level of delivery means the provision of a service regardless of its quality of service (QoS).

4. **Accountability**: is the property that ensures that actions of an entity may be traced uniquely to that entity. This refers to the actions of individuals or of the system.

Attacks against the availability of a system are called denial of service (DoS) attacks. These attacks are divided into three categories.

1. Resource allocation attacks, in which the attacker repeatedly uses the limited resources of the system and does not give enough time to the server to release such resources. Therefore, a new genuine request for a service that requires such resources for its execution is rejected.

2. Resource destruction attacks, in which the conditions for the correct execution of some services are ignored. Thus, the attacker provides a scenario resulting in an unconformable execution of a service. The unconformable execution of a service engages some resources for an infinite duration of time. Hence, the request for a service requiring such resources is rejected during this period (Menezes *et al.*, 1996).

3. Alteration or destruction of configuration information, in which the attacker alters some information required for the execution of services, or modifies the configuration of the operating system (Icove *et al.*, 1995).

Distributed Denial of Service (DDoS) attacks have become an increasingly frequent disturbance of the Internet (Ioannidis and Bellovin, 2002). DDoS attacks consist of two main phases. The first is breaking into hosts and installing slave programs, and the second is instructing these thousands or millions of slave programs to attack a particular destination. This artificial high load of traffic denies or severely degrades service to legitimate users of the targeted destination. Flash crowds are problems that occur as a consequence of a DDoS attack. That is when a large number of users try to access the same server simultaneously. Apart from over-loading at the server itself, the traffic from such flash crowds can overload the network links and thereby interfere with other, unrelated users on the Internet (Mahajan *et al.*, 2001). An example of a legitimate flash crowd was the degradation of Internet performance experienced during the NASA Pathfinder mission (Mahajan *et al.*, 2001).

DDoS attacks are very hard to defend against because they do not target specific vulnerabilities of systems, but rather the very fact that the target is connected to the network. All known DDoS attacks take advantage of the large number of hosts on the Internet that have poor security. The attack does not necessarily exploit a security hole at the target to cause a problem and the installed slave programs can remain in the host unnoticed for a long time.

2.2.2 Categories of Computer Abuse

Computer abuse can be categorised as follows (Commission, 1998), (Furnell, 2002).

- Fraud: usually for private and personal gain or benefit. It may involve altering input, data or programs in an unauthorised way. It may also involve destruction, suppression or misappropriation of output from a computer process. This excludes any alterations caused by virus infections.
- Theft: of data or software.
- The use of unlicensed software through the use of illicit copies.
- The unauthorised use of computer or computing facilities of an organisation or at the work place for private gain or benefit.
- Misuse of personal data through unauthorised browsing through computer records causing breaches of the data protection legislation.
- Hacking: the deliberate gaining of unauthorised access to a computer system.
- Sabotage: interfering with the computer process by causing deliberate damage to data, programs or equipment.
- Introducing illegal or unsuitable material such as pornographic material.
- Viruses, or the distribution of a program with the intention of corrupting computer processes.
- Web site defacement: This most common form of web-related security incident is on the increase. Web site defacement includes alteration of text and images on the web site.
- Denial of service impacting the availability of sites/systems for legitimate users. One form is known as SYN Flooding. The attacking machine sends many SYN packets as requests to establish a TCP connection with different source addresses and never responds with ACK.
- Remote administration, where users are deceived into downloading and installing software that enables their systems to be accessed remotely.

2.2.3 Type of Abusers

Abusers may be classified into the following (Furnell, 2002):

- External Penetrators: Those outsiders attempting or gaining unauthorised access to the system.
- Internal Penetrators: Authorised users of the system who access data, resources or programs to which they are not entitled.
- Misfeasors: Users who are authorised to use the system and resources accessed, but misuse their privileges.
- Malicious Processes: Software-based abuse, including viruses, worms, Trojan horses and logic or time bombs. Also known as Malware.
- Hackers: Persons deliberately gaining unauthorised access to a computer system. Generally, divided into Explorers (hackers) or malicious vandals (crackers).

Penetrators can be divided into two groups depending on the form of penetration:

1. Masqueraders: users who operate under the identity of another user.
2. Clandestine users: users who evade access controls and auditing.

The definition of "hacker" (Furnell *et al.*, 1999) has changed considerably over the last thirty years. In the 1960s, hackers were dedicated software and hardware gurus, and the term largely referred to persons capable of implementing elegant and/or technically advanced solutions to technologically complex problems. In the 1990s, however, the term implies something rather different and is commonly used to refer to persons dedicated to entering the system by identifying and exploiting security weaknesses. At the extreme, hackers are a subset (often distinguished by the term "crackers") who openly perform malicious actions upon the systems they enter, such as deleting files, modifying data and stealing information. Modern data hackers are one part of a so-called Computing Underground (Mizrach, 1997) referring to subgroups that would generally be classed as undesirable by society at large. These subgroups include crackers, phreakers, virus writers and software pirates. "Phreaking" is defined as (Humphrey and Gabrielson, 1995):

- The art and science of cracking the phone network (so as, for example to make free long-distance calls).
- By extension, security-cracking in any other context (especially, but not exclusively, on communications networks).

Intruders get into a computer system electronically in a number of ways:

1. Software bugs: buffer overflows and un-handled input.
2. System configuration: default configurations that allow access.
3. Password cracking: weak passwords, dictionary attacks and brute force attacks.
4. Sniffing unsecured traffic: using shared media, server sniffing and remote sniffing.
5. Design flaws: in the operating systems or in the different protocols.

2.2.4 System Vulnerability

A vulnerable system is a system where potential threats are likely to become a reality. The more protection that is provided the less vulnerable the system should be. A threat in this respect is any action or event that could cause damage to an information system. Many points of vulnerability have been identified:

1. Default installs of operating systems and applications, which normally includes un-needed sample programs/scripts, services and corresponding open ports. Such problems can be checked by running port and vulnerability scanning programs. Protection may be improved by removing unnecessary software, services and ports.

   A communicating computer has one IP (Internet Protocol) address, referred to as the the network address, so other computers on the network can communicate with. In addition to a network address a port number is used. A port is owned by a specific application on a single computer. Port numbers allows several different applications to communicate over the network even though theres only one IP address. Some ports are well known and are defined historically. For example, the FTP program uses port 21.

2. Accounts with no passwords or weak passwords. Easy to guess or default passwords are a major problem as are default or built-in accounts. An audit of all accounts on the system should be maintained and regularly validated. Passwords should be checked or assigned to accounts if they do not exist. Procedures for adding or removing authorized accounts should be developed. Password cracking tools should be regularly run against the accounts.

All user passwords should be validated after a change, new passwords not meeting the security policy should be rejected. User awareness and training is usually required.

3. Non-existent or incomplete backups. Even if backups exist, sometimes they are not up-to-date or are not verified, with no restoration policies and procedures. The backup medium may have insufficient physical protection.

4. Large number of open ports. The minimum set or subset of ports that must be open should be identified. All others should be closed with the corresponding services disabled or removed. Internal and external port scanners should be executed.

5. Not filtering packets for correct incoming and outgoing addresses, as decoy or spoofing of IP addresses is a common method used by attackers to hide their tracks. External firewalls or routers should be tested by sending decoy packets. An indication of dropped test packets should show in the log produced by the device.

6. Non-existent or incomplete logging. System logs should be regularly reviewed for each key system. Logs should be archived and backed up on a write once device so hackers cannot overwrite the logs to avoid detection.

7. Vulnerable Common Gateway Interface (CGI) programs. Most servers support CGI programs to provide interactivity in web pages enabling functions such as data collection and verification. Vulnerability scanning tools should be run to scan the system. All sample CGI programs should be removed from the production web server and auditing performed on the remaining scripts on all servers.

2.2.5 Goals of Security

A system can never be guaranteed to be one hundred per cent safe. Total security requires that all possible threats be accounted for, and all vulnerabilities be detected. But different levels of security can be achieved through imposing different levels of security measures. Security measures will always incur some type of cost and if there is a higher need to protect the assets then more security measures need be adopted.

Assets in this respect refer to an organisation's information systems including hardware, software, infrastructure, people and staff, information and data, goodwill, the organisation's income, integrity and image. The value of an asset is either its cost of replacement or its importance to the organisation. The value of an asset can be higher than its isolated value if there exists other assets in the system that depend on this particular asset.

The cost to fully protect an asset may be far higher than the overall value of the asset, therefore it may not always be the goal to fully protect an asset. A number of factors may affect the level of protection required or considered reasonable to be applied. Assessment of the threats, vulnerabilities, impact and consequences in relation to an asset will be an indication of how to:

- Minimize the probability of a threat occurring,
- Reduce vulnerabilities in relation to an asset,
- Reduce the impacts and/or consequences of an attack.

A security breach can have tremendous effects. The effects of a failure to preserve confidentiality, integrity and or availability are:

1. Disclosure of information to unauthorised entities
2. Denial of service to authorised entities
3. Destruction of information or a part of the system
4. Modification of system processes or data

The consequences of security failure may also cause financial loss, embarrassment, breach of personal privacy or commercial confidentiality, legal liability, disruption to activities or even a threat to personal safety. Security comes at a high cost and any approach to security will be based on identifying the assets requiring protection and determining the appropriate level of countermeasures to be applied. The following points are part of an approach to security in general:

1. Identify assets to be protected: Including data, software and physical equipment.
2. Valuation of Assets: Replacement value and the impact on the system that can be caused by loss or damage of an item.
3. Threats identification and assessment: Threats in all forms as deliberate attacks or wilful damage, theft, natural disaster and technical failures.

4. Identify vulnerabilities.

5. Estimate risks: Risk assessment should include combination of asset values. The value of a piece of equipment may be the combination of its purchase value and the value of the data stored in it. The estimate may include the impact of the threats and vulnerabilities on the system.

6. Select suitable protective measures: Formalising a customised security plan containing the type, number and complexity of countermeasures as well as the characteristics of the security policy. The plan indicates, the security measures for achieving the appropriate level of protection for the identified assets, as well as the roles and procedures for ensuring the effectiveness of the security measures.

7. Monitor security related events to take the appropriate responsive action that may include corrective measures, analysis or other actions such as prosecution, litigation or recovery.

People are one of the most important and difficult entities in any system from a security point of view. The organisation is dependent upon the attitudes of the end-users and their belief that the information that they use is important and in need of protection. Where the user is held accountable for the security of information, they are more likely to adhere to the policies and procedures that have been put in place (Beatson, 1992). Responsible end-users, who are motivated and well informed of the need for information security, are an organisation's best defence against security threats. Management has a crucial role to play in achieving such a desirable end-user environment (Beatson, 1992).

2.3 Firewalls and Packet Filtering

The use of packet filtering as a means of improving system security is well established. Although packet filtering has its limitations, low-level filtering has proved to be efficient and effective (Schuba and Spafford, 1997). The increase in communication traffic, and increased demand for security, will in the future place a heavy burden on communication systems if improvements are not found for existing schemes.

The commercial viability of the Internet in the future depends on its ability to provide a differentiated service to paying customers. The Internet at the present

delivers only so called best effort (undifferentiated) service. Differentiated service requires Internet routers to move from simple destination-based packet forwarding to a more complex form of forwarding, called layer 4 switching (Suri and Varghese, 1999).

Limiting packets that can pass or selectively allowing only some packets is a simple and easy method of providing many functionalities. Examples are security, network management and the reduction of congestion problems.

Firewall technology is used to protect networks, by being situated strategically at a single security screening station where the private network or the Intranet connects to the public Internet (ISO, 1988), see Figure 2.1. Firewalls can also be used to isolate or protect sub-networks. A firewall is a computer, router or other communication device that filters access to the protected network (Schreiner, 1998). (Cheswick and Bellovin, 1994) define a firewall as a collection of components or a system that is placed between two networks and possesses the following properties:

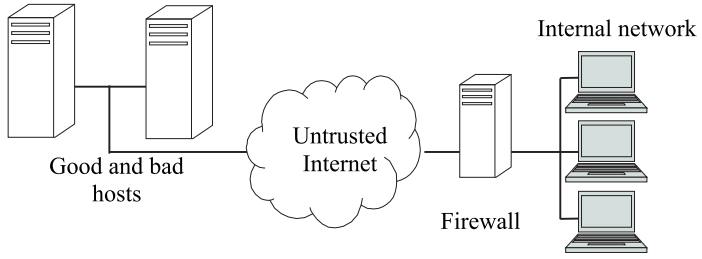

Figure 2.1: A typical firewall setup protecting an internal network [1]

- All traffic from inside to outside, and vice-versa, must pass through it.
- Only authorised traffic, as defined by the local security policy, is allowed to pass through it.
- The firewall itself is immune to penetration.

A firewall is simply a program or hardware device that filters the information coming through the Internet connection into a private network or into a computer system. If an incoming packet of information is flagged by the filters, it is not allowed through. Organisations may have large numbers of computers that have network cards connecting them together. Access to the Internet is provided through one or more connections like T1 or T3 lines. Without a firewall in place, all of those computers are directly accessible to anyone on the Internet. It is easy to make FTP or Telnet connections to such unprotected machines.

If a security hole is discovered, hackers can get to a machine and exploit the situation. With a firewall in place, the landscape is much different. An organisation will place a firewall at every connection to the Internet. Security rules can be implemented by the firewall. For example, only one of computers may be permitted to receive public FTP connections. An organisation can set up rules like this for FTP servers, Web servers, Telnet servers and so on. In addition, the organisation can control how employees connect to Web sites, whether files are allowed to leave the organisation over the network and so on. A firewall gives an organisation some control over how the network is used. The following are some examples of what the consequences can be for unprotected computers machines:

1. Remote login: When someone is able to connect to the victim computer and controls it in some form. This can range from being able to view or access files to actually executing programs on the computer.

2. Application back–doors: Some programs have special features that allow for remote access. Others contain bugs that provide a back–door, or hidden access, that provides some level of control of the program.

3. SMTP session hijacking: SMTP is the most common method of sending e-mail over the Internet. By gaining access to a list of e-mail addresses, a person can send unsolicited junk e-mail (spam) to thousands of users. This is done quite often by redirecting the e-mail through the SMTP server of an unsuspecting host, making the actual sender of the spam difficult to trace.

4. Operating system bugs: Like applications, some operating systems have backdoors. Others provide remote access with insufficient security controls or have bugs that an experienced hacker can take advantage of.

5. Denial of service: This type of attack is nearly impossible to counter. What happens is that the hacker sends a request to the server to connect to it. When the server responds with an acknowledgement and tries to establish a session, it cannot find the system that made the request. By inundating a server with these unanswerable session requests, a hacker causes the server to slow down or eventually crash.

6. E-mail bombs: An e-mail bomb is usually a personal attack. The same e-mail is sent hundreds or thousands of times until the victims e-mail system cannot accept any more messages.

7. Macros: To simplify complicated procedures, many applications allow the creation of a script of commands that the application can run. This script

is known as a macro. Hackers have taken advantage of this to create their own macros that, depending on the application, can destroy data or crash a computer.

8. Viruses: Probably the most well-known threat is the computer virus. A virus is a small program that can copy itself to other computers. This way it can spread quickly from one system to the next. Viruses range from harmless messages to erasing entire data on a machine.

9. Spam: Typically harmless but always annoying, spam is the electronic equivalent of junk mail. Spam can be dangerous. Quite often it contains links to Web sites. Clicking on these may cause a computer to accept a cookie that provides a backdoor to the computer.

10. Redirect bombs: Hackers can use the Internet Control Message Protocol (ICMP) to change (redirect) the path information takes by sending it to a different router. This is one of the ways that a denial of service attack is set up.

11. Source routing: In most cases, the path a packet travels over the Internet (or any other network) is determined by the routers along that path. But the source providing the packet can arbitrarily specify the route that the packet should travel. Hackers sometimes take advantage of this to make information appear to come from a trusted source or even from inside the network. Most firewall products disable source routing by default.

Packet filtering in the form of firewalls offers a level of security based on the following:

1. A packet filter is placed in a strategically selected place at the gateway of the network to be protected from the outside world, usually the Internet. Based on this, it is worth mentioning that any traffic that manages to bypass the filter is not inspected thereafter and can create a potential hazard.

2. Packets passing through the device in which the packet filter is placed are inspected against a list of rules called the access list. Each rule within the access list can decide to accept or refuse the packet passing through depending on the condition in the particular rule. A rule may have the following basic form: *if condition is met then accept-packet*. If the end of the access list is reached before a matching accept or refuse rule is found, in most implementations, the packet is refused.

3. Rules in access lists are generated based on a security policy. Simple security policies can be implemented easily with a small number of rules, while others can be very long. Very long lists can be difficult to comprehend and edit and can contain duplicated or contradicting rules and may not fully implement the security policy. They can also negatively affect performance, as they are a potential bottleneck for the communication channel.

Approaches used to provide security by firewalls are broadly classified according to Selani (1998) and CORBA (1998), into two types: transport level and application level. Packet filtering is used for transport level type firewalls.

Application level firewalls, also known as the proxy type or circuit proxy gateways, do not rely on general purpose mechanisms (access list rules) to allow traffic to pass but use special purpose code for each desired service. Proxies hide all addresses in the network they protect, and communicate with the outside world using the proxies' own IP (Internet Protocol) address. The circuit proxy replaces the original address on a packet (its own address) with the address of the intended destination (Habtamu, 2000).

In Network and Transport level packet filtering firewalls, packet filtering refers to the basic operation performed by the firewall to inspect the packet header, verifying any of the fields in the packet header, i.e. the IP address, the port or both, and then accepting or rejecting the packet. Filtering can be applied to incoming or outgoing packets or both. Packet filtering is transparent to the users or independent of the user's knowledge or intervention. Firewalls of this type are cheap, simple, fast, efficient and provide a good level of security (Habtamu, 2000). IP level filtering has proved to be efficient and effective at improving system security (Schuba and Spafford, 1997). A single rule can help protect an entire network by prohibiting connections between specific Internet sources and internal computers. Packet filters do not require client computers to be specifically configured; the packet filters do all of the work. But the cost of filtering may still be a significant bottleneck (Ballew, 1997). The time cost of performing a look-up on a rule list may become too high, particularly for routers where this may add significantly to the latency of the network.

The rules are implemented one at a time to check if the condition matches the incoming packet and consequently accept or reject it depending on the action. If the condition does not match the rule then the checking continues with the next rule. If none of the conditions in all the rules are matched then the packet is

usually rejected. The rules are checked in a specific order and the order is very critical. Changing the order can result in a different decision of acceptance or rejection for some packets (Hazelhurst, 2001).

The firewall design policy describes how the firewall will implement the network service access policy, and precisely how it will take access decisions in accordance with it. Typically, the policy is either "permit any service not expressly denied" or "deny any service not expressly permitted". This determines what happens to a packet if no rule is found in the list that will cause it to be accepted or rejected. An extreme access policy example allows only outgoing access to the Internet but no incoming access. Another example is to allow access only to certain selected services.

Although the main function of firewalls through packet filtering was originally a basic security implementation, packet filtering has now been applied in many other areas. The following are some of the services and applications where packet filtering and firewalling are applied:

- Logging of alerts and notification. Logging of events is important for future analyses of attacks.
- Virtual Private Networks (VPNs). A VPN is an encrypted tunnel over the Internet or untrusted network. A VPN provides confidentiality and integrity of transmissions, and logically all hosts in a VPN are in one Intranet (Schreiner, 1998). As well as the filtering of packets by accepting packets only from hosts who are members of the VPN, firewalls can implement other functionalities like strong authentication and encryption of all traffic.
- Quality of Service (QoS). Packet filtering can be used as a control to limit the use of certain network connections and therefore make a particular connection dedicated for a particular service. This allows administrators to control what proportion of a given network connection is to be dedicated to a given service.

2.3.1 Limitations of Firewalls

IP was not designed with security as a priority and is, as such, inherently insecure. Firewalls are a common method of addressing these insecurities and are

often regarded the only line of defence (Haeni, 1997). Firewalls do not eliminate these insecurities entirely, but do make unauthorised access to a system from the Internet far more difficult. Packet interception and filtering can suffer from some difficulties:

1. A firewall is not effective against users with authorised access.

2. Most firewalls are not a real defence against attacks using known software bugs or malicious problems such as mail bombs, ping floods, viruses and Trojan horses.

3. Firewalls rely on accurate IP source addresses for making filtering decisions. IP addresses can be faked (Bellovin, 1992).

4. Packet fragmentation complications. IP will break up large transmission units into small units of data known as fragments. This is done so that the data can be transferred over networks that support small maximum packet sizes. IP will re-assemble those fragment packets when the data is received at the destination (Tanenbaum, 1996). Packet fragmentation increases the complication for firewall filtering.

5. Scalability, with reference to having a single point (firewall) where all traffic flows through it makes it a central bandwidth bottleneck. Extra strain is created with the growing number of packets and computational cost due to increased sophistication of the filtering required. There is security a versus performance dilemma (Friedman and Nagle, 2001). Computational cost is influenced by how far up the network stack a packet must travel, as well as what level of security checks are being performed on each packet. Packet filter firewalls operate at the lowest level of all filters. They generally provide the highest performance (speed-wise), followed by circuit level firewalls and application layer firewalls. As packets pass through more protocol layers, they are inspected in more detail. As a result, application layer firewalls are considered more secure than circuit level firewalls. However, because a circuit level firewall does not perform extensive security checks, other than whether a network packet is associated with a valid connection, it can (and often does) perform faster than a packet filter firewall that contains a large set of accept and deny rules.

6. The process of configuring and testing packet filtering rules tends to be lengthy, difficult and complicated. The complexity of packet filtering consists of two parts. The first is the difficulty of correctly specifying filters (Chapman, 1992). The second is that reordering filtering rules makes

correctly specifying filters even more difficult (Hazelhurst, 1999). The order in which rules are specified is critical, to the extent that changing the order of the rules could result in some packets that were previously rejected being accepted and vice-versa. Consider the following simple example of a security policy "Accept packets from the domain D only if access is required through port P, while all other packets are refused" expressed by the following two rules and in this order:

(a) Accept any packet from domain D on port P

(b) Refuse all packets from domain D

In this order, the first rule above will cause acceptance of a packet from domain D if it requests access on port P. The second rule deals with all other packets from domain D and causes them to be refused. Consider the situation if the order of the rules is changed to the following:

(a) Refuse all packets from domain D

(b) Accept any packet from domain D on port P

The first rule now will cause the refusal of all packets arriving from domain D. Those packets requesting access for port P will not have a chance of reaching the second rule. The change in the ordering of the rules in this case causes the refusal of packets that according to security policy should have been accepted.

In general, application layer firewalls are the architecture with the lowest performance due to the fact that all network packets are sent up one network stack and down another, thus being treated as two separate network sessions. Application layer firewalls also implement the broadest set of security data checks, which increases the processing time required. Throughout the industry, application layer firewalls are generally considered to provide the best security.

2.3.2 Packet Classifications

The basic concept behind packet filtering consists of inspecting the header fields of incoming and/or outgoing packets and then applying rules from a rule base to determine whether to permit the packet or deny and drop it.

As opposed to a "best-effort" service, different qualities of service are requested from routers and are expected for different applications. Internet routers that

operate as firewalls, or provide a variety of service classes, perform different operations on different flows. A flow is defined to be all packets sharing common header characteristics (Gupta and McKeown, 1999); for example, a flow may be defined as all the packets between two specific IP addresses. Another flow may contain all of the packets that have the same source or destination IP addresses. Packets within a flow obey a pre-defined rule and are processed in a similar manner by the router.

In order to classify a packet, a router consults of a table (or classifier) using one or more fields from the packet header to search for the corresponding flow. The classifier is a set of rules that identifies each flow and the actions to be performed on each. Packet classification is the process of categorising packets into "flows" in an Internet router based on a specified collection of rules.

Packet classification is employed by Internet Routers and other communication devices to implement a number of Internet services and a wide variety of network security policies. Examples are to prevent unauthorized access from outside the protected network and to prevent attacks of DoS resulting in actual denial of service to legitimate users. For Virtual Private Networks (VPN), packet interception can ensure their privacy by only allowing packets sent by members of the same private network. Another use of packet interception is for Quality of Service (QoS) as a control to limit the use of certain network connections (dedication) to a particular service. Packet classification is a base for routing, rate limiting, access-control, resource reservation such as virtual bandwidth allocation, policy-based routing, service differentiation, load balancing, traffic shaping, and traffic billing services (Feldmann and Muthukrishnan, 2000).

All of these services require the devices to distinguish packets belonging to different flows. The router for example is required to classify incoming packets into different flows and then perform appropriate actions depending upon which flow the incoming packet has been identified as its flow.

For instance, each rule in a firewall access list could specify a set of source and destination addresses, and associate a corresponding 'deny' or 'permit' action with it. Alternatively the rules could be based on several fields of the packet containing addressing and protocol information. Traditionally, in firewalls, rules are mainly used to accept or deny a packet, and in a router they are used to ultimately find the IP address of the next hop of where the packet needs to be routed. The simplest and most well known form of packet classification is used to

route IP datagrams, were each rule specifies a destination prefix. The associated action is the IP address of the next hop. Generic packet classification requires the router to classify a packet based on multiple fields in its header. Each rule of the classifier specifies a class that is based on some criteria on some header fields, a packet may belong to, and associates with, each class an identifier. This identifier uniquely specifies the action associated with the rule. Each rule has a number of components. The ith component of rule R, referred to as R[i], is a regular expression on the ith field of the packet header. A packet P is said to match a particular rule R, if, the ith field of the header of P satisfies the regular expression R[i] (Gupta and McKeown, 1999).

2.4 Summary

This chapter presented issues related to computer network security. It described categories of abuse and attacks on computer networks from the Internet. Firewalls and packet filtering were discussed and some of the related problems outlined. The next chapter investigates simulation as a technique for system development, testing and performance analysis. Advantages and types of simulation. It will also looks into model development validation and verification.

Chapter 3

Performance Evaluation

3.1 Introduction

A number of approaches may be used to design, develop and evaluate the performance of a system. Such approaches range from the design and development of a real life system and testing it in real life environments; to using a model of the system and the environments. In this chapter we discuss the pros and cons of using simulation, performance evaluation using simulation and model development for simulation, and issues relating to using simulation in this research work.

3.2 Simulation and Modelling

Many definitions for simulation are available. The following is a general definition given by Shannon (1975) for simulation: "The use of a mathematical/logical model as an experimental vehicle to answer questions about a referent system". This definition does not specify computer based simulation. Essentially, simulation provides the basis for making some decision about a system in real life based on answers provided by the simulation. This decision taken based on some simulation can be of extreme importance importance. Not just the decision alone, but also the subsequent actions that follow up. A real system may be built on the basis of a wrong decision based on bad simulation. Often, simulation provides an assessment of some system which is not readily amenable to other types of analysis; thus the

simulation provides the only means by which to assess a given situation. The overriding objective of simulation is arriving at the correct decision. Simulation may also be desired to provide a variety of behaviors and capture a multitude of characteristics, but none of these should be achieved at the expense of a correct decision.

Computer simulation is the discipline that allows the study of the complex dynamics of a physical system by designing a model of the system, executing the model on a computer and analysing the execution output. A computer simulation or a computer model is a computer program which attempts to simulate an abstract model of a particular system. Computer simulations have become a useful part of modelling many systems to gain insight into the operation of those systems. Traditionally, the formal modelling of systems has been via a mathematical model, which attempts to find analytical solutions to problems which enables the prediction of the behaviour of the system from a set of parameters and initial conditions. Computer simulations build on, and are a useful adjunct to purely mathematical models.

According to Shannon (1975), digital computer simulation is the process of designing a model of a real system and conducting experiments with this model on a digital computer for the specific purpose of experimentation.

Conducting a performance evaluation of a system to establish which approach, which method or which design is best, can be a difficult task. The following three evaluation techniques are available:

1. **Analytical evaluation:** The analytical evaluation is based on representing the system in mathematical equations and functions. This mathematical representation can often be used to analytically predict the exact behaviour of the system. This method can be very accurate in small systems. As the system gets more complicated, this approach becomes very difficult, hard to comprehend and requires large amounts of computational resources.

2. **Real world evaluation:** This approach requires executing the system in the real world environment, and performing the necessary evaluation. This is almost impractical in many cases due to the time cost involved. In other cases, testing may not be permitted in the real world environment.

3. **Simulation:** Allows evaluation through experiments with a model of the system which allows observation of behaviour. Simulation in general is the

imitation of the operation of a real system. With reference to computers, a computer simulation is the execution of a model, represented by a computer program that gives information about the simulated system.

3.2.1 Advantages of Simulation

Simulation allows measurements and evaluation of a system at a suitable pace. In many cases simulation is less time consuming than real world measurement while in other cases it may suit to simulate things at a slower pace. Performance analysis of modern communication systems is a complex task, and one problem is that the time scales can differ by several orders of magnitude, further increasing the model complexity. However, given that one can reduce model complexity, it is still often the case that a precise and accurate model with known analytical results poses difficult problems in terms of the computability of the performance measures of interest (Lassila, 2001). Thus, approximative models or methods, such as simulation methods, need to be developed to obtain approximations or estimates of performance measures.

One of the key strengths of simulation is that it enables the study of a system over time. An approach for controlling the time advance in a simulation is to step the model into the future only at discrete, possibly random points in time when an event that could change the state of the system occurs.

In the field of simulation, the concept of "principle of computational equivalence" has beneficial implications for the decision-maker. Simulated experimentation accelerates and replaces effectively the "wait and see" anxieties in discovering new insights and explanations of future behavior of the real system.

Simulation is a powerful tool for evaluating and understanding the performance of existing and proposed systems. It is more desirable in many situations to use simulation. The following are general reasons why simulation may be a more desirable approach to adopt when developing or evaluating a system:

1. Where work on the real system is or is almost impossible due to either one or more of the following:

    - if there is a high level of danger involved,
    - when a high level of cost is involved,

- when the timespan is too short or too long to observe the real system.

2. Where simulation can be used as a tool to better understand and analyse systems. Non-experts in the topic of system analysis and users can model and analyse the operation of real systems using readily available simulation software. Such users might find the mathematical models built by experts difficult to comprehend. The simulation approach of analysing a model gives the analyst the advantage of visualising and understanding the system. The analytical approach for a system representation and a system analysis is purely theoretical, where as simulation can be used as a tool to optimise performance and/or the reliability of systems at very low costs.

3. Simulation is extensively used to verify the correctness of a design.

4. System modelling and simulation is helpful for designing and engineering real world jobs. Most digital integrated circuits manufactured today are first extensively simulated before they are manufactured to identify and correct design errors. Simulation early in the design cycle is important because the cost repairing mistakes increases dramatically the later in the product life cycle that the error is detected (Arsham, 1996).

5. Simulation is also used in developing "virtual environments" for training, with which a user can interact with some system simulating reality, like training of military personnel for battlefield situations at a fraction of the cost of running exercises involving real military equipment.

6. In cases where real system behaviour cannot be repeatedly observed due to the difficulty of establishing the exact environment, simulation can be used to repeat execution of a model in the exact environment. This makes it possible to observe and study the effect of individual factors.

7. In many cases, where a model is a good representation of the system under investigation, the simulation approach provides higher reliability, more flexibility and convenience than other methods in analysing and understanding the real system.

3.2.2 Types of Simulation

Based on the definitions given by Nance (1993), digital computer simulation may be divided into three categories:

1. **Monte Carlo:** Monte Carlo simulation is a method by which an inherently non-probabilistic problem is solved by a stochastic process; the explicit representation of time is not required.

2. **Discrete Event:** A discrete event simulation refers to where system changes are modelled to occur at discrete points in time. If changes to program variables occur at precise points in simulation time, the simulation is a discrete event.

3. **Continuous Time:** In a continuous time simulation, the variables within the simulation are continuous functions, e.g. a system of differential equations. The state of the system changes all the time, not just at the time of some discrete event. An example is the change of the height of liquid in a filling tank.

Nance (1993) notes that three further related forms of simulation are commonly used in the literature. A **combined** simulation refers generally to a simulation that has both discrete event and continuous components. **Hybrid** simulation refers to the use of an analytical sub-model within a discrete event model. Finally, **gaming** can have discrete event, continuous, and/or Monte Carlo modelling components.

3.3 Discrete Event Simulation (DES)

A discrete simulation, or stochastic simulation, manages only events and time. In this type of simulation, the simulator maintains a queue of events sorted by the simulated time they should occur. The simulator reads the queue and triggers new events as each event is processed. It is not important to execute the simulation in real time. It is often more important to be able to access the data produced by the simulation and, to discover logic defects in the design or in the sequence of events.

The physical system being modelled consists of a set of physical processes which are interacting in some way. If these interactions occur at discrete points in time then the system can be modelled as a discrete event system. The interactions change the state of the physical process and may cause further interactions. The simulation of such a system models the interactions between physical processes as events that carry information that changes the system state; processing events

can generate new events. These events occurs at discrete points in time and are processed in the same order as they would the real system. The DES system maintains a list of events sorted according to the event time-stamp. The time-stamp on the event is generated by the clock. The clock, which is a real value, gives the time at which the system is being simulated. The DES algorithm performs the following operation on the event-list with the clock initially set to zero.

1. Select the element with the minimum time-stamp from the event-list and remove it.
2. Update the clock's value to the time-stamp on the event.
3. Process the event and change the state if needed.
4. If the previous step generated new events then they are inserted into the event-list.
5. If there are more events to process then goto step 1, else stop.

Discrete event simulation is one which employs a next-event technique to control the behaviour of the model. Many applications of discrete simulation involve queuing systems of one kind or another. In the simplest case, the queue would operate with a first-in first-out (FIFO) discipline.

Discrete event simulation (DES) enables a model to be evaluated faster since only the significant points in time when something happens in the system are considered. DES is used in solving problems in a variety of domains such as service industries, manufacturing and telecommunications. Computer simulations of such applications are becoming larger and more complicated as people need to model more complex systems. Although computer power continually increases, simulations of such large systems are still time consuming.

Parallel discrete event simulation (PDES) has as its main goal the reduction of execution time of a DES application by executing it concurrently on multiprocessor computers. A large body of research has been conducted in this area and a number of approaches for solving the main issues related to PDES, such as synchronisation, scheduling, memory management, partitioning and load balancing, have been proposed. Although encouraging results have been obtained with different approaches, they are highly dependent on the application characteristics and none proves to be a universal solution that can be successfully applied to any type of simulation problem. Carrying out a DES experiment on a computer can

be done in different ways: using an available simulator or implementing the model using a special simulation language or general purpose programming language.

Within a discrete event simulation, the two concepts of time and state are of paramount importance. Nance (1981) identifies the following primitives which permit a precise definition of the relationship between these fundamental concepts:

- *An instant* is a value of system time at which the value of at least one attribute of an object can be altered.
- *An interval* is the duration between two successive instants.
- *A span* is the contiguous succession of one or more intervals.
- *The state* of an object is the enumeration of all attribute values of that object at a particular instant.

These definitions provide the basis for some widely used (and, historically, just as widely misused) simulation concepts:

- *An activity* is the state of an object over an interval.
- *An event* is a change in an object state, occurring at an instant, and initiates an activity precluded prior to that instant. An event is said to be determined if the only condition on event occurrence can be expressed strictly as a function of time. Otherwise, the event is contingent.
- *An object activity* is the state of an object between two events describing successive state changes for that object.
- *A process* is the succession of states of an object over a span (or the contiguous succession of one or more activities).

These concepts may be viewed as illustrated in Figure 3.1, keeping in mind that an activity for an object is bounded by two successive events for that object (Nance, 1981).

To briefly summarize, modelling is the process of describing a system (producing a model of that system) with the goal of experimenting with that model to gain some insight into the behavior of the system. The model itself is a collection of interacting objects, these objects being described by attributes. An object-based view of a model is not the only possible description of a system. For example, a system may be modelled as a set of functions that acts on streams of input to

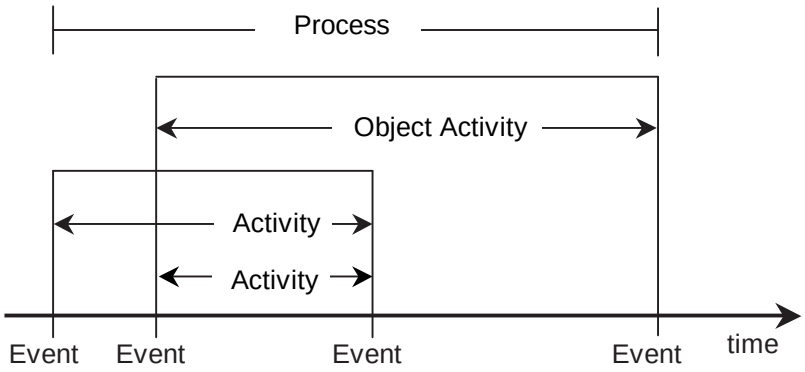

Figure 3.1: Illustration of Event, Activity and Process.

produce output (e.g. Hatley and Pirbhai (1987)), or as a set of data structures (e.g. Jackson (1983)) with some prescribed behavior.

There are three major conceptual approaches (world views) within discrete simulation. These are event scheduling, activity scanning, and process orientation (Derrick, 1992). Each approach is adopted by some programming language, and, more importantly, offers a different way to look at a simulation problem. Each, in its own way, suggests mechanisms to model real situations.

- In an event-scheduling world view, the modeler identifies when actions are to occur in a model. Event scheduling was one of the first simulation approaches to be developed. An event is anything that changes the state of the system other than the mere passage of time. The essential idea of event scheduling is to move along the time scale until an event occurs and then, depending on the event, modify the system state and possibly schedule new events.

- In an activity-scanning world view, the model designer identifies why actions are to occur in a model.

- In a process-interaction world view, the model designer identifies the components of a model and describes the sequence of actions of each one.

In general, a basic system consists of a number of entities. An entity is one or more objects which randomly enter the system and move through it and eventually depart. These entities get serviced and may be made to wait. Events are

occurrences within the system, at different points in time which can cause amongst other things, the state of the system to change. When events occur at distinct points of time then the state of the system changes at distinct points of time. If the events are finite, then the simulation is called a discrete event simulation. "Discrete-event" means that the system is viewed as progressing through time from one important event to another, rather than changing continuously.

Figure 3.2 represents the basic flow diagram of a simulation application on a computer. This is assuming a discrete simulation of a very simple system consisting of a single queue and a single server. Objects arriving into the system may be packets arriving at random time intervals. An arriving packet may be placed in the queue waiting to be assigned the server if there are other packets waiting in the queue and/or the server is busy serving another packet. The queue may be skipped if the server is free, and the arriving packet may be assigned the server immediately. A number of properties or characteristics are associated with each packet which distinguish one packet from another. These properties are created by the currently arriving packet. Packets waiting in the queue will be moved to the server by the packet departing the server. The simulation time shall start normally from time zero and shall be incremented by one unit of time. Each unit can represent a millisecond, microsecond or any other suitable real time unit.

In this basic system arriving objects may be treated similarly. In more sophisticated systems, many properties may be associated with these objects to allow different handling of differing objects. Examples of such properties are that the priority of each arriving object which may cause it to jump to the front of the queue; a type of object may need longer or shorter service time. A large number of different properties may be associated with objects to facilitate flexible and more realistic simulation.

The system may have multiple queues or multiple servers arranged in sequence or in parallel. When queues and servers are arranged in consecutive sequence, the layout is known as a network layout. Queues may also have a number of properties indicating how the system should operate or in someway affecting its operation. Examples of such properties are if the queue has a limited or limitless size, or a priority level of the queue if queue priority is to be implemented in a number of queues. Similarly, the servers may be assigned a number of properties such as the length of service time of the server and any other relevant properties.

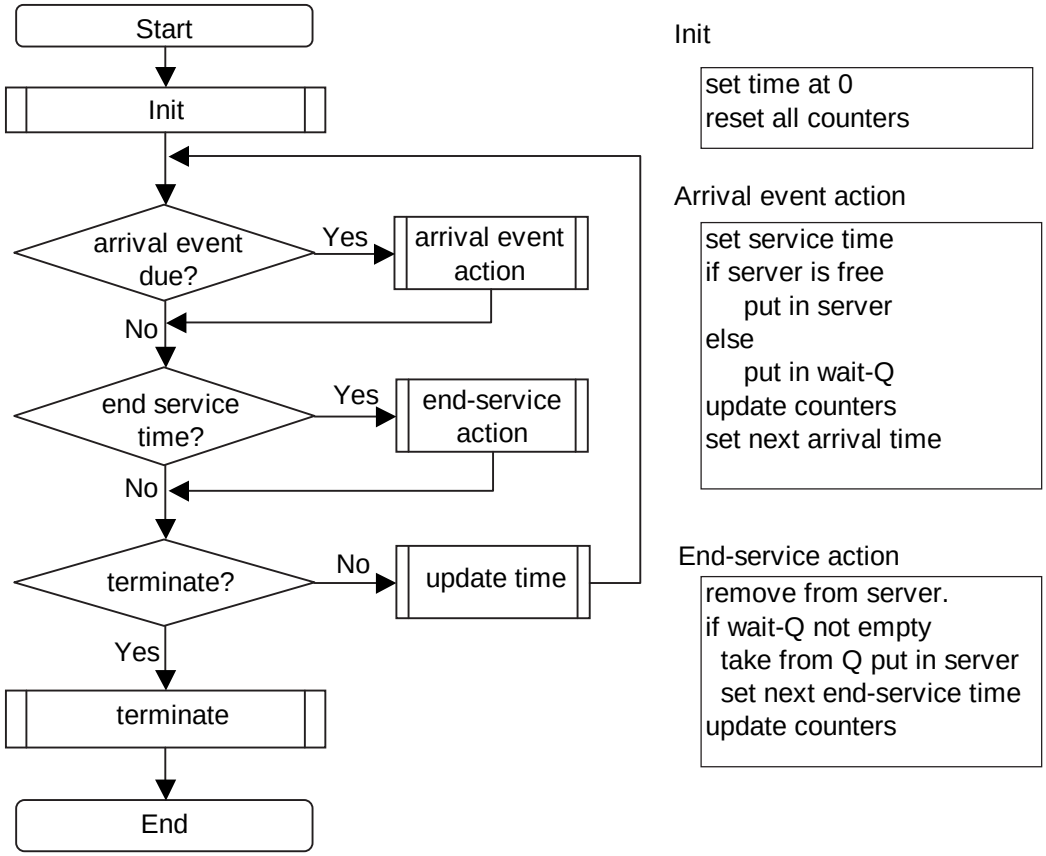

Figure 3.2: Basic logic of discrete event simulation

3.3.1 Parallel Simulation

Parallel simulation is the process of using multiple processors simultaneously for executing a single simulation, with the goal of reducing the total execution time (Bhatt *et al.*, 1998). Here, simulation execution time is reduced to tolerable levels, such as from several days down to a few hours. A parallel discrete event simulation program can be viewed as a collection of interacting sequential simulators, called logical processes (LPs). The computation performed by each logical process (LP) is a sequence of event computations, where each event represents some interesting action in the model e.g. the arrival of a new packet. Each event contains a time-stamp indicating when that event occurs. Logical processes (LPs) interact by scheduling events for each other. Ensuring time-stamp ordered processing of events is a fundamental problem in parallel discrete event simulation requiring complex techniques (Bhatt *et al.*, 1998).

3.4 System Modelling

A simulation involves modelling a system. Delta project report (Holbaek-Hanssen *et al.*, 1977) defines a system as: "A system is a part of the world which we choose to regard as a whole, separated from the rest of the world for some period of consideration, a whole which we choose to consider as containing a collection of components, each characterized by a selected set of data items and patterns, and by actions which may involve itself [a component] and other components. The system may be real or imagined and may receive input from, and/or produce output for, its environment".

A model is an abstraction of a system intended to replicate some properties of that system (Overstreet, 1982). The collection of properties the model is intended to replicate for the purpose of providing answers to specific questions about the system must include the modelling objective. The importance of the modelling objective cannot be overstated; a proper formulation of the objective is essential for any successful simulation study. Only through the objective can meaning be assigned to any given simulation program. Since by definition a model is an abstraction, details exist in the system that do not have representation in the model. In order to justify the level of abstraction, the model assumptions must be reconciled with the modelling objective.

According to Nance (1981), a model is comprised of objects and the relationships among objects. An object is anything characterised by one or more attributes to which values are assigned. The values assigned to attributes may conform to an attribute typing similar to that of conventional high level programming languages.

Studying an existing or a new system using simulation involves three steps:

1. Creating a system model: Those features within the system, which are deemed significant in determining its performance, are determined. These features are abstractly defined. This abstraction is called the system model.

2. Model implementation: A computer program is written; the execution of which mimics the behaviour of the model.

3. Performance estimation: An estimate of the performance of the original system is computed using the data collected during the simulation model execution. The accuracy of such estimates depends on the fidelity of the

model and the way in which the measurements are taken from the simulation execution.

The general role of system simulation in system design is depicted in Figure 3.3. The designer relies on the simulation model to provide guidance in choosing among alternative design choices, to detect bottlenecks in the system performance, or to support cost/benefit analysis. As part of this process, the designer may use the simulation output to modify the system model as opposed to the system itself, in order to include details that may have been omitted in the previous abstraction, or to change the implementation for example to collect additional or alternative types of data. Simulation may be used as a tool to help validate an analytic approach to performance evaluation.

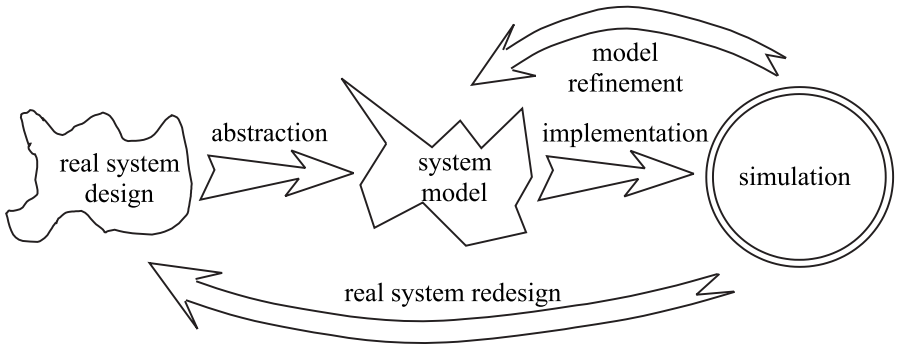

Figure 3.3: The role of simulation in design

Theoretically and ideally, outputs from a simulation should provide an indication about the behaviour of the real system being simulated. More precisely, the simulation outputs provide indication about the behaviour of the model. It is the task of the model designer to ensure that the model is sufficiently close to the system being modelled that the results obtained from the model can be applied to the system. That is why it is of the utmost importance to ensure that the model is a very close representation of the real system.

On the other hand, each output produced by a simulation of a model can be thought of as a measurement of its behaviour. It is from such output that the performance metrics of interest are generated.

Simulation models, in general, will produce random or stochastic output data. A major element of simulation analysis is the analysis of such output data, which controls the execution of simulation models, in order to obtain statistical data of acceptable quality. Parameters of the model are estimated by analysing the

output produced. From a statistical point of view, it is necessary to have both point and interval estimates of these parameters.

System Simulation is the mimicking of the operation of a real system in a computer, such as the day-to-day operation of a bank, or the value of a stock portfolio over a time period, or the running of an assembly line in a factory, or the staff assignment of a hospital or a security company. Instead of building extensive mathematical models by experts, readily available simulation software has made it possible to model and analyze the operation of a real system by non-experts.

A simulation is the execution of a model, represented by a computer program that gives information about the system being investigated. The simulation approach of analysing a model is different to the analytical approach, where the method of analyzing the system is purely theoretical. The simulation approach gives more flexibility and convenience. The activities of the model consist of events, which are activated at certain points in time and in this way affect the overall state of the system. The points in time that an event is activated are randomized, so that no input from outside the system is required. Events exist autonomously and are discrete, so between the execution of two events nothing happens. SIMSCRIPT provides a process-based approach of writing a simulation program. With this approach, the components of the program consist of entities, which combine several related events into one process.

There are many situations and scenarios where computer simulation can be effectively used. In addition to its use as a tool to better understand and optimize performance and/or reliability of systems, simulation is also extensively used to verify the correctness of designs. Digital integrated circuits manufactured today are first extensively simulated before they are manufactured to identify and correct design errors. Simulation early in the design cycle is important because the cost to repair mistakes increases dramatically the later in the product life cycle that the error is detected. Another important application of simulation is in developing virtual environments, simulations generate dynamic environments with which users can interact as if they were really there. Such simulations are used extensively today to train military personnel for battlefield situations, at a fraction of the cost of running exercises involving real tanks, aircraft, etc.

Dynamic modelling in organizations is the collective ability to understand the implications of change over time. This skill lies at the heart of a successful strategic decision process. The availability of effective visual modelling and simulation

enables the analyst and the decision-maker to boost their dynamic decision by making rehearsing strategies to avoid hidden pitfalls.

3.5 Model Validation and Verification

Simulation models are increasingly being used in problem solving and decision making. Decisions based on information derived from the results of models are used by many people developing new models and systems or for other purposes. It is a concern shared by all those affected by decisions based on such models whether those models and their results are "correct". Model validation is defined to mean "substantiation that a computerised model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model" (Schlesinger *et al.*, 1979). Model verification is often defined as "ensuring that the computer program of the computerised model and its implementation are correct". Model credibility is concerned with developing the confidence needed by potential users in a model that they are willing to use the model and the derived information (Sargent, 1998).

A model should be developed for a specific purpose or application and its validity determined with respect to that purpose. If the purpose of a model is to answer a variety of questions, the validity of the model needs to be determined with respect to each question. Numerous sets of experimental conditions are usually required to define the domain of a model's set of experimental conditions.

A model is considered valid for a set of experimental conditions if its accuracy is within its acceptable range, which is the amount of accuracy required for the model's intended purpose (Sargent, 1998). This usually requires that the model's variables of interest be identified and that their required amount of accuracy be specified. The amount of accuracy required should be specified prior to starting the development process. Several versions of a model are usually developed prior to obtaining a satisfactory valid model. The substantiation that a model is valid (model verification and validation) is generally considered to be a process and is usually part of the model development process.

It is often too costly and time consuming to determine that a model is absolutely valid over the complete domain of its intended applicability. Instead, test and

evaluations are conducted until sufficient confidence is obtained that a model can be considered valid for its intended application (Sargent, 1982; Shannon, 1975).

Deciding whether a simulation model is valid or invalid can follow either of the following approaches (Sargent, 1998):

1. The development team makes the decision as to whether the model is valid. This is a subjective decision based on the result of various test and evaluations conducted as part of the model development process. This is the most common approach.

2. Independent verification and validation (IV&V). A third independent party is used to decided whether the model is valid. The third party is independent of the both the model development team and the model sponsor or user. The third party conducts an evaluation to determine the model's validity. Wood (1986) makes the conclusion that a complete IV&V evaluation is extremely costly and time consuming for what is obtained.

3. Scoring model. Scores or weights are determined subjectively when conducting various aspects of the validation process and then combined to determine category scores and an overall score for the simulation model (Gass, 1993). A simulation model is considered valid if its overall and category scores are greater than some passing scores. This approach is infrequently used in practice. Sargent (1998) does not believe in the use of a scoring model for determining validity, for the following reasons:

   (a) the subjectiveness of this approach tends to be hidden and it thus appears merely to be objective,

   (b) the passing scores must be decided in some usually subjective way,

   (c) a model may receive a passing score and yet have a defect that needs correction,

   (d) the score(s) may cause overconfidence in a model or be used to argue falsly that one model is better than another.

3.6 Random Number Generation

A distinguishing feature of stochastic discrete-event simulation is the need to produce random values from one or more probability distributions. The algorithms

used to produce these values are called random number generators. Software random number generators are somewhat inappropriately named. A software random number generator is a numerical algorithm; and so the numbers it produces are completely deterministic, not random. For this reason, the algorithms are sometimes referred to as pseudo-random number generators. These pseudo-random typically start with a seed quantity and use numeric or logical operations to produce a sequence of values. In fact, statistical analyses of stochastic simulations are based on the assumption that the software can generate streams of independent random variables with specified distributions. But how is it possible to know that the random number generator is acceptable enough to provide reliable results for a given number of problems? The answer is that it is not possible to know (Page *et al.*, 1999). The quality of the random number generator may be measured through analysing the generator with respect to certain criteria. Building a random number generator that passes all statistical tests (or that gives the correct output distribution for all simulation problems, which is equivalent) is known to be impossible (Page *et al.*, 1999).

Since random number generators do not produce truly random sequences, it is possible that the produced results may be affected by the generator used. Many widely used random number generators have been shown to have quite poor randomness properties that lead to incorrect results in certain applications. It is therefore crucial to use a random number generator that has been thoroughly tested and recommended. Even then, it is still possible that generators that have passed standard tests may not be adequate for a particular application. It is therefore recommended that application programs be run using at least two different random number generators to check that the results are consistent.

True random numbers can only be produced through hardware. A hardware (true) random number generator is an electronic device that produces genuine random numbers as opposed to the pseudo-random numbers produced by a computer program. The usual method is to amplify noise generated by a resistor (Johnson noise) or a semi-conductor diode and feed this to a comparator or Schmitt trigger. If the output is sampled a series of bits are obtained which are statistically independent. These can be assembled into random numbers.

There exists a number of tests which can be used to test a list of random numbers generated by a hardware or pseudo-random number generator to verify that they are random with some degree of confidence.

3.7 Summary

This chapter discussed simulation and modelling. It outlined the different types of simulation relating to this research. The advantages of simulation were discussed. The steps of building a model of a system were outlined, pointing out problems found through other research work in the area. A brief study of random number generators and their effect on simulations was conducted. The next chapter analysis access lists and packet filtering and will outline the requirements and provides a specification for the model to be created.

Chapter 4

Requirements Analysis

4.1 Introduction

This chapter investigates evaluation techniques and specifies performance criteria. The chapter also describes the analysis of access lists, outlines existing problems and describes suggested solutions. Finally, detailed requirements specifications are defined.

4.2 Evaluation and Performance Criteria

To evaluate how well a system meets its specified requirements it will be necessary to measure, or at least estimate, the performance of the system. Performance evaluation is also an integral part of supporting an iterative design process. Optimisation of design cannot proceed without suitable techniques for evaluating the effects of design changes on system performance.

Performance evaluation of access list packet filtering devices is difficult due to their online operations and changeable nature as well as involvement in complex internet communication. Performance evaluation of packet filtering systems under a controlled simulation environment is a good option. Simulation is considered to be an effective tool for the performance evaluation of access list packet filtering. Simulation ensures that more tests can be performed rapidly than otherwise would

be possible. Repeated tests can be carried out under exact conditions allowing comparisons of different approaches.

Simulation requires the use of models of the real systems being simulated. In fact, the evaluation due to a simulation is only meaningful and realistic depending on how representative the model is of the real system. On the other hand, simulation can provide valuable analysis during the development phase of the simulated system. To obtain acceptable results from a simulation of the packet filter, packet streams to be filtered, and access lists as the classifiers, models must be designed and developed.

In order for a system to be developed and evaluated, certain levels of performance must be identified and used as measures of acceptance. Such performance criteria should be identified at an early stage of the system development. Such criteria may differ according to the motivations and objectives.

A rearranged access list will be evaluated with respect to how well it meets its requirements. These requirements will be specified later in this Chapter. Some of these requirements will be performance related. Such requirements may include the ability to process specified packet streams through some order of the access list. Evaluation includes processing different levels of loads and the ability to perform filtering within a specified time frame. The main performance measure in this work is the total processing time for a given packet stream to be processed through a given order of a given access list. This is how one access list can be compared to another access list from a performance point of view. Instead of the total processing time, performance is measured by the average processing time per packet.

The following variables have been identified to have a possible influence on performance of packet filtering:

- The number of rules in an access list and the number of classes into which rules are classified.
- The number of rules in each class and the length of time needed for each rule to process or inspect a packet.
- The characteristics of the packet stream, i.e. number of packets, average time between packet arrivals, number of classes and number of packets in each class.

- The consistency of the results returned by the simulation executions.
- The average processing time per packet when a packet stream is filtered by a given access list.

4.3 TCP/IP Packet Filtering

TCP/IP (Transmission Control Protocol/Internet Protocol) is actually a suite of protocols which work together in a consistent fashion and which has become the standard networking protocol used for Internet communications and networking (Dunkels, 2002). Services residing on network servers can be accessed using the TCP/IP protocol suite (Feit and Sidnie, 1993). Figure 4.1 shows the layered architecture of the TCP/IP suite. Distinct functions are implemented using protocols in different layers. The layered structure reflects the many different procedures which must be carried out to achieve end-to-end communication across a network.

At the Internet layer a number of protocols are used such as routing protocols Inter-Gateway Routing Protocol (IGRP) and the Open Shortest Path First (OSPF). Also the Internet Control Message Protocol (ICMP) is used to notify the presence of errors on the network. The main communication protocol is the Internet Protocol (IP). The IP protocol is a connectionless protocol and therefore does not guarantee the safe delivery of packets transmitted.

IP is the primary layer 3 protocol in the Internet suite. In addition to internetwork routing, IP provides error reporting, fragmentation and reassembly of information units called datagrams for transmission over networks with different maximum data unit sizes.

IP addresses are globally unique, 32-bit numbers assigned by the Network Information Center. Globally unique addresses permit IP networks anywhere in the world to communicate with each other. An IP address is divided into three parts. The first part designates the network address, the second part designates the subnet address, and the third part designates the host address (IETF, 1996).

IP addressing supports five different network classes A to E. Classes D and E are not available to the public. Class A networks are intended mainly for use with a

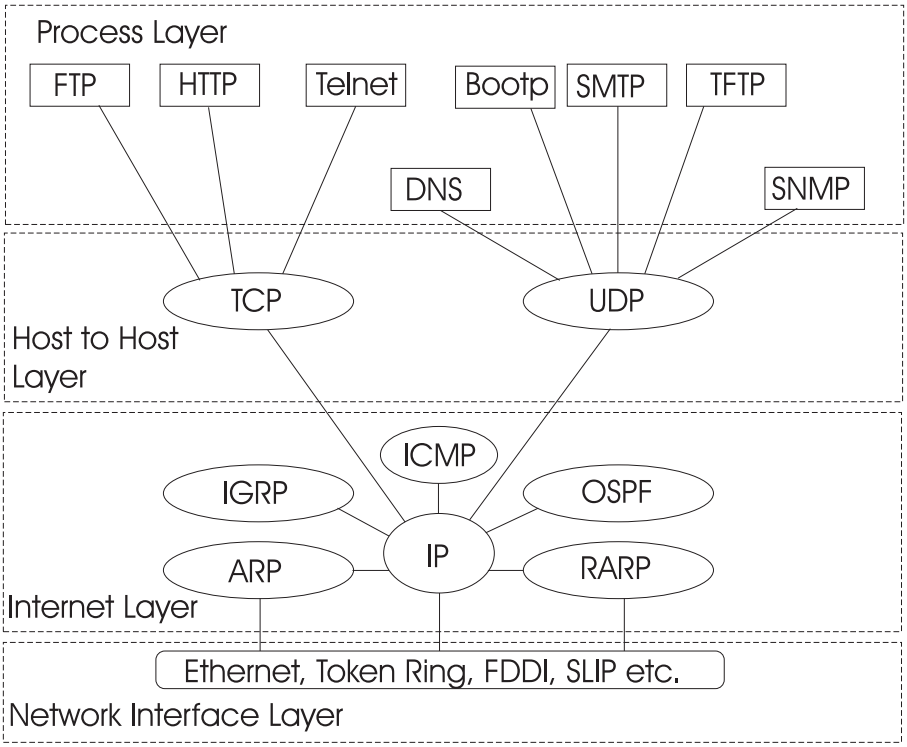

Figure 4.1: The TCP/IP Protocol suite

few very large networks, because they provide only 8 bits for the network address field. Class B networks allocate 16 bits, and Class C networks allocate 24 bits for the network address field. Class C networks only provide 8 bits for the host field, however, so the number of hosts per network may be a limiting factor. In all three cases, the leftmost bit(s) indicate the network class. IP addresses are written in dotted decimal format; for example, 34.0.0.1.

The mask is bit combination used to describe which portion of the address identifies the network and which portion of the address identifies the node. Class A, B, and C networks have default masks, also known as natural masks, as shown below.

Class A: 255.0.0.0
Class B: 255.255.0.0
Class C: 255.255.255.0

TCP/IP (Transmission Control Protocol/Internet Protocol) filtering is achieved by examining the Network and Transport layer packet headers depending on the protocol suite. The information in the packet header inspected by the access list

rules set may include the following:

1. the physical network interface that the packet arrives on,
2. the packet source IP address,
3. the packet destination IP address,
4. the protocol of the packet, type of transport layer (e.g. UDP, TCP etc.),
5. the transport layer source port (TCP/UDP),
6. the transport layer destination port (TCP/UDP),
7. some flags (e.g. a TCP packet contains connection status flags).

A rule set contains a list of rules, each of which is logically in the format: *If (condition) then (action)* where the action is either a packet acceptance or rejection.

An example of a rule for a Cisco router (Cheswick and Bellovin, 1994) is as follows: *Access-list 101 permit TCP 20.9.17.8 0.0.0.0 121.11.127.20 0.0.0.0 range 23 27*

The rule indicates that any TCP protocol packet with an IP source address 20.9.17.8 and destination IP address 121.11.127.20 is to be accepted provided the destination port is in the range 23 to 27.

The mask part of the address determines which part of the address is masked or ignored when the address is being processed. All zero as in the above example, indicates to ignore none of the address parts, which normally identifies a specific single computer. Masking is further discussed in section 4.3.3.

4.3.1 Firewall Approaches

Firewalls use one or more of three methods to control traffic flowing in and out of the network:

1. **Access list packet filtering:** Used in the Internet and Transport layers. This approach is based on using access lists or classifiers to block or to allow packets to pass. The access list consists of a number of rules designed to inspect the fields in the packet headers.

2. **Hiding local addresses:** Used by what is known as Proxies, where the the proxy uses its own address and hides the local machines' addresses when communicating through the Internet. Proxy servers are closely associated with, and are often combined with a firewall. The proxy server is used to access Web pages requested by other computers. When another computer requests a Web page, it is retrieved by the proxy server and then sent to the requesting computer. The net effect of this action is that the remote computer hosting the Web page never comes into direct contact with anything on a private network, other than the proxy server.

   Proxy servers can also make an Internet access work more efficiently. If a particular page is accessed on a Web site, it is cached (stored) on the proxy server. The next time the same page needs to be accessed, it is loaded from the cache from the server cache instead of the remote Web site.

3. **Stateful inspection:** In brief, a table is maintained of the state of all established communication connections. A packet must have the correct details for the specific state of communication to be allowed through, otherwise it is discarded. In most implementations, packet filtering using an access list only follows a successful packet state inspection.

   Packet stateful filtering can be used for any TCP flow to short-cut later filtering. For TCP flows, the filter will follow the ack/sequence numbers of exchanged packets and this information is maintained in a table. The "short-cuts" are extra tests performed on a packet. No alterations are done to the access list rules. Stateful filtering will inspect the packet's ack/sequence numbers and only allow through packets which fall inside the correct window. If a matching packet is found in the table, it is not passed through to the access list.

   For UDP packets, packet exchanges are effectively stateless. However, if a packet is first sent out from a given port, a reply is usually expected as a response, in the opposite direction.

4.3.2 TCP/IP Packets' Structures

Besides firewalls, packet filters are implemented in many other devices such as routers and gateways. Basically, packet filtering restricts traffic of packets flowing between the different networks. The restriction of packets is based on inspecting

the packet header against a number of rules found in an access list. Access lists contain rules that specify which packets should be allowed to pass through and which should not. The rules are individually inspected until one is found which determines the acceptance or refusal of the packet; or until the end of the list is reached.

Traditionally, routers have forwarded packets based only on the destination address in the packet. Increasingly, however, users are demanding, and some router vendors are providing, a more discriminating form of router forwarding known as service differentiation. This process involves mapping different packets to different service classes.

Table 4.1 shows the Internet Protocol (IP) packet header structure. It contains the following fields:

1. Version [VR] (4 bits): it contains the value 4 for IP version 4, and 6 for the more recent version.

2. IP Header Length [HL] (4 bits): indicates the number of 32-bit words forming the header, usually five.

3. Type of Service, [TS] (8 bits): now known as Differentiated Services Code Point (DSCP). Usually set to 0, but may indicate a particular Quality of Service needed from the network, the DSCP defines one of a set of class of services.

4. Size of Datagram [SD](16 bits): contains the total size in bytes, this is the combined length of the header and the data.

5. Identification [ID] (16 bits): contains a 16-bit number which together with the source address uniquely identifies this packet. Used during reassembly of fragmented datagrams.

6. Flags [FLG](4 bits -1): a sequence of three flags (one of the 4 bits is unused). Used to control whether routers are allowed to fragment a packet (i.e. the Don't Fragment, [DF], flag).

7. Fragmentation Offset [FO] (12 bits): contains a count in bytes from the start of the original sent packet, set by any router which performs IP router fragmentation.

Octet	Bits 0	1	2	3	4	5	6	7
1	Version				Header length			
2	Type of service							
3	Total length							
4								
5	Identification							
6								
7	N/A	DF	MF	Fragment offset				
8								
9	Time to live							
10	Protocol							
11	Header checksum							
12								
13	Source IP address							
14								
15								
16								
17	Destination IP address							
18								
19								
20								
21	Options(0 or more of 32 bits words)							
22								
23								
24	padding							

Table 4.1: The Internet Protocol (IP) header structure.

8. Time To Live [TTL] (8 bits): indicates number of hops /links which the packet may be routed over, decremented by most routers. Used to prevent accidental routing loops.

9. Protocol [PTL] (8 bits): Service Access Point (SAP) which indicates the type of transport packet being carried (e.g. 1 = ICMP; 2= IGMP; 6 = TCP; 17= UDP).

10. Header Checksum [HC] (16 bits): a 2's complement checksum inserted by the sender and updated whenever the packet header is modified by a router. Used to detect processing errors introduced into the packet inside a router or bridge where the packet is not protected by a link layer cyclic redundancy check. Packets with an invalid checksum are discarded by all nodes in an IP network.

11. Source Address [SAD] (32 bits): the IP address of the original sender of the packet.

12. Destination Address [DAD] (32 bits): the IP address of the final destination of the packet.

13. Options [OP] (0 or more of 32-bits words): not normally used, but when used the IP header length will be greater than five 32-bit words to indicate the size of the options field.

Table 4.2 shows the Transmission Control Protocol (TCP) packet header structure. It contains contains the following fields:

1. Source Port [SP] (16 bits): when a connection is attempted, or being conducted, this specifies what port the local machine is waiting to listen for responses from the destination machine.

2. Destination Port [DP] (16 bits): when a user desires to connect to a service on a remote machine, the Application Layer program specifies what port initial connections should use. When not used as part of an initial connection, this specifies what port number is going to be used for the remote machine as a packet is being sent out to its destination.

3. Sequence Number [SN] (32 bits): in a sliding window protocol like TCP, the sequence number allows both TCP stacks to know what packets have been received and which ones have not.

4. Acknowledgment Number [TL] (32 bits): this works by acknowledging the sequence number as sent by the remote host. The local host's Acknowledgement Number is a reference to the remote machine's Sequence number, and the local machine's sequence number is related to the remote machine's acknowledgement number.

5. Header Length [HL] (4 bits): just as for the TCP Header Length, this also is a measure of the length of the header in 32-bit sized words.

6. Reserved (6 bits)

7. Urgent flag [URG] (1 bit): this specifies that the Urgent point included in this packet is valid.

8. Acknowledgement flag [ACK] (1 bit): this specifies that the portion of the header that has the acknowledgement number is is valid.

9. Push flag [PSH] (1 bit): this tells the TCP/IP stack that this should be pushed up to the Application Layer program that needs, or requires it as soon as time allows.

Octet	Bits 0	1	2	3	4	5	6	7
1	Source port							
2								
3	Destination port							
4								
5	Sequence number							
6								
7								
8								
9	Acknowledgement							
10								
11								
12								
13	Header length				reserved			
14	reserved		URG	ACK	PSH	RST	SYN	FIN
15	Window							
16								
17	Checksum							
18								
19	Urgent pointer							
20								
21	Options (0 or more of 32 bits words)							
22								
23								
24	padding							

Table 4.2: The Transmission Control Protocol (TCP) header structure.

10. Reset flag [RST] (1 bit): this is used to reset the connection.

11. Synthesis flag [SYN] (1 bit): this is more commonly used to synchronise sequence numbers with acknowledgement numbers for both hosts. Sometimes it is referred to as the synthesis of a connection.

12. Finish Flag [FIN] (1 bit): this is to specify that a connection is finished according to the side that sent the packet with the FIN flag set.

13. Window size [WS] (16 bits): this specifies how many bytes may be received on the receiving side before being halted from sliding any further and receiving any more bytes as a result of a packet at the beginning of the sliding window not having been acknowledged or received.

14. TCP Checksum [TCPCS] (16 bits): this is a checksum that covers the header and data portion of a TCP packet to allow the receiving host to verify the

integrity of an incoming TCP packet.

15. Urgent Pointer [UP] (16 bits): this allows for a section of data as specified by the Urgent pointer to be passed up by the receiving host quickly. It points to the first urgent data byte in the packet.

16. Options [OP] (0 or more of 32-bits words): this specifies various TCP options that are not regularly used.

17. Data (Variable bits): this is the payload, or data portion of an TCP packet. The payload may be any number of application layer protocols. The most common are HTTP, Telnet, SSH, FTP, but other popular protocols also use TCP.

4.3.3 Filtering Mechanism

TCP/IP filtering means filtering using information found in the packet headers. An access list called the classifier is used and contains a list of rules. Filtering rules come in several formats, typically these are proprietary formats. While the expressiveness and syntax of the formats differ, the following is a generic description of how and what functions the rules do. A rule set consists of a list of rules each with a potential action to either accept or reject a packet. The rules are searched one by one to see whether the condition in the rule matches the incoming packet: if it does, the packet is accepted or rejected depending on the action specified, if the condition does not match the packet then the search continues with the remaining rules. If none of the rules match, then the packet is either accepted or rejected depending on the security policy being implemented, that is, to accept or reject all packets not identified by the rule set.

Here are some of the keywords used in some of the different applications for their rules' specifications:

- ***accept, allow, pass*** to accept a packet.
- ***block, deny, reject, drop, refuse*** to block a packet.
- ***in, out*** referring to the incoming and outgoing traffic
- ***all, *, any*** to indicate all packets matching the condition.

- ***from, to*** used to denote: from a source IP address or to a destination IP address.
- ***between, ..*** used to indicate a range for example in a port number. Also, a comma can be used. In some implementations the boolean operators or their textual equivalence are used.
- ***short, frag*** to indicated fragmentation of packets.
- ***opts, ipoptions*** to indicate a packet using the IP optional fields.
- ***quick*** used in filters where the entire list must be checked as the normal procedure. ***quick*** is used to instruct to terminate processing of subsequent rules if the condition in the current rule matches the packet.

In general, a rule can contain any number of keywords and any combination of some number of fields. However, in most implementations, there is a limit of eight fields. Some implementations offer the boolean or conditional operators for use: (<, >, <=, =, >=, <>, !).

Potentially any of the fields can be used, but the most commonly used fields are: *<Protocol>*, *<source IP address>*, *<destination IP address>*, *<source IP port>*, *<destination IP port> and <TCP or ICMP flags>*.
host names are used in some implementations instead of the source or destination address. The following is a brief description of some of the fields, purpose for using them and alias words used to identify them. There are also examples of actual values used or some key words regularly found as values:

1. **Protocol:**

   This is used to specify the protocol in a filter rule. Packets belonging to a particular protocol can be blocked or permitted. The name of the protocol can be any valid name or number. For example, to block all incoming ICMP packets:
   block in proto icmp
   or to allow all incoming IP packets which are of protocol IP version 4
   pass in proto 4

   Some computers might be set to handle a specific protocol and ban that protocol on all other machines.

   Some common protocols that firewall filters can be set for include:

(a) **IP (Internet Protocol)** - the main delivery system over the Internet,

(b) **TCP (Transmission Control Protocol)** - used to split and rebuild transmitted information,

(c) **HTTP (Hyper Text Transfer Protocol)** - used for Web pages,

(d) **FTP (File Transfer Protocol)** - used to download and upload files,

(e) **UDP (User Datagram Protocol)** - used for information that requires no response, such as streaming audio and video,

(f) **ICMP (Internet Control Message Protocol)** - used by a router to exchange the information with other routers,

(g) **SMTP (Simple Mail Transport Protocol)** - used to send text-based information (e-mail),

(h) **SNMP (Simple Network Management Protocol)** - used to collect system information from a remote computer,

(i) **Telnet** - used to perform commands on a remote computer,

2. **IP addresses and Masking:**

Each machine on the Internet is assigned a unique address called an IP address. IP addresses in Version 4 are 32-bit numbers and 128 bits in Version 6. Many applications are now available for handling IP V6. Extended access lists which allows address fields and header format handling. Our research interest will apply similarly to both versions of the protocol. The 32-bit number in Version 4, is normally expressed as four "octets" in a "dotted decimal number". A typical IP address is: 216.27.61.137. For example, if a certain IP address outside the organisation is reading too many files from a server, the firewall can be used to block all traffic to or from that IP address.

The mask is bit combination used to describe which portion of an address refers to the subnet and which part refers to the host. It is also used in filtering as a mechanism to refer to (accept of deny) receiving or sending packets to or from a single computer node or a group of computers. A rule can specify a range of addresses by using masking for both the source and destination addresses. The mask 0.0.0.0, means no masking of any part of the address given. An address is actually a 32 bit number (in IP version 4), which is expressed in the quad notation (four numbers each in the range 0..255). A mask is expressed similarly. It is the decimal representation of the a four 8-bit binary numbers. When expressed in binary format, if a "1"

appears in the mask, then the value of the corresponding bit in the address is ignored when performing the matching.

For example, with the address 20.9.17.8 and mask 0.0.0.0, the address must match exactly. While, with the address 20.9.17.8 and mask 0.0.0.255, is an indication that any address with 20.9.17 as a prefix would match.

The mask is not always expressed in the same format. Many implementations use the long or short format. The short format specifies the number of bits that must match. Other implementations use the hexadecimal form of expressing the mask. The following is an example of three ways of expressing the same mask for the same address:

(a) 20.9.17.8 mask 0.0.0.255

(b) 20.9.17.8/24

(c) 20.9.17.8 mask 0x000000ff

In some implementations, it is possible to use in a rule the domain name of a server instead its IP source or destination address. It is possible in filters to block or accept packets based on the domain name.

3. **Network Interfaces:**

   This is used to identify the interface that the packet is coming from or going to. For example, to drop all inbound packets from a local host coming from the *Ethernet* interface with the name *et10*:

   block in on et10 from localhost to any

4. **IP Fragments:**

   IP fragments in general are difficult to handle. Recent studies have shown that IP fragments can pose a large threat to Internet firewalls, if there are rules used which rely on data which may be distributed across fragments. Fragments can be non-malicious and normal. Some forms of attack produce large numbers of short fragmented packets. Fragments are indicated by the TCP flags field of the TCP packet. It is possible to filter out all fragmented packets or block only the short fragments. To block all incoming packets that are fragments:

   block in all with frag

   To block all incoming packets using protocol TCP that are fragments:

   block in proto tcp all with short

5. **IP Options:**

   IP options are potential risks from a security point of view. Some applications can block packets based on individual options and some collectively block on IP options. For example to block packets that have the "options" set; or to pass packets that do not have the "options" set:

   block in log all with ipopts
   pass in proto tcp from any to any port = 23 with no ipopts

6. **Filtering by Ports:**

   Any server machine makes its services available to the Internet using numbered ports, one for each service that is available on the server. Filtering by port number only works with the TCP and UDP IP protocols. Usually it is used in conjunction with the Protocol field in a filter rule. When using the port number, it is used with a logical comparison operator or with a range indicator.

   To accept incoming packets using protocol TCP with a destination address and mask 10.1.1.2/32 using port number 66667:

   pass in proto tcp to 10.1.1.2/32 port = 6667

   To block incoming packets using protocol TCP from any source address to any destination address using port number 2049:

   block in proto tcp from any to any port = 2049

   To accept incoming packets using protocol UDP with a from source address and mask 10.1.1.2/32 using port number not equal to 53:

   pass in proto udp from 10.1.1.2/32 port != 53 to localhost

   To block incoming packets using protocol TCP with any source addrss and any destination address as long as the port number is greater than 5999 and less than 6010:

   block in proto tcp from any to any port 5999 >< 6010
   block in proto tcp from any to any port 5999 .. 6010

   To pass incoming packets using protocol TCP and UDP on port number is greater than 512 and less than 515:

pass in proto tcp/udp port 512 <> 515

7. **TCP Flags:**

   It is possible with some implementations to compare the flags present in each TCP packet header. It is also possible to inspect individual flags. The flags are represented in the rule by a specific character for each flag. "A" for ACK and "S" for SYN etc. Some applications allow a TCP flag masks to be specified after a "/" character.

   The **ACK** flag allows filtering out of TCP packets which belong to an established connection. This is filtering on packets which have the ACK bit set. The ACK bit is only set in packets transmitted during the life cycle of a TCP connection. It is necessary for this flag to be present from either end for data to be transferred. When the "A" is present in the rule, then it is required to inspecting if the ACK bit to see if it is set, for example to allow incoming (or outgoing in the second example) packets to pass as long as the packet uses the protocol TCP, has the source address and mask 10.1.0.0/16 the port number 23 and the destination address and mask 10.2.0.0/16 and as long as the ACK flag bit and the ACK mask bit are both set:

   pass in proto tcp 10.1.0.0/16 port = 23 10.2.0.0/16 flags A/A
   pass out proto tcp 10.1.0.0/16 port = 23 10.2.0.0/16 flags A/A

   The **SYN** flag is only set during the initial stages of connection negotiation, and for the very first packet of a new TCP connection, it is the only flag set. At all other times, an ACK or an URG/PUSH flag may be set to stop connections being made through incoming requests to the internal network (10.1.0.0) from any where outside network, the rule will be something like: (two flags are to be inspected are: Syn and Ack)

   To block incoming packets on interface le0 using proctocol TCP using any source address with a destination address and mask 10.1.0.0/16 as long as the SYN flag and it mask are set, and the ACK mask bit is set.

   block in on le0 proto tcp from any to 10.1.0.0/16 flags S/SA

   To block the replies to this request (the SYN-ACK's), the rule can be as follows (notice all SYN and ACK flags and masks are set):

   block out on le0 proto tcp from 10.1.0.0 to any flags SA/SA

4.4 Previous Work

This part looks at work done by others in the same field. A lot of work was done aiming at improving performance of packet filters. This section shall be more specific to access list packet filters.

In Denial of Service attacks (DoS), the victim machine is bombarded with packets causing other packets belonging to normal communiation to obey the congestion control algorithms to hold back and eventually starve. Large-scale DoS attacks also interferes with other traffic in that part of the network that is being heavily congested. Mahajan *et al.* (2001) introduced a network-based solution, called *Pushback*, to defend against Distributed DoS attacks. Ioannidis and Bellovin (2002) presented an implementation of the *push-back* concept and the mechanisms involved. The push-back concept treats DDoS attacks as a congestion control problem and involves identifying and preferentially dropping traffic aggregates responsible for the caused congestions.

With multiple levels of fire walling it is possible to increase the level of protection or to provide more protection at different layers (Habtamu, 2000). In fact, distributed firewalls based on Network Interface Cards (NIC) have been proposed, that support the same functions as a centralised firewall (Friedman and Nagle, 2001). This can eliminate the problem of bypassing the firewall in single point of access or centralised implementations. It also eliminates the problem of the single point of failure of protection leaving the entire system exposed.

Packet classification using ad hoc mechanisms like a linear search through an entire list of filtering rules is too slow in practice and a significant source of bottlenecks. In recent years the problem has been receiving some attention. In particular, in the tuple space framework proposed by Srinivasan (1999), the associated simulation results suggest that a significant reduction in search space is achieved, while keeping memory requirements almost linear. The existing schemes for packet classification either have bad worst-case lookup times, or suffer from memory explosion (Warkhede, 2001). Moreover there is evidence to suggest that the time-space trade off for the general packet classification problem is hard to overcome (Warkhede, 2001).

Many of the algorithms which provide fast lookup performance require $O(n^k)$ memory in the worst case, where n is the number of rules and k is the number of

fields in the packet header to be inspected. The following is a list of a number of approaches:

- The simplest approach to packet classification is to perform a linear search through all the filters. This requires $O(n)$ memory, but also takes $O(n)$ lookup time, which can be unacceptably large even for modestly sized filter sets. Caching is a technique that aims at improving the performance of linear search. The idea is that if packets from the same flow have identical headers and a corresponding classification solution, then they can be cached. However, performance is dependant upon how large the number of packets in each flow is. The higher the number of packets, the better the performance becomes. If the number of simultaneous flows becomes larger than the cache size, then performance degrades sharply.

- Some solutions are hardware-based. A large degree of parallelism can be implemented in hardware to gain a speed advantage. Content Addressable Memories (CAMs) can be used very effectively for filter lookup (Lakshman and Stidialis, 1998). However, it is difficult to manufacture CAMs with a wide enough word to contain all of the bits in a filter. Many hardware-oriented schemes rely on heavy parallelism, and represent significant hardware constraints. The flexibility and scalability of hardware solutions is very limited (Warkhede, 2001).

- Srinivasan (1999) presented the Tuple Space Search algorithm and described a heuristic called "*tuple space pruning*" which performs best matching prefix lookups on individual fields to eliminate prefix length combinations that cannot match the query. This heuristic is expected to reduce search space on average, but does not provide any improvement in the worst case.

- Several existing firewall implementations do a linear search of the database of rules, and keep track of the best matching filter. Some implementations use caching of the full packet header to speed up the process of future lookups (Srinivasan, 2001). The cache hit rate is between 80 and 90 percent, and is likely to be much worse for caching full headers. Incurring a linear search cost to search through 10,000 filters is a bottleneck even if it occurs for only 10 to 20% of the packets (Srinivasan, 2001).

- Binary decision diagrams are another way of implementing packet filtering to solve a number of the problems. A decision diagram is a method of representing a Boolean expression. Each bit in the packet header is represented as a Boolean variable. The groups of bits are represented by a

group of Boolean expressions over these variables. Each rule in the list is represented as a condition on the bits in the packet header. The rule will indicate acceptance if the expression evaluates to true. Dynamic access lists have been proposed by Hazelhurst *et al.* (1998) to increase flexibility and security (Wulf, 1997).

Bryant (1992) introduced the concept of a reduced, ordered binary decision diagram that can reduce the decision diagram to represent the Boolean expression $((x_1 \cup x_2) \cap x_3)$ from the diagram in Figure 4.2 to that in Figure 4.3. The dashed and solid lines in the diagrams indicate the branch when the decision variable is 0 (No) or 1 (Yes).

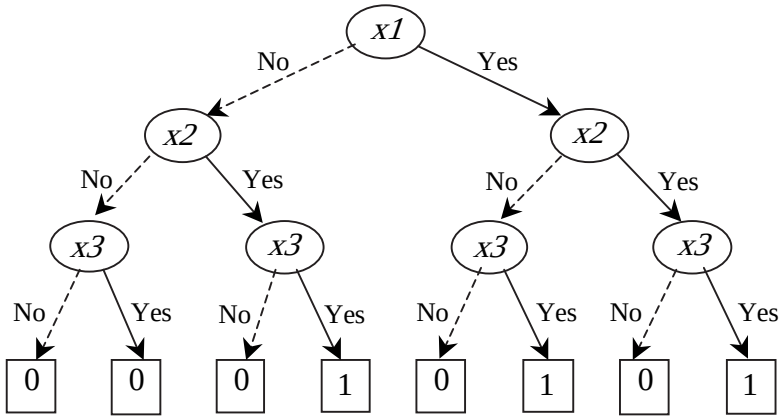

Figure 4.2: Binary Decision Diagrams representation for $((x_1 \cup x_2) \cap x_3)$

Algorithms for converting rule sets into Boolean formula have been produced and tested. Binary decision diagrams produce encouraging results (Hazelhurst, 1999).

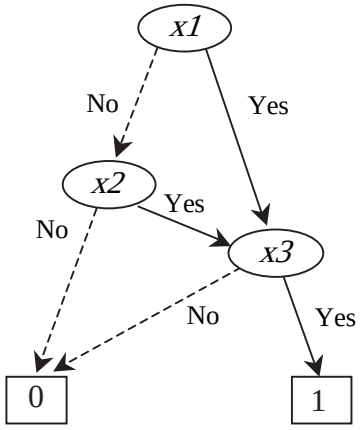

Figure 4.3: The reduced ordered Binary Decision Diagrams for $((x_1 \cup x_2) \cap x_3)$

Binary decision diagrams solve a number of the problems in packet filtering but there are other problems that need to be investigated. The problems

relate to the effect of the ordering of variables on the size of the diagram produced, the effect on the overall lookup performance, and on memory requirements.

4.5 Proposed Approach

All implementations of access list firewalls comprise a list of rules that are applied to inspect the incoming or outgoing packets. Most implementations of packet filtering incur a lookup latency linear in the number of rules in the access list.

A linked list structure is used to store the access list rules. Searching is performed sequentially through the list. This has the advantage of using a small amount of memory for storing the list, but the lookup time is linear in the size of the list.

Access list rules operate by inspecting packets against each rule in the access list until a clear accept or reject rule is met which applies to the packet. Otherwise, all rules in the list are checked. From a performance point of view, it is desirable that each packet meets that critical (accept or refuse) rule at or near the start of the list. This will reduce the time required for the packet to be inspected and consequently improve performance. If the critical rule for a packet is met at or near the end of the list, or the rule is never met, the processing time for the packet will be very high and performance will degrade.

It is feasible to classify access list rules into different classes based on what fields they inspect. Rules in an access list are classified into different classes and are then ordered according to the class. In this research, the performance of different orders or classes filtering the same packet stream were tested.

It is also assumed that packets flowing into a network device can be monitored through a log over a period of time. It is possible to predict the numbers and classes of arriving packets and develop a packet stream profile. If it is assumed that most of the packets in a packet stream are of class "x", then placing all class "x" rules in the access list on the top, in such a way that they are the first rules to filter an incoming packet will improve performance, see Figure 4.4. All or most packets would therefore need only a few rules to be checked at the start of the list before a critical rule for acceptance or refusal is reached. While, if the list was ordered in such a way that class "x" rules are placed at the end of the list, then

for each packet arriving, most of the rules in the list have to be checked before eventually reaching the needed class "x" rule at the end of the list. The more rules in an access list that are checked, the higher the processing cost for a given packet.

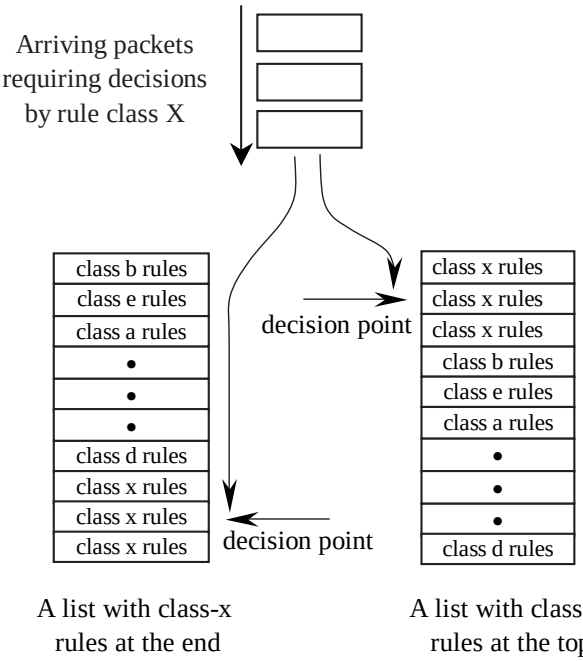

Figure 4.4: Effects of rules list type ordering on arriving packets

The researcher is suggesting that if a profile of arriving packets can be developed for a network device then it will be possible to determine the pattern of these packets (Gupta and McKeown, 2001). Classification of packets arriving in the past can determine the types and numbers of packets arriving. This pattern can periodically be inspected to determine the most effective ordering of the rules list.

4.5.1 Rules Reordering

An assumption is made here that access lists can relatively easily be reordered to facilitate better performance for a particular pattern of arriving packets. However, that is not the case, as changing the order of the rules within the list can in fact change the acceptance or refusal of a packet (Hazelhurst *et al.*, 1998). Reordering access rules can be a difficult and dangerous operation. The real difficulty is in ensuring that the list is or remains an accurate translation of the security policy in as far as permitting those packets intended to pass and blocking those intended

to be blocked. Ordered lists require some form of validation to ensure truthful adherence to the security policy. Part of the problem is the time constraints on actually reordering and verifying. This problem of the effects of ordering the list on the security policy is not the subject under research in this thesis. One way of eliminating this time constraint problem is to have a number of versions of the access list pre-ordered and verified (Hazelhurst, 2001), so in real time no actual ordering takes place, only a selection of the most suitable access lists is made.

4.5.2 Internet Survey

The following details were collated after considering results collected from over 793 packet classifiers from 101 ISPs (Internet Service Providers), with a total of 41,505 rules (Stoica, 2001). The following are some of the characteristics extracted from those surveys which will be referenced throughout the simulation:

- The mean number of rules in an access list is 50 rules.
- The maximum number of rules in an access list is 2734.
- Only 0.7% of the access lists contained over 1000 rules.
- Many fields are specified by ranges, i.e. port numbers.
- Rules in the same classifier tend to share the same fields.
- 8% of the rules are redundant, i.e. they can be eliminated without changing the behaviour of the access list.
- Rules with one field per rule form 17% of all rules.
- Rules with three fields combinations in a single rule form 23% of all rules.
- Rules with four fields combinations in a single rule form 60% of all rules.
- The maximum number of fields that can be specified in a single rule is limited to eight, though there are a lot more fields that can be tested (Gupta and McKeown, 1999):
    1. Source address field (Network-Layer) (32-bits).
    2. Destination address field (Network-Layer) (32-bits).
    3. Source port number field (Transport-Layer) (16-bits).

4. Destination port number field (Transport-Layer) (16-bits).
5. Type-Of-Service field (8-bits).
6. Protocol field (8-bits).
7. Flags (Transport-Layer) protocol flags (8-bits).
8. Fragmentation ID (16-bit).
9. Interface used.

Consider an access list of 1000 rules, with only one rule in the list which can determine the fate of an incoming packet. If this rule is at the beginning of the list and happens to be the first rule the incoming packet meets, then processing is very fast. If this rule is last in the list and happens to be the last rule the packet meets, then processing will take the maximum possible processing time. If this particular rule can be identified out of all the other rules through some form of classification of rules, then it can easily be placed at the beginning of the list. Similar rules classified in a similar class can all be moved up to the top of the list.

Grouping of rules is best based on what inspection the rules perform. Rules that perform similar inspections may be classified into the same group or class. The number of classes may vary from one network device to another as inspections performed by different devices may differ. The difference depends on the nature, function and location of the device in a communication system. The number of classes can take any value between two extremes, that is one class or as many classes as the number of rules. In other words, all the rules are classified into a single class or many classes with each class containing one rule only.

The optimum number of classes is difficult to define, and will vary from one network device to another. But the general guideline is that classification is based upon the fields inspected in the particular list at the particular device.

This maximum number of classes in an access list will be the sum of all the unique combinations of every number of fields taken 1, 2, 3, etc. at a time.

4.5.3 Packet Frequency

The frequency of packet arrivals should reflect the ongoing operational network device, a device that can usually handle the traffic it is receiving. The processing

ability of the device must be more than the processing required by the arriving packets or else the device will not be a practically operational device in a network. The simulation system must cater for the queueing of arriving packets that arrive into a busy device. Queued packets wait until they can be handled without any loss of packets. Such a situation may occur with high surge of communication activities. The total processing time for a packet is the time between packet arrival to the device and the packet departure including waiting time:

$$\text{Processing time} = \text{waiting time} + \text{servicing time}.$$

The frequency of arrival can dramatically change the performance values. One must be careful when measuring average processing time per packet for a number of packets that arrive in a burst and other packets that arrive well dispersed. Consider the following example depicting two different groups with extreme frequencies of arrival. In one group, a packet arrives at or after the previous packets finishes processing, packets never have to wait. With the second group all packets arrive at very short time intervals. Except for the first packet in group 2, all other packets are forced to wait.

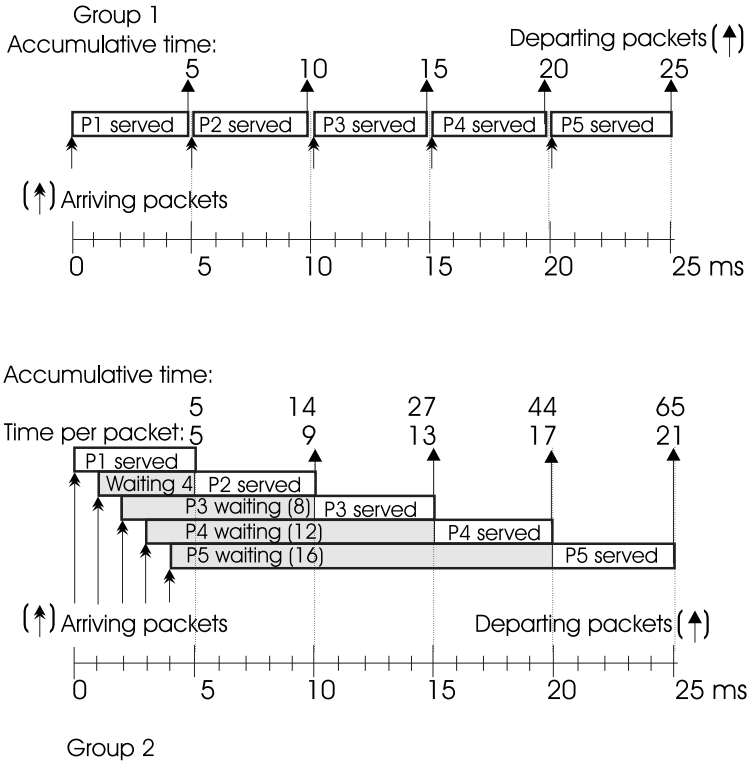

Figure 4.5: Effects of rate of arrival of packets on average processing time

Assume the time is measured in one μs (1/1,000,000 of a second) and that all packets will need exactly the same processing time of 5 μs per packet. The Gant charts in Figure 4.5 show the arrival, waiting, service and departure times of each packet. In group 1, the chart reflects servicing time only and none of the packets are forced to wait. Every packet spends 5 μs in the system from its time of arrival to its time of departure.

However, in group 2, the chart reflects a waiting time and a servicing time. The waiting time is 0, 4, 8, 12 and 16 μs for packets 1 to 5 consecutively. Table 4.3 shows the details of waiting, service times and their sum, represented by the total processing time column. It also shows the arrival and departure times for each packet also add to its total processing time. The total processing time for all five packets is 65 μs.

Packet number	arrival time	waiting time	service time	departure time	total processing time (depart time - arrive time) or (wait+service times)
Packet 1	0	0	5	5	5
Packet 2	1	4	5	10	9
Packet 3	2	8	5	15	13
Packet 4	3	12	5	20	17
Packet 5	4	16	5	25	21
Totals					65

Table 4.3: Effects of frequency of arrival of packets on average processing time.

After processing the five packets in both cases, a large difference in the average processing time is noticed between the two groups:

Group 1, average processing time per packet = $\frac{(5+5+5+5)}{5} = 5\mu s$.

Group 2, average processing time per packet = $\frac{(5+9+13+17+21)}{5} = \frac{65}{5} = 13\mu s$.

When all other variables unchanged, this difference in performance is attributed solely to the different rate of arrival of packets in the two groups. This must be taken into consideration in our simulation to ensure that any changes in performance that are due to different rates of arrival are not attributed to the rearrangement of access lists.

4.6 Problem Definition

Access list packet filters can be fast, efficient and effective. That does not mean they are free of problems. On the other hand, simulation is an extremely useful means of developing and testing systems and concepts. It is not free from problems. Also a number of problems were identified with both simulation and access list packet filtering. Some of the problems identified which relate to access lists are:

1. As access lists grow in size their level of complexity increases and the task of modifying them becomes difficult.

2. One difficulty is in ensuring that the list is an accurate translation of the security policy, permitting those packets intended to pass and blocking those intended to be blocked.

3. Access lists may last for several years and so may be changed from time to time (rules may be added or deleted, old rules changed, or the order of the rules change). Nevertheless, an access list is relatively static. The problem with a static access list is that the level of security is relatively static also (Hazelhurst, 2001).

4. Since the rules are checked in order, the order in which they are specified is critical. Changing the order of the rules could result in some packets that were previously rejected being accepted (or vice-versa).

5. Large access lists also suffer from the length of time required to inspect each packet against all the rules in the list. Filtering becomes a potential bottleneck for communication. This is the problem the research work is concerned to address.

There are a number of problems with simulation and simulation models:

1. Complexity of the model representing the real system. One problem with network models can be due to the large number of parameters (Jobin *et al.*, 2001). Another problem is that realistic values for such parameters are not always obvious. Further more, the sensitivity of the results to these parameters is not well established. These issues and the cumbersome number of parameters makes it difficult to compare different models against each other. Ideally, it is desirable to have as few parameters and metrics in the model as possible. It is of the utmost importance to maintain the model in

the simplest representation possible and yet still emulate the real system it is meant to represent.

2. Difficulty in interpreting and analysing the results obtained using the model. Simulation can produce large amounts of data. The data often includes many different values for many different variables or parameters. Knowing that a change in the data produced is due to changes in some group of variables and not due to changes in another group of variables can be difficult.

3. Model and simulation abstraction. The idea of simulation abstraction has been discussed in Huang *et al.* (1998). Attempts were made to abstract unnecessary details from a simulation. Fall (1997) suggests decreasing the number of objects in a simulation by aggregating some of them, without loosing the integrity of the model and as long as it remains a true representation of the system it simulates.

4. Model Conformity. The correctness of the results obtained from a simulation execution of a model will depend on the validity of the model itself. The validity of a model depends on how much it conforms with the real system it intends to represent. Obtained results can be misleading if the model is not a true representation of the system.

The task at hand in this research work can be specified by a number of steps. The first is the creation of a correct model for a network device which performs packet filtering. This is followed by execution of a number of experiments as simulations of incoming packet streams being filtered by different access lists. Those access lists are arranged differently based on the rules' classes. The results of these simulations are anlaysed to confirm or otherwise that better performance for access list filters can be obtained through some specific order of the access list classes. The next step is attempting to obtain an algorithm that will give the arrangement which will produce the best performance possible. Finally, if possible, seek more efficient methods of the algorithm itself.

4.7 Requirements Specifications

The requirement specifications is divided into three parts. The Simulation model requirements, the System requirements and the Interface requirements.

4.7.1 Simulation and Model Requirements

This lists requirements which must be implemented as a minimum standard for the model design and the simulation operation. This includes the following:

1. The model must include the packet streams, access lists and the actual simulation of the filtering operation.

2. Packet streams used must include small and large numbers of packets.

3. The average time between arrivals of packets in any packet stream must be larger than the time taken by any single packet to be filtered by all rules in an access list. This ensures that the queuing of packets waiting to be serviced only happens in normal packet arriving bursts. Consequently, unnecessary queuing will not influence performance. Elimination of all waiting time from calculations must be tested, so pure filtering service time is taken into consideration.

4. The number of packet classes in packet streams as well as the number of packets in each class must test as wide a range of values as possible.

5. In packet streams there should be some packets which may not be identifiable by the access list in use.

6. The access lists used must include a wide variety of access list specifications including total number of rules, each to include a variety of number of classes; with different numbers of rules, and different average processing times for rules in each class. For each access list produced, there should be many copies which only differ in the way rules are arranged based on their class.

7. The access list model must allow for equal processing time of all rules in the list; to allow for some appropriate tests to be done.

8. All of the access list rules and packet streams must be saved to allow for inspection, reuse of combinations and repeating or re-execution of the same simulation.

4.7.2 User Interface Requirements

This lists requirements which must be implemented as a minimum standard for the simulation interface. This includes the following:

1. The user interface to the system must be simple, clear, easy to use and functional.

2. The Interface must allow for easy creation of single or multiple access lists. It must also allow for the display and re-organisation of any access list based on the classes, in any order.

3. The interface must allow the user to create, display, edit and store the average processing times for rules in each class of the different access lists.

4. The interface must allow the user to create a single packet stream, to create many packet streams and to display any packet stream.

5. The interface must allow for the execution of a single simulation or a multiple number of simulations. It also must allow the user to display the simulation results for a single simulation or a summary of a number of simulations.

4.7.3 Functional Requirements

This lists the requirements of the functions which must be implemented as a minimum standard for the system. This includes the following:

1. The system must allow for model evaluation and verification through testing and modifying the model.

2. The system must be able to compute the total processing time for any given packet stream when filtered by any specific arrangement of any given access list.

3. The system must be able to compute the permutation for any given access list which will yield the lowest processing time when filtering any specific packet stream.

4. The system must produce the permutation for any given access list which will yield the lowest processing time when filtering any specific packet stream in ways other than through exhaustive cost calculations for all permutations.

5. The system must be able to perform the simulation operation of packet filtering for a given access list and a given packet stream.

6. The system must be able to produce a summary of a simulation execution results containing at least the average processing time per packet for any given simulation.

7. The system must be capable of performing the needed simulations and providing the results in order to reach definite conclusions with regard to the effects of rearranged classes in an access lists.

4.8 Summary

This chapter provided a detailed analysis of access lists and packet filtering. TCP/IP protocols were described and evaluation criteria were considered. Considerations were given to what other research has been carried out and looked at, and identified potential problem areas. The proposed approach was outlined and the requirements for the simulation, interface and system functionality were listed. The next chapter will outline the design of the system model.

Chapter 5

Design

5.1 Introduction

The previous chapter presented an analysis of packet filtering on the Internet. It outlined some problems and potential improvements. It stated the requirements for an improved packet classifier. This chapter starts by describing the design related aspects of packet filtering. It continues by describing the models created to perform the simulation of the proposed access list filtering. Detailed algorithms and flowcharts are presented. The design also outlines the test specification for the created model and the new implemented method.

5.2 Access List Filtering

The way access list rules operate is that a packet is inspected against each rule in the access list until a clear accept or reject rule is met which applies to the packet. Otherwise, all rules in the list are checked. From a performance point of view, it is desirable that each packet meets that critical (accept or refuse) rule at or near the start of the list. This would reduce the time required for the packet inspection and consequently improve performance. If the critical rule for a packet is met at or near the end of the list, or the rule is never met, the processing time for the packet will relatively be very high and performance will degrade.

An access list consists of a number of rules. The number of rules varies from one

implementation to another. The sophistication level of the security required to be achieved also determines the number of rules in the access list. Unsophisticated security is considered very simple security. Such simple security is that which can be expressed with very few rules. Extreme cases such as "Allow access to all" or "Block all" will require no rules or one rule at the most. Another example that may require a single rule is allowing only packets from one particular network or blocking only packets from one particular network. As security becomes more sophisticated, more rules are needed: for example, if a number of different groups or different individuals from the same network need to be allowed or to be blocked. The level of sophistication may also increase when more conditions are required: for example if the same groups or individuals are only allowed or blocked when using specific port numbers or a certain service.

5.3 Solution Design Overview

Rearrangement of an access list based on the classes will suit an arriving packet by reducing the processing time for such a packet. Ideally, for each arriving packet, if its filtering rule requirement is known, then those rules are used first to inspect the packet. If every time a packet arrives it is possible to recognise the class of rules it requires, and if the list of rules then can be rearranged accordingly to suit the packet requirement before the filtering operation for the packet starts, then packets will meet their critical rules in the list early on at the start of the list. But, there are a number of problems with this approach:

1. In order to identify the class which a packet belongs to, it is necessary first for the packet to go through the list. After all, the list is the classifier. To overcome this problem it is suggested to resort to packet flow profiling for packets arriving into a network device based on observations of past packet classifications. This packet flow profile will for an individual network device provide some idea of the expected numbers and classes of packets. When a packet flow profile is developed or can be identified, some arrangement of the rules is selected to suit this profile. The arrangement will need to be changed when the packet flow profile changes.

2. The second problem is that once a packet flow profile for a device is identified, it is difficult to predict the best way to arrange the classes in the list to produce the best performance possible for such a pattern. This is one of

the main objectives of this thesis, i.e. to find out what is considered to be the most efficient arrangement as far as performance is concerned.

3. The third problem is that even if a packet profile requirement becomes known, the difficulty is in the time constraints on actually reordering and verifying the validity of the new order in its truthful adherence with the security policy. One way of dealing with this is to have a few versions of the access list pre-ordered and verified to suit the expected few packet flow profiles. A similar approach was suggested by Hazelhurst (2001). So, in real time neither actual arrangement nor verification takes place; only a selection of the most suitable arrangement of the access list is made.

Reordering access rules can be a difficult and dangerous operation. The real difficulty is in ensuring that the list is or remains an accurate translation of the security policy in permitting those packets intended to pass and blocking those intended to be blocked. This problem is minimized by the fact that access list rules are not ordered in their entirety. The rearranging of the list is based on the rules' classes. That is, all the rules belonging to one class are moved up or down in the list as a unit. This means that rules of the same class retain their order relative to the rules of the same class. This will have fewer negative consequences on security than a full reordering of the rules in the list. Rules in separate classes are expected to be independent of one another and mostly are expected to have no effect on each other. There may be cases where rules in different classes can change the security policy if their order is changed. Such cases need to be included in the same class or such rules are broken into more basic rules. More investigation will be needed with regard to the effects of reordering. The topic of verification of access list adherence to the security policy is a separate issue and is outside the scope of this thesis.

One way of benefiting from the rearranged rules of an access list is in a system which experiences regular profile changes of the incoming packet stream. Figure 5.1 represents a block diagram of how rearranged access lists can be incorporated in a system. An example is a network device receiving a stream of packets with the numbers of packets in the different classes fluctuate, while the same security policy is used. In such a system, based upon this given security policy an access list is created. Multiple versions of the same list are created, verified and stored. During normal operation, classifications of incoming packets are analysed. When a profile change is sensed in the incoming packet stream, the most suitable list organisation is selected to be used as the classifier. Such a system consists

of two main parts. One part performs the packet classification and the second maintains logs of classifications of arriving packets, performs profile analyses and determines the best access list arrangement to be used by the classifier.

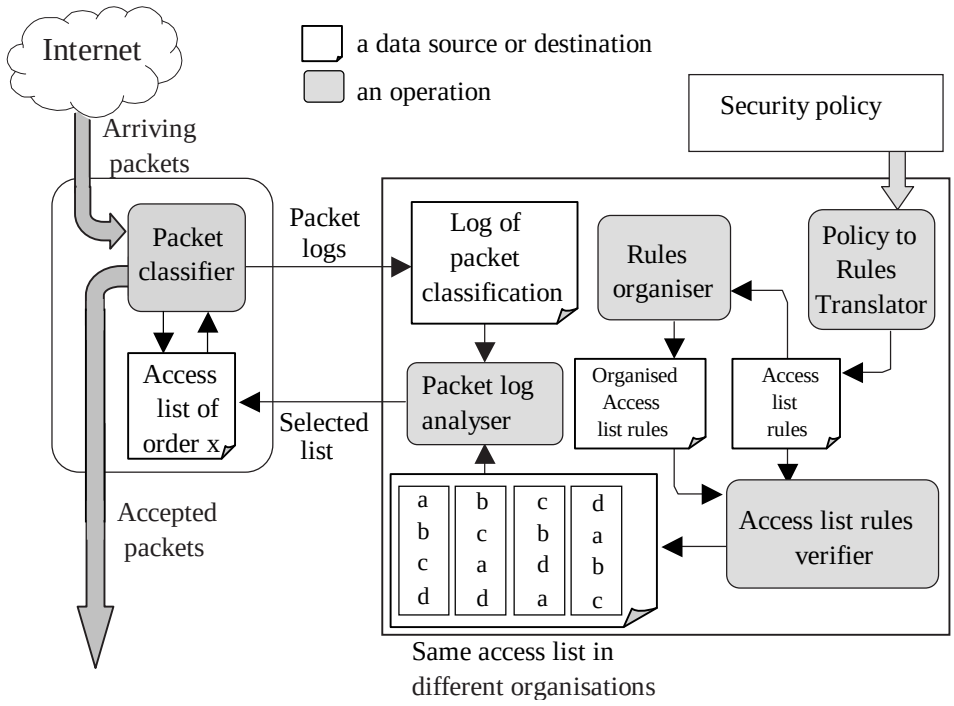

Figure 5.1: Outline of the proposed ordered access list filtering system

In this system, the security policy is translated into a list of rules called an access list. Many copies of this access list are made in different arrangements. The access list rules verifier compares two access lists and verifies that they implement the same security policy. Basically, it verifies that a particular change in organisation does not alter the security functionality of a particular access list. Accepted lists made up of the same rules but in different organisations are saved by the verifier for future use by the packet classifier. The packet log analyser can be a real-time analyser of packet classifications. The packet log analyser determines the best version of the access list to use based on the results of the analysis of the packets logs. This selected order of the access list rules remains in operation for a period until the next analysis is performed. This period of time can be different depending on the profile stability of arriving packets arriving into an individual network device and is expected to vary from one network to another, or from one filtering device to another on the same network. The packet classifier starts using the selected version of the list to classify packets arriving from the Internet. The rules organiser produces different lists with different class rules placed at the top of the list in each of the lists. It is possible to implement this system in an

effort to produce a faster classifier provided that rearranging the rules based on their classes will actually produce faster processing time. The rest of this thesis describes the work done to test if organizing access list rules in such a way can produce lower average processing time per packet for a given packet stream and outlines the actual outcome.

One of the objectives of this research work is to find out if changing the order of the rules in an access list based on their class will change the performance of packet filtering. A second objective is to find out the best arrangement of the classes of the rules of a particular access list to filter a particular stream of packets, i.e. the arrangement which will give the least average processing time per packet for a particular packet stream. This work is done through a simulation of the filtering operation. It simulates the filtering of a stream of packets through a list of rules in an access list. The performance is observed, then the simulation is repeated using the same stream of packets being filtered by different arrangements of the same access list. The more different arrangements created of the same access list and used to filter same stream of packets, the more comprehensive the test.

This approach is based on the assumption that it is feasible to classify access list rules into different classes. This classification will be based on real classifications observed on real classifiers on the Internet. The approach is also based on two points with regard to packets. The first is that packets can be classified into classes, and the second is that a profile of a packet stream arriving into a network device can be identified.

So, it is proposed for the purpose of this work to first classify rules in an access list into different classes. Secondly, a number of access lists are generated, and each is reproduced in as many different organisations as possible based on the rules classes. Also a number of packet streams are generated with many different profiles. All those generated access lists and packet streams are saved so either can be reused to perform any filtering simulation.

Simulation runs are then carried out to test the performance of different arrangements of access lists. Results are saved, and performance as well as the different configurations of the access lists and packet streams are analysed. Conclusions are drawn, and when necessary, more access lists and packet streams are generated and more simulation runs are executed to verify or refute such conclusions.

5.4 Modelling the Filtering System

The simulation consisted of three main integrated models namely, the access list, the packet streams and the filtering and reporting models. One other part was specifically introduced for analysing the results of the simulation. The three models are integrated to reflect the operation of a network device performing access list based packet filtering on received packet streams. The models allow specific streams of arriving packets to be checked against specific lists of access list rules. Figure 5.2 shows the general structure of the models and how they relate but each model will be described in detail in the next three sections. The operation of the different models can be summarised by the following steps:

1. The generation, classification and reorganistion of access list rules.
2. Classification and generation of arriving packets and the creation of different packet flow patterns.
3. Provision of some performance measuring parameters to determine the effects of changes in access lists.
4. Investigation of problems and solutions for arranging access lists.
5. Analysis of results achieved and the determination of the validity of the hypothesis of improved performance due to the change in arrangement of access list rules.

The models are formulated based on detailed surveys of studies conducted by Gupta and McKeown (1999). The details of the findings were described in the previous Chapter. Below are some of the more relevant findings:

- The mean number of rules is 50.
- Only 0.7% of the access lists (classifiers) contained over 1000 rules.
- Rules in the same classifier tend to share the same fields.
- 8% of the rules are redundant, i.e. they can be eliminated without changing the behaviour of the access list.
- Rules with one field per rule form 17% of all rules.
- Rules with three-field combinations in a single rule form 23% of all rules.
- Rules with four-field combinations in a single rule form 60% of all rules.

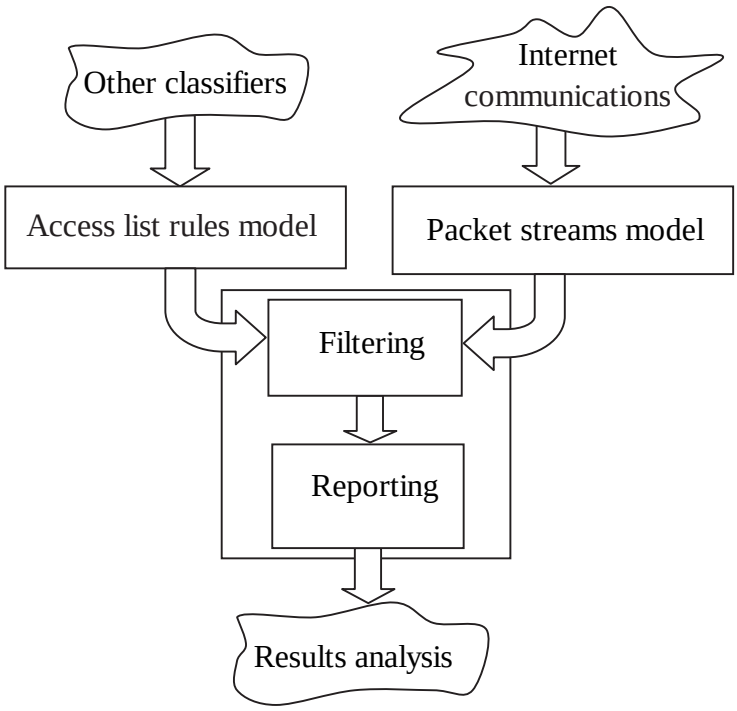

Figure 5.2: Block diagram of the system's simulation model

- The maximum number of fields that can be specified in a single rule is limited by eight.

The survey showed that rules were generally divided into groups of 17%, 23% and 60% based on the number of fields they inspected, one, three or four fields respectively. Apparently, there were no rules; or very few indeed; that had two or five fields.

5.5 The Access List Rules Model

The access list rules model is concerned with the generation of a number of lists each containing a number of rules specifications. This involves the classification, organisation (or ordering), the generation and storage of the rules. The rules model is specified by the following:

1. Rules classification

2. Access list rules specifications

This is defined by a number of parameters the values for which can be specified during the generation operation. The parameters that determine the specifications for a set of rules are:

(a) Total number of rules in the list.

(b) Number of classes within the list.

(c) Number of rules in each class.

(d) Average processing time for rules in each class.

3. Access list rules organisation.

4. Rules definition specifications.

5. Access list files specifications.

5.5.1 Rules Classification

Classifying the rules is based on what inspection the rules perform. Rules performing similar inspections on the same packet fields may be classified into the same class. In the real world, the number of classes will vary from one network device to another. The difference will be due to the nature, function and location of the device. The number of classes can vary between two extreme values, One-class and as many classes as there are rules. That is when all the rules are grouped into one class, or many classes with each class containing only one rule. Neither of these two extreme cases is useful for our study. In the first example all the rules perform the same type of inspection, i.e. all rules check a port number. In such cases, it is possible to classify rules based on something else, like different ranges of the port number. The other extreme case is where the number of classes approaches the number of rules. Such cases defeat the purpose of classification. The optimum number of classes is difficult to define, and may differ from one device to another.

Rules classification is really based on the packet header fields that are inspected by the rules in a particular list in a particular device. Some rules will inspect one field, others will inspect two fields and so on. Some fields are more common than others. The survey showed that rules were generally divided into three groups based on the number of fields they inspected, one, three or four fields. These three groups made 17%, 23% and 60% of the total number of rules respectively.

It is worth pointing out that this 3-field rules that makes up 23% of rules refers to a list that may contain many different 3-fields classes, not the same 3 fields. The number of classes containing three fields will depend on the number of different combinations (or permutations) of three fields that can be made up of eight or more fields. The numbers of classes will be discussed later.

5.5.2 Access List Specifications

1. **Total number of rules:**

   Based on the survey results carried out by Gupta and McKeown (1999), the maximum number of rules in an access list found was a rare 2734, with an average of 50 and very few indeed (0.7%) over 1000 rules. If the number of rules is very small, as in 2 or 3 rules, then the savings in processing time will not warrant the cost of classification and organising. Only sizes of 20 rules or higher will be considered.

   The main reason for keeping a low number of rules (an average of 50) is thought to be the difficulty of maintaining and formulating a large list. The need for larger lists is expected to increase in the future, as higher levels of security will be needed. On the other hand, it is expected that better systems will be developed for simplifying the task of creating and maintaining long access lists. For the purpose of our simulation, larger list sizes are considered up to a size of 2500 rules. Therefore, list sizes included vary between 20 and 2500. Other values in between reflect the average or the expected increase in the average number of rules to be somewhere between 50 and 100 rules. Thus, starting with 20, multiples of 5 will provide close enough values to test making the values to be tested as follows: 20, 100, 500 and 2500 rules in access lists.

2. **Number of classes:**

   The number of rules depends on the amount of security checks that need to be carried out, but the maximum number of classes the rules are divided into is determined by the maximum number of different combinations of the packet fields that can exist. That can be easily calculated as the sum of all the unique combinations of all the fields, given by: $2^n - 1$ where n is the available number of fields to select from. For example, with four fields to select from, the formula will calculate how many unique combinations

can be obtained from a total of four fields when one field is picked, two fields, three fields and four fields are picked at a time. In this example, the combinations would be: $2^4 - 1 = 15$ combinations (1, 2, 3, 4, 12, 13, 14, 23, 24, 34, 123, 124, 134, 234, 1234), as shown in Table 5.1. Also notice that it is not a *permutation* needed here, it is a *combination*, for the selection of the two fields: field-1 followed by field-2, (1,2) is the same as selecting field-2 followed by field-1 (2,1). This formula is helpful in computing all the possible combinations for a given number of fields.

It is rather more relevant to find the combinations when an individual number of fields (k) are picked at a given time from a total n number of fields. In the example above, the number of combinations when only two fields are picked from the available four fields, were 6 combinations. For this reason the formula is reconsidered in a different format:

$$\sum_{k=1}^{n} c_n^k$$

Where n is the entire number of fields to select from at any time, c is the unique combination of n fields taken k fields at a time.

The maximum number of unique combinations for one given number of fields (k) selected from any one given available number of fields (n) can be calculated as:

The number of unique combinations for any single number of fields

$$c_n^k = \frac{n!}{k!\,(n-k)!}$$

Where: c is the maximum number of combinations, n is the number of fields to select from, k is the size of each combination.

Table 5.1 shows the maximum number of possible combinations, which can be obtained from (n) number of available fields for different combinations taken (k) fields at a time. Note that the number of different combinations found in a single access list is not necessarily equal to the maximum possible. It is also impossible for some access lists to have the maximum possible combinations used, as the number of rules in an access list (average 50) is much less than the maximum number of combinations for 6 fields. It is logical for the maximum number of classifications of rules to be less than the number of rules, to avoid the extreme case of having one rule per class. In fact, because a class may contain a number of rules, that will leave many

other classes with zero rules. Even in a case of say 100 rules divided into 3 classes, it is possible for one class to have 98 rules, and the other two classes have one rule each. In such cases, reorganising the list will have a very minor effect. Though it may be normal to have some classes with only one rule in each, it is not desirable to have one overwhelming class with a relatively very large number of the rules. The author is suggesting that no single class will make up more than 75% of the rules, and therefore, in any list the number of classes should not exceed one quarter of the number of rules in that list.

Based on this maximum value for the number of classes in an access list, (that is not being more than 25% of the number of rules in the list) and also, based on the fact that most access lists on the Web have a number of rules of 50 (being the average according to the survey) one would conclude that in fact they would have had rules classes of between 2 and 13. The value two is the minimum number of classes. The value 13 is approximately 25% of the average number of rules in the Web, 50. Most likely the number of classes is some number between 2 and 13; averaging around $(2 + 13)/2 = 6$ or 7 classes.

The survey (Gupta and McKeown, 1999) indicated two important points. First is that almost all combinations of fields used in rules are made up of one-field, three-field and four-field. The second is that the maximum number of fields that can be specified in a rule is limited to 8. This was due to syntax limitations in the specifications of the rules analysed. It is unlikely that more than 4 fields are used in many rules.

Considerations is given to the maximum number of combinations of one-field, three-field and four-field picked from any number of fields. Looking at Table 5.1, it is found that the maximum number of combinations when 7 fields are available to select from is 77. This represents a maximum value of the combinations, while the average value will most likely be a lot less.

To summarise, the following points are considered when determining the number of classes present in an access list:

(a) that the average number of classes in an access list on the Web is 6 or 7.

(b) that the maximum number of classes in an access list is to be no more than 0.25 of the total number of rules in the list.

n (number of fields available)	k(fields combined in a rule)								Maximum Combinations	
	1	2	3	4	5	6	7	8	Using all fields	1,3 or 4 fields
1	1	n/a							1	1
2	2	1	n/a						3	2
3	3	3	1	n/a					7	4
4	4	6	4	1	n/a				15	9
5	5	10	10	5	1	n/a			31	20
6	6	15	20	15	6	1	n/a		63	41
7	7	21	35	35	21	7	1	n/a	127	77
8	8	28	56	70	56	28	8	1	255	134

Table 5.1: Maximum number of combinations of fields.

(c) that too many classes in an access list defeats the purpose of classification.

(d) that too few classes in an access list (as in one class) eliminates advantages of classification.

The values for classes found to be adequate to perform the simulation meet the following conditions:

- The numbers of classes to be considered in any list are: 2, 4, 8, 16, 32, 64 and 128. These values are good samples to test and also cover the values shown in the "*Maximum Combinations for 1,3 and 4 fields*" column in Table 5.1.
- To include only those values for classes which are less than 0.25 of the total number of rules in a list.

3. **Number of rules in each class:**

It has already been said above that the total number of rules to be considered will vary between 20 and 2500, while the number of classes for each of those will vary between 2 and 128 (provided it does not exceed 0.25 of the number of rules within the list). The problem now is to determine how many of the rules in a list will be classified into each of the classes. For example, consider an access list of 100 rules to be distributed into 4 classes. The problem is to determine how many of these 100 rules will be classified into each of the 4 classes. It is suggested to perform the following repeatedly as many times as there are classes to calculate the number of rules for each class based on the results of the previous calculation. Table 5.2 shows the full steps of the example.

Starting with the first class, the minimum number of rules to have is one rule per class. Considering the example of 100 rules and 4 classes, placing 1 rule in each of 3 classes leaves one class with the remaining rules (97 rules). This defines the minimum number of rules as 1 and the maximum as 97. In other words, the maximum number of rules in any given class can be defined as follows:

$$\text{Max rules in a class} = r - (c - 1), \tag{5.1}$$

where r is the number of rules to distribute, c is the number of classes to consider.

Class no.	Rules to distribute r	Classes left c	Max rules assign r-(c-1)	Rules assigned
1	100	4	97	66
2	44	3	40	7
3	37	2	36	25
4	12	1	–	12

Table 5.2: Distribution of rules between classes.

In this case, the number of rules to be assigned to class 1 will be a random number between 1 and 97. According to the example in Table 5.2, this random number is 66. If out of 100 rules, 66 are assigned to class 1, then that leaves 44 rules to distribute between the other three classes. Therefore, class 2 can have a maximum value of 42. If the maximum is assigned for class two, that would still leave classes three and four with one rule each. But the number of rules assigned to class two can be a value generated randomly between 1 and 42 (Minimum and maximum). In the example in Table 5.2, this value is 7 rules. The steps are repeated until it is time to deal with the last class. In the case of the last class, the remaining rules left are assigned instead of using a random number. This entire operation can be repeated to generate other access lists distributions of 100 rules for 4 classes.

The actual number of rules for each class is generated randomly. The random number must be checked in such a way that it is not too large so that the remaining classes are left with no rules. Algorithm 5.1 shows how values were assigned to classes for any number of classes. The different numbers of the rules assigned to each class are placed in an array of a size equal to the number of classes, (class_array[number of classes]). Two parameters are needed, the total number of rules to distribute and the number of classes.

```
let rules_left = total rules to distribute
let c = total number of classes
let classes_left = c
let class_array[c] // define an array of size equal to number of classes
for all (i = 1toc) do
  if (i == c) then
    /* the last class to be assigned a number? assign all remaining rules.
    class_array[i] = rules_left;
  else
    {
    /* if there are as many classes as there are rules
    /* to distribute, then every class will have one rule
    if (rules_left == classes_left) then
      {
      class_array[i] = 1;
      rules_left --;
      classes_left --;
      }
    else
      {
      x = (rules_left − classes_left + 1) /* maximum number of rules in a class
      r = random(x) /* generate a random number equal to or less than x
      /* this ensures that at least one rule will be left for each class remaining
      class_array[i] = r;
      rules_left = rules_left − r;
      classes_left --;
      }
    end if
    }
  end if
end for
```

Algorithm 5.1: Distributing the number of rules into the classes

4. **Average processing time for rules in each class:**

   When a packet is being inspected by a rule, all its header fields are available for inspection. Each particular rule will perform some inspection on one or more of the fields. Depending on which field and on the number of fields, the processing time required for inspection will differ between one rule and another. Values assigned as the processing time for a rule will represent a relative time compared with the processing time for other rules. For example if the values 100 and 200 are assigned as processing times for rules A and B, then rule B will take twice as much processing time as that of rule A.

   Let us initially assume that it is possible to state that the processing time taken by a rule to inspect a packet depends on the fields that need to be inspected. Then it can also be stated that all rules inspecting the same field or fields on different packets will require exactly the same processing time. Rules have already been divided into classes based on the fields in packets they inspect. Therefore, rules within the same class are inspecting the same fields and will take the same processing time for all packets. This statement is not entirely true, and it will be seen why in a moment.

   The actual value assigned as the processing time for each class of rules is based on the number of bits in a packet that normally need to be inspected by a rule of this class. This is a very good estimation of the processing time suggested by Hazelhurst (1999). But, even if the estimation is not an accurate reflection of the actual processing time, it will still provide a good way to compare the performance of the same rules when organised differently, for rules of the same list inspecting the same stream of packets.

   For the purpose of this simulation, the application allows for a configuration file to be created, edited and saved which contains different average processing times for each class. This helps in reducing the effort required in the generation of access lists' rules having different average processing times. It also allows using other numbers not directly related to the bits in packets. Sometimes equal processing time for rules was required for simulations to make other comparisons.

   The statement made earlier that within the same class, the rules would need the same processing time to inspect all packets, may best be rephrased to, almost the same processing time. For example, consider a rule inspecting the source or target address field on an IP packet header. The inspection will require first inspecting the Mask part of the address to determine how many bits of the address need to be inspected. The following two mask

values: 0.255.255.255 and 0.0.0.255, will in the first case check one part of the address while in the second case it will check three parts of it. This leads to slightly different processing time when different packets are inspected, according to Hazelhurst (1999). It is clear that processing times will be different for rules of different classes. But, even for those rules in the same class inspecting the same fields, processing time will be slightly different from one packet to another. Therefore, the processing time for each classes rules will be based on a value for that particular class with a smaller random value added to allow for differences.

Processing time is assigned to each class of rules. For each particular rule within each class the same assigned value is used with a plus or minus randomly generated value. For example, suppose the processing time for class x is 100, with a possibility of variation of up to $\pm$ 25% between different rules in that class. That is indicating that the minimum and maximum values rules can assume are 75 and 125. The processing time for any rule in this class is calculated to be equal to be 75 + (a random value between 0 and 50).

5.5.3 Organisation of Access List Rules

It is obvious that the location of rules within the list will have an effect on the total processing time for each packet to pass through the list. The earlier the critical rule for a packet is executed to inspect the packet the shorter the processing time will be for that packet. There is only one case where the organisation or order of the rules within the list will have no effect on the processing time of a packet. That is the case where there is no rule within the list that will determine the fate of the packet. In such a case, the packet will have to traverse through the entire list of rules. No matter what order the rules are in, there will still not be a rule that will recognise the packet. Unfortunately, it can never be known for sure if there will or will not be a rule in the list which will be the critical rule for a particular packet until the packet goes through the list. The other unfortunate consequence of such packets is that the processing time for such packets is going to be the maximum of that for any packet. If these packets make up the majority of a packet stream, it is going to increase the average processing time per packet dramatically. The solution to such problems may come in two ways. The first is to have a better analysis of the packet stream flow and making a better prediction of future packet streams. The second is to add new classes of rules to include

(accept or refuse) such packets or most of those packets, even if such classes of rules have little or no security significance.

To help establish if the organisation of the classes of rules makes any significant difference on the performance, and more specifically on the average processing time per packet for a packet stream, it is necessary for each and the same packet stream to be tested many times using the same particular list of rules but in a different organisation each time. That is the same list is used once with no particular order or organisation. Rules belonging to one particular class are picked out and placed at the top of the list one at a time in new versions of the same list. Other versions should place all rules of the same class at the bottom of the list. Such organisations should not change the order of any rules belonging to the same class. During the initial research experiments considered organisations are such that the entire set of rules belonging to one class are moved to the top or bottom of the list without any change in the order of its rules or the rule order of other classes. For example, for a single 4-classes list of rules, 4 versions of the list will be created with classes A, B, C and D placed on top in each list respectively. Then 4 more versions will be created with classes A, B, C and D placed at the bottom of each list. That is a total of 8 versions of the same list; plus the original randomly organised list. In this example this indicates that for a c number of classes in a single access list, the minimum number of versions created for that access list is $2c + 1$. For an individual say 2500 rules access list, with 16 classes, there will be 33 versions created. To start with, there may be 10 or more different original randomly organised access lists, each having 16 classes of rules, with 33 versions made for each.

The application is implemented in such a way that a list of rules may be taken in as input, and output can be produced for that list but in any given sequence of arrangement of its classes. All access lists in all organisations produced must be saved for future reference and reuse.

5.5.4 Rules Definition Specifications

This refers to actual individual rules in the access list. Each rule used for the simulation is specified through three parameters. Every rule in the list is defined by the following:

1. **Rule class:**

Every rule is assigned to a class represented by a number between 1 and the maximum number of classes allowed for this access list. The parameter is randomly generated for each rule and assigned to it during the generation process. This rule class is based on two values, namely the maximum number of classes allowed for this access list and the number of rules of this class in relation with the total rules in the list. Both those values are part of the access list definition and are requested when the access list is being generated.

2. **Rule processing time:**

   This is a random number generated based on a mean processing time assigned to the class. Every class is assigned such a value during the creation of the access list. When rules are generated, every rule is assigned a class to which it belongs. Processing time for the rule is computed based on two values, the first is the average processing time for rules in that class, the second is the $\pm value$ attached with the average processing time. This $\pm value$ is what gives uniqueness to processing time for individual rules within the same class. Details of processing time computation were discussed under Item 4 (**Average processing time for rules in each class**) on page 89 in subsection 5.5.2.

3. **Rule decision to Accept or Deny:**

   Every rule that identifies a packet into some packet flow will do so by deciding to accept or deny the packet based on some conditional comparison of some fields. Normally, the decision to accept or deny is a part of the syntax of every rule in the list. During this simulation the decision to accept or deny is made to be an integral part of the rule. This parameter is randomly generated for each rule during the generation of the access list. When an access list is being created, a value is accepted as the percentage of rules that will be assigned the value "accept". Based on that value, some of the rules created in that particular access list are assigned an "accept" decision and the rest are assigned a "deny" decision.

   The values for the decisions "accept" and "deny" are represented by the 1 and 0 respectively. In fact, for the purpose of this simulation, these values to accept or deny will have no consequence on performance. Once, a packet is accepted or denied, processing is finished. This field is maintained because it formed part of the model verification experiments through monitoring the number of accepted and denied packets.

5.5.5 Access Lists File Specifications

The actual rules' definitions of a list are kept in a file as part of the access list definitions, which allows the simulation application to read these rules' details. The file's header provides details about the entire access list like the total number of rules and number of classes in the list. An access list specifications file consists of a header and a number of records. The header provides details about the entire access list like the total number of rules and the number of classes used etc. The rest of the records provide information about each rule in the list. The file structure as well as some example values are shown in Table 5.3, while the header and record have the following syntax:

1. The access list specifications file header:
   (a) Total number of rules in the access stream.
   (b) Number of classes used for rules.
   (c) Details of each class of rules: (This record is repeated a number of times as indicated by the field "*Number of classes used for rules*").
      i. Number of rules in this class.
      ii. Mean Processing Time of rules in this class.
      iii. Percentage of "Accept" rules in this class.
   (d) Order of classes (This field is repeated a number of times as indicated by the field "*Number of classes used for packets*").
2. The file records, one record per rule in the list:
   (a) Accept or Deny rule.
   (b) Rule class.
   (c) Processing time for this rule.

For every access list generated, a number of different organisations are generated that differ only in having the rules of different classes moved up towards the top or bottom of the list in relation with other classes. In other words, the access list is ordered based on the class of the rules, while the rules belonging to one class maintain their original order in the list. Two types of files are required in the access list model.

The first file is used to hold details of an access list and its rules. All those lists are saved individually in separate files with file names which are descriptive of

The field explanation	Example values			
The total number of rules in the list	100			
The number of classes in the list	4			
Details about every class in the list (These three fields are repeated for each class)	Class	Num of rules	Process. time	% of accept
. Number of rules in this class	1	24	120	99%
. Mean processing time for rules in this class	2	59	100	19%
. Percentage of "accept" decision in this class	3	16	80	87%
	4	1	70	55%
Class organisation:(0 to indicate randomness)	Class 2 rules on top of list			
. Class placed as first in list	2			
. Class placed as second in list	4			
. Class place as second-last in list	1			
. Class placed as last in list	3			
Specifications of each rule in the list: (these three fields are repeated for each rule)	Accept or deny	Rule class	Processing rule time	
. Accept or deny decision.	1	2	76	
. The class or this rule	1	1	111	
. Processing time.	0	2	118	
	1	3	63	

Table 5.3: File structure containing the randomly generated rules of an access list.

the access list contained within. For example: al_100_8_1 indicates that the file contains an (al) access list of a total of (100) rules, containing (8) different classes of rules. This is followed by a serial number to identify the different access lists with 100 rules and 8 classes.

When an access list is class organised, more files are generated. All the generated files contain the same list but in different organisations. Consequently, all files will share the file name as the original access list file after appending another serial number. This number will identify each individual organisation of the same access list. The syntax for the access list file will be as follows:

al_<*number of rules*>_<*number of classes*>_<*serial number of access list*>[_<*serial number of organisation*>]

Example of access list files names are:

```
al\_100\_8\_1000
al\_100\_8\_1000\_1
al\_100\_8\_1000\_2
al\_100\_8\_1000\_2\_2001
```

al_100_8_1000_2_2002
al_50_4_2000
al_50_4_2000_505
al_50_4_2000_506

The second file is the processing time configuration file for an access list. This file is used to save the mean processing time of each class in an access list. This file must be generated prior to the generation of the access list specification file described above. All or some of the 256 values contained in this file are used to calculate the processing time of each individual rule in the list based on the class of the rule.

5.6 The Packet Stream Model

The packet stream model is concerned with the generation of lists to represent the packets arriving into a network device. For the purpose of this simulation each list represents a stream of packets from a realistic environment. Each list contains the specification for a packet stream. The packet stream model involves the classification and generation of packet streams, and includes the following:

1. Packet classification.
2. Packet streams specifications. This is defined by a number of parameters, the values for which can be specified during the generation operation. The parameters that determine the specifications for a packet stream are:
   (a) Total number of packets.
   (b) Number of classes within the packet stream.
   (c) Number of packets in each class.
   (d) Mean time between arrivals.
3. Organisation of packets within a packet stream.
4. Packet definition specifications.

5.6.1 Packet Classification

In communication systems, classifiers classify packets into different packet flows. The classifiers are the access lists that determine what class a packet will belong to. No matter what size a packet stream is, or what types of packets it is made up of, it is still the access list rules which ultimately will decide how many classes the packets in a packet stream will be divided into. The same packet flow when filtered by two different access lists will result in two different classifications. But for the purpose of this simulation, it is rather important to know in advance the class of each and every packet in the packet stream. It is important to control the pattern of packets to enable the testing of different aspects and behaviour of access lists. The number of classes defined for packets is the same as that defined for access list rules based on observations of the survey outlined above. In fact, it is worth pointing out that to simplify matters, packet classes and access list rules classes will use a similar numbering to identify them. In other words, a packet identified by a class A access list rule will also be called a class A packet.

5.6.2 Stream Specifications

Every packet stream is defined and identified for the purpose of this simulation study by the following four definitions:

1. **Total number of packets:**

   The packet streams should contain as many packets as possible. The more packets in a packet stream the more comprehensive the simulation. On the other hand, the more packets in a stream, the longer the simulation time becomes, especially when tested against a very large access list. To allow for more comprehensive analysis of simulation results, it is best to have as many simulations performed as possible. Packet stream sizes of 1,000, 10000 and 1,000,000 packets are thought to be sufficient to reflect the performance of access lists because larger size packet streams increased simulation times to unreasonable limits with no further benefits.

2. **Number of classes in the packet stream:**

   The main purpose of this simulation analysis is to study the performance of different organisations of access lists on classifying packets. For that purpose many different packet streams must be tested. In fact the more varied the

packet streams tested, the more comprehensive the results. In as far as classifications of packets are concerned, all possible packet classifications in all possible combinations must be used. The maximum number of classes for access lists was determined as seen earlier to be 128. That will be the same for the packet streams. Packet streams containing 2, 4, 8, 16, 32, 64 and 128 packet classes must be generated. In fact a number of streams must be generated containing each of those number of classes.

3. **Number of packets in each class:**

   The different packet streams generated must have many different mixtures of packets within them. The different packet classes within each packet stream will each have a number of packets other than zero assigned to them. With the same number of packets, many packet streams with different numbers of classes must be generated. Also for streams with the same number of classes, many streams must be produced with different numbers of packets in each of these classes. Number of packets are supplied to the application during the generation of packet streams. The different packet streams to be generated are to reflect the different possible packet streams that can be received by a network device of a real live connection. More importantly, it is important to be able to identify a particular packet stream to be mostly of a particular class. For example, if a packet stream consists mostly of class C packets, then it is possible to perform specific tests by using access lists with for example, class C rules at the top or bottom of the list, or access lists with more or less class C rules. It is also important to be able to identify a stream by the number of packets in each of its classes.

4. **Mean time between arrivals:**

   This indicates the value from which each arriving packet time is based upon. This can be different from one network device to another. If the time between arrivals is a small value compared with the average processing time of a packet by the access list, this will allow more packets arrive than can be handled immediately. This means that very many packets will be queued on arrival to wait for their turn to be processed. This increases the overall time a packet spends in a classifier from the time it arrives until it leaves. Even though some waiting time may be acceptable and can be seen as normal for the operation of packet filters, excessive waiting time in the simulation can produce misleading results. Long average processing time produced in a simulation can be attributed to long waiting times due to a very short time

between arrival of packets, rather than to the performance of the access list itself.

To eliminate such problems, two solutions are available, the first is to ignore all waiting times and use pure processing time. This concept will ignore the fact that in real life, packets can actually arrive in close sequences and will have to wait in the device. The processing speed of the access list can make the time for the waiting packets short or long. This solution is not used. The second solution is that it is best to select a (relatively) large mean time between arrivals for packet streams. This value should be higher than the average time for processing a packet for a given access list, because the average processing time for a packet through an access list is difficult to predict due to the fact that it depends on the packet flow and differs from one flow to another. It is possible instead to make the mean time between arrivals for packets in a packet stream to be higher than that of the total processing time required by an access list. That is the processing time required for a packet from a class that does not exist in the list. Such a packet will have to be examined by every rule in the list. In our system, such a value is displayed as part of an access list definition when an access list is being viewed. It was necessary to use the mean time between arrivals to allow for those packets that arrive in real life at very short time intervals.

5.6.3 Organisation of a Packet Stream

Packets in most real communication systems do not arrive in any particular order. Packet streams are left in random order as they are randomly generated.

5.6.4 Packet Definition Specifications

The actual packets that are used for the simulation are specified through three parameters per packet. Every packet in a packet stream is defined by the following:

1. **Time since last arrival:**

   This is a random number generated based on the mean time between arrivals assigned as part of the definition of the packet stream. Every packet stream class is assigned such a value randomly during the creation of the packet

stream. The arrival time of the packet in the simulation is calculated from this value and the actual clock time of the simulation.

2. **Packet class:**

   Every packet is assigned a class represented by a number between one and the maximum number of classes allowed for this packet stream. This parameter is randomly generated for each packet during the generation of the packet stream and it is based on two values, namely, the number of packets in the packet stream and the number of packets in each class. Both of those two values are part of the packet stream definition and are both accepted when the packet stream is being generated. In the simulation, it is important to ensure that packets that get accepted are always the same packets when the same packet stream is filtered by any arrangement of the same access list. In other words, changing the arrangement of an access list will not change the output when filtering the same packet stream. This is of greater importance when investigating the effects of changing the order on the security implementation.

3. **Percentage of rules in the class to be processed:**

   This is a random number generated to indicate how many rules of the relevant class (represented as a percentage) need to be inspected before the critical rule for this packet is reached. This parameter ensures that the same packet will be identified by the same rule regardless of the organisation of an access list. The reason is that the order of rules within a particular class does not change. This value has no effect on whether the packet gets accepted or not, that is a function for the the packet class parameter and access list.

   This parameter (Percentage of rules in the class) for a packet mainly ensures that any changes in processing time of a packet by different copies of the same access list are due to the organisation alone. A value of 1% will ensure on all versions of the same access list, this packet will be processed at very early on when meeting rules similar to its class. A value of 100 % or over will indicate that no critical rule is ever found and ensures all the rules in the list are involved in processing this packet.

   The parameter serves two other purposes. First, it allows for situations of packets which are not recognised and traverse through the entire list of rules. For example, a packet with an unrecognised class. The second, is that for those packets that traverse the entire list of rules to be accepted (or refused)

at the end, it might be worth inserting a rule to actually accepts (or refuses) them earlier on in the list. From a security point of view, the same packets will still be accepted (or refused) with or without the rule. This makes the rule of no value from a security prospective, but from a performance point of view it can make a difference.

5.6.5 Packet Stream File Specifications

The actual packets' definitions for a packet stream are kept in a file as part of the packet stream definitions, which allow the simulation application to read these details. A packet stream specifications file consists of a header and a number of records. The header provides details about the entire packet stream like the total number of packets and the mean time between arrival of packets etc. The rest of the records provide information about each packet in the stream. The file structure as well as some example values are shown in Table 5.4, while the header and record have the following syntax:

1. The file header:
   (a) Total number of packets in the packet stream.
   (b) Mean Time Between Arrivals.
   (c) Number of classes used for packets.
   (d) Number of packets in each class ((This field will be repeated a number of times as that indicated by the previous field "*Number of classes used for packets*".
2. The file records, one record per packet in the file:
   (a) Time since last packet arrival.
   (b) Packet class.
   (c) Percentage of rules to be inspected in the relevant class of rules.

For every packet stream generated with a particular number of classes, a number of similar streams are generated that have the same number of classes but differ in the number of packets that make up the classes. Packets are generated randomly and are kept at that organisation. All packet streams are saved in files with file names, which are descriptive of the packet streams contained within. For example: pk_1000000_10000_8_1 indicates that the file contains a (pk) packet stream of a

total of (1000,000) packets, with a mean time between arrival of 10,000 time units, and there are (8) different classes of packets. This is followed by a serial number to identify the different packet streams with 1000,000 and 10,000 mean time between arrivals and 8 classes. The syntax for the packet stream file will be as follows:

pk_<*number of packets*>_<*Mean Time Between Arrivals*>_<*number of classes*>_<*serial number of packet strems*>

Example of packet stream files names are:

`pk_1000000_10000_8_555`
`pk_1000000_10000_8_556`
`pk_1000000_10000_8_557`
`pk_100000_20000_64_707`
`pk_100000_20000_64_708`

<table>
<tr><td>The field explanation</td><td colspan="3">Example values</td></tr>
<tr><td>Total number of packets in the stream</td><td colspan="3">10,000</td></tr>
<tr><td>Mean time between arrivals of packets</td><td colspan="3">9,000</td></tr>
<tr><td>Number of packet classes in the stream</td><td colspan="3">4</td></tr>
<tr><td rowspan="5">Details about every class in the packet stream
(This field is repeated for each class)

• Number of packets in each class</td><td colspan="2"></td><td>Number of packets</td></tr>
<tr><td colspan="2">Class 1 packets</td><td>2500</td></tr>
<tr><td colspan="2">Class 2 packets</td><td>1600</td></tr>
<tr><td colspan="2">Class 3 packets</td><td>4000</td></tr>
<tr><td colspan="2">Class 4 packets</td><td>1900</td></tr>
<tr><td rowspan="5">Specifications of each packet in the stream:
(The three fields are repeated for each packet)

• Time since last packet arrival (TSLPA)
• Packet class (1 to maximum classes in stream)
• Percent of rules in access list to be checked</td><td>TSLPA</td><td>Packet class</td><td>% rules to be checked</td></tr>
<tr><td>13,203</td><td>2</td><td>76 %</td></tr>
<tr><td>2,479</td><td>3</td><td>12 %</td></tr>
<tr><td>10,547</td><td>3</td><td>103 %</td></tr>
<tr><td>6,000</td><td>1</td><td>25 %</td></tr>
</table>

Table 5.4: File structure of packet stream definitions,and example data.

5.7 The Filtering and Reporting Model

The Filtering model is that part of the system which performs the actual simulation of the filtering operation. This part of the system is the mechanism of inspecting arriving packets against rules in a list. It must also be clear that in an

access list filtering system, this part of the system does not have an effect on the result if a packet is to be accepted or refused, even though, it is the module that does the actual filtering. The outcome will in depend on the inspected packet and the list of rules used to inspect the packet.

The reporting part of the model is concerned with capturing the information about every simulation run and the results produced. This model is configured so it can be used to capture a minimum amount of information in the form of a one line summary about each simulation run. It can also be configured to capture a maximum amount of details about a single simulation in the form of a line summary about each individual packet.

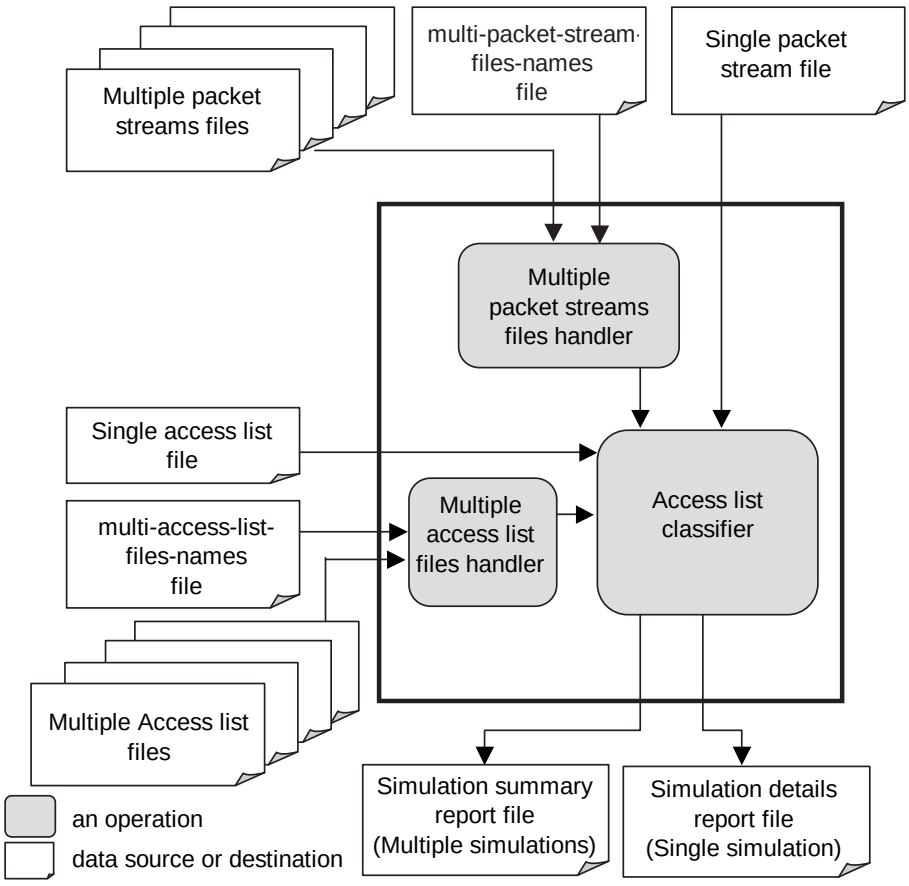

Figure 5.3: The Filtering and Reporting Model input and output files

The mechanism in filtering and reporting model will accept as input one or more stream of packet to be filtered and one or more access list to use as the classifier. The model will produce detail or summary information about the simulation. Figure 5.3 shows the filtering and reporting model with its input and output files. Two input files are required to perform any simulation, a file containing the packet

stream to be filtered, and a file containing the specific access list in its specific order to be used for filtering the packets in the incoming stream. The packet stream to be filtered can be freshly generated during the simulation itself, and in such a case a previously saved file will not be needed. An option is made available to save a freshly generated stream packet used in the simulation run for future reuse.

In the case of multiple simulations, the names of the access lists files can be placed in a single text file. Similarly, the names of all the packet stream files can be placed in a single text file. These two files are generated automatically when multiple simulation is selected.

Two output files are also produced. The first contains a summary of the simulation run. This contains the following information about each individual simulation of a single packet stream filtered by a single arrangement of an access list:

Simulation Summary file

Details about the access list:

- Serial number of the simulation operation.
- Number of rules in the access list.
- Number of classes in the access list.
- Number of rules in each class.
- Class order in the access list.

Details about the packet stream:

- Number of packets in the packet stream.
- Mean time between arrivals of packets in the stream.
- Number of classes in the packet stream.
- Number of packets in each class.
- Total service time of all packets in the stream by all rules in the list.
- Average processing time per packet.

The second output file is optional. It contains all the details of the processing of each packet in the stream. It contains the following information about each individual packet: The name of the file contains the names of both the packet stream and the access list files. ie: sim_al_20_2_2_pk_1000_600_2_707 This file will contain the following details report in text format: **Simulation details file:**

1. Report header:

    - File name of access list used.
    - Number of rules in the list.
    - Number of classes in the list.
    - Specifications of each class (number of rules, Mean processing time and percentage of the "Accept" rules).
    - Order of classes in the access list.
    - File name of packet stream used.
    - Number of packets in packet stream.
    - Mean time between arrival in packet stream.
    - Number of classes in packet stream.
    - Number of packets in each class.

2. Report main body:

    - For each packet arrival, the following information is kept:
        (a) serial number of arriving packet.
        (b) packet arrival time
        (c) time since last arrival
        (d) packet class
        (e) number of rules in the critical class to reach critical rule.
        (f) packet service time
        (g) packet departure time
        (h) accumulated service time for all packets

3. Report footer:

    - Total number of packets received.
    - Total number of packets served.

- Total service time.
- Maximum size of the Wait-Queue.
- Packets still remaining in the Wait-Queue.
- Packets still remaining in the Server.
- Total service time for all packets.
- Average service time per packet.

5.8 System Parameters

The following is a reduced list of the parameters found to be the minimum which will define the required model and help interpret its behaviour. Parameters included also allow the redefinition of the model characteristics for two purposes:

1. to control the behaviour of the model through selecting the required configuration dictating its affect on the operation,

2. to observe the behaviour of the model as a response to changes during operation.

The performance of the models' operation is measured by recording a number of the following parameters through a simulation:

1. **Maximum Packets**: indicates the number of packets arriving to the system. This controls the length of the the simulation run.

2. **Server Free**: the server is represented by the process that performs the actual inspection of packet against the access list rules. This variable will indicate "busy" if the process is actually busy inspecting a packet. Any arriving packets must enter the waiting queue.

3. **Time of Arrival**: for each packet indicates the time the packet has arrived in the system.

4. **Time to Next Arrival**: this is calculated at the time of arrival of every packet. It is used to decide at what time the next packet will arrive. This is a random number generated based upon the rate of arrival for packets into the system.

5. **Time in Queue**: if a packet can not be handled immediately upon arrival it is queued in a queue waiting to be served.

6. **Time in Service**: this is the actual processing time taken by the server to determine the fate of a packet. This represents the time taken for a packet to be checked against the rules in an access list. This is calculated for each packet depending on the packet class and some of the access list specifications. The access list specifications which can influence the time in service are the number of classes, whether or not the same packet class exists in the list and if so, its position within the list.

7. **Time of Departure**: the time a packet leaves the system after waiting in the queue, if so needed and after being checked through an access list.

8. **Total Packets Arrived**: represents the total number of packets that have arrived in the system.

9. **Total Packets Served**: represents the total number of packets that were served by the server. It may be less than the total packets arrived in the system, if by the end of the simulation there were still packets in the queue waiting to be served.

10. **Total Queuing Time**: total time spent in the queue waiting to be served by all packets that had to wait in the queue.

11. **Total Service Time**: time required for service by all packet that receive service. This will not include packets that are still remaining in the queue waiting to be served.

12. **Max waiting queue**: Indicates the maximum number allowed of packets waiting for to be served.

5.9 Model Conformance

In order to ensure that simulation provides meaningful results, it is vital to ensure that the model does behave like the real system for the purpose it was built. The model must be a correct representation of the real system in the filtering operations on incoming packets. A number of tests must be carried out to ensure the correct behavior of the model. The tests must include the separate testing of

each individual part or model of the system, as in the access list model, packet stream model and the packet filtering model in their individual operations. The tests must ensure that their cooperative operation is also correct. Conformance tests are done by executing the testing module and comparing the results it produces against a previously calculated and expected result. Some of the tests are performed directly with no previous calculations, for example, to check for a computation result of an expected single value, or testing the model by requesting it to generate one ten packets, and actually counting the number of packets produced.

For testing to be done and results to be verified, it is necessary to include other functions in the system for displaying and browsing in all modules. As an example the access list model provides facilities to display the actual details for an access list and all the values for all its rules, with counters and totals. This is important to verify that the access list model performed the requested operations generating a single or multiple access list or the rearrangement of an access list. The tests include the following:

1. **Access list generation:** Single and multiple access lists generation of predefined specifications. This ensures the access list model produces the correct access list based on some given specification. Those specifications include:

   (a) access lists containing different number of rules.

   (b) access lists with different number of classes.

   (c) access lists with different number of rules in the different classes.

   (d) access lists with the same processing time for all its classes and also lists with different values of processing times for each of the classes of rules.

2. **Rearrangement of an access list:** The model must be able to produce the same access list in any arrangement based on the classes of the rules within the list. Both operations, the single and multiple rearrangement of access list is verified.

3. **Access list processing time for each class:** For ease of use and efficiency of the model, the processing time of each of the classes of rules in multi-access list generation is taken from file rather than being entered manually. Many different values of processing time for different classes of rules must be specified in the file. The generated access lists must be checked for adherence.

4. **Packet stream generation:** Single and multiple packet stream generation of predefined specifications. This ensures the packet stream model produces the correct packet stream based on some given specification. Those specifications include:

   (a) Packet streams containing different number of packets.

   (b) Packet streams with different average time between arrival for packets.

   (c) Packet streams with different number of classes.

   (d) Packet streams with different number of packets in the different classes.

5. **Packet filtering:** simulation executions of single and multiple packet filtering operations. Different packet streams are tested by filtering through different access lists and different arrangements of the same access list.

5.10 Summary

This chapter discussed the design issues of the simulation model for the filtering system. It included the model for access lists and the packet streams. This design was based on the requirements specified in the previous chapter. The next chapter discusses the implementation of this design and the results obtained.

Chapter 6

Implementation

6.1 Introduction

This chapter presents the implementation of both the simulation model and the actual filtering. It describes how the simulation model of the filtering system was implemented and how it was validated. It then discusses how the simulation is used to implement the filtering operation of the different arrangements of access lists.

6.2 Implementation Environment

The implementation involved two separate stages: the first was implementing the simulation model of the system, and the second was using the model to implement the simulation of the packet filtering operation. The first step of the implementation was to build a simulation model of a packet filtering device. The simulation model must emulate the performance of a device connected to the internet, and be capable of receiving a flow of packets for the purpose of classifying them. The implementation builds the full model including the packet flow and the access lists used for filtering.

The purpose of the implementation is to build the model and perform simulation executions of the model in order to:

- determine the feasibility of the model development for the proposed system.
- enable the execution of test filtering operations for the purpose of model verification and if required modification.
- produce and allow manipulation of execution results in order to determine the validity of the model.
- use the developed model for the simulation executions using packet streams and access lists based on the models.
- produce and allow manipulation of simulation results in order to study and analyse such results for the purpose of making conclusions about performance of different arrangements of access lists.

The steps involved in developing, testing and verifying the model were based on the idea of continual verification during development. Each and every part of the model was tested and verified for correct operation while being developed and as its development was completed. Those parts included access list generation, rearrangement and packet stream generation; as well as the filtering part of the model. Predetermined input was supplied and the results were compared with precalculated results. Every part of the model was verified for conformance with the relevant part of the system for as much conformance as can be achieved on individual bases. Then, related parts of the model were tested to ensure correct cooperation and finally the full model was validated.

The steps involved in using the model for testing the effects of arranging the lists can be summarised as follows:

1. Generation of access lists based on the access list model, and the production of different copies of each of the lists in different ordering based on the classes of rules.
2. Generation and storage of a number of streams of packets based on the patterns according to the packet stream model. This will enable the same patterns to be used with different orders of the same access list.
3. Extensive simulation executions of a variety of access lists and packet streams. The results of simulation executions were analysed and conclusions drawn.

6.3 Model Creation and Validation

The initial part of the implementation focused on the creation and validation of the access list, packet stream and filtering models in the packet filtering simulation system. The model creation for the packet filtering system involved the logic of the simulation itself, and the inputs required (the packet stream and the access list). The algorithm used for the actual simulation is the Discrete Event Simulation (DES) standard algorithm (Banks and Carson, 1996) discussed on page 32 in section 3.3. In its simplest form, it involves setting a simulation system clock, which is incremented in every loop. Every time the clock is incremented, the system is checked to see whether a packet is due to arrive at that point in time, or whether a packet is due to depart the system. Those are the two major events that would cause the system to change its state; or they may cause other events.

The arrival time of the next packet is available in a file which contains details of all packets in a packet stream. In cases where packets are freshly generated, the time of the next arriving packet is available as it is generated when the previous packet arrived. Whether the packet stream is freshly generated or a previously saved packet stream is used, every time a packet arrives, the value for the arrival time of the next packet is freshly generated or taken from the saved packet stream.

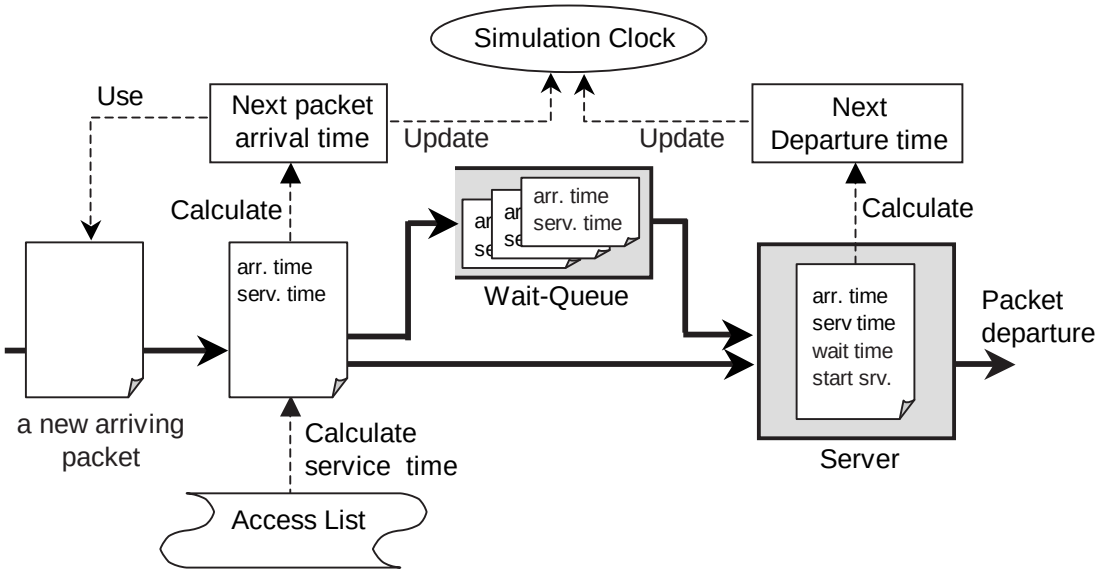

Figure 6.1: Discrete event logic for packet filtering simulation

If the server is free an arriving packet will be placed into the server to start the service. It will remain in the server a length of time equivalent to the time taken

for this particular packet to be classified by the access list in use at the time. If the server is busy serving another packet then an arriving packet is placed in a waiting queue, see Figure 6.1. The waiting queue is operated on a First-In-First-Out (FIFO) basis. Packets are removed from the queue and are placed in the server as the server becomes available. The length of time a packet remains in the server is based on the length of service time for that packet determined by the access list currently being used. The service time indicating the time a packet requires to be filtered by the access list is calculated once the packet has arrived, and is stamped on the packet, taking into consideration the packet class and the class arrangements in the access list. Once a packet is placed in the server to be served, its departure time becomes known (time of starting of service + length of service time for this packet). Therefore at any time during a simulation, two pieces of information are always available, namely: the arrival time of the next packet, and the departure time of the packet that is currently being served. In such a system, the system will remain idle until one of those two events occurs. In such a system, incrementing the system clock in steps until the time of one of those events occurs is somehow inefficient. The system clock is advanced in one operation to the lower value of either of those two events, that is provided the server is currently serving a packet, and provided that some packet is still expected to arrive, i.e. the end of the packet stream has not yet been reached.

6.4 Simulation Executions of Packet Filtering

Once the model's validity was established, then the other part of the implementation could start, i.e. performing the simulation executions to study the effect of rearrangement of access lists on their performance when filtering. To perform the simulation a number of access lists of varying sizes, number of classes and mixture of contents of classes were generated. For each of the lists a number of copies were regenerated in different orders based on the classes of rules.

A number of packet streams representing arriving packets were also generated, each stream resembling a different pattern of packets arriving into a network device. The patterns were characterised by the different numbers of packets in each packet stream, mean time between arrivals, number of classes and number of packets in each class depending on the packet classification. In this sense a packet classification reflects the class of rule that will decide the fate of the packet (accept or reject).

For the same access list, all copies of its individually rearranged lists were used in simulation executions to filter each of the packet patterns. In other words, for a large number of possible patterns of packet streams, all available different orders of the same access list were used for a simulated execution. The main factor deciding the performance in this case was the length of time taken to process all packets in a stream, and therefore, the average processing time for each packet.

All other variables and conditions that may influence the results were kept unchanged in all simulations where appropriate. For the same packet pattern the variables are the total number of packets, the number of classes in a packet stream, the number of packets in each class and the frequency of packets arriving (time between arrivals). While for access lists the total number of rules, the number of classes, the number of rules in each class, also the processing time for rules of similar class and for rules in different classes.

A simulation unit of time was used because the performance comparison is made by checking average processing time per packet. This reduces complications of differing real life data observed from different sources due to different performance speeds.

For the purpose of all our simulations, two main inputs are needed, a list of access list rules and a list of arriving packets. The rules list, referred to as an access list contains rules that are first classified according to what inspection the rule performs. This list of rules is ordered in such a way that all rules of a particular class are placed at the top of the list. On the other hand, the list of packets representing a particular pattern based on past real live Internet analysis is referred to as a packet stream. Packets can also be classified by classifiers based on one or more of the values in the fields in the packet headers. A few different patterns are needed, and therefore, a few different lists of packets are generated which only differ in the number of packets in each of the classes.

Table 6.1 shows the results of six simulation executions performed. Two packet streams were generated both having 1,000 packets with 4 classes in each (A, B, C and D) and an average time between arrivals of 3,000 units. One packet stream is mostly a class A stream containing 80% class A packets. The second is mostly a class D stream containing 80% class D packets. The other 20% of the packets contained all other classes of packets that require other classes of rules, the percentages for the remaining classes in both packet streams were 5%, 7% and 8%. The files specifications for the packet streams were:

pk_1000_3000_4_001 where 80% of the packets are class A packets.
pk_1000_3000_4_002 where 80% of the packets are class D packets.

An access list was generated which contained 200 rules in 4 classes. Class A, B, C and D dividing the rules into 60, 40, 40 and 60 consecutively. Five copies of the list were made available. one where all the rules were randomly organised. Rules of class A were placed at the top in the second list; and at the bottom in the third list. Rules of class D were placed at the top in the second list; and at the bottom in the third list. The files specifications for the access lists were:

al_200_4_001 with randomly arranged rules
al_200_4_100_1001 arranged with class A rules on top
al_200_4_100_1002 arranged with class A rules at end
al_200_4_100_1003 arranged with class D rules on top
al_200_4_100_1004 arranged with class D rules at end

It is expected that the access list with class A on top would give better performance than the same list with class A at the bottom when filtering a mostly class A packet stream. Similarly, better results are expected when class D is on top of the list during filtering a mostly class D packet stream.

Access list (200 rules)	80% class A packet pattern (mostly Class A)	80% class D Packet Pattern (mostly Class D)
	Average processing time per packet	
Randomly arranged rules	2633	2465
Arranged with class A on top	1054	
Arranged with class A at bottom	4338	
Arranged with Class D on top		1461
Arranged with Class D at bottom		3881

Table 6.1: Effects of class position in an access list.

Similar results in line with these results in Table 6.1 were observed on many other executions with varying other values such as the number of packets, other classifications of packets on one hand, and the varying numbers and types of access list rules on the other.

In Table 6.1, the two best performances are represented by the lowest average processing time per packet in each column. The better performances were obtained when the class of rules in the access list placed on top of the list matched that of the class which is predominant in the packet stream. The worst situation is

where the class placed at the bottom of the access list is the class that matches the predominant class of packets in the stream. The unsorted access list performance falls between the other two best and worst order scenarios. The unsorted list performance is near the average performance of the other two arrangements of the list. In fact, in some of the cases it is actually slightly worse than the average of all the different orders used.

Extreme domination of a single class packet in a packet stream is not the norm. This simulation was used to show the idea that better performance is expected to be achieved by placing on top of the list the class that matches the dominant class in the packet stream. But, in most cases, the dominance of the class in the packet stream is not as high. Examples are where three or more classes dominate the packet stream at round 30% each. Also, the cost of each of the rules within a class can fluctuate to the point where one class can have more effect on performance than another class with a higher number of rules but of less computing cost. In fact, more simulation experiments showed different behaviour that was explained by considering a class by other than just the number of rules it contains. The term *rule-weight* was used to better reflect the effect of rules in a class taking in consideration their number and computing cost on a single packet.

6.5 Packet-Rules Cost Weight Method (PRCW)

Consider a packet arriving into an access list. It is not entirely true to say that the total processing time for such a packet is dependent on the number of rules alone. The processing time of each rule must be taken into consideration for a more accurate calculation. An arriving packet getting filtered by 5 rules, which require 1 microsecond each, will have the same processing time if it was being filtered by 10 rules requiring 0.5 microsecond each. Therefore, it can be stated that the processing time of a packet depends on the number of rules as well as the processing time of each rule.

Consider a packet passing through a list of rules. If there were m rules of one class in the list with a processing time (or an average processing time) r for each rule, then the total cost of processing C, for a single packet passing through the rules of a particular class in the list would be calculated as: $C = m \times r$. If the packet has to pass through more rules belonging to different classes s then the

total cost can be expressed as:

$$C_{total} = \sum_{s=1}^{n} m_s \times r_s,$$

where n is the number of classes.

But, for the class that contains the critical rule for the packet, processing is different. That is where the rules' class is the same as the packet's class. The critical rule may be located as the first rule in the class, last or somewhere in the middle. Let us assume that on average the critical rule will be in the middle of the class's rules. This means that on average processing time for the class, which will accept or refuse the packet, will be presented as: $C = 0.5mr$.

Let us consider an example of a packet stream and an access list both having only two classes A and B. The packet stream consists of only 10 packets dominated by class A. It contained 6 packets of class A and 4 packets of class B. For the access list, the number of rules and processing time of each has been taken into consideration to produce a total cost ($m_s \times r_s$) for each of the classes A and B.

The left-hand side of Figure 6.2 shows the list with the cost for classes A and B as 100 and 500 units of time respectively, while on the right-hand side of the figure, the cost for classes A and B is reversed to 500 and 100 respectively. In the case of the access list, the cost 100 units of time can refer to 20 rules with an average processing time of 5 units each; 50 rules with an average processing time of 2 units each; or 100 rules with an average processing time of 1 unit each (As long as it takes a single packet of 100 units of time to pass through). To simplify this example, consider the 100 and 500 in the case of the access list to refer to the number of rules each with a processing time of 1 unit.

Based on what has previously been seen, to obtain the better performance it would be recommended to place rules of class A on top of the list because there are more class A packets in the packet stream. With class A rules on top of the list, remember that class A packets passing through the list will only go through class A rules and will not get to class B rules, while class B packets will be processed by class A rules followed by class B rules. In other words, every packet will continue processing until it is processed by a rules of its class. The example on the left–hand side; shows that the total cost is less if class A rules are placed on top of the list (C=1700). While on the right–hand side of Figure 6.2, the total

processing cost is less when class A rules are placed at the bottom of the list (C=2300).

It has been established that the amount of processing did not depend on the number of rules alone. It has also been established that the amount of processing did not depend on the costing weight of the rules alone. The number and class of packets need also to be considered in any particular situation. The term Packet-Rule Cost Weight became the factor for determining the cost in a particular packet stream filtered by a particular permutation of an access list.

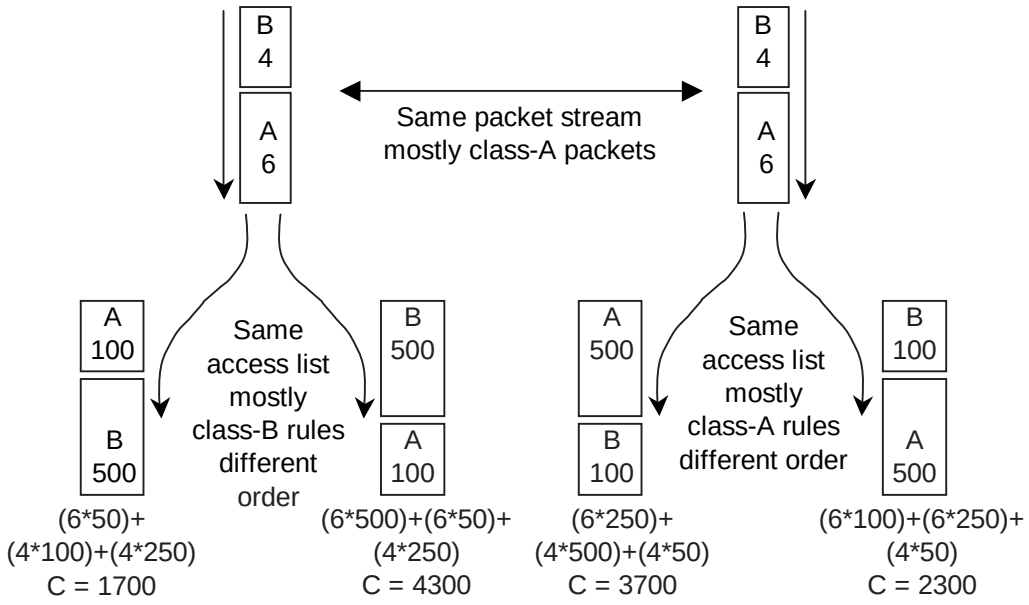

Figure 6.2: Calculating processing cost for different organisations or different lists.

6.5.1 Total Cost Calculation

It is possible to calculate the total cost for all the different possibilities of rules' classes combinations of an access list for a given packet stream. Combinations of the rules classes in this sense are best referred to as the permutations of the access list classes. Consider a packet stream made up of 3 classes of packets (p_1, p_2 and p_3) and an access list of 3 classes of rules with a calculated cost of (r_1, r_2 and r_3). Notice that r_x now refers to the cost for all the rules in a class rather than the cost of a single rule. The maximum number of permutations for the 3 access list classes would be $3! = 6$ different possibilities. The best permutation for our purpose; is the one that produces the lowest processing time (or least cost) for a given packet stream. The cost for each permutation is the sum of the cost

for every class of packets passing through the list, as was seen in Figure 6.2. The cost can be calculated as follows with reference to Figure 6.3 and the following formulae. Remember a p_1 class packet is processed through the rules and will stop at rule class r_1. Similarly, a p_2 packet will stop processing at r_2 and a p_3 will stop at r_3. Also, $C(r_1, r_2, r_3)$ refers to the cost when the access list rules classes are arranged with r_1 class on top, followed by r_2 and then r_3.

$$C_{(r_1,r_2,r_3)} = p_1\frac{1}{2}r_1 + p_2r_1 + p_2\frac{1}{2}r_2 + p_3r_1 + p_3r_2 + p_3\frac{1}{2}r_3$$
$$C_{(r_1,r_3,r_2)} = p_1\frac{1}{2}r_1 + p_2r_1 + p_2r_3 + p_2\frac{1}{2}r_2 + p_3r_1 + p_3\frac{1}{2}r_3$$
$$C_{(r_2,r_1,r_3)} = p_1r_2 + p_1\frac{1}{2}r_1 + p_2\frac{1}{2}r_2 + p_3r_2 + p_3r_1 + p_3\frac{1}{2}r_3$$
$$C_{(r_2,r_3,r_1)} = p_1r_2 + p_1r_3 + p_1\frac{1}{2}r_1 + p_2\frac{1}{2}r_2 + p_3r_2 + p_3\frac{1}{2}r_3$$
$$C_{(r_3,r_1,r_2)} = p_1r_3 + P_1\frac{1}{2}r_1 + p_2r_3 + p_2r_1 + p_2\frac{1}{2}r_2 + p_3\frac{1}{2}r_3$$
$$C_{(r_3,r_2,r_1)} = p_1r_3 + p_1r_2 + p_1\frac{1}{2}r_1 + p_2r_3 + p_2\frac{1}{2}r_2 + p_3\frac{1}{2}r_3$$

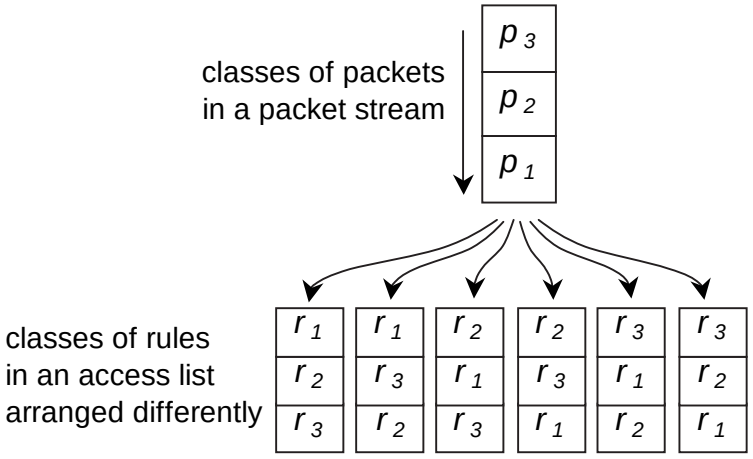

Figure 6.3: Calculation of cost for all different permutations.

The above formulae can be simplified to extract a general format that calculates the cost for any single permutation by subtracting the cost values appearing in all of them: $p_1\frac{1}{2}r_1$, $p_2\frac{1}{2}r_2$ *and* $p_3\frac{1}{2}r_3$. As the same value will be taken from every formula, it will have no effect on the final comparison to find the formula with the smallest value. Now, the general formula to calculate the cost for any number of classes of rules and packet streams can be expressed as follows:

The time cost for a permutation t of rules classes $= \sum_{j=1}^{n-1} [r_{t_j} . \sum_{z=j+1}^{n} p_z]$

Where:

t	is the permutation used i.e. (r_2, r_1, r_3)
n	is the number of classes
r_{t_j}	is the cost of r rule class in position j in the permutation t
p_z	is the number of packets of class z

The PRCW algorithm for calculating the cost of each permutation of an access list classes for a given packet stream may be expressed in two parts. The first part produces all the different permutations possible for the access list classes. This is done by recursively shifting the numbers that make up the classes. For example from the 4 classes: 1, 2, 3 and 4; the permutations in the left column in Figure 6.4 are extracted. From each one of those permutations, a further 3 permutations are extracted (middle column) by shifting only the right 3 elements. From each of those, two more permutations are extracted (right column) by shifting only the right 2 elements. A total of 24 different permutations are extracted from 4 numbers. In other words, the first part of the algorithm can be described by the recursive function shown in Algorithm 6.1 on page 121.

The second part of the algorithm actually calculates the total cost of processing the packets through the access list using each of those permutations. For this purpose two more lists of information are needed, the first is the number of packets in each class in the tested packet stream. The second is the total processing cost of rules for each class in the list. This processing cost is a simplification of two values namely the number of rules in the class multiplied by the average processing cost of the rules in that class.

Algorithm 6.2 shows the algorithm for calculating the processing cost for packets filtered by one permutation of an access list. The algorithm uses the arrays: pkt[ ] to hold the number of packets of each class in the packet stream, al_class_cost[ ] to hold the processing cost of each class of rules and permutation_array[ ] to hold the classes' permutation of the access list for which cost is being calculated.

This algorithm works well in producing the permutation of classes in an access list which yields the lowest processing cost for a given packet stream. In fact, this algorithm computes the processing cost for each and every permutation of a given access list classes for a given packet stream.

Figure 6.5 shows the results of simulations for a packet stream containing 10000

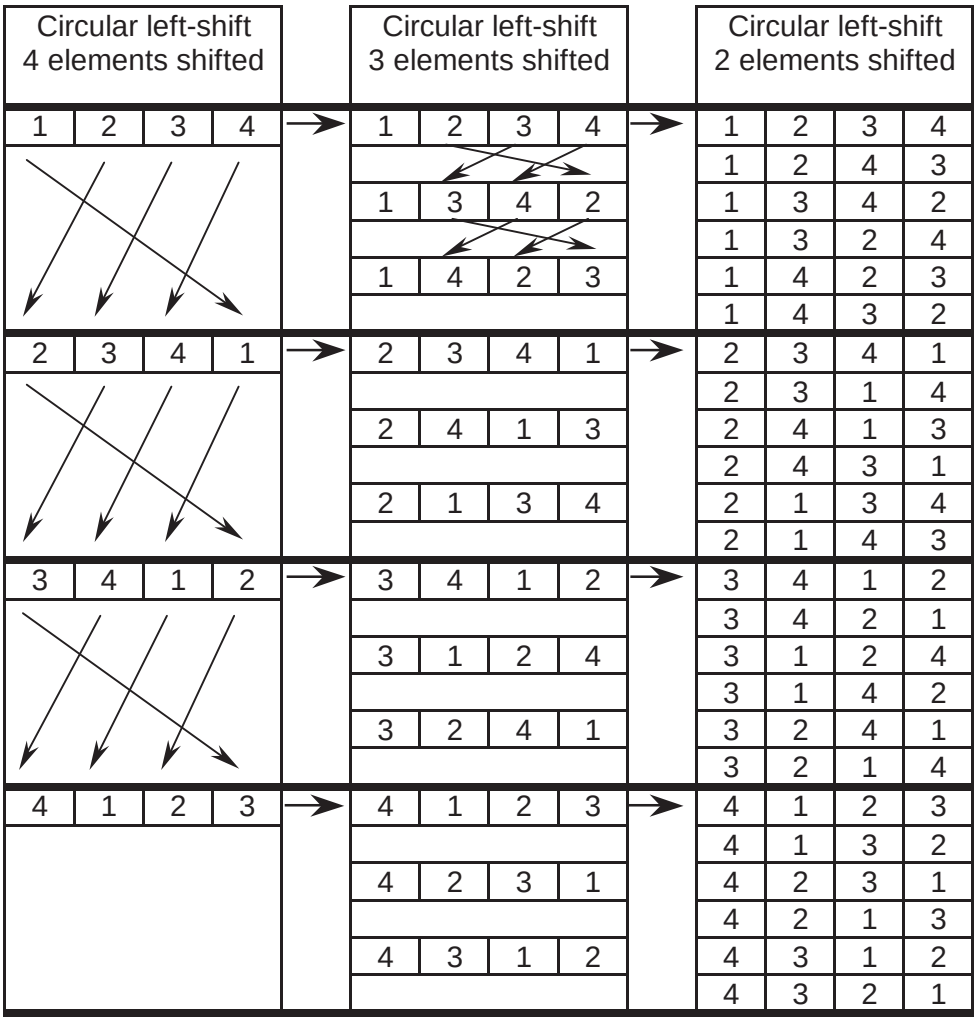

Figure 6.4: Extracting all possible permutations for numbers 1 to 4.

packets with 8 classes, class A making up most of the packets (91%), class B makes 3%, the rest of the 6 classes make up 1% of the packets each. The access list has 500 rules, classified into 8 classes (class A to H) having the following percentages and in the same order: 23%, 48%, 12%, 10%, 3%, 2%, 1% and 1%.

Figure 6.5 shows the performance curve of the access list with the described packet stream in many different organisations of the access list classes. Execution run number 10 (on the x-axis) represents the organisation of the classes when class A rules are at the top of the list, with the rest of the classes in the list in random order. Because the packet stream is mostly a class A packet, the performance is very good.

The second last point (run 21) shows the performance when the all rules classes in the access list are organised based on the numbers of packet's classes in the packet

```
Assume initially:
n = number_of_classes    /* is number of classes to get permutations for.
permutation_array    /* is the array holding first permutation, i.e. [1,2,3,4]

Func permutation(permutation_array, n, number_of_classes)
{
if (n > 1) then
  {                  /* circular right-shift numbers in the array in indices < n
  circular_shift_array(array, n)
  Call  permutation(permutation_array, n − 1, number_of_classes);
  }
else
  {
  return(permutation_array);   /* this is a permutation
  }
end if
}
```

Algorithm 6.1: The recursive function for generating the permutations.

stream. The last point on the curve represent the best performance obtained by the organisation based on the calculation for the lowest processing time using the processing cost weight method described above.

6.6 The High-Cost Elimination (HCE)

The Packet-Rule Cost Weighing (PRCW) method was proposed and investigated early on during the research work. It was implemented and was successful in producing the best permutation. The PRCW method is particularly helpful for automating the solution when considering that the number of possible permutations of the classes in the access list can be large.

The major problem with the PRCW algorithm described above is the amount of processing time required. This is because computation is performed to calculate the cost for every possible permutation. A 30-class access list will have over 2×10^{32} different permutations. A 16-class access list can have more than 2×10^{13} permutations. Each permutation contains 136 calculations (the sum of all the numbers from 16 to 1). An access list with n-classes has $n!$ different permutations; each with a number of actual calculations equal to the sum of the numbers from

```
// 3 arrays are assumed:
//  permutation_array[ ]  holding class numbers in the permutation to test
//  pkt[ ]                 numbers of packets in each class in a packet stream
//  sl_class_cost[ ]       holds cost of rules for each class in an access list
let minimum_cost = max_double;                  /* largest possible value
let cost =0
for (pk_index = 1 to number_of_classes) do
          /* repeat this for each packet class indicated by pk_index in pkt[ ] array
  al_index = 1    /* start with first class in the given access list permutation
  repeat
         /* accumulated cost of packet passing through classes of an access list
    actual_index = permutation_array[al_index]    /* take the class in list
    if (pk_index == permutation_array[al_index]) then
                    /* same packet class as access list class, half processing cost
      cost = cost + (pkt[pk_index]) * al_class_cost[actual_index]/2)
    else
                                   /* different classes, full cost
      cost = cost + (pkt[pk_index] * al_class_cost[actual_index])
    end if
    increment  al_index;
  until  (pk_index == permutation_array[al_index])
         /* stop when processed access list class is the same as the packet class
  if (cost <= mimimum_cost) then
    {     print("found a lower cost : ", cost)
          minimum_cost = cost;
          print(permutation_array)      }
  end if
end for
```

Algorithm 6.2: Calculating the processing cost for one permutation.

1 to n.

Some approach must be found to reduce this enormous size of computation. The PRCW algorithm is successful in getting the best permutation with the least processing time, provided this processing is done off line and in advance for preparing the permutation. The PRCW algorithm will not be successful for use in a dynamic situation to calculate the best permutation due to changes in characteristics of continually flowing packet stream. That is especially true in the case of access lists with large number of classes. A number of algorithms were suggested, implemented and tested. The following is a brief description of these algorithms:

1. **Minimum Processing to Next Class:** This is based on the idea of selecting the access list class that filters the packets at the minimum processing cost at each stage. The cost for all different classes of packets is calculated as if there was only one class in the access list. This is done for each class in the access list and the class with the least processing time is selected to be on

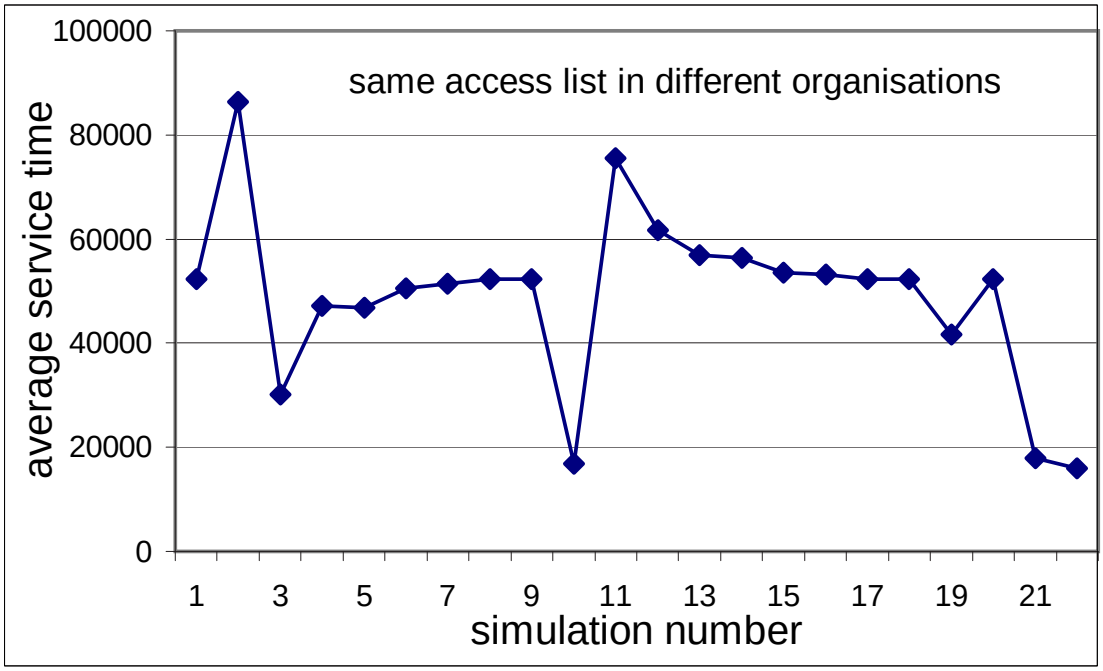

Figure 6.5: Different organisations of an access list with same packet stream.

top of the list. Packets in the packet stream and the access list rules for the selected class are ignored in the next round of filtering. The class with the minimum processing time is selected each time and dropped from the next round until a single class is left. This algorithm performed well in a lot of cases yet it did not produce the permutation with the minimum processing time in other cases. This is a typical minimization problem, which suffered from local minima, that is when the minimum value at a local point may not be on the overall minimum route line. Solving the minimum cost for all access lists and packets streams using this algorithm was not possible.

2. **Minimum remaining processing algorithm:** This looked at the filtering operation in reverse, and starting to find out the last class in the filter. The idea is that the worst case may be viewed when a large number of packets pass through all the classes producing heavy cost. Those packets that have not been filtered yet will reach as far as the last class and need to be filtered. Logically, a good algorithm will have fewer packets reaching this class and will have the least amount of useless processing done. It would also mean that all other packets have already been filtered by their corresponding access list classes. Only packets belonging to the last class will reach the last class. Based on this, the question remained was, which class of packets is preferred to have as last? The answer was; naturally: the

one that would give the least amount of useless processing.

This algorithm starts by calculating the processing required for each class of packets (knowing the number of packets) by the cost of rules for that class. The processing cost is calculated for each class separately. The class with the least amount of processing cost (packets and rules) is taken to be the last class.

This class is eliminated from the list of classes and the operation is repeated for the remaining classes to find out the last class in the remaining set. When only one class is left, the order of sequence of classes it identified in reverse. This method gave good results in many classes; but failed at further tests and was not pursued (See results in the figures in appendix A).

3. **The High-Cost Elimination (HCE):** This algorithm is a new approach and is based on the elimination of unnecessary processing of packet stream classes with large numbers of packets by the access list classes with high cost rules.

   Consider having a packet stream with two classes of packets and an access list with two classes of rules. Assuming the larger number of packets belong to class B, and that the highest cost class of rules in the access list are of class A. This means that of all the processing of the different classes, the highest processing cost is done when packets of class B are processed by rules of class A. It is also a fact that this processing is an extra cost of no value to the filtering process; it is already known that class B packets are only filtered by class B rules. Ideally, in this case it is not desirable for class B packets to reach class A rules in the list. Therefore, class B rules must be located before class A rules in the access list.

   Where there are more classes of packets and access list rules, similar comparisons in cost between class A and each of the other classes are made. That determines position of class A relative to each of the other classes. The following algorithm implements the High-Cost Elimination (HCE):

   (a) The classes in the packet stream are arranged in a descending order based on the number of packets in each class.

   (b) The access list classes are arranged in a descending order based on the cost weight of the rules in each class.

   (c) Each of the packet stream classes is checked by calculating the cost of filtering with each of the classes in the access list. Two cost calculations

are made for a given packet class number and a rule class number. In other words, cost is calculated when packets of class A are filtered by rules of class B. Then they are swapped so that packets of class B are filtered by rules of class A and the cost is calculated.

(d) The number of times are noted when a higher cost value is calculated for a class when it is used as a packet stream class than when it is an access list class. The number of times will indicate how high up the class will move in the desired access list class permutation.

To describe the High-Cost Elimination (HCE) algorithm using an example, assume a 4-class packet stream containing the following number of packets: 10, 30, 40 and 10 for classes A, B, C and D. Also assume a 4-class access list containing the following rules' cost values: 8, 2, 4 and 10 for classes A, B, C and D consecutively. Table 6.2 shows the packet stream classes ordered based on the number of packets in each class; and also shows the access list classes ordered based on the costing of rules in each class. This ordering is helpful in identifying the highest cost class combination of packets and access list. But it is not a necessary step for the rest of the algorithm implementation.

Packet stream			Access list	
Packet classes	Number of packets		Access list classes	number of rules
C	40		D	10
B	30		A	8
A	10		C	4
D	10		B	2

Table 6.2: The ordered packet stream and access list classes.

Taking the packet stream classes one at a time and calculating the cost against each of the access list classes, the left part of Table 6.3, shows the packet stream classes and the cost calculations with each of the remaining classes of the access list. In the right side of the table, the roles of packets and access list classes are reversed. For example, class C packets filtered by class D rules become class D packets filtered by class C rules, and the cost is calculated. When the two cost columns in the Table 6.3 are compared; the higher values of the cost in the left side of the table are noted and marked in the column [*position in list*] with the (*) sign. The number of (*) signs for a given class indicates the index of its place in the list. The higher the index number is; the higher the position of the relevant class in the list. A value of 0 indicates that the class is placed at the bottom of the access list. The position refers to the class shown in the [*Rules class*] column on

the right side of the table. In the example in Table 6.3, the permutation obtained for the access list class is: classes *B, C, A, D*. Class *B* being on top of the list and class *D* at the bottom.

Packet class into access list class				Reversed roles classes				
Packet class	Rules class	Cost calculation	cost	Position in list	Rules class	Packet class	Cost calculation	cost
C	D	40 x 10	400	*	C	D	04 x 10	040
C	A	40 x 08	320	* 2	C	A	04 x 10	040
C	B	40 x 02	080		C	B	04 x 30	120
B	D	30 x 10	300	*	B	D	02 x 10	020
B	A	30 x 08	240	* 3	B	A	02 x 10	020
B	C	30 x 04	120	*	B	C	02 x 40	080
A	D	10 x 10	100	*	A	D	08 x 10	080
A	C	10 x 04	040	1	A	C	08 x 40	320
A	B	10 x 02	020		A	B	08 x 30	240
D	A	10 x 08	080		D	A	10 x 10	100
D	C	10 x 04	040	0	D	C	10 x 40	400
D	B	10 x 02	020		D	B	10 x 30	300

Table 6.3: Cost calculations and defining the position in the access list order.

When performing the cost computations, no computation needs to be done if the packet class and rules class are the same. On the other hand, classes of rules in an access list for which there are no matching classes of packets in the packet stream shall be ignored, for they will be placed at the bottom of the list. They are an extra overhead on the access list, but will have no effect on the arriving packets with one exception. The exception is those classes of packets, which do not have matching classes in the list. Such class of packets will be ignored because their cost is going to be constant for the same access list regardless of its class permutation. A packet of such a class will simply pass through each and every rule in the list all the time.

One more reduction is for the computations of processing time for any packet class to be filtered by the same class rules in the list. This would be the same value for all different permutations of the access list when filtering the same packet stream. Eliminating that from all permutations will not affect the final output. This High-Cost Elimination (HCE) algorithm produced the best permutation with the minimum processing cost in all tested cases.

6.7 Summary

This chapter described the model creation and validation implementation. The simulation implementation was also discussed in detail showing the environment, the problems and solutions. The chapter outlined the progress in performance as it occurred during research work. The idea, development and implementation of both algorithms the Packet-Rules Cost Weight (PRCW) and the High-Cost Elimination (HCE) were described in detail. The The next chapter looks at the evaluation of the simulation results. Conclusions are drawn as to the effects of rearranged access list classes on performance.

Chapter 7

Evaluation

7.1 Introduction

The previous chapter outlined the implementation of the packet filtering model and its simulation. A discussion was presented of how the development progressed through a number of stages.

This chapter presents the evaluation of the research work from the performance aspect of access lists. This is based on the results of simulation executions using the model. This chapter outlines and analyses the results obtained. The performance evaluation of different arrangements of access lists is analysed in detail.

7.2 Model Validation and Verification

More time was spent on the model design, development and verification than on all other simulation experiments put together. It was absolutely vital to ensure that the model was a correct reflection of a real network device performing access list based packet filtering. The verification involved all the operations of the model; including all access list operations, packet stream operations and the filtering operations.

The access list operations included the creation, modification, reordering and display of access lists. It included the verification of many possible numbers

of classes, number of rules, processing time and distributions of rules into the different classes. It included the saving of such access list specifications and the reading and use of such lists.

The packet stream operations included the creation, modification display, saving, reading and using in filtering operations or other wise. Also included in verification were the numbers of packets in a stream, classification and the distribution of packets into the different classes. Other validation was for the value indicating the position of the critical rule for each packet in the relevant class in an access list.

The verification of the correct operation of the simulation operation of the model included the testing of a very large number of access lists using many packet streams. Step-by-step verification was needed many times to verify some of the results obtained. The simulation was verified for correct input and interpretations of access lists and packet stream specifications, the logic of the simulation for correctness then the relevant and correct results produced. Detailed reports of simulation operations were developed specifically to show step-by-step progress of a simulation per packet.

It was necessary to ensure the model was verified for correct operation and to ensure it produced valid results. Once, this was achieved, then it was possible to start simulations for actual performance measurement and analysis of access list.

The following are some of the tests carried out for the purpose of validation and verification of the different models:

1. **The Access List Rules Model:**

   (a) The processing time for rules in a given class is a random number based on a mean processing value for each individual class in the access list. It is one of the requirements that all classes can as an option have the same mean processing time. The mean processing time for all classes is created and kept in a file. Many files can be created any of which can be used during the creation of a new access list. To ensure the model produces the correct mean processing times, a number of files are generated. Specifically, the files had different mean processing time values. For some files the selection was for equal mean processing times, while for others the selection was for different processing times. When files were displayed, all files showed correct results.

(b) The creation and re-ordering of an access list was tested to ensure that the access list is created and the correct details is generated as well as the correct arrangement is obtained. Thousands of access lists were created in different sizes, specifications and arrangements. An example is the creation of an access list with 20 rules and 4 classes. The four classes (1, 2, 3 and 4) had each: 8, 1, 4 and 7 rules, and a mean processing time of: 20, 35, 40 and 12 time units and finally, a percentage of rules to have the value "Accept" represented by "1" as follows: 50%, 100%, 75% and 0%. The access list is created and saved and re-arranged using all possible permutations available. It was possible to view an access list at every stage. In this example this following report was produced when a request to display the final version of the list after the reordering (The 9th out of 24 permutations):

```
Number of rules are:  20
Number of Classes are:  4

class    Number     mean time of   accept
 no      of rules    processing     rate
-------------------------------------------------
 1          8            20           50 %
 2          1            35           100 %
 3          4            40           75 %
 4          7            12           0 %

The Rule's classes are Organised as follows:
 3 ,    1 ,    4 ,    2 ,

The exact number of rules in each class are as follows:
Class No: 1 :  8  Rules.
Class No: 2 :  1  Rules.
Class No: 3 :  4  Rules.
Class No: 4 :  7  Rules.
With a total processing time of          400

S/no   accept/deny     class   processing time

  1        1           3       38
  2        1           3       49
  3        0           3       31
  4        1           3       34
  5        0           1       17
  6        1           1       19
```

```
7       1       1       18
8       0       1       19
9       1       1       16
10      0       1       15
11      0       1       16
12      1       1       23
13      0       4       9
14      0       4       11
15      0       4       12
16      0       4       12
17      0       4       10
18      0       4       12
19      0       4       9
20      1       2       30
```

2. **The Packet stream Model:**

   (a) The creation of a packet stream was tested to ensure that it was created and the correct details were generated. Thousands of packet streams were created in different sizes and specifications. An example is the creation of a packet stream with 20 packets, mean time between arrival of 10,000 time units and 4 classes. The four classes (1, 2, 3 and 4) had each: 2, 11, 3 and 4 packets. The maximum number of rules in an appropriate class rules that can be checked when filtering a packet is set at 100%. The packet stream is created and saved. In this example this following report was produced when a request to display the packet stream:

```
Number of packets:  20
Average time between arrivals:  10000
Number of packet classifications:  4

Number of packets meant to be in class 1 = 2
Number of packets meant to be in class 2 = 11
Number of packets meant to be in class 3 = 3
Number of packets meant to be in class 4 = 4

Exact number of packets in each class of packets in this packet
stream file:

 Class No: 1 :  2  packets.
 Class No: 2 :  11  packets.
 Class No: 3 :  3  packets.
```

```
Class No: 4 :  4  packets.

List of Packets:
```

pk no	Time since last packet	Packet Class	% of rules checked
1	212	3	84
2	8199	3	47
3	3047	3	84
4	11936	4	33
5	2545	1	44
6	9761	1	2
7	15323	2	64
8	4049	4	100
9	8599	4	88
10	13331	2	69
11	7590	4	81
12	1613	2	75
13	887	2	21
14	533	2	45
15	15470	2	17
16	18892	2	12
17	1417	2	2
18	19901	2	23
19	4516	2	93
20	5127	2	89

3. **The Filtering and Reporting Model:**

   (a) The user interface is a menu driven interface which is relatively simple to use, clear and functional.

   (b) With the filtering model it was possible to manually compute the cost of packet filtering operations of small packet streams filtered by small access lists. The results were compared against those produced by the model. The model also produces detailed account of a filtering operation one packet at a time. The following is a sample result of the detailed report of a simulation operation:

```
Access list used was from file name: al\_20\_4\_021\_88

Number of rules in access list:  20
```

```
Number of Classes in list:  4
Class 1: makes: 8    , processing time: 20 , accept rate: 50 \%
Class 2: makes: 1    , processing time: 35 , accept rate: 100 \%
Class 3: makes: 4    , processing time: 40 , accept rate: 75 \%
Class 4: makes: 7    , processing time: 12 , accept rate: 10 \%
 Class Order of the access list is :
 4,   1,   3,   2,

There are exactly  8 rules of the Class 1
There are exactly  1 rules of the Class 2
There are exactly  4 rules of the Class 3
There are exactly  7 rules of the Class 4

Packet stream used was from file name: pk\_20\_10000\_277

Number of packets:  20
Mean time between arrivals:  10000
Number of packet classifications:  4

class 1, makes 2 of packet stream
class 2, makes 11 of the packet stream
class 3, makes 3 of the packet stream
class 4, makes 4 of the packet stream

 List of Packets:
```

pk no	arr time	since last pkt	Packet Class	decide rule	service time	deprt. time	accum. serv time
1	212	212	3	3	336	548	336
2	8623	8411	3	1	256	8667	592
3	20081	11458	3	3	336	11794	928
4	43475	23394	4	2	20	23414	948
5	69414	25939	1	3	129	26068	1077
6	105114	35700	1	20	400	36100	1477
7	156137	51023	2	1	400	51423	1877
8	211209	55072	4	7	75	55147	1952
9	274880	63671	4	6	66	63737	2018
10	351882	77002	2	1	400	77402	2418
11	436474	84592	4	5	54	84646	2472
12	522679	86205	2	1	400	86605	2872
13	609771	87092	2	1	400	87492	3272
14	697396	87625	2	1	400	88025	3672
15	800491	103095	2	1	400	103495	4072
16	922478	121987	2	1	400	122387	4472
17	1045882	123404	2	1	400	123804	4872

```
18 1189187    143305     2      1      400   143705     5272
19 1337008    147821     2      1      400   148221     5672
20 1489956    152948     2      1      400   153348     6072


Total number of packets  received:     20
Total number of packets  Served:       20
Total service time:                    5672.000000
Maximum waiting Queue s ize:           0
Packets still in Waitin g Queue:       0
Packets still in the Server:           0

Actual average time between arrivals:  283.6000
Average service time (calculated):     303.6000061,    303.600000
```

7.3 Performance of Arranged Access Lists

The evaluation of the filtering operation covers both the functionality and performance of the filtering system. Functionality refers to two aspects: the behaviour of the service which the function (object, or application software) provides to its user; and the action or set of actions completed by the function when it is executed.

Performance refers in our case to the speed at which a function performs the service at hand. Performance is inversely proportional to the time taken by a function to perform a service in its entirety. The longer it takes to do the job, the lower the performance. In our case, it is the average processing time per packet when filtering a stream of packets by an access list.

Reorganising access lists based on rule classification is one way of improving performance particularly where there are a large number of rules. A particular arrangement of rules is only relevant to one particular characteristic of an incoming stream of packets. The research work progressed so that ultimately, it is one particular arrangement or permutation that is sought after, the permutation which yields the best performance. The first hypothesis (see Chapter 1) stated in the research objectives to test if access list performance can be improved by rearrangement of the access list. The second hypothesis relates to finding out the best way to arrange an access list to achieve the lowest average processing time needed per

packet.

The requirements specifications stated in Section 4.7 of chapter 4 was verified to ensure all requirements were met. The following are some of the observations worth mentioning:

1. Requirement number 3 in section 4.7.1, was about having an average time between arrivals of packets in a given packet stream to be larger than the time taken by a single packet to be filtered by all rules in a list. To ensure that this was possible, it was made to be a standard to display either of those two values whenever any access list or any packet stream is displayed. The access list model was made to produce the total cost of processing a single packet by all rules in the list under consideration. The Packet stream displayed the mean time between arrivals.

2. Requirement number 4 in section 4.7.1, was to test as wide a range as possible of number of packets in packet steams. Values were tested and verified ranging from a few packets about 10, to a maximum tested value of 10 million packets. All models were capable of handling such range of values. It was required at some stage to change the type of the memory storage variable holding the total cost of processing due to the size.

3. Requirement number 5 in section 4.7.1, was concerned with allowing some packets to remain unidentifiable by the access list in use. For each packet specification in a packet stream a value is assigned to the packet which states the percentage of rules to be applied to the packet before the critical rule is reached. This value only relates to the rules in the class which is the same class as the packet. A value of 50% for packet class A means that packet A will go through 50% of the rules in class A. This simulates that the critical rule for the packet exists approximately half way through the list. A value of 100% or higher, will make the packet go through the entire list of rules in its class without finding the critical rule. This value for individual packets is assigned randomly based on a value given as part of the stream specification during creation. The higher the value above 100% given during creation the more packets that will have to traverse the full length of their class.

   It was found that the same effect can be obtained by specifying more classes of packets in the packet stream than there are in the access list. A packet of class "x" will traverse through the entire list if no rules of class "x" exist in the access list.

But, this value of percentage of rules to be checked was used for a different purpose. It was used to specify for all packets in a particular class in a packet stream to have values below a certain value, i.e. 70%. This means that all packets will never go pass the rule marked as number 70% on the list. This makes the other 30% of the rules in a redundant situation. They have no effect on packets of similar class, but have a negative effect on all other packets. It was possible to determine the percentage of redundant rules in any given class in a given access list through controlling this variable in the packet stream.

4. Requirement number 7 in section 4.7.1, was concerned with being able to allow for equal processing time of rules in a list. This was achieved through the ability to create and edit a separate file to hold the processing time for each class (256 classes). These values can be selected to be equal or different. Each individual value can be accessed and changed to allow for certain requirement. When an access list is created, the file to use for assigning the processing time for the rules must be specified. It is possible to create many different files and use either when it suits.

5. Requirement number 2 in section 4.7.2, was concerned with the ease to create, display and re-arrange access lists. It was possible to allow creation and display much earlier on during the research work. With regard to the re-arrangement of the access list it was difficult to determine the method which would satisfy the requirements. Initially, the option was to arrange a list into a specific single arrangement. It was possible to achieve that, but was very impractical when having to deal with hundreds of re-arranged access lists. So, the next method was developed to re-arrange a list in all its possible permutations. That was successful with access lists of low number of classes, but was impossible to finish the job in a reasonable time for access lists with large number of classes. At the initial stages of the work lists were arranged by placing each of the classes once at the top and once at the bottom of the list. This was practical for initial experiments but of lesser value at later stages. One of the most successful ways to organise access lists with large number of classes was the random arrangement of the lists. The developed algorithm requests for the number of required arrangements to produce. The algorithm generates random numbers to represent the classes in the list and determine their new arrangement. This was especially helpful for example in producing 300 random different arrangements for an access list with 64 classes and higher.

7.4 Placing Class With Most Packets On Top

Figure 7.1 shows a typical observed result. This particular one involved the same access list which has 100 rules in two classes, class A and class B, with 56% of all the rules in the list belong to class B. Simulations were conducted once with the classes being randomly arranged and once with class B rules placed on top of the access list. The two access lists were used in the simulation to filter a number of packet streams. These packet streams have the same number of classes and all consist of mostly class B rules. They only differ in the number of class B packets within the stream as shown on the x-axis.

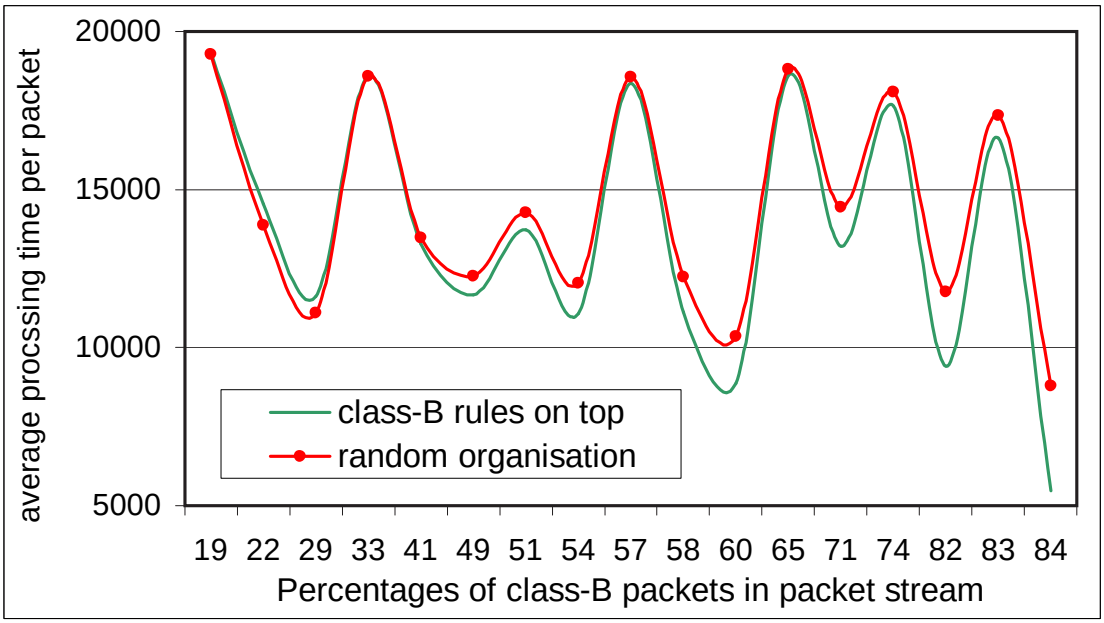

Figure 7.1: Access lists with class B rules on top or randomly mixed

The results indicate that when a packet stream is highly class B oriented, performance improves when the access list has the class B on top of the list. This improvement is indicated by the lower average processing time as depicted by the middle and right parts of the curve in Figure 7.1.

Notice the performance deterioration for some access list with the class B rules on top, see the left side of the Figure 7.1. The reason is that these packet streams contain very low number of class B packets. In other words, they are no longer class B oriented streams. In such cases, the randomly arranged class of the access list provided better performance. In fact, placing class B rules on top of the access list ceased to have an advantage when number of class B packets dropped below

41%.

Figure 7.2 shows the same class B oriented packet stream being filtered by the same access list. Simulations were conducted with class B rules once on top and once at the bottom of the access list. The results indicated a larger difference in performance was obtained in this access list when class B rules were placed at the top and at the bottom. The better performance is attributed to the fact that the packet stream is class B oriented, and likewise the access list. This result was expected, but as will later be demonstrated, is not always the case.

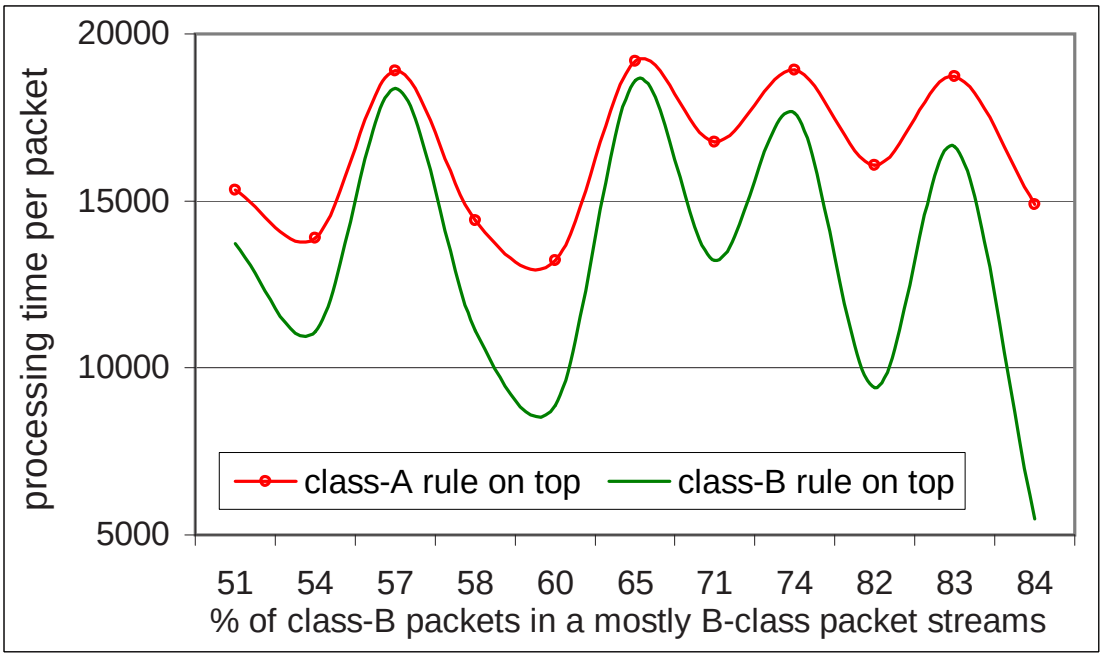

Figure 7.2: Access lists with class B rules on top or randomly mixed

Simulation experiments have shown that placing the class of rules which matches the majority of packets in a packet stream seems to work in most cases. Figure 7.3 shows the performance of a number of simulations for the same incoming packet stream having 10,000 packets and 8 classes of rules in an access list, being filtered by different organisations of the same access list. Most of the packets in the packet stream are class C packets. The access list has 500 rules and 8 classes of rules; it also has more class C rules than any other. For each simulation, access list rules belonging to a different class are placed on top of the access list as indicated on the x-axis in Figure 7.3. In this particular experiment, two access list organizations with class C rules on top gave different performance levels. Also some random organizations of the access list had a high-level performance, see the left part of Figure 7.3.

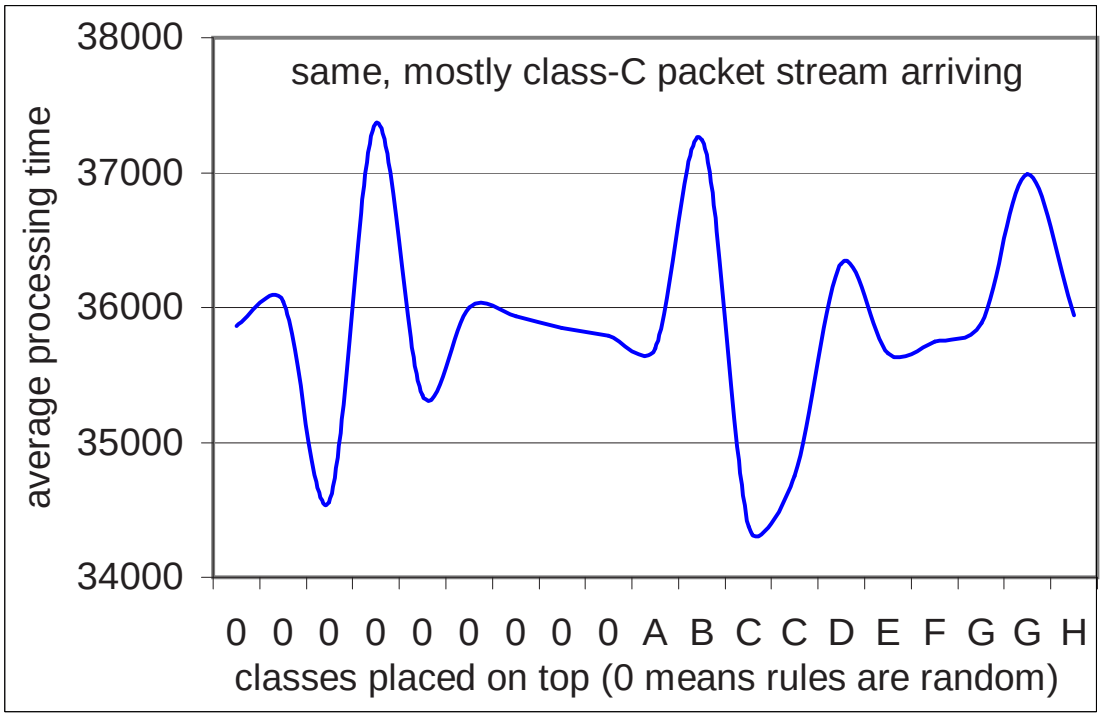

Figure 7.3: Access list of 500 rules with 8 classes in different access list organisations

Figure 7.4 represents the results of many packet streams each consisting of 10,000 packets in 64 classes, with varying percentages of class *A* rules; being filtered by a two–class access list. Better performance is achieved in most cases when Class *A* rules are placed on top of the access list as was expected. But, other varying levels of performance was noticeable indicating that performance must be determined by more than just placement on top of the access list the class that is dominant in the packet stream.

7.5 Class Ordering According to Number of Packets

Many of the results obtained did not support the hypothesis, i.e. having a particular class of rules on top of the list did not always produce the best results for a particular pattern. It seemed that it was possible to obtain better performance by placing some class rules on top of the access list, provided that the very same class is also the most common class of packets in the packet stream. Looking at the remaining rules in the access list, can they be arranged similarly to improve

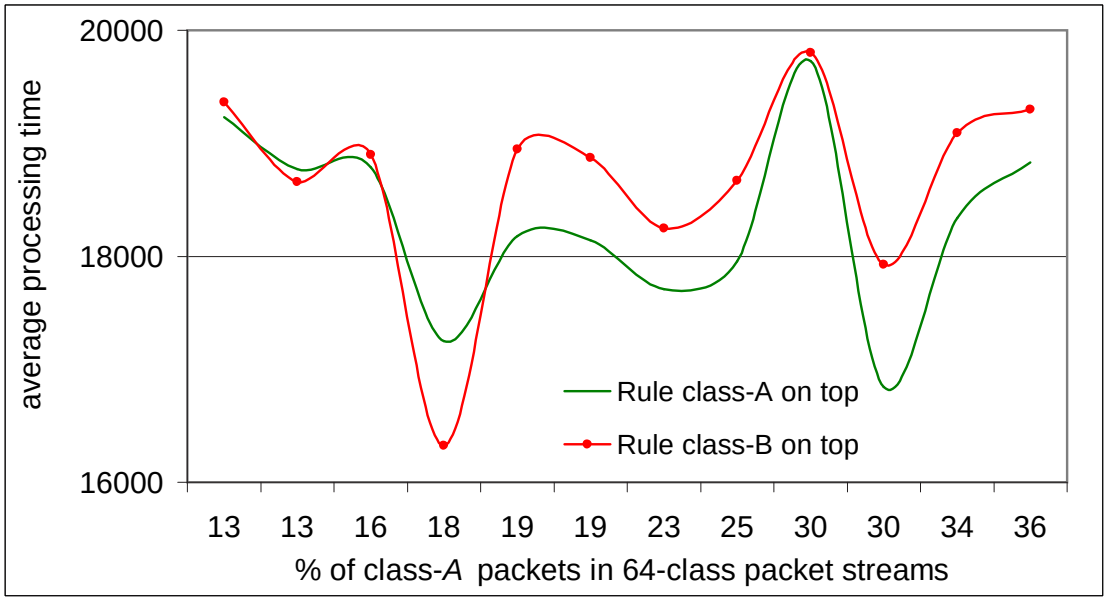

Figure 7.4: Packet streams of 64 classes and an access list in two organisations

performance? If the second most common class of packets in a packet stream can be identified, then in the access list the rules of the same class can be placed next to the previous class, and so on with the rest of the classes.

In other words, if the most common classes in a packet stream were *B, C, A* and *D* and in this order, then the access list rules are arranged according to their class to match the same order of the packets, *B, C, A* and *D*. When the lowest processing time points were inspected, most of them turned out to belong to access lists whose classes were arranged in the same organisation of those classes of the packet stream being filtered, but not for all cases.

Figure 7.5 shows the performance obtained by an access list with its rules classes arranged in the order *B, C, A* and *D*. The packet streams all had more packets of class *B*. But, the lowest processing time was obtained by filtering the packet stream with the classes percentages being: 16, 58, 21 and 5% for classes *A, B, C* and *D*. According to the number of packets they would be arranged as: *B, C, A* and *D*. Following closely behind is the stream with the class percentages: 17, 51, 19 and 13% whose classes can also be arranged as: *B, C, A* and *D* based on the number of packets. In this experiment, this makes the access list classes ordered to match the number of packets in the packet stream yield the best performance.

Here is an example where hypothesis 1 did not hold. Figure 7.6 shows a 10,000 packet stream of 4 classes mostly class *B*. The packets classes have the order: *B,*

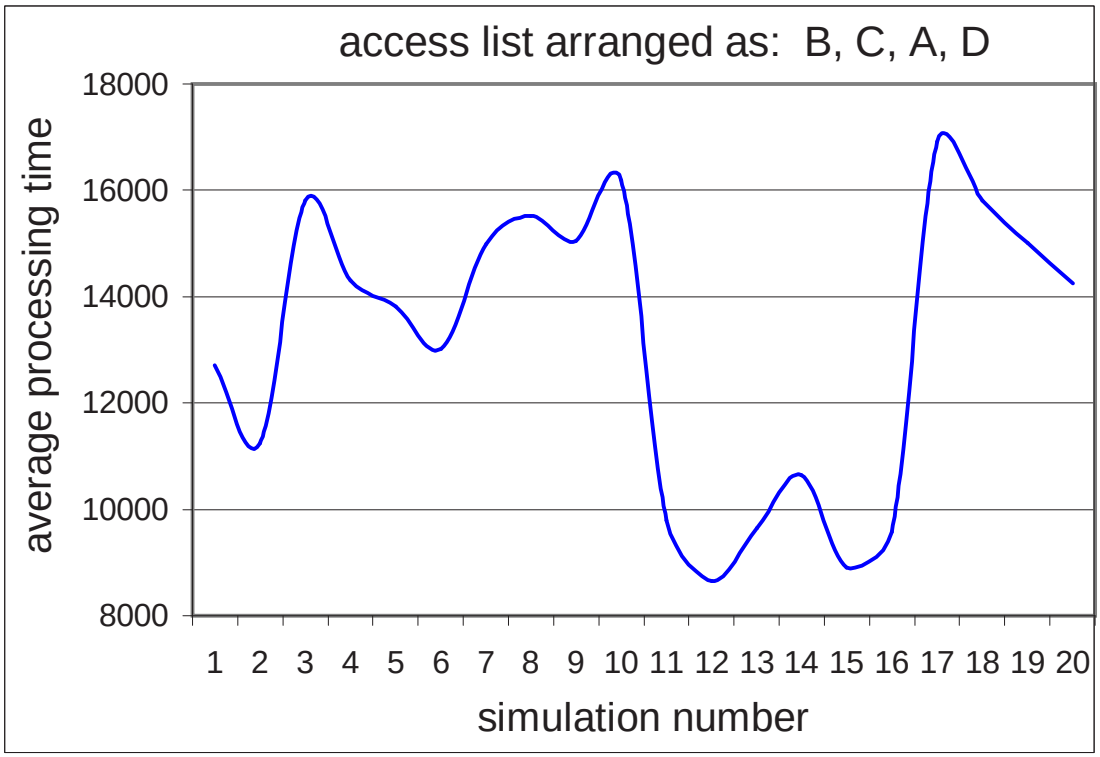

Figure 7.5: Access list class in order: B, C, A and D.

C, A and *D* when based on the number of packets in each class. That is, in the packet stream there were more packets in class *B* then class *A* then *C* and finally *D*. The packet stream is being filtered by an access list of four classes arranged randomly, then with class *B* once at the top and once at the bottom and one arrangement which matched the presence of packets in the stream that is (*B, A, C* and *D*).

In this particular example, the permutation of the access list classes based on the number of packets in the packet stream classes did not yield the lowest average processing time per packet.

7.6 Packet-Rule Cost Weight (PRCW)

Ordering an access list classes to match the number of packets in the packet stream was thought to produce the best results all the time, but that was not the case. In many of the situations inspected, still lower values of processing time were observed where the organisation did not exactly match the number of packets in

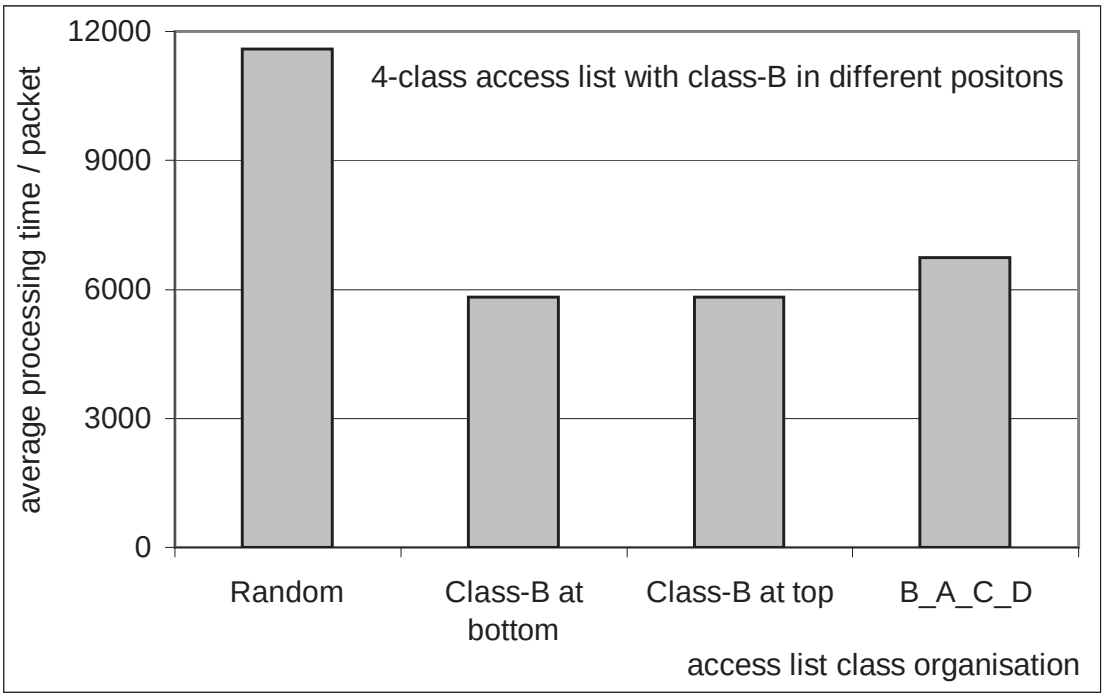

Figure 7.6: Arranged classes based on the number of packets.

each class in the packet stream. Yet, it is clear that performance is affected by rearrangement of access list rules based on their class.

The number of packets in a given class in a packet stream will have more relevance when their processing time by the rules in the classes is taken into consideration. Packet-Rule Cost Weight (PRCW) refers to the processing cost for a packet filtered by rules in an list in a particular organisation.

Figure 7.7 shows results of simulations for the same packet stream containing 10,000 packets with 4 classes, class B making up most of the packets. This is the same example as given in Figure 7.6. The PRCW method was used to calculate the processing time for all the different permutations and to provide the permutation yielding the lowest time for the given access list for a particular packet stream specification. In this case, using 4 classes of rules, the processing time for 24 permutations (4!) were calculated. The PRCW method defined the permutation for the shortest time was classes: A, B, C and D represented by the right column in Figure 7.7.

Figure 7.8 provides a quick comparison for performance represented by the average processing time per packet for a number of organisations. The packet stream contains 8 classes and is made up of mostly class A packets. The access list is

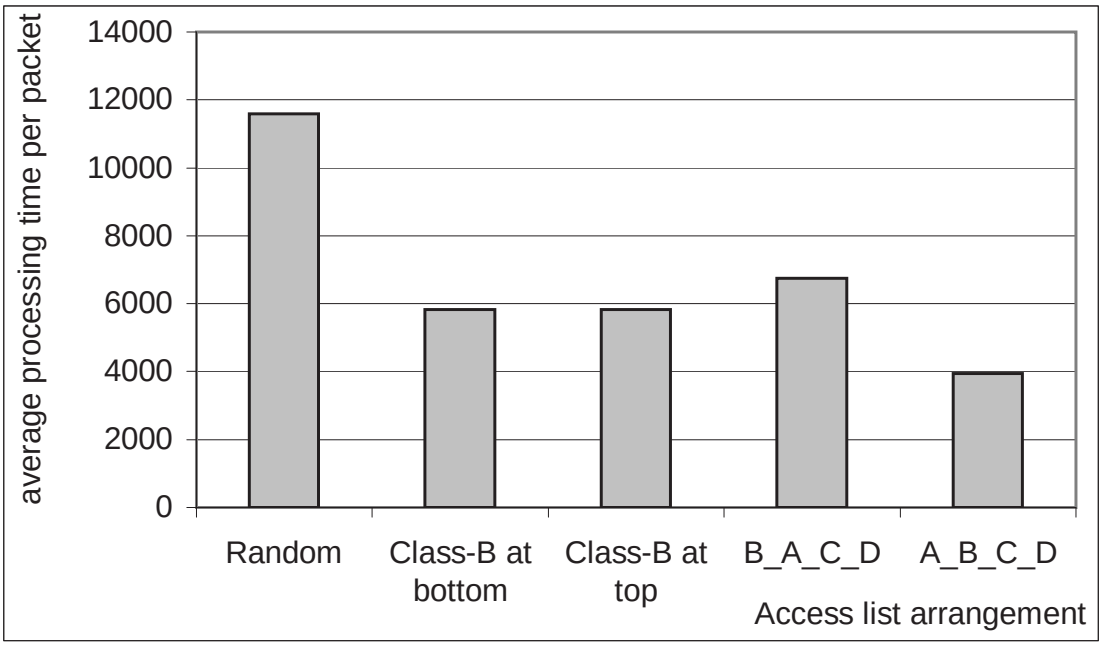

Figure 7.7: Access list with many arrangements including PRCW calculated

also an 8-class list, organised with class *A* rules at the bottom and at the top of the list. Class *A* at the bottom of the list gave the worst performance. The best performance obtained through the organisation based on the computation of the processing time using the PRCW method according to the permutation with the lowest cost.

The PRCW method was also used to compute the processing time for all the different permutations and to provide the permutation yielding the lowest time for the given access list for an access list with 8 classes of rules, the processing time for 40320 permutations (8!) were calculated. The permutation for the shortest time was classes: *A, H, G, F, E, D, C,* and *B*.

7.7 High-Cost Elimination (HCE)

The PRCW works everytime in producing the best permutation of classes for an access list to filter a particular packet stream. Yet, the drawback with the PRCW algorithm is in that it produces the cost for each and every permutation possible for a particular packet stream. The High-Cost Elimination (HCE) algorithm limits calculations to a fraction of that needed by the PRCW algorithm. The

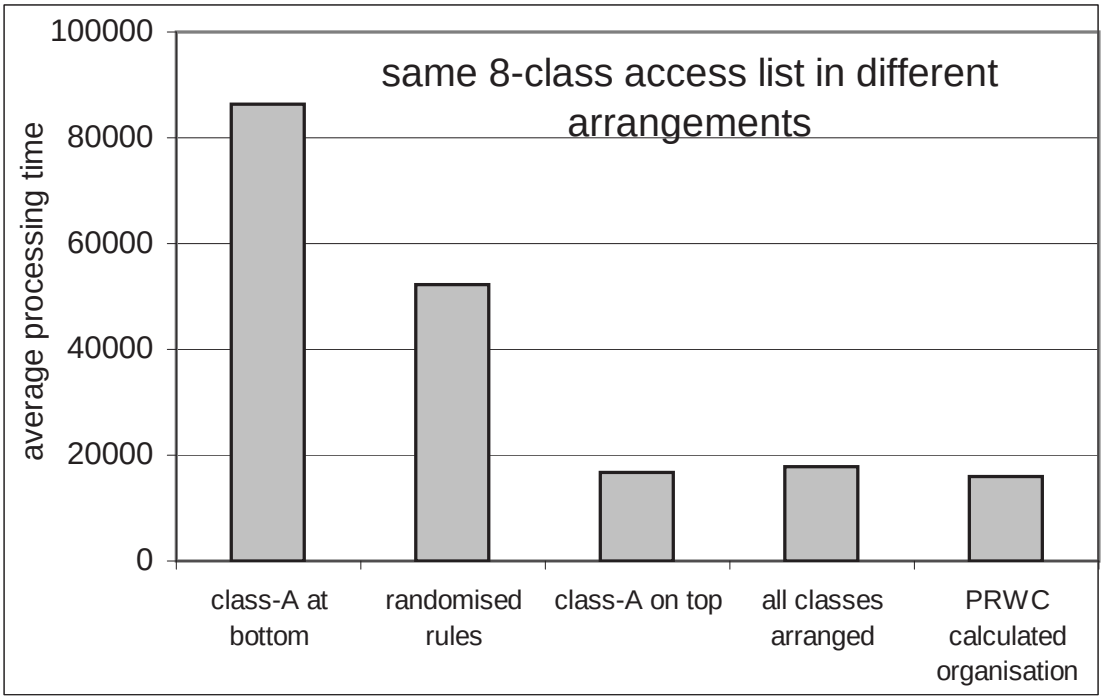

Figure 7.8: Performance comparison of different organisations of an 8-class list.

HCE algorithm has been tested and verified in tests against the original PRCW algorithm.

The HCE algorithm was used in a large number of experiments to produce the best permutation which will yield the best performance for a given access list filtering a given packet stream. Appendix A shows a large number of charts representing results obtained from simulation executions of different specifications of access lists and packet filters. The outcome was always compared to the permutation produced by the PRCW algorithm. The HCE was successful every time in producing the correct permutation.

As an example of many of the test carried out, an access list is assumed to have 100 rules with 4 classes and it was a class *C* oriented. The rules in each of the classes made 1%, 36%, 40% and 23% for classes *A, B, C* and *D*. The packet stream contained 10,000 packets with the percentages of packets in each of the classes *A, B, C* and *D*: 37%, 50%, 10% and 3%. The HCE algorithm produced the permutation classes: *A, B, C* and *D*. Figure 7.9 shows the results of the filtering processing cost of the packet stream by each and every permutation of the access list using the PRCW algorithm. The lowest value of the "Average processing time per packet" is that of the access list permutation classes: *A, B, C* and *D* which

was the same produced by the HCE.

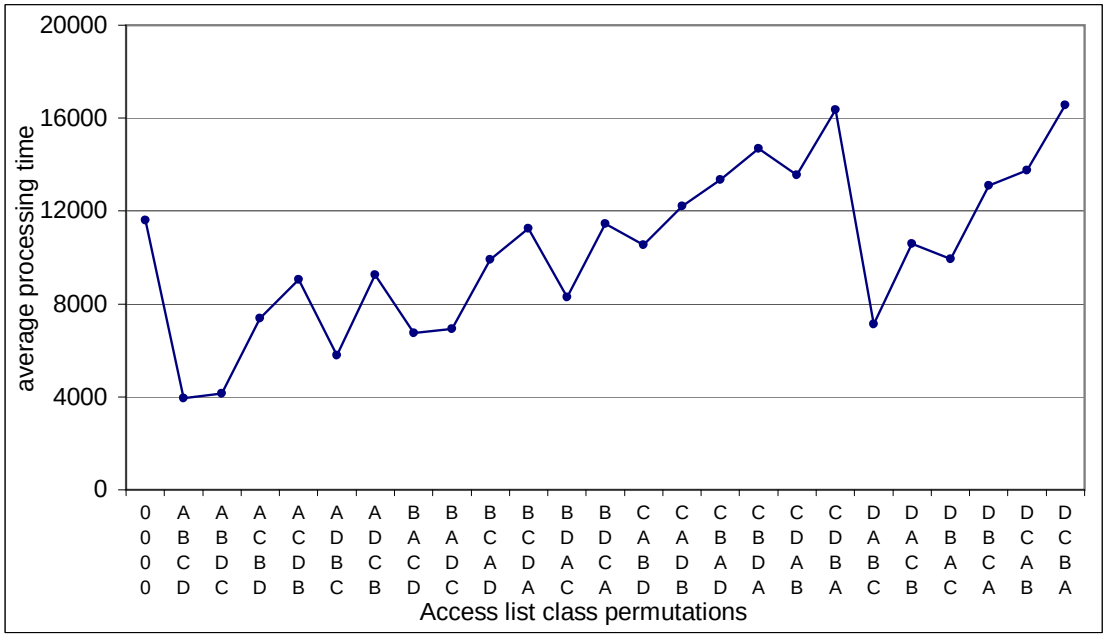

Figure 7.9: Performance of all permutations of a 4-class access list.

7.8 Summary

This work involved the design and development of a simulation model for packet filtering. More evaluation time was actually spent on verification of the model during its implementation phase. Once correct operation of the model was verified, the filtering simulation operations was carried out. In this chapter the results of tests and evaluation were described. Sample results from thousands of simulation executions of filtering operations were selected. They included many access lists in many organisations filtering many streams of packets with different characteristics. The results were outlined in the logical sequence in which the development process progressed. The next chapter will discuss achievements and problems encountered during this research work. It will also discuss future work and possible implementations.

Chapter 8

Conclusion

8.1 Introduction

This chapter presents the overall conclusions from the work described in this thesis. It outlines the objectives and aims of the research and discusses what was achieved and what was learned during the research work. It also gives directions for future work.

8.2 Aims of Research

As was outlined in Chapter 1, the main objectives of this research were summarised in three hypotheses:

Hypothesis One:
"Performance of packet filtering using access lists is improved by the ordering or by the re-arrangement of the rules in an access list based on their classes."

Hypothesis Two:
"When filtering a specific stream of packets, if performance of an access list can be improved by class reordering of the list, then a method can be devised which will find the best order of the classes in an access list to give the best performance possible."

Hypothesis Three:
"If a method can be devised which will find the order of the classes in an access to give the best performance when filtering a packet stream, then an efficient way must exist in finding that order."

The first hypothesis states that improvement in performance is achieved by re-ordering the access list bases on the classes. Observation and analysis of experiments carried out did not fully support this hypothesis. The research work established that improved performance depended on the re-arrangement of the access list only when taken in relation with the characteristics of the packet stream being filtered. The re-arrangement of access list alone may improve performance for one packet stream, or may cause it to deteriorate for another.

If the hypothesis is rephrased such that reordering of an access list will improve performance when the hypothesis applies to a single particular access list and a single particular packet stream of specific characteristics, then the results obtained can support such a hypothesis. The modified hypothesis can be stated as follows: "*Class re-ordering of the same access list when filtering the same packet stream will effect performance*". Changing the order can improve or deteriorate performance. In order to improve performance the ideal permutation of the classes in the list must first be determined for the specific access list and packet stream.

Hypothesis two refers to finding an algorithm which computes the best order of the classes in an access list which will give the best performance when filtering a particular packet stream. After establishing that the cost of processing arriving packets depended on the number of rules, the processing cost of the rules and the number of packets, the Packet-Rule Cost Weighing (PRCW) algorithm was developed. The PRCW made it possible to correctly calculate the cost for a packet stream when passing through a specific permutation of an access list. The best organisation of the access list to filter a specific packet stream is computed by using the PRCW algorithm to compute the cost using every arrangement possible for the access list. The best arrangement is the one that caused the lowest cost to be computed. In thousands of experiments carried out, the PRCW produced the best organisation. The development of the Packet-Rule Cost Weighing algorithm supports the second hypothesis.

Hypothesis three refers to finding an efficient algorithm which can in a relative short time find the best arrangement of an access list when filtering a specific packet stream. It was established that efficiency here refers to performance of the

algorithm. Here, it is taken as a measure of the length of time (or amount of processing) needed for an algorithm to find the best arrangement for an access list. While the best arrangement of an access list will refer to that arrangement which when used to filter a specific packet stream, will give the lowest average processing time per packet. In both cases, the lower the value, the more efficient either the algorithm or the arrangement.

Both algorithms (PRCW and HCE) were based on establishing a measure for cost that ties both packets and access list rules. The PRCW algorithm used the hard-headed approach of computing the cost for each and every possible permutation of the access list classes to find the best. This can take a long time when the access list contains a large number of classes. To add efficiency to this algorithm a way was needed to reduce the number of calculations required. The High-Cost Elimination (HCE) algorithm was developed which reduced the number of calculations to a fraction compared with the PRCE algorithm. The HCE is also based on establishing a measure for cost that combines both packet streams and access list rules. The High-Cost Elimination (HCE) algorithm supports the third hypothesis.

Chapter 1 also outlined the following sub-objectives:

1. Full investigation and understanding of the topics of network security, network communication including Internet protocols.

2. Investigation of existing methods and future expectations of packet filtering in network devices such as firewalls, routers, bridges and gateways.

3. Design and develop a model of the operation of a network device that uses access lists to filter a stream of packets.

4. Perform detailed simulation experiments using different combinations of packets and filtering rules perform analyses of the experimental results and draw the appropriate conclusions and recommendations.

5. Produce conclusions of the effect of reordering of access lists on performance of packet filtering operations.

All other sub-objectives outlined above were achieved through the course of this research work.

8.3 Achievements

With respect to the aforementioned objectives, and the requirements stated in Section 4.7, the achievements of this research are as follows:

- The current state of the art in access lists based packet filtering was examined in this thesis by analysing relevant work in the literature and by analysing existing work described by other researchers. The work carried out included the classification of packets, description of access list filtering mechanisms as well as different approaches to filtering.

- The access list performance issues were discussed and the hypotheses were stated in relation to the effects of class ordering of the access lists rules on performance.

- A model of the operation of a network device that uses an access list based packet filter was designed. This was based on the analysis of access lists, and access list based filtering devices. The design included specifications of simulation inputs and parameters. Also the specification requirements of performance. The design included the different tests and verification methods of results obtained.

- The model was verified for correctness and closeness to the real life system through a number of experimental simulation executions. The results of such experiments were compared to expected results and accordingly the model was modified. The experimental executions were repeated until the model was found to behave in a correct manner.

- The developed model was used to experiment and evaluate different access list structures, types, sizes, contents, classifications and orders, with many different packet streams. A very large number of simulation executions were performed and the results were analysed.

- The evaluation results obtained have been discussed in detail and overall conclusions regarding the obtained results were made.

- Progress by stages was a feature of this research work in the area of access list performance. The work started with simple ordering and went on to develop a costing method for filtering called the Packet-Rule Cost Weighing (PRCW) method to find the best permutation. This progressed to formulate the highly efficient and innovative High-Cost Elimination (HCE) algorithm.

In this thesis it was possible to show the effects of ordering the rules in an access list on the performance of access list based filtering communication devices. It was possible to show this through simulation of the operation of such devices using different possible packet patterns and large numbers of access lists. The results of the simulation showed clearly that ordering the access list can have an effect on performance when jointly considered with the pattern of the packet stream arriving to the device. The results indicate that different patterns of packets arriving into a network device will require a specific access list order to yield the best performance. Performance in this case is measured by how long it takes a packet to be filtered through the list of rules. When considering a number of packets, then the measurement criteria used is the average time needed to process each of the packets, which was referred to as the average processing time per packet.

The pattern of the packet stream arriving to a communication device may be difficult to predict and consequently the most suitable order of the access list can not be determined. The decision must be made based upon a historical knowledge of the packet patterns in the past. This accumulative historical knowledge can be considered by performing the analysis (of the packets received) at suitable intervals. This time span can vary from monthly, weekly, daily, hourly intervals etc. The online continuous analysis of packets arriving is also possible. As long as for the particular communication device it can be determined how often packet patterns normally change then it the analyser can be invoked to determine the best order of the access list for the existing (or most recently recognised) packet pattern. The new innovative High-Cost Elimination (HCE) algorithm can allow such an on line analyser to determine the new class order to suit changes in packet stream characteristics

8.4 Future work

A number of points have been identified and are worth considering for future work. The following is a brief discussion:

1. A natural step to follow this work is to have the implementation done on real life network device that performs actual packet filtering. Prediction of packet stream characteristics will be needed based on analysis of logs of a real classifier. Then testing the performance obtained through the

precalculated access list organisation based on the High-Cost Elimination (HCE) algorithm must be carried out.

2. The development of an automatic analyser for packet streams arriving to a network device. Such an application would be expected to produce the specifications of the packet stream being analysed. Specifications may include, rate of arrival, number of classes and the distribution of packets into the classes. Such an application will require the log of packets arriving to a communication device to be kept for analysis.

3. Automatic ordering of an access list for a given access list can be performed based on the packet stream specification produced by the stream anlyser. The developed High-Cost Elimination (HCE) algorithm can be used as the basis for selecting the most suitable class order of the access list. The HCE algorithm is fast enough to be able to make the selection suitable for an efficient operation. The HCE is suitable for both dynamic and less dynamic network devices where changes in packet stream characteristics differ in how long they remain unchanged.

4. The automatic generation of an access list based on some given security policy. The task of manually generating an access list is a very difficult task except for the simplest policies. It is thought that some syntax for specifying security policies can be used as input to produce the required access list.

5. The verification of an access list from a security point of view. There is a need for ensuring that a generated access list or a reordered access list is adhering to the security policy for which it was created. The security specification and the access list to be tested may be used to produce some testing packet stream. This testing packet stream will include as many packets with all the required values to test for all possible conditions in the security policy.

6. Given two versions of two different orders of the same access list, there should be an application which can verify that both copies will implement the same security policy. This will help ensuring that a reordered list will not deviate from the security policy its original list was hopefully successfully implementing.

7. As a part of system management, an off line tool can easily be developed to provide information of how to arrange an access list in order to achieve best performance for a given specification of an access list.

8.5 Summary

The work had two major parts, development of a model and the simulation executions of this model. The model was developed to reflect the operation of a network device performing packet filtering operations. Validating the model for correct operations was vital to accept results from further experiments that were carried out. Model execution produced helpful results in formulating and verifying both the Packet-Rule Cost Weighting algorithm and the High-Cost Elimination algorithm.

This research succeeded in producing strong support for considering class reordering as a method of improving the performance for access list packet filtering. A new algorithm for deciding the best order was also developed.

As far as access list-based filtering is concerned this research reflects how much more effort is needed for improvements to packet filtering.

Bibliography

Alexander D.S., Menage P.B., Keromytis A.D., a. Arbaugh W., Anagnostakis K.G. and Smith J.M., 2001. The Price of Safety in an Active Network. *Journal of Communications and Networks (JCN)*.

Arsham D.H., 1996. Systems simulations: The shortest path from learning to applications. University of Baltimore, Maryland, USA. Available online at `http://ubmail.ubalt.edu/~harsham/simulation/sim.htm`. [Last Accessed May 2004].

Ballew S.M., 1997. *Managing IP Networks with Cisco Routers*. 1st ed. O'Reilly.

Banks J. and Carson J.S., 1996. *Discrete-event system simulation*. 2nd ed. Englewood Cliffs, N.J.: Prentice Hall.

Beatson J.G., 1992. Information Security: The Impact of End User Computing. *In: 8th International Federation for Information Processing (IFIP) Technical Committee: TC11*. Singapore: International conference on Information Security.

Bellovin, 1992. There Be Dragons. *In: Proceedings of the Third USENIX UNIX Security Symposium*. USENIX, Baltimore, MD: Proceedings of the Third USENIX UNIX Security Symposium.

Bhatt S., Fujimoto R., Ogielski A. and Perumalla K., 1998. Parallel simulation techniques for large scale networks. *IEEE Communications Magazine*, 36 (8), 42–47.

Bryant R., 1992. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *Communications of the ACM*, 24 (3), 293–318. ACM Computing Surveys.

CERT, 1996. *Denial of service attacks via Ping*. Computer Emergency Readiness Team Advisory, CA-96.26, US-CERT. Available online at `http://www.cert.org`.

Chapman D., 1992. Network (In)Security Through IP Packet Filtering. *In: Third USENIX UNIX Security Symposium*. Baltimore, MD: USENIX.

Cheswick W. and Bellovin S., 1994. *Firewalls and Internet Security, Repelling the Wily Hacker*. Addison–Wesley.

Commission, 1998. *Ghost in the Machine, An Analysis of IT Fraud and Abuse*. Tech. rep., Audit Commission Publications, UK.

CORBA, 1998. *CORBA/Firewall Security+Errata, Object Management Group, Joint Revised Submission orbos/98-07-03*. Joint revised submission orbos/98-7-03, The Common Object Request Broker Architecture (CORBA).

Derrick E.J., 1992. *A Visual Simulation Support Environment Based on a Multifaceted Conceptual framework*. Ph.D. thesis, Department of Computer Science, Viginia Tech, Blacksburg, VA.

Dunkels A., 2002. *Minimal TCP/IP implementation with proxy support*. Techincal Report T2001-20, Swedish Institute of Computer Science.

Fall T.C., 1997. A Framework for the Simulation Experimentation Process. *In: Proceedings of the 1997 Winter Simulation Conference*.

Feit and Sidnie, 1993. *TCP/IP Architecture, Protocols and Implementation*. McGraw-Hill.

Feldmann A. and Muthukrishnan S., 2000. Tadeoffs for Packet Classification. *In: IEEE Infocom 2000*, vol. March. IEEE, IEEE.

Friedman D. and Nagle D., 2001. *Building Firewalls with Intelligent Network Interface Cards.*. Techincal Report CMU-CS-00-173, Carnegie Mellon University.

Furnell S., 2002. *Cypercrive:Vandalizing the Information Society*. Addison Wesley Professional.

Furnell S.M., Dwland P.S. and Sanders P.W., 1999. Dissecting The 'Hackers Manifesto'. *In: Information Management & Computer Security*, vol. 7 of *2*.

Gass S.I., 1993. Model Accreditation: A Rationale and Process for Determining a Numerical Rating. *European Journal of Operational Research*, 66 (2), 250–258.

Gupta P. and McKeown N., 1999. Packet Classification using Hierarchical Intelligent Cuttings. *In: Proceedings of Hot Interconnects VII*, vol. August.

Gupta P. and McKeown N., 2001. Algorithms for Packet Classification. *In: IEEE Network Special Issue*, vol. 15. IEEE Computer Society, 24–32.

Habtamu A., 2000. *An Overview of Firewall Technologies*. Tech. rep., Norwegian Computing Centre, Norway.

Haeni T., 1997. Firewall Penetration Testing. The George Washington University. Available online at `http://www.mabuse.de/sources/firewall.pdf`. [Last Accessed May 2004].

Hatley D. and Pirbhai I., 1987. *Strategies For Real-Time System Specification*. Dorset House Publishing, New Yourk.

Hazelhurst S., 1999. *Algorithms for Analysing Firewall and Router Access Lists*. Techincal Report TR-Wits-Cs-1999-5, Audit Commission Publications, UK, Dept. of Computer Science, University of Witwatersand, South Africa.

Hazelhurst S., 2001. A Proposal for Dynamic Access Lists for TCP/IP Packet Filtering. *In: South African Institute of Computer Scientists and Information Technologists (SAICSIT 2001)*. South Africa.

Hazelhurst S., Fatti A. and Henwood A., 1998. *Binary Decision Diagram Representations of Firewall and Router Access Lists*. Tech. rep., University of Witwatersrand, South Africa.

Highland H.J., 1990. *Computer virus handbook*. Oxford, England: Elsevier Advanced Technology.

Holbaek-Hanssen E., Handlykken P. and Nygaard K., 1977. *System Description and the Delta Language*. Tech. Rep. Report No 4, Robin Hills (Consultants) Ltd., Surrey, England, UK.

Howard J.D., 1998. *An Analysis of Security Incidents on the Internet.*. Ph.D. thesis, Carnegie Mellon University.

Huang P., Estrin D. and Heidemann J., 1998. Enabling large-scale simulation: Selective abstraction approach to the study of multicast protocols. *In:*

Proceedings of the Sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '98.

Humphrey J. and Gabrielson B.C., 1995. Phreakers, Trashers and Hackers. South Africa: AFSEA INFOSEC.

Icove D., Seger K. and Vonstorch W., 1995. *Computer Crime: A Crime-fighter's Handbook*. Sebastopol, California, USA: O'Reilly & Associates Inc.

IETF, 1996. RFC 1918, Address Allocation for Private Internets . Internet Engineering Task Force. Available online at `http://www.ietf.org/rfc/rfc1918.txt`. [Last Accessed May 2004].

Ioannidis J. and Bellovin S., 2002. Implementing Pushback: Router-Based Defense Against DDOS Attacks. *In: The Network and Distributed System Security Symposium Conference (NDSS)*.

Jackson M., 1983. *System Development*. Prentice-Hall, Englewood Cliffs, NJ.

Jobin J., Faloutsos M. and Tripathi S., 2001. Performance Evaluation of Mobile Wireless Networks: A New Perspective. *In: The Fourth ACM International Workshop on Modeling (MSWiM)*.

Lakshman T. and Stidialis D., 1998. High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. *In: Special Interest Group on Data Communications (SIGCOMM)*.

Landwehr C., Drewer P., Taylor V., Devlin P., Scott A., Allain D., Harrison M. and McLean J., 1997. *Safe Use of the Internet for Defence Purposes*. Tech. Rep. DOC - C3I - 1 1997, THE TECHNICAL COOPERATION PROGRAM, A Report from the Panel on Secure Information Systems, SUBCOMMITTEE ON NON-ATOMIC MILITARY RESEARCH AND DEVELOPMENT. STP-11, C3I Group.

Lassila P., 2001. *Methods for Performance Evaluation of Networks: Fast simulation of loss systems and analysis of internet congestion control*. Phd, Department of Electrical and Communications Engineering, Helsinki University of Technology, Helsinki Universtiy of Technology, Espoo, Finland.

Mahajan R., Bellovin S.M., Floyd S., Ioannidis J., Paxson V. and Shenker S., 2001. *Controlling High Bandwidth Aggregates in the Network*. Techincal report (draft), AT&T Center for Internet Research at ICSI (ACIRI) and AT&T Labs Research.

Marzo J.L., Vila P., Fabrega L. and Massaguer D., 2003. A distributed simulator for network resource management investigation. *Computer Communications*, 26 (15), 1782–1791.

Menezes A., Oorschot P.V. and Vanstone S., 1996. *Handbook of Applied Cryptography*. CRC Press.

Mizrach S., 1997. Is there a Hacker Ethic for 90s Hackers? Florida International University. Available online at `http://www.fiu.edu/~mizrachs/hackethic.html`. [Last Accessed May 2004].

Muftic S., 1994. Seurity Architecture for Open Distributed Systems. *In: Computer Networks and ISDN Systems*, vol. 26. 1343–1349.

Nance R., 1981. The Time and State Relationships in Simulation Modeling. *Communications of the ACM*, 24 (4), 173–179.

Nance R., 1993. A History of Discrete Event Simulation Programming Languages. *In: The second ACM Special Interest Group for Programming Languages (SIGPLAN): History of Programming Languages Conference*, Notices, 28(3). Cambridge: ACM SIGPLAN, 149–175.

Overstreet C., 1982. *Model Specification and Analysis for Discrete Event Simulation*. Ph.D. thesis, Department of Computer Science, Virginia Tech, Blacksburg, VA.

Page E., Nicol D., Balci O., Fujimoto R., Fishwick P., L'Ecuryer P. and Smith R., 1999. PANEL: Strategic dirctions in simulation research. *In: Proceedings of the 1999 Winter Simulation Conference, 1999*. IEEE, Pheonix, Az, USA: IEEE.

Sargent R.G., 1982. *Verification and Validation of Simulation Models*. F. E. Cellier,, Academic Press, London.

Sargent R.G., 1998. Terminology for Model Credibility of Simulations. *In: Proceedings of the 1998 Winter Simulation Conference,*. IEEE, 121–130.

Schlesinger S., Crosbie R., Gagn R., Innis G., Lalwani C., Loch J., Sylvester J., Wright R., Kheir N. and Bartos D., 1979. Terminology for Model Credibility. SCS Technical Committee on Model Credibility. *Simulation*, 32 (3), 103–104.

Schreiner R., 1998. CORBA Firewall White Papers. Internet. Available online at http://www.technosec.com/whitepapers/corba/fw/main.html. [Last Accessed March 2003].

Schuba C. and Spafford E., 1997. A Reference Model for Firewall Technology. *In: Thirteenth Annual Computer Security Applications Conference*. Dept. of Comput. Sci., Purdue Univ., West Lafayette, IN, USA: IEEE Computer Society. Available online at http://www.computer.org/proceedings/acsac/8274/8274toc.htm.

Selani, 1998. ICSA Firewall Policy Guide v2.00 . International Computer Security Association (ICSA). Available online at http://www.du.edu/~dcelani/paper.htm. [Last Accessed May 2003].

Shannon R.E., 1975. *Systems Simulation: The Art and Science*. Prentice-Hall, Englewood Cliffs, NJ.

Srinivasan V., 1999. Packet Classification using Tuple Space Search . *In: Special Interest Group on Data Communications (SIGCOMM)*.

Srinivasan V., 2001. A Packet Classification and Filter Management System. *In: IEEE International Conference on Computer Communication (INFOCOM)*.

Stoica I., 2001. Rout Lookup and Packet Classification CS268. University of California, Berkely. Available online at http://www-inst.eecs.berkely.edu/~cs2268/sp03/. [Last Accessed 30 July 2003].

Suri S. and Varghese G., 1999. Packet filtering in High Speed Networks. *In: Symposium on Discrete Algorithms*. Baltimore, Maryland, USA: ACM-SIAM.

Tanenbaum A., 1996. *Computer Networks*. Prentice Hall, Upper Saddle River, New Jersey, USA.

Warkhede P., 2001. Fast Packet Classification for Two-Dimentional Conflict-Free Filters. *In: IEEE International Conference on Computer Communication (INFOCOM)*.

Wood D.O., 1986. MIT Model Analysis Program: What we have learned about policy model review. *In: Proceedings of the 1986 Winter Simulation Conference, 1986*. Washington D.C., USA: IEEE, 248–252.

Wulf L., 1997. *Interaction and Security in Distributed Computing*. Ph.D. thesis, Wolfson College, University of Oxford. Available online at `http://web.comlab.ox.ac.uk/oucl/publications/books/concurrency/examples/security/index.html`.

Appendix A

Sample of Simulation Results

All the following charts show results of simulated filtering operations. They represent a sample of hundreds of thousands of simulation executions carried through out the research. In each of the experiments both the Packet Rule Cost Weight (PRCW) and the High Cost Elimination (HCE) algorithms were used to compute the best possible permutation. In every case both algorithms predicted the best permutation which on the chart showed the lowest average processing cost per packet. On some of the charts, the number of permutations is too large to show each individual one.

Figure A.1 shows an access list of 100 rules with 4 classes of rules made up of the following number of rules: 33, 41, 11 and 15. The list was filtering a packet stream of 10,000 packets with an mean time between arrival of 10,000 units of time. The packet stream had 4 classes of packets made up of the following number of packets: 1495, 5544, 1024 and 1937. The best access list class permutation predicted was: classes 4, 2, 1 and 3.

Figure A.2 shows an access list of 100 rules with 4 classes of rules made up of the following number of rules: 33, 41, 11 and 15. The list was filtering a packet stream of 10,000 packets with an mean time between arrival of 10,000 units of time. The packet stream had 4 classes of packets made up of an extreme distribution of packets where most packets belonged to class-1. The number of packets were: 9793, 115, 91 and 1 for classes 1,2,3 and 4. The best access list class permutation predicted was: classes 1, 3, 2 and 4.

Figure A.3 shows an access list of 100 rules with 4 classes of rules made up of the

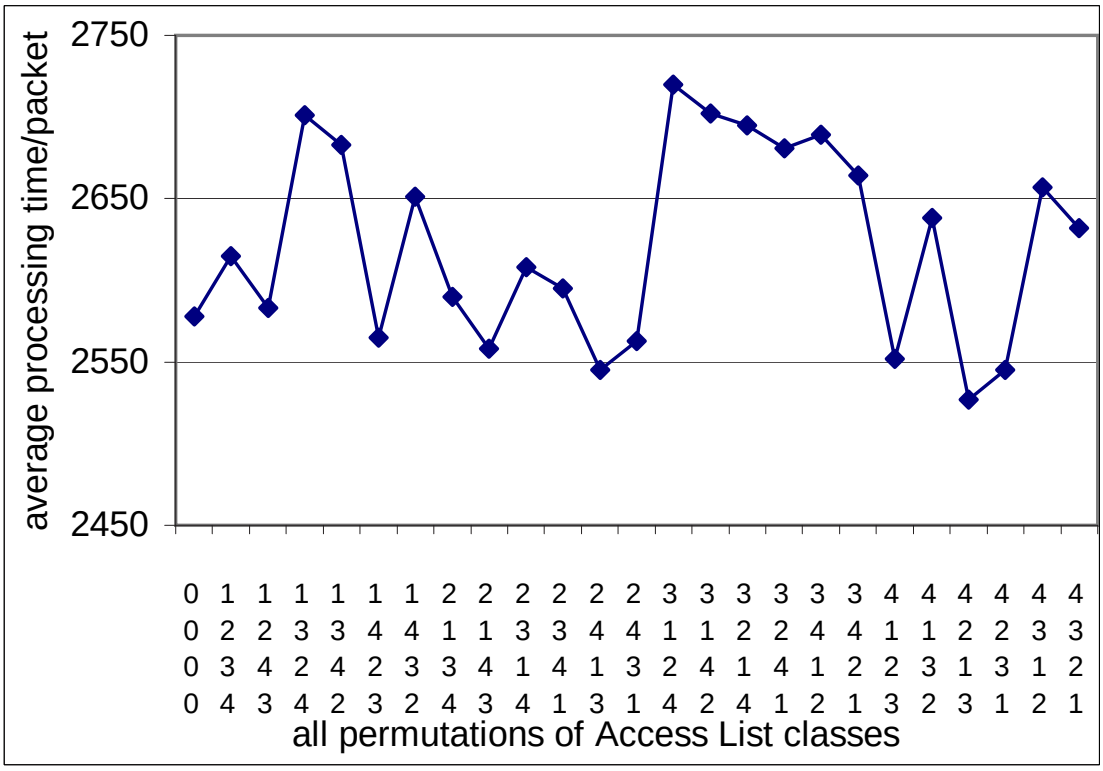

Figure A.1: All permutations of 4-class access list filtering a 4-class stream.

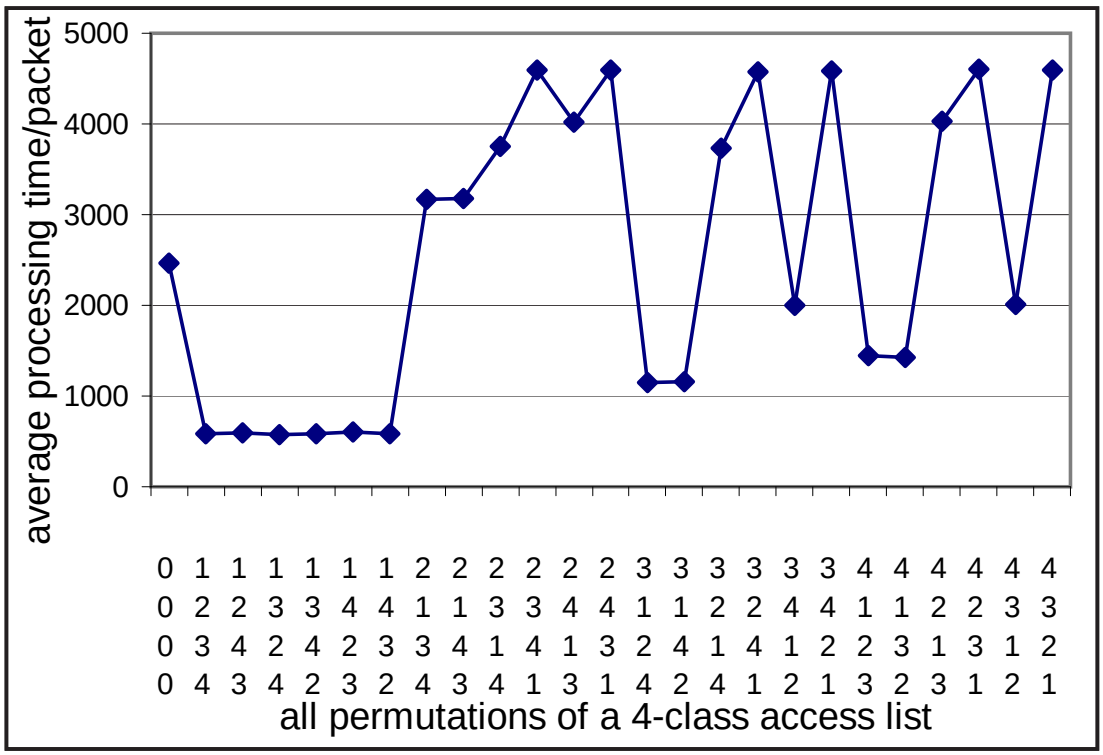

Figure A.2: All permutations of 4-class access list filtering a 4-class stream.

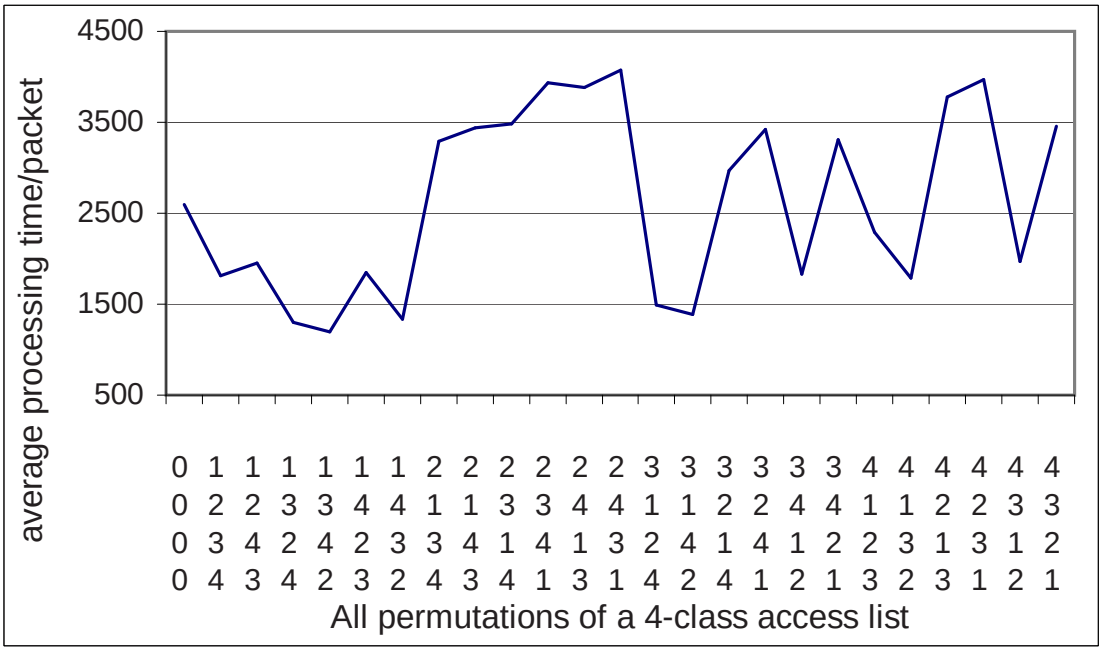

Figure A.3: All permutations of 4-class access list filtering a 4-class stream.

following number of rules: 33, 41, 11 and 15. The list was filtering a packet stream of 10,000 packets with an mean time between arrival of 10,000 units of time. The packet stream had 4 classes of packets made up of the following number of packets: 5901, 1040, 2290 and 769. The best access list class permutation predicted was: classes 1, 3, 4 and 2.

Figure A.4 shows an access list of 100 rules with 4 classes of rules made up of the following number of rules: 15, 14, 4 and 67. The list was filtering a packet stream of 10,000 packets with an mean time between arrival of 10,000 units of time. The packet stream had 4 classes made up of the following number of packets: 5901, 1040, 2290 and 769. The best access list class permutation predicted was: classes 4, 1, 3 and 2.

Figure A.5 shows an access list of 100 rules with 8 classes of rules made up of the number of rules: 12, 22, 5, 13, 13, 18, 8 and 9. This has 40,320 different permutations. The list was filtering an 8-class packet stream of 10,000 packets with an mean time between arrival of 10,000 units of time. The packets classes had the following percentages: 130, 15, 140, 175, 264, 31, 58 and 118. The best access list class permutation predicted was: classes 3, 4, 1, 5, 8, 7, 6 and 2, on the chart it is the execution number 11569.

Figure A.6 shows an access list of 500 rules with 64 classes of rules made up of

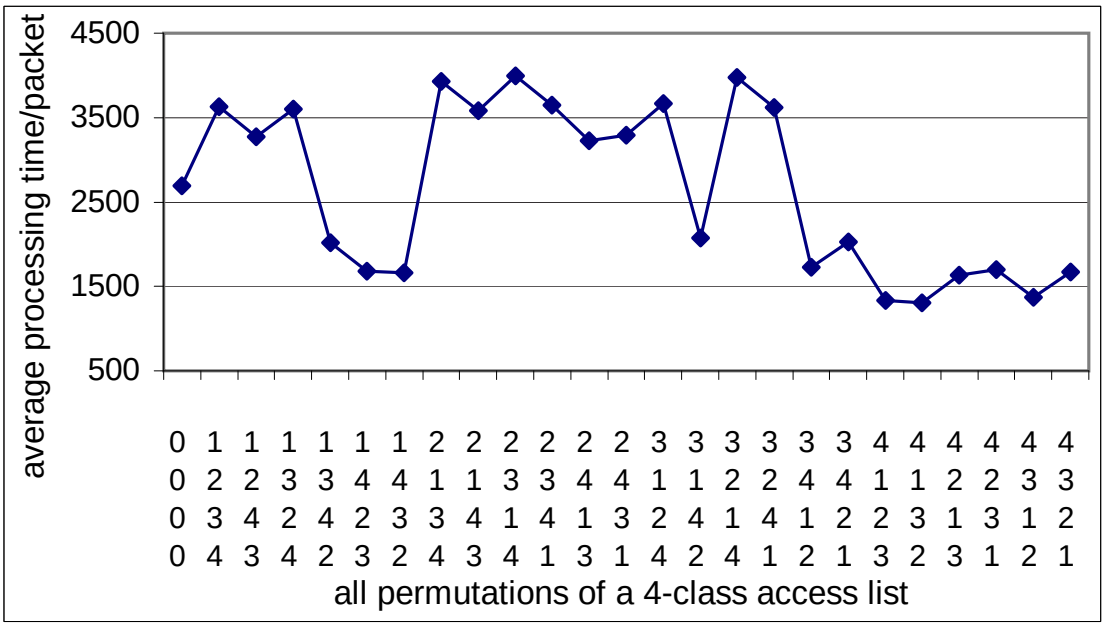

Figure A.4: Access list with 8 classes filtering an 8-class packet stream.

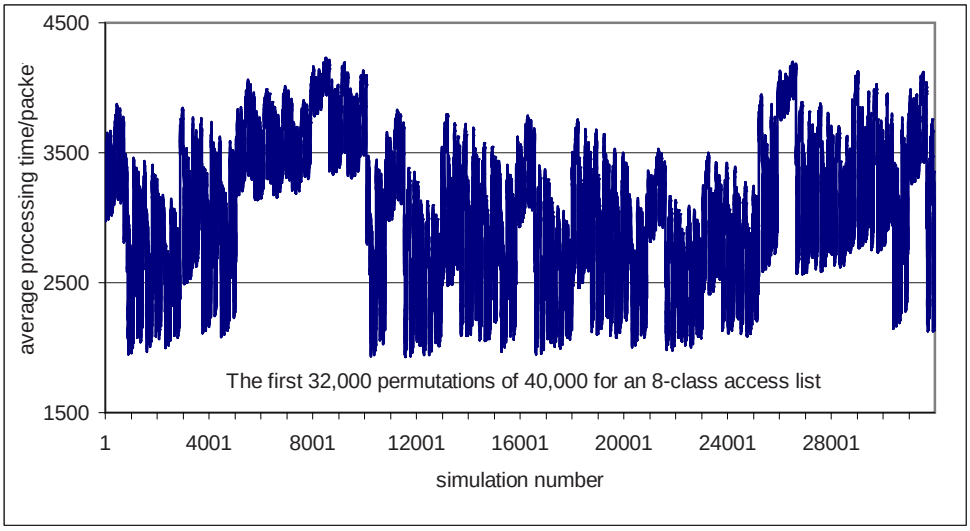

Figure A.5: Access list with 8 classes filtering an 8-class packet stream.

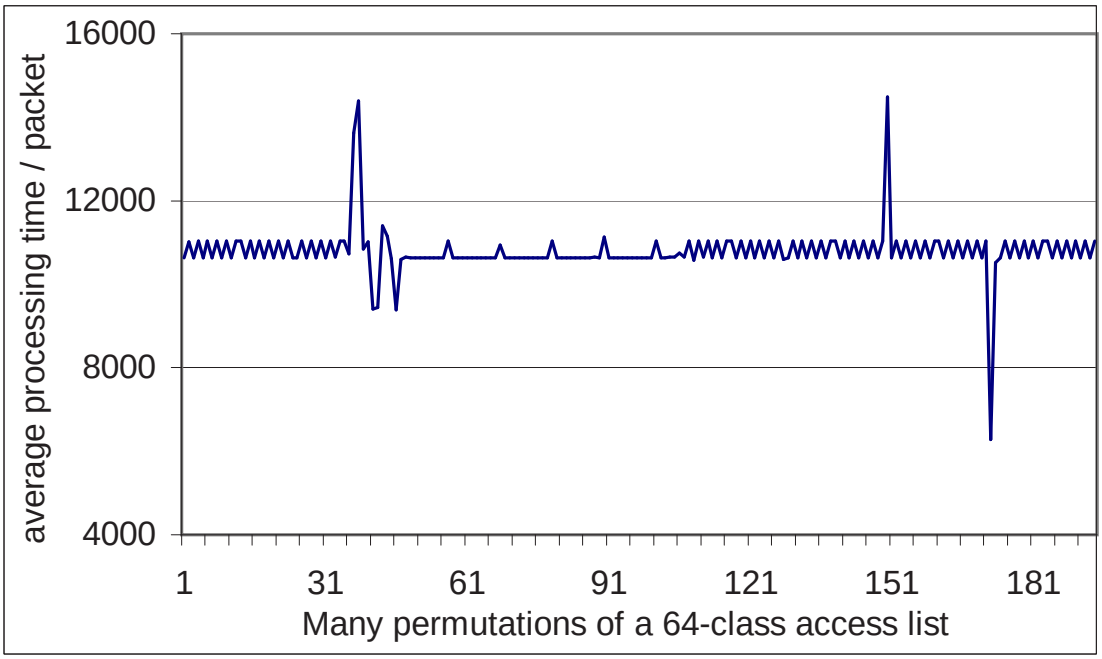

Figure A.6: Access list with 8 classes filtering an 8-class packet stream.

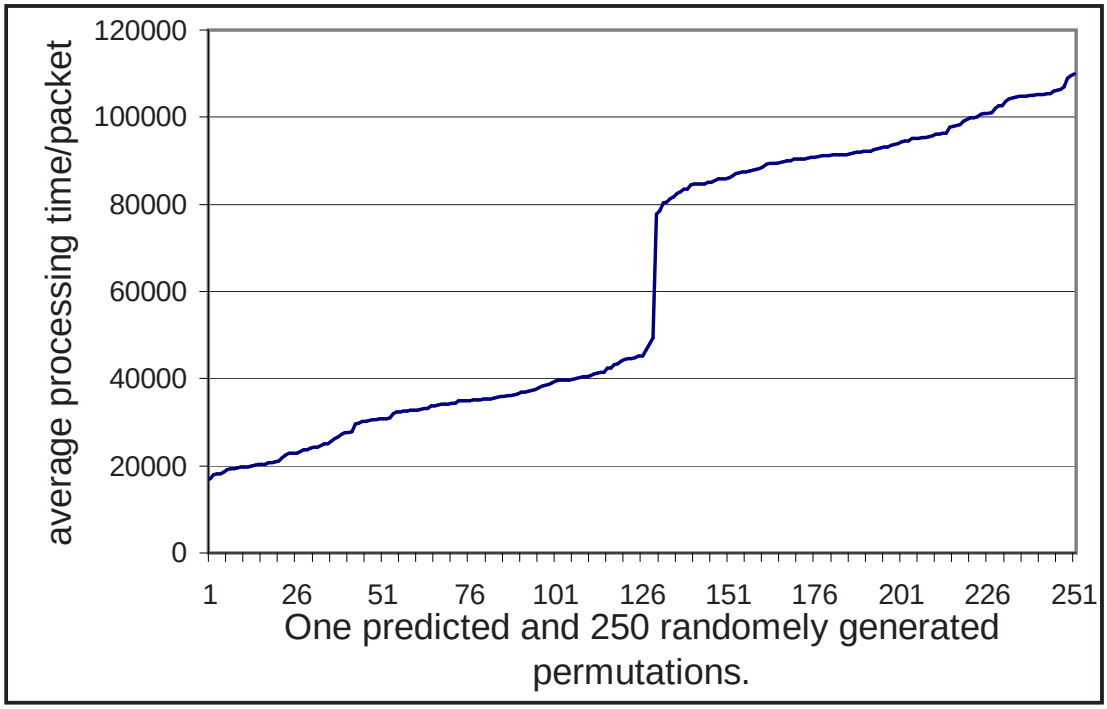

Figure A.7: Access list with 8 classes filtering an 8-class packet stream.

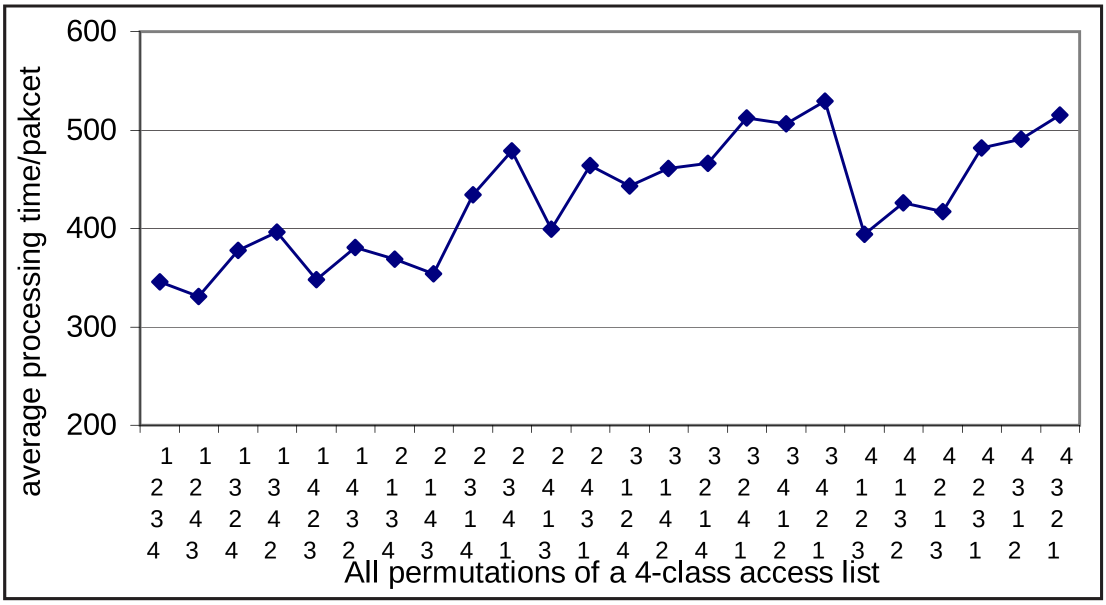

Figure A.8: Access list with 4 classes filtering an 8-class 1,000,000 packet stream.

the following number of rules each: 173, 76, 99, 11, 19, 2, 12, 23, 22, 6, 1, 2, 1, 1, 1, 1 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 1 1 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 1 1 1 1 1 1, 1, 1 and 3. The list was filtering a 64-class packet stream of 1,000 packets with an mean time between arrival of 50,000 units of time. The packets classes had the following numbers of packets: 179, 234, 22, 63, 27, 58, 285, 50, 2, 2, 18, 6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 and 3. The best access list class permutation predicted was classes: 7, 6, 11, 4, 12, 2, 24, 1, 13, 62, 44, 56, 34, 26, 25, 63, 31, 22, 40, 29, 8, 42, 50, 52, 45, 27, 59, 57, 35, 33, 5, 28, 55, 21, 60, 51, 32, 19, 61, 39, 23, 54, 47, 46, 64, 41, 36, 30, 15, 49, 48, 37, 20, 58, 43, 16, 14, 10, 53, 17, 18, 38, 3 and 9. On the chart this is at point 172 on the X-axis with a value of 6289 unit time.

Figure A.7 shows an access list of 2500 rules with 16 classes of rules made up of the following number of rules each: 593, 257, 1172, 106, 152, 65, 11, 6, 81, 18, 21, 11, 2, 1, 1 and 3. The list was filtering a 16-class packet stream of 10,000 packets with an mean time between arrival of 50,000 units of time. The packets classes had the following numbers of packets: 1591, 7041, 336, 347, 369, 21, 196, 24, 22, 3, 38, 1, 5, 2, 1 and 3. The best access list class permutation predicted was classes: 7, 2, 1, 13, 8, 4, 11, 5, 14, 16, 15, 10, 6, 3, 9 and 1. On the chart this is at point 1 on the X-axis with a value of 16,900 unit time.

Figure A.8 shows an access list of 50 rules with 4 classes of rules made up of the following number of rules: 8, 12, 20 and 10. The list was filtering a packet stream of 1000,000 packets with an mean time between arrival of 2,000 units of time. The packet stream had 4 classes of packets made up of the following number of packets: 400000, 250000, 150000 and 200000. The best access list class permutation predicted was: classes 1, 2, 4 and 3.