

PERFORMANCE ANALYSIS OF REORGANISED ACCESS LISTS IN PACKET FILTERING

Faheem Bukhatwa
*Department of Computer Science
University College Dublin
Belfield, Dublin 4, Ireland.
Email: faheemfb@gmail.com*

ABSTRACT

For many years protection of networks has been provided by packet filtering firewalls mostly using access lists. Packet filtering firewalls are efficient, fast, provide a good level of security and have withstood the test of time. But things are changing, communication is greatly expanding and the number of users continues to increase. The number of attacks from the Internet has also increased. All this places more pressure on firewalls to provide greater security at higher performance levels without becoming the bottleneck of communication. Packet filtering is also becoming popular in many other areas of communication systems. Examples of that are rate limiting, access control, resource reservation, routing, service differentiation, Virtual Private Networking (VPN), IP security gateways, load balancing, traffic shaping and Quality of Service (QoS). This enforces the need to search for better or more efficient methods of implementing packet filtering. In this paper we propose a scheme to improve performance of access list-based packet filtering. This scheme is based on two steps. The first is observing and recognising patterns of packet streams arriving into a network device, then predicting future patterns. The second is classifying and reorganizing the access list rules so that the reorganised list will give the best performance possible with the specific predicted packet pattern. We are proposing a new method to find the best organisation of an access list that has the lowest processing cost for a given pattern of arriving packets. The performance of such a scheme and the evaluation of the new reorganising method were implemented through the simulation of a network device performing packet filtering using many access list organisations to filter many different packet stream patterns. Simulation enables more experiments to be carried out under exact repeatable conditions which otherwise may not be possible. Detailed performance analysis of access lists filtering operations is outlined.

KEYWORDS

Firewalls, packet filtering, access lists, packet classification.

1. INTRODUCTION

In recent years, advances in computing and telecommunication technologies have expanded the computer systems capabilities and greatly expanded user requirements. As a consequence, users and their organisations are becoming more and more dependent on the services provided by their systems and computer networks. Data, programs and information critical to the functioning or even survival 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 the information.

Computer networks are becoming very convenient targets for attacks and illegal operations. The security of a system must be addressed to prevent, detect and correct different forms of attacks. The required level of security for networks and organisations differ but generally the principal reasons for security are

confidentiality, integrity, availability and accountability. Looking at availability in more detail, 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 of 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). Attacks against the availability of a system are called Denial of Service (DoS) attacks.

DoS 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 DoS attacks take advantage of the large number of hosts on the Internet that have poor or no security. The attack does not necessarily exploit a security hole at the target to cause a problem but it can overwhelm a host with too many requests for resources.

Firewall technology can be 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, see Figure 1. It can also be used to isolate and protect sub-networks. A firewall is a computer, router or other communication device that filters access to the protected network (Schreiner, 1998). Cheswick & Bellovin (1994) define a firewall as a collection of components or a system that is placed between two networks and possesses the following properties:

- 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.

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

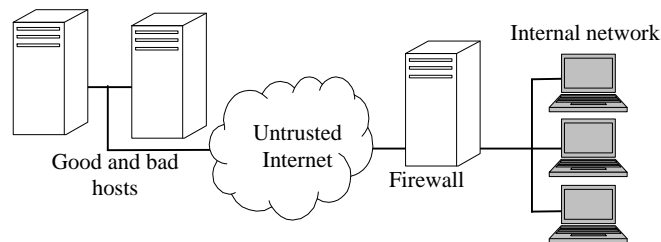


Figure 1: A typical firewall set up protecting an internal network

Application level firewalls are also known by a few other names, such as proxy firewalls. Those firewalls do not rely on 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 proxy's own IP (Internet Protocol) address. The proxy uses only its own address when communicating with the outside world. It keeps replacing the addresses of the machines it is protecting on the outgoing packets with its own address. While, on incoming packets it replaces its address with the addresses of the machines it is protecting. So addresses on machines in the network they protect are not made known outside the proxy (Habtamu, 2000). Because different types of firewalls operate at different layers, they differ in the level of security and performance speeds. Higher-level firewalls (application level) provide higher security at lower speeds, while the lower level firewalls (transport level) perform much faster but at lesser levels of security.

In packet filtering firewalls, packet filtering refers to the basic operation performed by the firewall to inspect the packet header, verifying any number of the fields in the packet header i.e. the IP address, the port or both then accepting or rejecting the packet with no changes made. 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. Habtamu (2000) stated that firewalls of this type are cheap, simple, fast, efficient and provide a good level of security. Packet filtering has proved to be efficient and effective at improving system security (Schuba & Spafford, 1997). A single access list 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 involved may still be a significant bottleneck when a lot of inspections need to be performed on a lot of packets (Ballew, 1997).

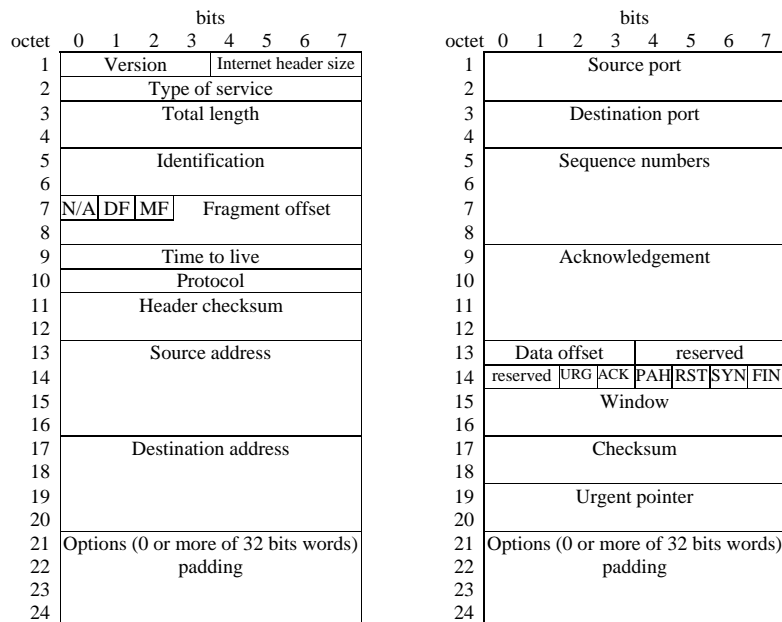


Figure 2, The Internet Protocol (IP) packet header structure (left) and the Transport Control Protocol (TCP) packet header structure (right).

TCP/IP (Transport Control Protocol/Internet Protocol) filtering is done by inspecting the Network and Transport layer packet headers depending on the protocol suite. Figure 2 shows the header fields and their width in bits for packets at two levels of the protocol stack TCP and IP. Packet filters typically manipulate (that is, accept or reject) packets based on a combination of some of the following controls:

- the packet source IP address
- the packet target IP address
- the protocol of the packet, type of transport layer (e.g. UDP, TCP etc.)
- the transport layer source port (TCP/UDP)
- the transport layer destination port (TCP/UDP)
- some flags (e.g. a TCP packet contains connection status flags).

A rule set contains a list of rules that can be looked at in the following format:

If (condition) then action where the action is either a packet *accept* or *reject*.

A real example of a rule for a Cisco router (Cheswick & 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 of IP address 121.11.127.20 is to be accepted provided the destination port address is in the range 23..27.

Addresses in the above rule are given together with their masks (0.0.0.0). Every part of the mask maps to its corresponding part of the address. When the mask is converted into binary, then for each bit of the mask, the binary value “1” is an indication to ignore during matching the corresponding bit in the address, and the binary value “0” is an indication that the corresponding address bit must match exactly. Since the masks have all zeros in the above example, then the addresses must match exactly. In the above example, if the source address was 20.9.17.8 with the mask 0.0.0.255, (the 255 would be 1111 1111 in binary) then it indicates to match the first three parts of the address only, and ignore the last part. It means that any packet with the address 20.9.17.xxx would match the condition, where xxx means any value.

The rules in a rule list are consulted one at a time to check if the condition matches the incoming packet. A matching condition will accept or reject the packet depending on the action specified in the rule. If the condition does not match the packet then the checking continues with the next rule in the list. The rules are checked in a specific order and that order is very critical. Changing the order of the rules could result in a different decision of acceptance or rejection for some packets (Hazelhurst, 2001), an example is shown later.

The security policy describes how the device will implement the network service access policy, and precisely how it will take access decisions in accordance with it. Typically, the policy is either “*permitting 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 is allowing only outgoing access to the Internet but no incoming access. Another example is allowing access only to certain selected services.

Even though the main function of firewalls through packet filtering started as a basic security implementation, packet filtering has been applied in many other areas. Packet filtering can be used as a mechanism for the implementation of a wide variety of network security policies such as to prevent unauthorized access from outside the protected network. It can also be used to prevent attacks of DoS resulting in actual denial of service to legitimate users. For Virtual Private Networks (VPN), packet filtering can ensure their privacy by only allowing private communication. A VPN is an encrypted tunnel over the Internet or another un-trusted 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 strong authentication and encryption of all traffic. Packet filtering is also used for Quality of Service (QoS) as a control to limit the use of certain network connections for a particular service (dedication). Packet filtering is used as a control to limit the use of certain network connections and therefore it makes 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. LIMITATIONS OF FIREWALLS

The Internet Protocol was not designed with security as a priority and is, as a result, inherently insecure. Firewalls are a common method of addressing these insecurities but are often regarded as the only line of defence (Haeni, 1997). Firewalls make unauthorised access to a system from the Internet far more difficult but do not eliminate these insecurities entirely, for example:

- A firewall is not effective against users with authorised access or what is known as internal attacks.
- Most firewalls are not real defence against attacks on Internet-accessible services like mail bombs, ping floods, viruses and Trojan horses etc.

- IP addresses can be faked (Bellovin, 1992) when firewalls rely on accurate IP source addresses for making filtering decisions.
- The process of configuring and testing packet filtering rules tends to be lengthy, difficult and complicated. Complexity of access lists configuring and testing consists of two parts, the first, is the difficulty of correctly specifying filters (Chapman, 1992) which correctly reflect the security policy intended. Second, reordering filtering rules makes correctly specifying filters even more difficult (Hazelhurst, 1999). The order in which rules are specified is so 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:

Consider the security policy “Accept packets from the domain D only if access is required through port P, while all other packets from domain D are refused” expressed by the following two rules and in this order:

- 1) Accept any packet from domain D on port P
- 2) 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 if the order of the rules is changed to the following:

- 1) Refuse all packets from domain D
- 2) Accept any packet from domain D on port P

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

- Scalability, with reference to having a single point (firewall) where all traffic flowing through makes it a central bandwidth bottleneck. Putting extra strain on keeping up with the growing packets processing computational cost due to increased traffic and increased sophistication of the required filtering. The fact that there are more users on the Internet and computers are getting more powerful means there is more communication consequently more packets traversing. The increased number of attacks over the years requires more security, which means more inspections through more rules at firewalls. Higher security will mean lower performance leading to the Security versus performance dilemma (Friedman, 2001). Firewalls are expected to be the bottleneck of communication systems in the future. The time cost of performing look-up on a rule list may become too high particularly for routers, where this may add significantly to latency in the network.
- Packet fragmentation complications. IP will fragment large transmission frames by breaking them into small units known as datagrams, as well as perform the re-assembly of those datagrams when the data is received at the destination (Tanenbaum, 1996). This is done so the data can be transferred over networks that support small maximum packet sizes. This increases the complication for firewalls; in one aspect it increases the number of packets inspected. Other situations like when a packet fragment is accepted and another fragment of the same packet is denied.

3. PACKET CLASSIFICATIONS

Up until recently, Internet routers provided only “best-effort” service. Nowadays, different qualities of services are requested from routers and are expected from them for different applications. Internet routers that operate as firewalls, or provide a variety of service classes, perform different operations on different packet flows. A packet flow is defined to be all the packets sharing common header characteristics; for example a flow may be defined as all the packets between two specific IP addresses. Another flow may contain all 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. Packet classification is the process of categorising packets into “flows” in an Internet router or other network devices based on a specified collection of rules. One of the many different services provided is determined for a packet depending upon what class or flow the packet is categorised into. While in firewalls, rules are mainly used to accept or deny a packet, traditionally, in a router they are used to ultimately find the IP address of the next hop where the packet needs to be routed to.

The simplest, and most well known form of packet classification is used in routing IP datagrams, where 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 a packet may belong to and associates with each class an identifier that is based on some criteria on some fields of the packet header. This identifier uniquely specifies the action associated with the rule (Gupta & McKeown, 1999).

4. PACKET FILTERING IMPLEMENTATIONS

In DoS attacks, the victim machine is bombarded with packets causing normal packets that obey the end-to-end congestion control algorithms to back off and eventually starve. Large-scale DDoS (Distributed denial of service) attack also interferes with other traffic in that part of the network, which is being heavily congested. Mahajan et al. (2001) introduced a network-based solution, called Pushback, to defend against DDoS attacks. Ioannidis & 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 congestions.

Multiple levels of firewalling can increase the level of protection or provide more protection at different layers (Habtamu, 2000). In fact, distributed firewalls based on the Network Interface Card (NIC) have been proposed supporting at least the same functions as a centralised firewall (Friedman, 2001). This can eliminate the problem of bypassing the firewall in a single point of access or centralised implementations. It also eliminates the problem of the single point failure of protection leaving the entire system exposed.

Packet classification using ad hoc mechanisms like 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 et al (1999), the associated simulation results suggest significant reduction in search space, while keeping the memory requirement 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 bridge (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. Here are a number of approaches:

- **Linear Search and Caching:** 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 modest size filter sets. Caching is a technique that aims at improving performance of linear search. The idea is that if packets from the same flow that have identical headers and corresponding classification solutions then they can be cached. However, performance is dependent upon how large the number of packets in each flow are. The higher the number of packets, the better the performance becomes. If the number of simultaneous flows becomes larger than cache size, performance degrades sharply.
- Some solutions are hardware-based. The large degree of parallelism can be implemented in hardware to gain speed advantages. Contents Addressable Memories (CAMs) can be used very effectively for filter

lookup (Lakshman & Stidialis, 1998). However, it is difficult to manufacture CAMs with wide enough words to contain all the bits in a filter. The many hardware-oriented schemes rely on heavy parallelism, and represent significant hardware constraints. Flexibility and scalability of hardware solutions is very limited (Warkhede, 2001).

- Srinivasan et al, (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 the search space on average, but does not provide any improvement in the worst case.

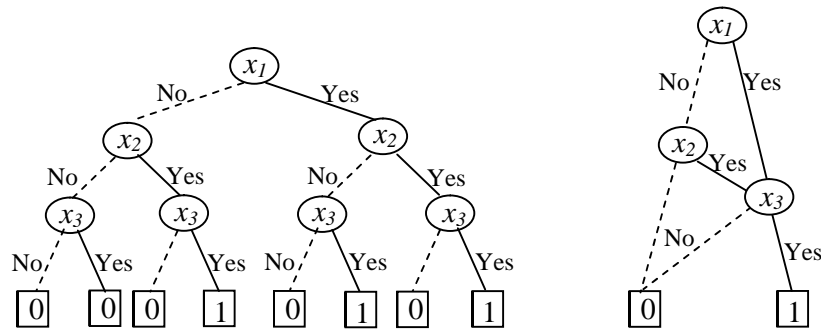


Figure 3, A simple binary decision diagram for $(x_1 \cup x_2) \cap x_3$ (left) and the reduced ordered binary decision diagram (right).

- Binary decision diagrams are another way of implementing packet filtering to solve a number of the problems. Decision diagrams are a method of representing a Boolean expression. 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 on the left in Figure 3 to that on the right. The dashed and solid lines in the diagrams indicate the branch when the decision variable is 0 (No) or 1 (Yes).

Algorithms for converting rule sets into Boolean formulae have been produced and tested. Binary decision diagrams produced encouraging results (Hazelhurst, 1999). They 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 produced diagram, the effect on the overall lookup performance and on memory requirements.

5. IDEA BEHIND 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 lookup latency linear in the number of rules in the access list. The ultimate aim is to reduce to a minimum the processing time needed for each packet to be inspected, with a reasonable cost in memory requirements.

Let us define the ‘critical rule’ for a packet to be ‘that rule in an access list that will identify the packet and causes it to be accepted or refused’. Until that rule is found, which may not, there is a processing time cost added every time the packet is inspected by an individual rule. From a performance point of view, it is desirable that each packet meets that critical rule at or near the start of the list. This would reduce the time

required for 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 be very high and performance will degrade. The idea in our proposed concept is to have most packets meet their determining rule at or near the start of the list of rules.

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 are intended for inspecting and filtering packets based only on the source address or only on the target address or on both. Also, rules can be intended for checking and filtering based on the port number in the packet or the protocol used. So, it is proposed for the purpose of this work to first classify rules in an access list into different classes and secondly, rearrange the rules in the list in such a way that rules of a particular class are placed on top of the list. Test runs are then carried out to test the performance when rules of each class are placed on top of the list. Those runs are executed using the same set of packets arriving into the system.

Considering this simple approach then a number of assumptions can be made here. Consider that rules are ordered in a particular way such that all rules of a certain class, let us say class “X” are made to be located at the top of the access list. When packets start arriving, and if all arriving packets or most of them require the critical rule for acceptance or refusal to be of class “X”, then our ordering of the rules list was a successful choice and performance is expected to improve, see the right side of Figure 4. The reason is that all or most packets need only a few rules to be checked at the start of the list before a deciding rule for acceptance or refusal is reached. While, if the list was ordered in such a way that “X” class rules are placed at the end of the list, see the left-hand side of Figure 4; then for each packet arriving, most of the rules in the list have to be checked before eventually reaching the needed “X” class rule at the end of the list.

Each rule used to inspect a packet has a time cost attached to it, and the more rules the packet has to pass through the more time is wasted. The ultimate aim here is to reduce this time required for processing each packet.

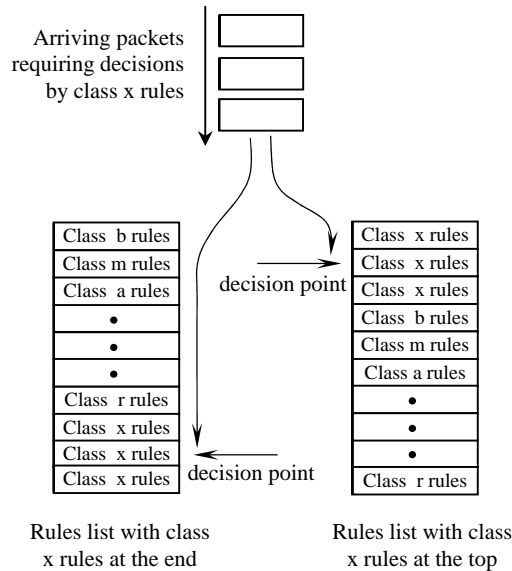


Figure 4, Effects of class arrangement of the rules in a list on processing time of arriving packets.

An assumption was made here that it was known in advance for each arriving packet what class of a filtering rule it required to determine acceptance or refusal of the packet. If every time a packet arrives it is

possible to recognize the class of rules it requires, and, if the list of rules then can be rearranged according to the packet requirement, then packets will meet their critical rules in the list early at the start of the list. There are two problems with this approach. The first is that to recognize the class of a rule for a packet it is necessary for the packet to go through the list. The second is that this approach of rearranging or reordering the list of rules based on individual packet requirement is neither practical nor possible from a performance point of view.

We are suggesting that if a profile of arriving packets can be developed from past observations for a network device, then it will be possible to make a prediction about the expected pattern of these packets in the future. Profile development is possible and can easily be established (Gupta & McKewon, 2001). Classification of packets arriving in the past can help to determine expected classes and numbers of packets arriving in the future. On the other hand, real time pattern recognition can be done based on classifying arriving packets. Packet patterns can periodically be inspected to determine the most effective order of the rules list.

Another assumption being made here is that access lists can relatively easily be rearranged or reordered to facilitate better performance for a particular pattern of arriving packets. That is not quite the case, as changing the order of the rules within the list can in fact change the acceptance or refusal of a packet (Hazelhurst, 1999).

Reordering access rules can be a difficult and dangerous operation. The real difficulty is in ensuring that the list is and remains to be 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 with the security policy. Part of the problem is the time constraints on the actual reordering; another part of the problem is verifying that the new reordered list reacts the same as the original to any arriving packet. This problem concerning the effects of ordering the list on the security policy is not the subject for discussion in this paper. It is however worth mentioning here that rearranging the classes of rules are what is needed here and not the full reordering the list or rules. Even though rearranging can be seen as some form of reordering, but all the rules belonging to a single class are moved up or down through the list in their same order of appearance in the original list. Though eliminating many of the problems caused by changing the order of rules. It is obvious that the reordering of unrelated rules do not have as many problems from a security point of view. Consider the following 4 rules belonging to two classes, one class containing rules inspecting source addresses and the other class containing rules inspecting port numbers. If the following two packets are inspected by these rules in this order, both packets are accepted. Packet 1 has the source address 98.6.22.17, while packet two is requesting a service on port number 50.

- 1) Accept if packet has the source address 98.6.22. (class 1 rule)
- 2) Accept if port number requested is in range 40..60 (class 2 rule)
- 3) Refuse if packet has the source address 98.6. . (class 1 rule)
- 4) Refuse if port number requested is in range 1..90 (class 2 rule)

If the rules are rearranged according to their class without the change of order of the rules belonging to the same class. The access list will appear in the following order, and will still accept both packets 1 and 2:

- 1) Accept if packet has the source address 98.6.22. (class 1 rule)
- 3) Refuse if packet has the source address 98.6. . (class 1 rule)
- 2) Accept if port number requested is in range 40..60 (class 2 rule)
- 4) Refuse if port number requested is in range 1..90 (class 2 rule)

If the rules within each class are ordered (sorted) in such a way that the order or appearance of rules (in similar class) becomes different to their order of appearance in the original list then this can have serious effects. Both packets 1 and 2 will now be refused access when inspected by the list after changing the order of the rules within each class:

- 1) Refuse if packet has the source address 98.6. . (class 1 rule)

- 2) Accept if packet has the source address 98.6.22. (class 1 rule)
- 3) Refuse if port number requested is in range 1..90 (class 2 rule)
- 4) Accept if port number requested is in range 40..60 (class 2 rule)

The point to be stressed here is that the proposed rearranging of the list has less negative consequences on security than the full reordering of the rules in the list. More investigation will still be needed when considering reordering as proposed here. Though, one way of eliminating the time constraints problem of rearranging and verifying, is to have few versions of the access list pre-arranged and pre-verified, an almost similar approach was suggested by Hazelhurst (2001). So, in real time neither actual arranging nor verification takes place; only a selection of the most suitable arrangement of access list is made.

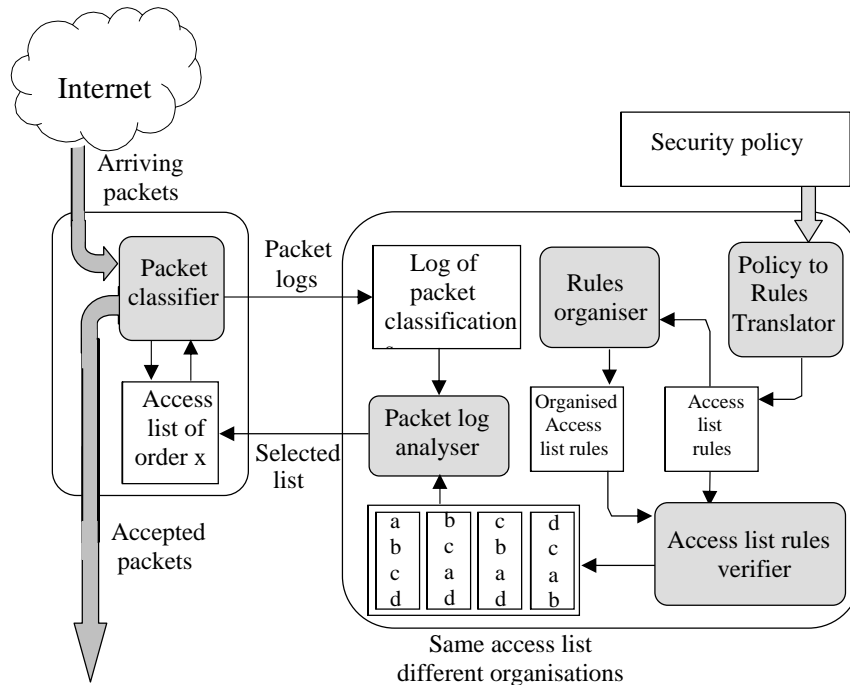


Figure 5, The proposed reorganised access list packet filtering system.

The proposed system consists mainly of two parts. One part that performs the actual packet classification and the second part maintains logs of classifications of arriving packets, performs log analyses and determines the best access list arrangement to be used by the classifier. Figure 5 provides a block diagram of the proposed 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 organisations. The access list rules verifier compares two access lists and verifies that they implement the same security. Basically, it verifies that a particular change in organisation does not alter the security functionality of a particular access list. For future use by the packet classifier, accepted lists made up of the same rules but in different organisations are saved by the verifier. The packet log analyser is 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. 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 placing the rules of a specific class on top of the list will actually produce faster processing time. The rest of this paper 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.

6. MODELLING THE ACCESS LIST PACKET FILTERING

The operation of a network device receiving streams of packets and performing packet filtering is simulated to allow for more rigorous testing and better comparisons. The simulation consisted of three main integrated models; namely, the access list, the packet streams and the filtering models. One other part was specifically for capturing and analysing the results of the simulation. Figure 6 shows a general outline of the models in the simulation system and how they relate.

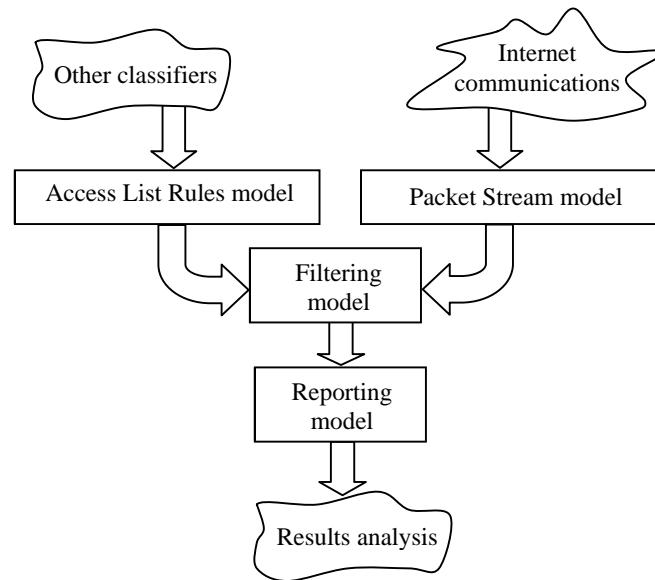


Figure 6, Modelling the access list filtering simulation system.

The models are mostly formulated based on detailed surveys of studies conducted by Gupta & McKewon (1999). These details were found 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 through out the simulation:

- The mean number of rules = 50 rules.
- The maximum number of rules = 2734.
- Only 0.7% of the access lists (classifiers) 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 a classifier's behaviour.
- 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 by eight.

The survey showed that rules were generally divided into 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.

6.1 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 (or percentages of rules) in each class.
 - d) Average processing time for rules in each class.
3. Access list rules organisation.
4. Rules definition specifications.

The following is a discussion in some details of each of the above points in the access list model.

6.1.1 Rules classification

Classifying the rules is best 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, 1 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. While 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 get 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. This would be a good base to start our classification of rules. It is worth pointing out that this 3-field rules that makes up 23% or 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.

6.1.2 Access list specifications

a) Total number of rules: Based on the survey results, 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%) are over 1000 rules. If the number of rules was 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 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 going to be considered. It is suggested that the size of 2500 rules should be included. Therefore, list sizes to be included will vary between 20 and 2500. Other values in between reflect the average or the expected increase in average number of rules to be somewhere between 50 and 100 rules. Another value somewhere between 500 and 1000 which reflects the large list. Looking at those few numbers, starting with 20, multiples of 5 will provide close enough values to test and will be easy to program. Making the values to be tested as follows: 20, 100, 500 and 2500 rules in access lists.

b) 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 calculated as the sum of all the unique combinations of all the fields, which can be calculated as: $2^n - 1$ where n is the available number of the fields to select from. We shall opt for the following formula, as it enables getting sums of all combinations from available selected particular number of fields. In other words, it allows calculating all possible combinations from say 4 fields. That is how many combinations of 1 field, 2 fields, 3 fields and 4 fields can be obtained from a total of 4 fields. In this example, the combinations would be : 9 combinations (1, 2, 3, 4, 12, 13, 14, 23, 24,34, 123, 124, 134, 234, 1234), see Table 1. Also notice that it is not a permutation needed here, it is a combination, for the selection of fields 1 and 2, (12) is the same as (21). The formula is expressed as follows:

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

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:

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

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 1 shows the maximum number of possible combinations, which can be obtained from (n) number of available fields for different combinations of these fields (k). 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 some class may contain a number of rules, that will leave many other classes with a zero number of 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 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. We are 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, it is possible to imply that most access lists on the web have a number of rules of or around 50; being the average according to the survey, in fact had rules classes of between 2 and 13. Most likely some number between those two; averaging round 7 or 8 classes.

The survey also indicates 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 rule analysed. It is unlikely that 8 fields are used in many rules. In fact, even if 8 fields were used, only a few of the combinations of eight fields would be present. We are going to go as far as including the combination of 7 fields within a rule. Looking at Table 1, we find that the maximum number of combinations for the 7 fields is 77. Taking the following considerations into account:

1. that the average number of rules in an access list on the web has on average 7 or 8 classes.
2. that the maximum number of classes in an access list is to be no more than $\frac{1}{4}$ of the total number of rules in the list.
3. that too many classes in an access list defeats the purpose of classification.
4. that too few classes in an access list (as in 1 class) eliminates advantages of classification.

Based on that, it is worth considering, the values for classes found to be adequate to perform the simulation will meet the following conditions:

- 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 column in Table 1.
- To include only those values which are less than $\frac{1}{4}$ of the total number of rules in a list.

Table 1. Maximum number of combinations of fields

<i>n</i> number of fields available	<i>k</i> (fields combined in a rule)								Max combinations	
	1	2	3	4	5	6	7	8	Any number of 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

c) 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 $\frac{1}{4}$ 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 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 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 can be defined as follows:

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

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

In this case, the number of rules to be assigned to the class 1 will be a random number between 1 and 97. According to Table 2, this random number is 66. For class 2, the number of rules available to distribute will be $(100 - 66) 44$ rules. Therefore, class 2 can have a maximum value of 42. So the number of rules assigned can be a value generated randomly between 1 and 42. In the example in Table 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 rules for 4 classes.

Table 2. Distribution of rules between classes

Class no.	Rules to distribute r	Classes left c	Max rules to 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

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 none of the remaining classes are left without rules. Algorithm 1 shows how values were assigned to classes for any number of classes. The different number 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])

```

C = total number of classes
Classes_left = C
Class_array[C]
Remaining_rules = total_rules to distribute; // total number of rules to distribute

For (I=1 to C) {
  if (I == C) // the last class to be assigned a number? assign all remaining rules.
    Class_array[I] = Remaining_rules;
  Else {
    *** if there are as many classes as there are rules to distribute
    *** then every class will have one rule
    if (remaining_rules == classes_left) {
      Class_array[I] = 1;
      remaining_rules--;
      classes_left--;
    }
    else {
      r = generate a random number less than (remaining_rules - classes_left - 1)
      ** this ensures that at least one rule will be left for each class remaining
      Class_array[I] = r;
      Remaining_rules = remaining_rules - r;
      Classes_left--;
    }
  }
}

```

Algorithm 1, Distributing the number of rules into the classes.

d) Average processing time for rules in each class:

When a packet is being inspected, 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 processing time for other rules. For example the values 100 and 200 assigned as processing time for rules A and B will indicate that rule B will take twice as much processing time as that of rule A.

Let us initially assume that we can state that processing time taken by a rule to inspect a packet depends on the fields that need to be inspected. Then we can also state 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, we will see why not in a moment.

The actual value assigned as 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 was not an accurate reflection of the actual processing time, it will still provide a good way to compare performance of the same rules when organised differently, 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 of 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 cause in the first to check one part of the address while in the second to check three parts of it. This leads to slightly different processing time when different packets are inspected, according to Hazelhurst (1999) in his binary diagrams study of access lists. It is clear that processing time 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 class rules will be based on a value for that particular class with a smaller random value added to allow for differences.

Processing time will be assigned to each class of rules. For each particular rule within each class the same assigned value will be used with a plus or minus randomly generated value. For example, suppose the processing time for class $x = 100$, 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 resume 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).

6.1.3 Access list rules organisation

It is obvious that the location of rules within the list will have an effect on the total processing time for each packet 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 there 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 flowing 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 every 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. The organisation is such that the entire rules belonging to one class are moved to the top or bottom of 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 1, 2, 3 and 4 placed on top in each list respectively. Then 4 more versions will be created with classes 1, 2, 3 and 4 placed at the bottom of each list. That is a total of 8 versions of the same list; plus the original randomly organised list. 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. There can be 10 or more original randomly organised; different 16-class access lists to start with, with 33 versions made for each. More specific organisations were needed and were produced individually as the research progressed.

6.1.4 Rules definition specifications

The actual rules that are used for the simulation are specified through three parameters per rule. Every rule in the list is defined by the following:

- **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 percentage 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.
- **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. Every rule, once assigned a class, then depending on that class, the rule processing time is calculated. This allows rules from similar class to have slightly different processing time.
- **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 the rules that will be assigned the value "accept". Based on that value, some of the rules created in that particular access list are assigned with an "accept" decision and the rest are assigned with a "deny".

The values “accept” and “deny” are represented by the values 1 and 0 consecutively. In fact, for the purpose of this simulation, these values to accept or deny will have no consequence on performance and of no further value for this simulation. Once, the packet is accepted or denied, processing is finished. This field is maintained because it formed part of other related experiments in inspecting the number of accepted and denied packets.

The actual rules’ definitions of a list are kept in a file 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. The structure of the file, the details and the fields, as well as some example values are shown in Table 3.

Table 3, The structure of the file containing the randomly generated rules of an access list.

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	% of rules	Processing time	% of accept
<ul style="list-style-type: none"> Percentage of the rules in this class of the total number of rules Mean processing time for rules in this class Percentage of accept to deny decision in rules in this class 	1	24%	120	99%
	2	59%	100	19%
	3	16%	80	87%
	4	1%	70	55%
Class organisation: (0 to indicate random or undefined)	Rules of class 2 are on top of the list			
<ul style="list-style-type: none"> What class is placed at top (first) place in list What class is placed at second place in the list What class is at the place before last (second-last place) in list What class is placed at bottom (last) place in list 	2			
	0			
	0			
	0			
Specifications of each rule in the list: (the following three fields are repeated for each rule in the list)	Accept or deny	Rule Class	Rule Processing time	
<ul style="list-style-type: none"> Accept or deny decision. (1= accept, 0=deny) The class this rule belongs to. (1 to max number of classes in list) Processing time. (a random number based on [mean processing time of rules in the class (discussed above)]) 	1	2	76	
	1	2	91	
	0	2	118	
	1	3	63	

6.2 Packet stream model

The packet streams 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 shall represent 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 (or percentages of packets) in each class.
 - d) Mean time between arrivals.
3. Organisation of packets within a packet stream.
 4. Packet definition specifications.

6.2.1 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 entirely 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 1 access list rule will also be called class 1 packet, class 2 packet by a class 2 rule and so on.

6.2.2 Packet stream specifications

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

a) 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 1000, 10000 and 1000000 packets are thought to be sufficient to reflect the performance of access lists.

b) Number of packet classes within 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.

c) Number of packets (or percentage 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 must have different percentages of the total number of packets in a particular stream. 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 percentages of these classes. Percentages 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 of utmost importance 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 3 packets, then it is possible to perform specific tests by using access list with for example, class 3 rules at the top or bottom of the list, or access lists with higher or lower percentages of class 3 rules.

d) 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, but 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 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 concept 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 by 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 displayed. 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.

6.2.3 Organisation of packets within a packet stream

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

6.2.4 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 are defined by the following:

a) 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.

b) Packet class:

Every packet is assigned a class represented by a number between 1 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 percentage values of packets in each class. Both those two values are part of the packet stream definition and are both accepted when the packet stream is being generated.

c) 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. This value has no effect on whether the packet gets accepted or not, that is a function for the access list. Fixing this for a packet 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 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. This allows for situations of packets of a known class, but yet fail to be identified by the rules in the access list.

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. The file header provides details about the entire packet stream like the total number of packets, mean time between arrival of packets and number of classes in the packet stream. The structure of the file, the details and the fields, as well as some example values are shown in Table 4.

7. EVALUATION

To perform the simulation a number of access lists of varying sizes and mixtures of rule classes were generated. For each of the lists a number of copies were regenerated and reorganised differently based on the classes of rules. In total, over 640 different access lists were produced and saved. Also a number of packet streams were generated, each stream resembling a different pattern of packets arriving into a network device, over 234 packet patterns were generated. The different number of packets, classes, rate of arrival and different percentages of packets in each stream characterise the packet patterns.

One important identifier in packet streams turned out to be the percentages of packets in each class. These numbers gave clear indication to the orientation of the packet stream. For example, a higher number (or percentage) of packets of class 3 allows us to say that the stream is mostly a class-3 stream. This induces the hypothesis that this type of packet stream will generally perform better against access lists that are organised in such a way that class-3 rules are positioned at the start of the list. In order to test the correctness of such hypotheses, thousands of simulations were executed using the pre-generated access lists and packet streams. During the results analysis, concentration was on performance. For example, a class-3 packet stream filtered through the same access lists with its class-3 rules on top, and the same access list in other organisations. Also, the performance of a class-3 organised access list against the same similar packet streams that differ only in the percentage of their rules, that is to say class-2 and class-1 as well as class-3 oriented packet streams.

Table 4, The structure of the file containing the definitions of a packet stream and the packets within.

The field explanation	Example values		
The total number of packets in the packet stream	10,000		
The mean time between arrivals of packets	9,000		
The number of packet classes in the packet stream	4		
Details about every class in the packet stream (This field is repeated for each class)	% of class rules		
• Percentage of the packets in this class of total number of packets	Class 1 packets	38%	
	Class 2 packets	16%	
	Class 3 packets	45%	
	Class 4 packets	01%	
Specifications of each packet in the packet stream: (the next three fields are repeated for each packet in the stream)	TSLPA	Packet class	% rules to be checked

<ul style="list-style-type: none"> • Time since last packet arrival • The packet class (1 to maximum classes in this packet stream) • Percentage of the rules in the access list to be checked, before reaching the critical rule. 	13203	2	76 %
	2479	3	12 %
	10574	3	103 %
	6000	1	25 %

For the same access list, all copies of its individually organised rules were tested with each of the packet patterns. In other words, for most possible patterns of packet streams, all different organisations of the same access list were simulated. Over 150,000 simulations were executed and their results were saved and analysed. 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 per packet. Obviously, the desirable value is the lower average processing time per packet.

All other variables and conditions that may influence the results were kept unchanged in all simulations where appropriate. For the same packet pattern such variables are the total number of packets, frequency of packets arriving (mean time between arrival) and number of packet classes. While for the access lists, the same total number of rules, number of classes of rules and the same number of rules for each class, also the processing time for same class of rules. The unit of time itself that is most suitable to use was the μsec (1/1000,000 of a second) as is used in other similar experiments. But, in this simulation, because the figures are only relevant to this experiment and the comparison is made between the same variable namely the average processing time per packet, a generic unit of time sufficed. It reduces complications of differing real life data observed from different source due to different machines or line speeds. It would be more accurate for us to make a statement such that *processing using method A takes 60% of the time taken using method B*. This will give a definitive indication that method A has faster performance regardless of the machine speed.

7.1 RESULTS

Figure 7 shows a typical result observed, this particular one involved the same access list which has 100 rules in two classes, class 1 and class 2, with class 2 making up 56% of all the rules in the list. Simulations were conducted with class 2 rules once on top and once at the bottom of the access list. The two access lists were used in the simulation to filter the same number of packet streams. All packet streams are similar in the number of classes and all consist of mostly class 2 rules. They differ only in the percentage of class 2 packets within the stream as shown on the x-axis. The results indicate that when the packet stream is highly class 2 oriented, performance improves when the access list has the class 2 on top of the list. This improvement is indicated by the lower average processing time.



Figure 7, Performance results of the simulation for a 2-class access.

Notice that at lower percentages of class-2 packets in the packet streams (less than 33%), the performance degrades. That is because at those lower percentages, there are more class-1 packets in the packet streams than the class-2 packets, and the stream is no longer a class-2 oriented. In fact, the improved performance was only expected at percentages of 50% or higher of class-2 oriented streams, but the improvement was clearly noticeable even when class-2 packets are as low as 33% in the stream. Notice that these two curves are the two extreme cases where all class-2 rules are either at the top or the bottom of the access list. The unorganised list where the rules are randomly located in the list, produced a curve located between those two curves.

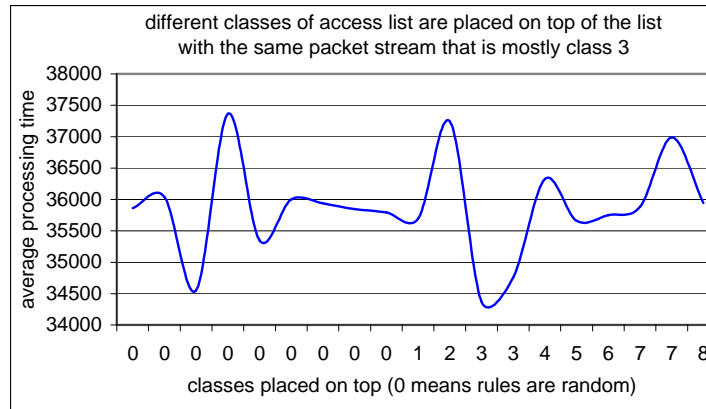


Figure 8, Simulations of the same packet stream of 10000 packets, 8 classes filtered by the same access list of 500 rules with 8 classes in different access list organisations.

7.2.1 Best class of rules on top

Figure 8 shows performance of a number of simulations for the same incoming packet stream having 10000 packets and 8 classes being filtered by different organisations of the same access list. The packet

stream is mostly class-3 packet. The access list has 500 rules and 8 classes of rules, it also has more class-3 rules. 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 8. A number of simulations were also performed with total randomness of the access list rules, indicated by “0” on the X-axis. It is clear that the lowest average processing time per packet was achieved when rule class 3 was placed on top of the access list. Remember the packet streams tested is a mostly rule-3 packet stream. Two versions of the access list with class-3 rules on top of the list produced some of the best results, but one organization was better than the other (see Figure 8). Another random organization, which happened to have rule-3 on top, produced very good results.

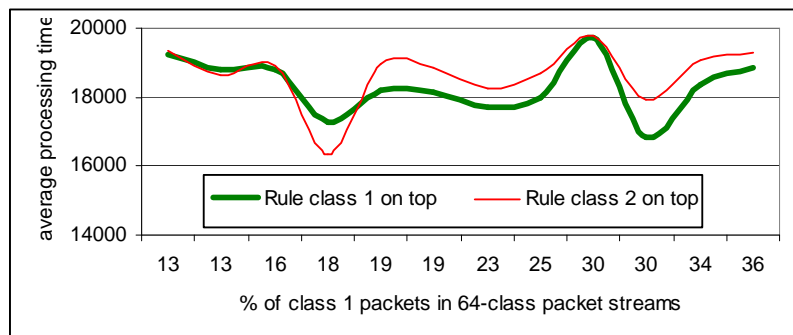
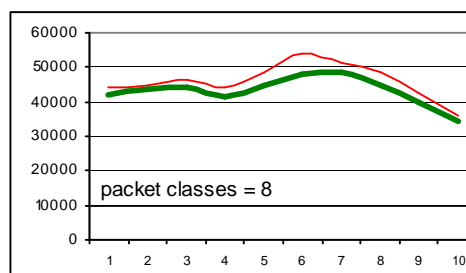
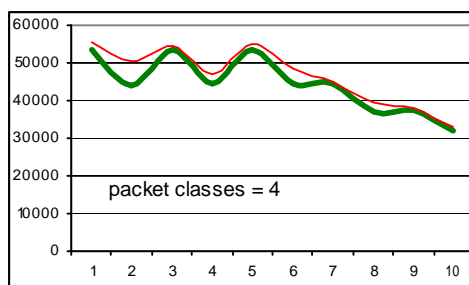


Figure 9, packet streams of 10000 packets, 64 classes filtered by a 100 rules with 2 classes access list in two organisations.

Figure 9 represents a bad case of 64-class packet streams each consisting of 10000 packets being filtered by a 2-class access list. This is a case where the access list does not know about packets belonging to 62 other classes. In such cases most of the packets will go through the entire list. This case was intentionally selected as an extreme case of a badly predicted packet stream and yet, even with an unsuitable access list some slight improvement was still possible. Improved performance is where the packet stream is mostly a class-1 stream; that is at the higher percentages of class-1 within the stream. These results emphasize the point that even if the pattern of arriving packets in real life in a network device fluctuates from the expected, better performance can sometimes be obtained from an organised list even if it is not suitably organised to match the packet pattern. That is provided the packet stream of arriving packets does not go in a complete pattern reversal for the entire period of operation. In fact, such changes of patterns should be the cue for the change of the access list used for the more suitable organisation. A pattern analyzer program can make the selection of the best organisation of an access list for the new pattern.



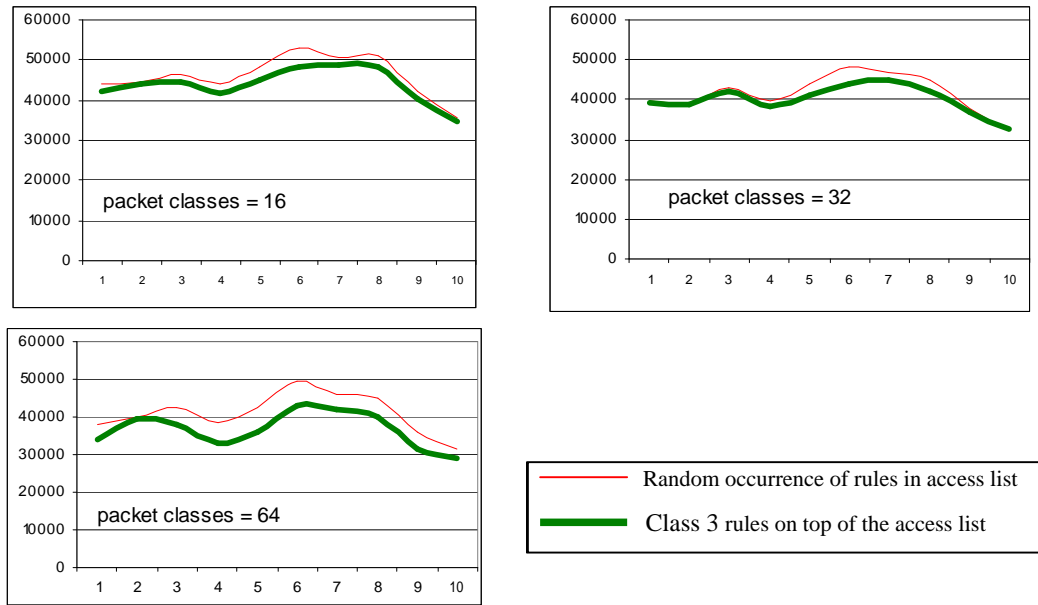


Figure 10, packet streams of 10000 packets mostly class-3 in different classes (4, 8, 16, 32 and 64), filtered by a 500 rules access list with 8 classes in two organisations, random and class 3 on top.

Many results observed still did not quite agree with the expected outcome. More simulations need to be carried out to confirm if placing the rules of a particular class at the top of an access list can improve performance regardless of the number of classes present in the packet stream. A number of simulations were executed which involved a number of packet streams that have similar configuration but differ in the number of classes. The packet streams used had 10000 packets and they all had more class-3 packets than any other class but each had different numbers of classes, between 4 and 64. The access list had 500 rules in 8 classes with class-3 rules located at the top of the access list. Figure 10 shows the performance for each of the cases. It shows that some improvement occurs in all cases where class-3 in the access list is placed on top of the list.

7.2.2 Reorganising all the classes of an access list

Many of the results obtained did not support the assumed hypothesis. That is 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 performance? If the second most common class of packets in a packet stream can be identified, then in the access list we can place the rules of the same class 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 2,3,1 and 4 and in this order. Then the access list rules are arranged according to their class to match the same order of the packets, 2,3,1 and 4. Figures 11 shows the performance obtained by filtering many packet streams that are either mostly class-1 (left) or mostly class-2 (right). Filtering is performed by the same access list in different organisations. When the lowest processing time points were inspected, they turned out to belong to access lists whose classes were arranged in the same organisation of those classes of the packet stream being filtered.

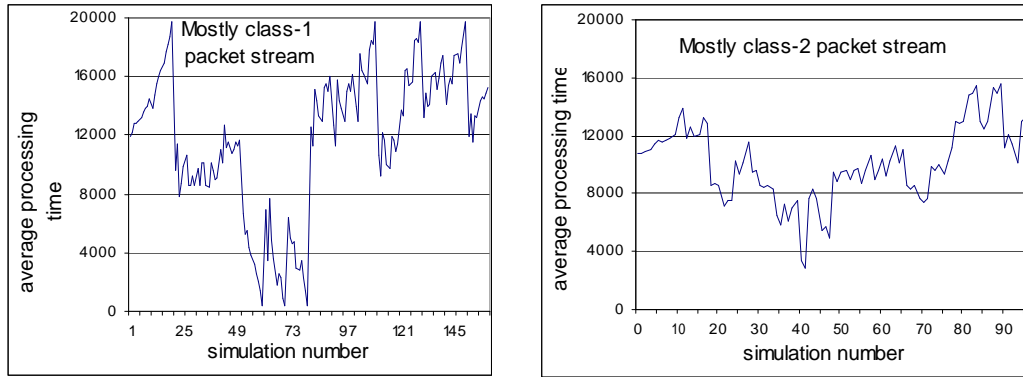


Figure 11, many packet streams mostly class-1 (left) and mostly class-2 (right), filtered by different organisations of the same assess list.

Figure 12 shows performance obtained by an access list with its rules classes arranged in the order 2,3,1 and 4. The packet streams all had more packets of class 2. But, the lowest processing time was obtained by filtering the packet stream with the classes percentages being: 16, 58, 21 and 5% for classes 1, 2, 3 and 4. Following closely behind is the stream with the class percentages: 17, 51, 19 and 13%. This 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 of each class in the packet stream.

7.2.3 Classes' processing cost weight

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. 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 will require 1 microsecond each, will have the same processing time if it was being filtered by 10 rules having 1/2 microsecond each. Therefore, the processing time of a packet depends on the number of rules as well as the processing time of each rule.

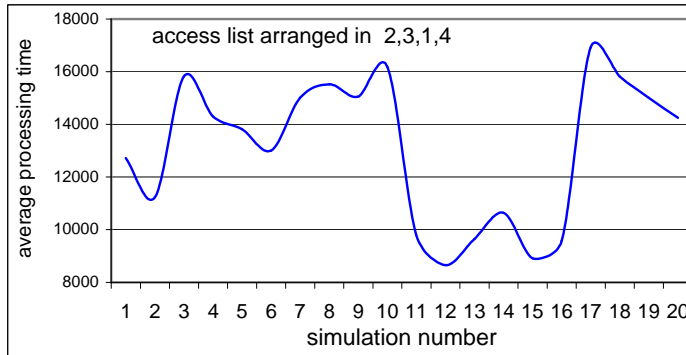


Figure 12, lowest average processing times are for exact matching of classes between access lists and packet stream.

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) 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: $C = m \cdot r$. If the packet has to pass through more rules belonging to different classes s then the total cost can be expressed as:

$$c_{total} = \sum_{s=1}^n m_s \cdot r_s \quad \text{where } n \text{ is the number of classes}$$

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 can be 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 processing time for the class, which will accept or refuse the packet, will be presented as:

$$C = \frac{1}{2} m \cdot r$$

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, 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 ($m_s \cdot r_s$) for each of the classes A and B. The left-hand side of Figure 13 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. Based on what we have seen previously, to obtain the better performance it is best 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 a rules' class similar to the packet's class processes the packet. 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 13, the total processing cost is less when class A rules are placed at the bottom of the list ($C=1200$).

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. Assume we have 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 one rule. The maximum number of permutations for the 3 access list rules' classes would be ($3! = 3 \cdot 2 \cdot 1 = 6$) different possibilities. The best permutation would be the one that would produce the lowest processing time (or least cost) with a given packet stream. The cost for each permutation would be the total cost for each class of packets passing through the list, as was seen in Figure 13. The cost can be calculated as follows with reference to Figure 14:

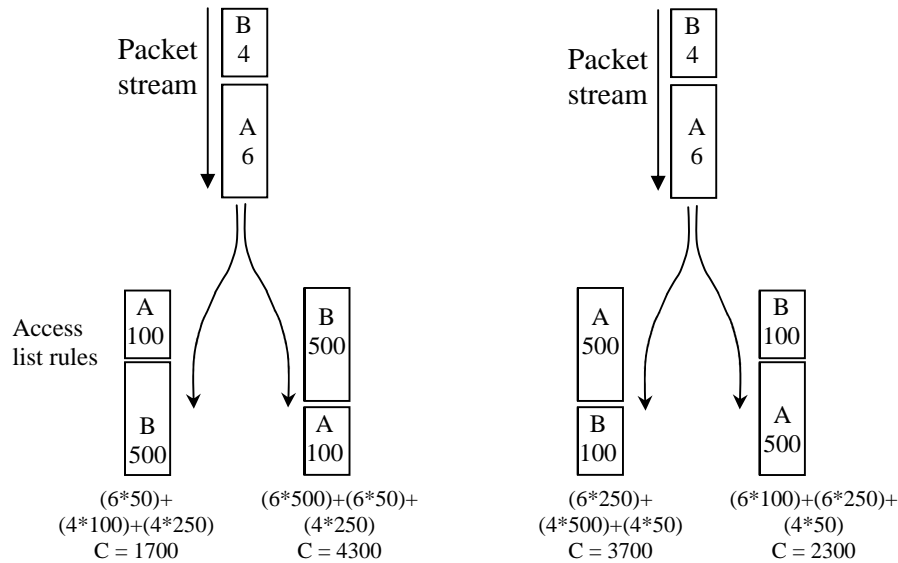


Figure 13, the same packet stream arriving into two different access list (the left and the right), each is organised different.

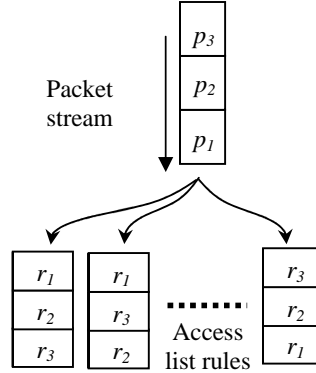


Figure 14, Calculation of cost for all different permutations.

$$\begin{aligned}
 C(r_1, r_2, r_3) &= p_1 \frac{1}{2} r_1 + p_2 r_1 + p_2 \frac{1}{2} r_2 + p_3 r_1 + p_3 r_2 + p_3 \frac{1}{2} r_3 \\
 C(r_1, r_3, r_2) &= p_1 \frac{1}{2} r_1 + p_2 r_1 + p_2 r_3 + p_2 \frac{1}{2} r_2 + p_3 r_1 + p_3 \frac{1}{2} r_3 \\
 C(r_2, r_1, r_3) &= p_1 r_2 + p_1 \frac{1}{2} r_1 + p_2 \frac{1}{2} r_2 + p_3 r_2 + p_3 r_1 + p_3 \frac{1}{2} r_3 \\
 C(r_2, r_3, r_1) &= p_1 r_2 + p_1 r_3 + p_1 \frac{1}{2} r_1 + p_2 \frac{1}{2} r_2 + p_3 r_2 + p_3 \frac{1}{2} r_3 \\
 C(r_3, r_1, r_2) &= p_1 r_3 + p_1 \frac{1}{2} r_1 + p_2 r_3 + p_2 r_1 + p_2 \frac{1}{2} r_2 + p_3 \frac{1}{2} r_3 \\
 C(r_3, r_2, r_1) &= p_1 r_3 + p_1 r_2 + p_1 \frac{1}{2} r_1 + p_2 r_3 + p_2 \frac{1}{2} r_2 + p_3 \frac{1}{2} r_3
 \end{aligned}$$

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: $p_1 \frac{1}{2} r_1$, $p_2 \frac{1}{2} r_2$, $p_3 \frac{1}{2} r_3$. As each value will be taken from every formula, they 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:

$$\text{The time cost for a permutation } t \text{ of rules classes} = \sum_{j=1}^{n-1} \left(r_{tj} \cdot \sum_{z=j+1}^n p_z \right)$$

Where n is the number of classes.
 r_{tj} is the time cost for rule class in position j in the combination t
 p_z is the number of packets of class z

Figure 14 shows the results of a simulation for a packet stream containing 10000 packets with 8 classes, class 1 making up most of the packets (91%), class 2 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 1 to 8) having the following percentages and in the same order: 23%, 48%, 12%, 10%, 3%, 2%, 1% and 1%. Figure 14 shows the performance curve of the access list with the described packet stream in many different organisations of the access list classes. Execution run 10 (on the x-axis) represents the organisation of the classes when class 1

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 1 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 percentages of packet's classes in the packet stream. The last point on the curve represent the best performance obtained by the organisation based on the calculation for the lowest processing time with the method described above.

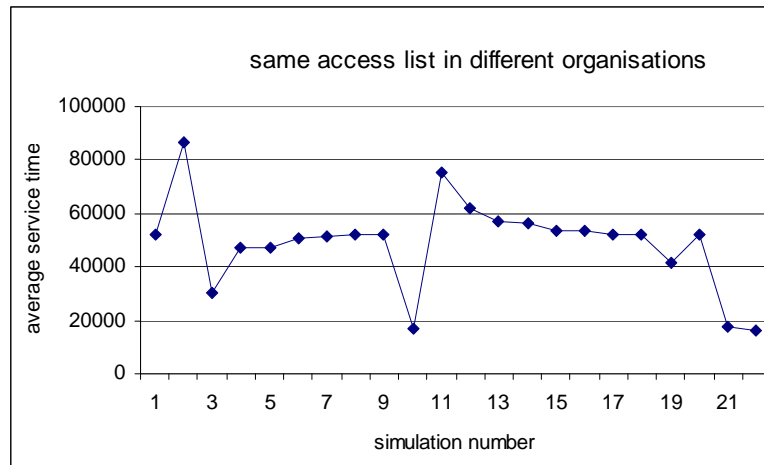


Figure 14, performance of the access list with the same packet stream at many different organisations of the rules' classes.

A function was developed to calculate the processing time for all the different permutations and provide the permutation yielding the lowest time for the given access list for a particular packet stream specifications. In this case, using 8 classes of rules, the processing time for 40320 permutations (8!) were calculated. The permutation for the shortest time was: 1, 8, 7, 6, 5, 4, 3 and 2.

Finally, Figure 15 provides a quick comparison for performance represented by average processing time per packet for a few organisations. The packet stream is made up of mostly class 1 packets. The access list is organised with class 1 rules at bottom and at top of the list. Class 1 at the bottom of the list gave the worst performance. The best performance obtained through the organisation based on the calculation of the processing time using the developed function according to the permutation with the lowest cost.

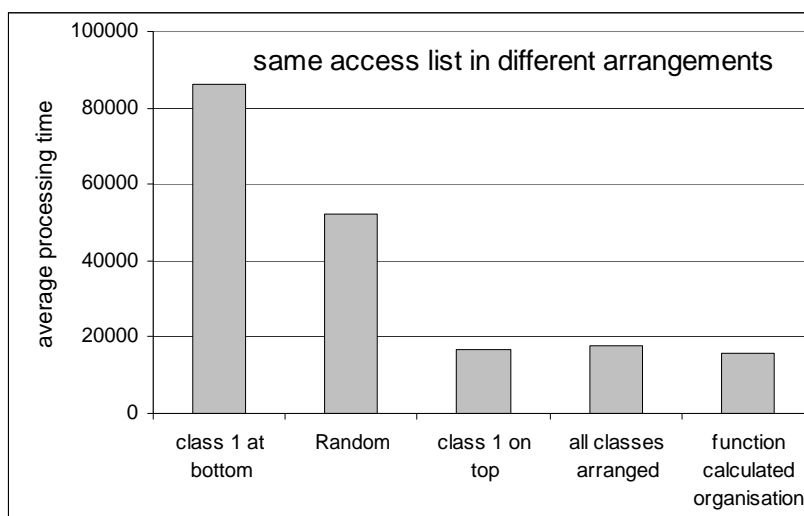


Figure 15, Performance comparison of different organisations.

8. CONCLUSION

This paper has described the simulation performed and analysed the results of the reorganisation of access lists. The results of the simulation showed clearly that reorganising access lists can have a large effect on performance when considered with the pattern of the packets arriving into network devices. The results indicate that different patterns of packets arriving into a network device will require a specific organisation of a given access list to yield the best performance. Performance is measured by the average processing time per packet to filter through the list. Given a particular packet stream specification and an access list specification, an algorithm has been developed to predict the best list rules arrangement. This proposed cost-weighting method is proved to produce the arrangement of access list rules that has the lowest average processing time per packet for a particular packet pattern.

Future work involves two specific points that came up as a result of this work. The first is the development of an automatic analyser of packet patterns arriving to a network device and making the appropriate selection of the most suitable organised version of the access list available. Such an application can be executed at different time intervals depending on the nature of changing patterns of packets arriving at a particular network device. Such an application will require the log of packets arriving to a communication device to be kept for analysis.

The second area to be considered for future work is the operation of reorganising and reordering of an access list and the verification of adherence to the security policy. Such applications would prove to be invaluable tools for network managers.

REFERENCES

- Ballew, Scott M. 1997. *Managing IP Networks with Cisco routers*. O'Reilly, 1st edition. October 1997.
- Bellovin, 1992. *There Be Dragons*, Proceedings of the Third USENIX UNIX Security Symposium, Baltimore, MD, September 1992
- Bryant, R., 1992. *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*, ACM Computing Surveys, 24 (3):293-318, September 1992.

- Chapman, D., 1992. *Network (In)Security Through IP Packet Filtering* Proceedings of third USENIX UNIX Security. Symposium, Baltimore, MD September 1992
- Cheswick, W.R. and Bellovin, S.M. 1994. *Firewalls and Internet Security, Repelling the Wily Hacker*, Addison-Wesley.
- Friedman David, and Nagle David, 2001. *Building Firewalls with Intelligent Network Interface Cards*. Technical Report CMU-CS-00-173. CMU, May 2001.
- Gupta, P. and McKeown, N., 1999. *Packet classification using hierarchical intelligent cuttings*. In Proc.Hot Interconnects VII ,Aug.1999
- Gupta, P., and McKewon, N., 2001. *Algorithms for Packet Classification* IEEE Network Special Issue, March/April 2001, vol. 15, no. 2, pp 24-32
- Habtam, Abie, 2000. *An Overview of Firewall Technologies* Norwegian Computing Centre January 2000.
- Haeni, T., 1997. *Firewall Penetration Testing* The George Washington University January 1997
- Hazelhurst, S., 1999. *Algorithms for Analysing Firewall and Router Access Lists*”, Techinal Report TR-Wits-Cs-1999-5, Dept. of Computer Science, University of Witwatersand, South Africa July 1999.
- Hazelhurst, S., 2001. A proposal for Dynamic Access Lists for TCP/IP Packet Filtering. *SAICSIT 2001*. Pretoria, South Africa.
- Ioannidis, J., and Bellovin, S., 2002. Implementing Pushback: Router-Based Defense Against DDoS Attacks. in *Proceedings of NDSS '02, Feb. 2002*
- Lakshman, T., and Stidialis, D., 1998. *High speed policy-based packet forwarding using efficient multi-dimensional range matching* in proceedings of SIGCOMM'98.
- Mahajan Ratual, Bellovin Steven M., Floyd Sally, Ioannidis John, Paxson Vern, ShenkerControlling Scott, High Bandwidth Aggregates in the Network. *AT&T Center for Internet Research at ICSI (ACIRI) and AT&T Labs Research, Technique Report (Draft), Feb. 2001*
- OMG, 1998. *CORBA/Firewall Security+Errata*, Object Management Group, Joint Revised Submission orbos/98-07-03, July 6, 1998
- Schreiner, R., 1998, *CORBA Firewall White Papers*, TechnoSec Ltd.
- Schuba, C.L. and Spafford, E.H., 1997, A Reference Model for Firewall Technology. In *Proceedings of the Thirteenth Annual Computer Security Applications Conference*, December 1997.
- Selani, Deb, 1998. *ICSA Firewall Policy Guide v2.00*. Available from: <http://www.du.edu/~dcelani/paper.htm> [Accessed 06 May 2003].
- Srinivasan, S., et al, 1999. Packet classification using tupe spce search in *Proceedings of SIGCOMM'99*.
- Stoica, Ion, 2001. Rout lookup and packet classification CS 268, February 2001. Available from: <http://www-inst.eecs.berkeley.edu/~cs268/sp03/> [Accessed 06 May 2003]
- Tanenbaum, A., 1996 *Computer Networks*, Prentice Hall, Upper Saddle River, New Jersey, USA.
- Warkhede, P., 2001. Fast Packet Classification for Two-Dimensional Conflict-Free Filters. *INFOCOM*
- Wulf, L., 1997. *Interaction and security in distributed computing.*” Dphil., Wolfson College, University of Oxford, Hilary Oct. Available from: <http://web.comlab.ox.ac.uk/oucl/publications/books/concurrency/textmat/> [Accessed 06 May 2003].