Tree Rule Firewall

A Thesis Submitted for the Degree of

Doctor of Philosophy

By

Thawatchai Chomsiri

in

Faculty of Engineering and Information Technology

UNIVERSITY OF TECHNOLOGY, SYDNEY

17th November 2016

CERTIFICATE

Date: 17th November 2016

Author: Thawatchai Chomsiri

Title: Tree Rule Firewall

Degree: Ph.D.

I certify that this thesis has not already been submitted for any degree and is not being submitted as part of candidature for any other degree.

I also certify that the thesis has been written by me and that any help that I have received in preparing this thesis, and all sources used, have been acknowledged in this thesis.

Signature of Author

Acknowledgements

I am greatly indebted to my supervisor, Professor Xiangjian (Sean) He, for his scholarly guidance, inexhaustible patience, constant encouragement, and invaluable suggestions. He is a nice, generous, helpful and hospitable person. I feel happy and comfortable with him during my PhD study. Many thanks are also expressed to my co-supervisors, Dr. Priyadarsi Nanda, for his valued suggestions and constantly technical supports. I appreciatively acknowledge the useful suggestion from Dr. Zhiyuan (Thomas) Tan. His suggestions help me to improve many paper publications of my research. I also gratefully acknowledge useful discussions with my team mainly consisting of Dr. Mian Ahmad Jan, Dr. Mohammed Ambusaidi and Dr. Aruna Jamdagni. I appreciate the supports, which I received from the Faculty of Engineering and Information Technology, and its staff. I appreciate the financial assistance, International Research Scholarship (IRS), which I have received from Graduate Research School - University of Technology Sydney. Lastly, I would like to thank my father (Mr. Weera Chomsiri), my mother (Ms. Darawadee Chomsiri) and members of my family for everything.

Table of Contents

List of Tables

List of Figures

Abbreviation

Abbreviations	Descriptions
2D-Box	Two Dimensions Box
ACL	Access Control List
Cent OS	Community ENTerprise Operating System
ConnTrack	Connection Tracking
CPU	Central Processing Unit
DEC	Digital Equipment Corporation
Dest IP	Destination IP Address
Dest Port	Destination Port
DIP	Destination IP Address
DMZ	Demilitarized Zone
DPT	Destination Port
DstIP	Destination IP Address
DstPT	Destination Port
ESXi	Elastic Sky X – integrated (ESXi is the primary component in the VMware Infrastructure software suite)
FDD	Firewall Decision Diagram
GHz	Gigahertz
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol

Abbreviations	Descriptions
Hyper-V	is formerly known as Windows Server Virtualization
IOS	Cisco's Internetworking Operating System
IP	Internet Protocol
IPSEC	Internet Protocol Security
LAN	Local Area Network
Max	Maximum
Mbps	Megabits per second
Min	Minimum
NICs	Network Interface Cards
OS	Operating System
PVFS	Procfs Virtual File System
RAM	Random-Access Memory
SIP	Source IP Address
SrcIP	Source IP Address
SrcPT	Source Port
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine

Abstract

Firewall is a network component for deciding packets whether they will be accepted or denied. The packet decision results are dependent on rule policy pre-defined by firewall administrators. In traditional firewalls, the rule policy will be arranged in a list of rule line called 'listed rule'. The listed rule can cause three significant problems consisting of speed, security, and user friendly problems. The speed problems can occur because many packets will be matched with the rule positioned in bottom positions. Firewall may waste time to verify packets with many rules positioned above the matched rule. Moreover, the traditional firewalls also face to rule conflicts, e.g., shadowed rules. Many rules written to prevent attacking packets may be shadowed by some rules above them and cannot block any packet so that dangerous packets originated from outside can reach internal networks. Additionally, the traditional firewalls are involved with the lack of user-friendly features because administrators must have enough experience in order to create enough efficiency rules.

This research proposes a novel firewall by using a tree structure of rules to solve the above problems. In the proposed approach, firewall administrators are able to design rules in the tree format, and then a core processor of firewall will process packets according to this format. The tree structure can be seen in both users' view and firewall's view. Packets will be verified with the tree shape of rule called 'tree rule'. To decide packet, searching for a data in the tree rule can be done quickly in comparison to searching data in the listed rule of traditional firewalls. This is because searching data in the Tree is faster than sequential searching data in Arrays. Moreover, rule conflicts can be eradicated, since each packet will be verified with the corresponding 'rule path' in the tree rule. This can avoid rule conflicts and shadowed rules. Thus, security problems

caused by shadowed rules cannot be found in the tree rule firewall. Moreover, administrators can create rules easier with the GUI (Graphical User Interface) rule editor. They can design tree rule by creating nodes and links. There are ranges of IP addresses or ports inside each node. The GUI can sort the data inside nodes automatically and maintain consistency of the rule. Thus, the tree rule can be designed easily.

Therefore, the Tree-Rule firewall can provide faster functional speed, be more secure, and be easier to use compared to traditional Listed-Rule firewalls.

Papers from the Thesis

1. **T. Chomsiri**, X. He, P. Nanda, "Limitation of listed-rule firewall and the design of tree-rule firewall", in: Proceedings of the 5th International Conference on Internet and Distributed Computing Systems, China, 2012, pp. 275–287.

2. X. He, **T. Chomsiri**, P. Nanda, Z. Tan, "Improving cloud network security using the Tree-Rule firewall", **Future Generation Computer Systems**, Elsevier, 30 (2014) 116-126. [ERA **Tier-A** Journal, **IF=2.430**, SCImago Journal Rank: **Q1**]

3. **T. Chomsiri**, X. He, P. Nanda, Z. Tan, "A Stateful Mechanism for the Tree-Rule Firewall", IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom2014), 2014, pp. 122-129. [ERA **Tier-A** conference]

4. **T. Chomsiri**, X. He, P. Nanda, Z. Tan, "Hybrid Tree-rule Firewall for High Speed Data Transmission", **IEEE Transactions on Cloud Computing**, 2016, no. 1, pp. 1, PrePrints, doi:10.1109/TCC.2016.2554548. [SCImago Journal Rank: **Q1**]

5. **T. Chomsiri**, X. He, P. Nanda, Z. Tan, "An Improvement of Tree-Rule Firewall for a Large Network: Supporting Large Rule Size and Low Delay", IEEE 15th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom2016), 2016, pp. 178-184. [ERA **Tier-A** conference]

Chapter 1 Introduction

Nowadays, the Internet is increasingly used, and almost all computer networks are connected to the Internet. These connections combine every isolated network to be the one large network. In fact, this connectivity has emerged since 1980s by academia after the first time of Internet use in 1960s. At that time, the Internet seemed to be open trusting community until the late of 1980's since Clifford Stoll has discovered that German spies [1] was tampering with his system. Consequently, the first firewall was first proposed by engineers from Digital Equipment Corporation (DEC) in 1988. They developed filtering systems which were known as packet filtering firewalls. At the same time, Bill Cheswick and Steve Bellovin at AT&T Bell Labs were conducting their research in packet filtering firewalls [2] and developed a working model for their own packet filtering firewalls.

The first generation of firewall is the packet filtering firewalls [2], which examine a packet using a packet's header information (e.g., source IP address, destination IP address, and destination port) without considering TCP connection states. A few years later, the stateful firewalls (the second generation firewall) were invented. This type of firewall examines connection states such as ‘start of a new connection’, ‘being part of an existing connection’, and ‘not being part of any connection’. Thus, the stateful firewalls are faster and more secure than the packet filtering firewalls. Today, almost all firewalls are the stateful firewalls [3].

1.1 Background and Motivation

According to the Ingham's article [4], the first hacking attempt has been reported in late 1980's since Clifford Stoll discovered German spies [1]. To create a protection, router devices have been adopted to be the first generation firewalls. There were many cases of Internet threats. For example, computer networks of the NASA’s Ames Research Center have been attacked by Internet viruses and the first famous Internet worm named 'Morris' [5]. These serious threats caused computer scientists to develop more efficient firewalls. Therefore, the firewalls were developed and improved by many computer companies including DEC and AT&T Bell LAB.

The firewalls invented in the first era operated with uncomplicated tasks. They verified source/destination IP addresses and ports without considering state of connections. Thus, the first generation of firewalls has been called the 'Packet Filtering firewall'. Many years later, the firewalls have been developed to be able to make a judgment using connection states [6]. Even though this generation of firewall has been called 'stateful packet inspection', today many people call it a 'stateful firewall'.

1.1.1 Firewall in General Terms

Firewall is an important network device placed between a network that you want to protect (e.g., company network) and a network that you cannot trust (e.g., Internet network), for regulating the packets travelling between networks. The firewall can be used to be a gateway to link two or more networks together, and it verifies packets travelling across networks using its rule pre-defined by firewall administrators. Firewalls today decide packets by comparing packets' header information with its rule. The rule of traditional firewall consists of a list of condition lines and actions. An example of traditional firewall's rule is shown below.

```
==========================================================================
   No.  Protocol Source IP         Destination IP    Dest. Port  Action
==========================================================================
   1       TCP   10.1.1.1          20.1.1.1          80          Accept
   2       TCP   10.1.1.2          20.1.1.1          80          Deny
   3       TCP   10.1.1.0/24       20.1.1.1          80          Deny
   4       TCP   10.1.1.3          20.1.1.1          80          Accept
   5       TCP   10.2.2.0/24       20.2.2.5          80          Deny
   6       TCP   10.2.2.5          20.2.2.0/24       80          Deny
   7       TCP   10.3.3.0/24       20.3.3.9          80          Accept
   8       TCP   10.3.3.9          20.3.3.0/24       80          Deny
   9       IP    0.0.0.0/0         0.0.0.0/0         0-65535     Deny
==========================================================================
```

To regulate packets travelling pass through the firewall, header information of the packets (e.g., source IP address, destination IP address, and destination port) must be compared with conditions in each rule in firewall's rule list. The comparison will begin from the first rule to the last rule, rule by rule, or until the comparison result is matched. An action specified in a matched rule will determine the packets whether they will be accepted or denied.

A format of firewall rules in one brand may be slightly different from other brands. For example, the format of Cisco ACL (Access Control List) is not complicated compared to IPTABLES' rule format. Rules of some firewall products may be seen as GUI (e.g., rules of Check Point Firewall-1 [7] and Juniper NetScreen [8]). However, they are still considered as a list of condition lines.

1.1.2 Packet Filtering Firewall

A packet filtering firewall [2] has been designed and created in the first decade of firewall age. This type of firewall uses an uncomplicated operation to examine incoming and outgoing packets. Due to the simple operation, each line of a firewall rules can regulate packet flows in only one direction [9]. In a context of packet filtering firewalls, Johansson and Riley [9] state that a rule, which allows an outbound access, requires a mirror rule for allowing an inbound access to permit a replied packet to enter the network. Even though some packet filtering firewalls can add mirror rules automatically, some firewalls need firewall administrators to do this task manually [10].

The main problem of the packet filtering firewalls is a security problem because the mirror rules generated by the firewalls or even administrators can cause a big security hole. This hole can allow attackers to access opened TCP ports on protected computers in internal networks. For example, in the rule list below, rule #1 which allows internal users to connect every IP addresses on HTTP port (port number 80) corresponds to its mirror rule (rule #2).

No.	Protocol	Source_IP	Source_Port	Dest_IP	Dest_Port	Action
1	TCP	10.1.1.0/24	0-65535	0.0.0.0/0	80	Accept
2	TCP	0.0.0.0	80	10.1.1.0/24	0-65535	Accept
3	IP	0.0.0.0/0	0-65535	0.0.0.0/0	0-65535	Deny

Rule #2 allows attackers from any IP address to connect with any ports of internal computers using the source port number 80. Note that some firewalls generate mirror rules automatically and these rules may be invisible for users.

Another main problem of the packet filtering firewall is that the firewall will allow all packets, which are accepted by the matched rules without verifying packets' connection states. Thus, packets classified in non-existing sessions [3][6] can travel through the firewall as well.

Although the packet filtering firewall was obsolete and has been replaced with the Stateful firewall [3], its basic concepts are still used on some OS (Operating System) features such as IPSEC of MS Windows, and NAT of Cisco IOS.

In early 1990s, the problems of packet filtering firewalls were solved through improving a model inside the firewalls. This improvement changes a mechanism inside the packet filtering firewall to operate more complex for security enhancement. The new firewall was called 'Stateful firewall' [3].

1.1.3 Stateful Firewall

Similar to the Packet Filtering firewalls, Stateful firewalls examine packet's header (e.g., IP, TCP and UDP header). However, stateful firewalls are slightly smarter because they understand TCP and UDP connection status by watching all states of connections [9]. If clients behind the firewall create a new connection to communicate with Web servers on the outside, the firewall will create inbound filters to allow relevant reply packets from the Web server to the clients. The stateful firewall will keep track the states of network connection which travel across it. The firewall may check TCP sequence numbers for observing the connection status. Some examples of stateful firewalls are Check Point Firewall-1, Juniper Net Screen, and IPTABLES.

The IPTABLES is the most popular open-source firewalls. There are many sources of documents providing understanding about its stateful mechanisms inside. The article entitled "Netfilter's Connection Tracking System" by Pablo Neira Ayuso [11] offers useful information about a stateful mechanism within the IPTABLES. This article reveals that the IPTABLES uses hashing algorithm to digest important packet header information (e.g., source IP address, destination IP address, source port, and destination port), and then use the hashed result as an entry pointing to a node of packet header information. Figure 1.1 shows the stateful mechanism [11] in IPTABLES firewall.

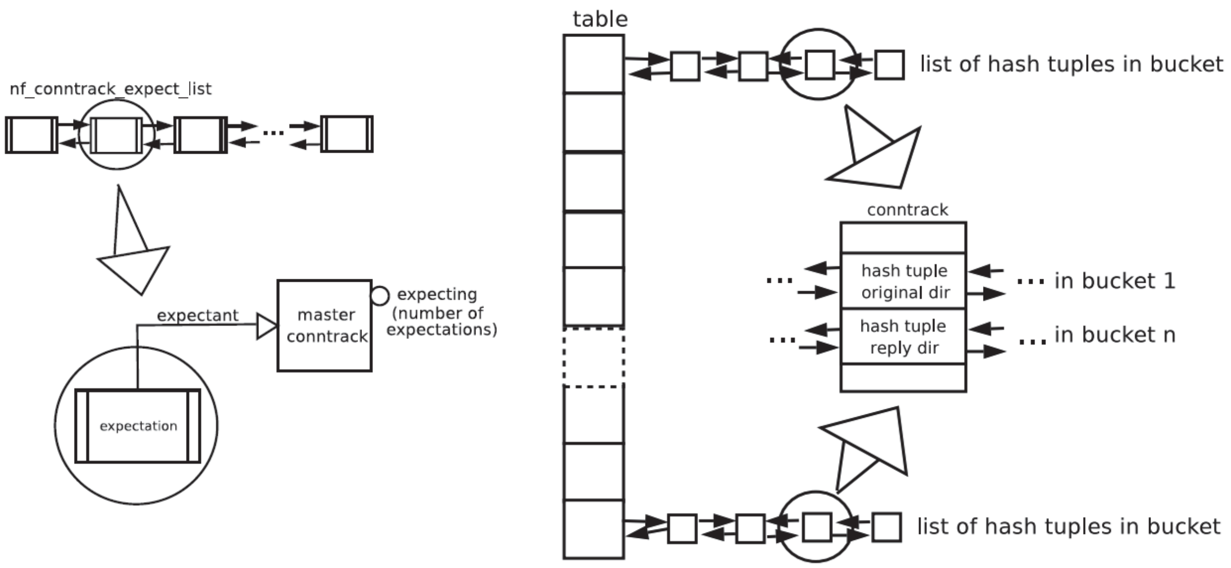


Figure 1.1: A stateful mechanism in IPTABLES firewall [11].

To prevent a hashing collision, the IPTABLES uses buckets of linked lists to carry many nodes in the same entry (see Figure 1.1). If a new connection is created, firewall will calculate a hashing result and find a location for putting necessary information of the new connection. If a reply packet reaches to a firewall, the firewall will verify that the packet belongs to the existing connection or not. Finally, the node corresponding to the connection will be destroyed when the connection is disconnected or expires. The first packet of connection (e.g., the packet having TCP flag SYN=1, ACK=0) will be examined using the firewall's rule list while others packets will be examined using the hashing tables and nodes in buckets.

1.1.4 Anomaly Detection

In this context, an anomaly means a conflict between rules. Many researchers have defined types of anomalies and proposed algorithms to discover them.

Ehab Al-Shaer and Hazem Hamed used SET theory and their 'policy tree' [12] to discover rule conflicts. They created special software to map the original listed rules to the 'policy tree'. Consequently, they used algorithms applied from their proposed 'state diagram' to identify conflict rules. An output of the algorithms can report the anomaly types and the rules involving conflicts. Moreover, their tool named "Policy Advisor" also helped firewall administrators to create firewall rules with minimal conflicts.

Alex Liu [13] proposed a tool applied from his 'decision diagrams' to verify firewall policy (i.e., firewall rule). The tool used a firewall policy as an input, then returns an output indicating whether the policy was satisfied or not. He has evaluated the tool with real-life networks. Furthermore, he has experiment it with a synthetic large size of firewall policies. Although the results show that his algorithm is very efficient, it cannot identify any type of anomalies.

Alex Liu and Mohamed Gouda [14] presented an algorithm to detect all redundant rules within a firewall policy. They provided necessary conditions for discovering all redundant rules. They categorised redundant rules into two main groups. The first group is called 'upward redundant rules' which are the rules overlaid by some rules positioned above them. The second group is called 'downward redundant rules' which are the rules overlaying on other rules below. They applied their 'firewall decision trees' to identify the group of redundant rules. However, their algorithm cannot discover other anomalies such as a correlation anomaly defined by Ehab Al-Shaer and Hazem Hamed [12].

Thanasegaran et al. [15] investigated anomalies within Time-Based firewall policies. They proposed the 'Mapping Mechanism for Periodic Filters' to detect rule conflicts. Their algorithm can minimise a number of repetitions and can reduce the huge computational time and memory. However, they categorised conflicts of Time-Based firewall policies on only one group called the 'overlapping confliction'. Besides, they did not mention the conflict which can happen from swapping the positions of rules. Therefore, their theory cannot cover the concept of Ehab Al-Shaer's shadowing anomaly [12].

1.1.5 Motivation

The traditional firewalls, including packet filtering and stateful firewall, are based on the list of condition lines [2]. These firewalls have to process packets by comparing packet header information with the listed rules, rule by rule, until the packets match with a corresponding rule, or the comparison reaches the last rule. This comparison method can be considered as a sequential searching which gives low performance compared to other searching algorithms [3]. Considering the situation that our network is under attacking from outside hackers, viruses or worms, a huge number of attacking packets are arriving to our firewall with random destination ports or random IP addresses of our network. These packets will not match with any rules except the last rule, which has been pre-defined as 'deny all'. If a percentage of these packets are high (i.e., 50-90%) compared to a normal traffic, the firewall will waste its processing time to verify these packets with all unmatched rules in order to be dropped by the last rule. This can cause a significant network delay to our network. In a normal situation of traffic (i.e., no attacking packets), firewall still wastes its computational time to verify packets with many rules positioned above the real matched rule. Therefore, it can be claimed that the listed rule of traditional firewalls can cause functional speed problems to networks.

Apart from functional speed problem, traditional firewalls' listed rules can cause rule conflicts such as shadowed rules and redundant rules, especially the shadowed rule. A shadowed rule is the rule that cannot be matched with any packets because all packets are already matched with other rules above it. This can cause security problems because some rules, which will protect the network against hackers, may be shadowed and will not be processed by the firewall for protection against dangerous packets. Consequently, the dangerous packets can travel to internal networks.

Moreover, a listed rule in a traditional firewall is quite hard for the firewall administrators to design an efficient set of rules to protect a large network, which requires a large number of rules without any rule conflicts. This is because firewall administrators have to be careful about rule positioning to avoid rule conflicts, which is hard to do if the number of rules is massive. Thus, the listed rule of traditional firewalls is considered to be difficult to design.

Thus, it is important to solve these problems by inventing a new kind of firewall, which can provide a higher speed and less rule conflicts, and on which the rules can be easily designed for large networks. The new firewall model will be designed to support the three goals mentioned above.

1.2 Objectives

This research aims to study disadvantages and limitations of traditional firewalls, and re-design the architecture of a novel firewall to minimise these disadvantages. A rule of the proposed firewall will be based on a tree structure. The original design of the novel firewall will be improved and we will add necessary features (e. g., stateful features) to the firewall. Finally, experiments will be conducted by testing and evaluating the performance of the firewall, and comparison with various traditional firewalls will be made.

The detailed objectives of this research are listed as follows.

1) To develop a model for designing and representing the tree-based firewall. The model has to be easy for users to understand rule policies, even though the rule tree may be large with many groups of IP addresses in a large network.

2) To improve the proposed model to be understandable and computable for the 'firewall core processor'. The term 'firewall core processor' means some procedures of executable codes of firewall software, which operates in a kernel level to decide packets travelling through the firewall. The improved model (i.e., tree rule) can be easy to understand by both users and computers.

3) To develop additional features inside the proposed firewall that is able to handle a stateful operation. A stateful mechanism of the proposed firewall is extended from a stateful mechanism of IPTABLES (the most famous open-source firewall).

4) To prove that rule conflicts can occur in the traditional listed-rule firewalls and bring disadvantages to them, and to illustrate that these rule conflicts can be avoided by using the proposed firewall.

5) To implement the proposed firewall in various environments (e.g., virtual networks, real networks, and cloud networks) in order to evaluate its outperformance compared to the traditional firewalls (e.g., IPTABLES).

1.3 Contributions and Innovation

The proposed schemes originate from the tree structure of firewall rules. The contributions and innovation of the proposed scheme are presented as follows.

1) This thesis proposes a novel firewall called the 'Tree-Rule firewall', which is based on a tree structure of rules.

- A rule of the Tree-Rule firewall will be created (by firewall administrators) in a tree shape instead of the list of lines.
- A data structure of rules (in the memory of proposed firewall) is in a tree structure. Packets travelling through the firewall will be regulated and verified by the firewall using the tree structure of rules stored in its memory (RAM: Random Access Memory).
- A mechanism inside the Tree-Rule firewall is proposed and is compared with the mechanism in a traditional firewall.

2) This thesis proves that rule conflicts do not occur in the proposed Tree-Rule firewall.

- Shadowed rules and Redundant rules [12] are no longer found in the rule set of any Tree-Rule firewall.
- Firewall administrators can write the rules in a Tree-Rule firewall easier because they do not have to worry about rule conflicts. Therefore, the proposed firewall can operate faster and more securely. Moreover, its rules can be designed easier.

3) A stateful mechanism is proposed to search the rules in a Tree-Rule firewall quickly.

- Examining a packet using the tree rule is faster than examining a packet using traditional listed-rules. This is because searching data in the 'tree' structure is faster than searching data sequentially in an Array (or linked list).
- The stateful mechanism is improved from IPTABLES' stateful mechanism to decrease packet decision time and use fewer resources.

1.4 Structure of the Thesis

The thesis is organised as follows. Chapter 1 (this chapter) describes the background and motivation of this research. Chapter 2 reviews the existing research work. Anomaly detection methods for traditional firewall's rules are presented. Moreover, a stateful mechanism inside the traditional firewalls is discussed. Chapter 3 illustrates five limitations of the traditional firewalls. These limitations consist of (1) limitations caused by shadowed rules, (2) limitations caused by swapping positions of rules, (3) limitations caused by redundant rules, (4) limitations about rule design, and (5) limitation caused by sequential search computation for verifying packets. Chapter 4 proposes a novel firewall called the 'Tree-Rule firewall', which is based on a tree structure of rule. Firstly, the basic design is introduced. Then, the improvement of design is presented. Lastly, the implementation and experimentation are conducted. Chapter 5 proposes a stateful mechanism for the Tree-Rule firewall. Its performance in terms of functional speed is compared with IPTABLES (the most popular open-source firewall). In Chapter 6, some features of the Tree-Rule firewall are merged with some good features of the traditional firewall. This combination creates a hybrid firewall, which operates faster than traditional firewalls and pure Tree-Rule firewalls. A summary of this thesis and future work are presented in Chapter 7.

Chapter 2 Related Works

This chapter reviews the existing work related to this thesis. Rule conflicts identification, and firewall tools are presented. The hierarchical and tree models in the existing work are also discussed in this chapter. Moreover, a stateful mechanism of IPTABLES (the most popular open-source firewall) is also illustrated as well.

2.1 Rule Conflict Identifications and Firewall Tools

There are many researchers studying conflicts within the firewall rules in order to answer the questions "How many types of rule conflicts?" and "How to detect and remove them?" Many research projects focused on creating tools supporting firewall administrators to design firewall rules easily with minimal rule conflicts. This section reviews several significant and useful research topics, which can be applied and extended for the proposed firewall in this thesis.

The paper "Firewall Policy Advisor for anomaly Detection and Rule Editing", by Ehab Al-Shaer and Hazem Hamed, introduced the four types of anomalies [12] to identify rule conflicts occurring in traditional firewalls' rule lists. These types of rule conflicts consist of 'Shadowing Anomaly', 'Correlation Anomaly', 'Generalization Anomaly', and 'Redundancy Anomaly'. To explain the Shadowing Anomaly, the authors of the paper [12] described the concepts as follows. The shadowed rule is a rule that will never be executed because all packets that matched with this rule are already matched with one rule positioned above in the rule list. The authors also described the Correlation Anomaly. Two rules (e.g., Rule-x and Rule-y) are correlated if some fields (e.g., Source IP address, Destination IP address, Source Port, and Destination Port) in Rule-x are subsets of the same field in Rule-y, and some of the rest fields in Rule-x are supersets of the same fields in Rule-y. Similarly, a definition of Generalization Anomaly has been defined. A rule is said to be a generalization of a previous rule if it matches all the packets that can be matched with the previous rule. Redundancy Anomaly is also defined. Rule-x is redundant to Rule-y (x<y) if all fields of Rule-x are subsets of the same fields in Rule-y and actions of two rules are the same. With these definitions, it is possible to create algorithms for discovering the anomalies and rule conflicts inside firewalls' rule list. Apart from definitions of the four anomalies, they also proposed a tool to design anomaly-free firewall rules. Recently, they applied their definitions with the 'adaptive threshold tuning' [16] to real-time detection of the anomalies within a network.

Chen et al. [17] proposed a ‘fault model’ of firewall policies. This model consists of five types of faults, which are 'Wrong order', 'Missing rules', 'Wrong predicates', 'Wrong decisions', and 'Wrong extra rules'. Additionally, they provided an automatic correction technique, and proposed a systematic approach employing these five types of faults for automatic firewall rule correction.

Mansmann et al. [18] suggested a visualization tool, which can support firewall administrators to understand firewall rule set easily. This tool provides a hierarchical visualization of rules and objects, based on their common characteristics. Moreover, the tool can illustrate rules and object groups in classical tree view components.

Garcia-Alfaro et al. [19] proposed a management of stateful firewall misconfiguration using automation tools. They also validated the feasibility of their proposal using a proof of concept prototype. This method can parse existing firewall configuration files, and handle the discovery of flawed rules automatically.

Sylvain et al. [20] demonstrated the distributed firewall anomaly detection using LTL model checking. They suggested that the correct identification of anomalies must be taken into account of the routing function performed in each node of the network. They also implemented an anomaly detector, which can support firewall administrators to identify distributed anomalies with reasonable cost.

2.2 Hierarchical and Tree Models in Previous Works

There are several research papers in study areas of firewalls. These papers presented models look like the tree structure and a hierarchical firewall rule [12][13][21]. However, these are not considered as the Tree-Rule firewall proposed in this thesis. Details of these research studies are illustrated as follows.

Liang Zhao et al. [21] have proposed the use of 'goto' function inside traditional listed-rule firewalls (e.g., a 'jump' command in IPTABLES). They have guided firewall administrators to apply listed rules of traditional firewalls to be a simple tree by defining a list of relevant rules as a node and using 'goto' like function as a link between nodes. Although their rule structure looks like a tree structure, their sub-rules (or nodes) contain listed rules. Therefore, their firewalls are still deemed as the listed-rule firewalls. A group of listed rules in the node still performs sequential searching for examining a packet. Moreover, rule conflicts can occur in their firewall rules. The firewall rule designing proposed in this research is based on the traditional firewall, which depends on listed rules. A processing method inside the firewall presented in [21] still executes a rule in rule list, line-by-line, sequentially. This is because there is no modification of processing algorithm inside the firewall.

Alex Liu and Mohamed Gouda proposed 'Diverse Firewall Design' [13] using a tree structure to organise the rules translated from a traditional rule list. This method aims to discover and eradicate some of the rule conflicts. They suggest two main steps (as shown below) for firewall administrators to design firewall rule without rule conflicts.

- Step #1: design a firewall rule using a firewall decision diagram.
- Step #2: convert the firewall decision diagram into a compact sequence of 'output rules' using a software tool.

The software tool [13] uses the 'FDD reduction' and 'FDD marking' algorithms to combine rules together, and use their 'firewall compaction' algorithm to remove redundant rules. However, their work was still based on the traditional listed-rule firewall, and the firewall still decides packets using list of 'output rules' rule-by-rule sequentially.

Ehab Al-Shaer and Hazem Hamed proposed the tool called 'Firewall Policy Advisor' [12] to detect anomalies within rule list of firewalls. The tool can also be used for editing and verifying firewall rules. This tool begins with reading the listed rule, which was designed by firewall administrators and then convert it to a decision tree structure within RAM of a firewall administrator's computer. With the decision tree structure of rule, the tool can detect rule conflicts and alert firewall administrators to know and remove the conflicts manually. Finally, the rule with confliction free is exported from the tool to a real firewall. Although this process involves tree structure of rule, it only operates in firewall administrators' computers rather than the firewall. Moreover, the firewall still decides packets with a listed rule and traditional firewall's mechanism. With sequential rule processing, the firewall still faces significant time consumption.

Furthermore, the tool may not be able to find all of rule conflicts and allow these rule conflicts to appear in the firewall.

These research studies did not propose any new type of firewall but suggested merely some methodologies to detect and eradicate rule conflicts. Their output rules were still the listed rules, and then sequentially proceed by traditional firewalls.

2.3 Stateful mechanism inside IPTABLES

The paper entitled "Netfilter's Connection Tracking System" [11] has revealed a stateful mechanism inside IPTABLES. This paper has presented a model of the mechanism, which can be extended and improved to be integrated into the proposed firewall in this thesis. The two main points of stateful mechanism inside IPTABLES are as follows.

- It can identify a packet that it is in (1) a new connection, (2) an existing connection, or (3) an invalid connection. This can provide more security to the firewalls.
- For most packets, examining a packet (to decide whether it should be accepted or dropped) takes the same interval time. The time complexity (represented in terms of big O) for examining each packet is the O(1).

A packet creating a new connection will be examined with IPTABLES' rule list while other packets coming later will be examined with the 'hashing table' [11]. IPTABLES has methods to make sure which type of connection a packet is in. For examples, IPTABLES can verify TCP flags inside a packet's header to distinguish whether the packet is in a new connection or an existing connection. For example, the flags SYN=1 and ACK=0 can indicate that the packet is in the first packet of TCP connection. The packet having these flags is considered to be a new connection. Information from the paper [11] illustrates that the first packet of connection will be examined with a rule list of IPTABLES first. If the packet is allowed by the rule list, IPTABLES will perform hashing calculation and create a node for storing some necessary information of this packet. There is no performing of the hashing calculation or creating any node if the packet is dropped by the rule list. Hashing calculation is used for determining an entry number in which the node resides. Input parameters for the hashing calculation are taken from Later-3 and Layer-4 data in packet's header. The operation shown below can illustrate this method.

Entry number = Hash(SrcIP, DstIP, SrcPT, DstPT)

The hashing algorithm used in IPTABLES is Jenkins' hash [22]. Hashing calculation in the stateful mechanism inside IPTABLES for the first packet of a connection will perform two times. The first hashing result is used in an original direction. The second hashing result is used in a reply direction. In the second hashing calculation, there are swapping positions of input parameters. A source IP address will be swapped with a destination IP address, and a source port will be swapped with a destination port. Therefore, methods for calculating the two entry numbers can be illustrated as shown below.

Entry_number_1 = Hash(SrcIP, DstIP, SrcPT, DstPT)

Entry_number_2 = Hash(DstIP, SrcIP, DstPT, SrcPT)

Note: The 'SrcIP' is source IP address, the 'DstIP' is destination IP address, the ' SrcPT' is source port, and the 'DstPT' is destination port.

The 'Entry_number_1' is used for an original direction, and the 'Entry_number_2' is used for a reply direction. After the two entry numbers are calculated, two nodes will be created for recording the important information of a packet for both an original and a reply directions as presented in Figure 2.1.

There will be collisions of hashed results, e.g., two different packets can bring the same entry number. To solve this problem, IPTABLES uses the 'bucket' to store many nodes of packets in the same entry (same hashed result).

Note: The bucket is a linked list of nodes located in the same entry.

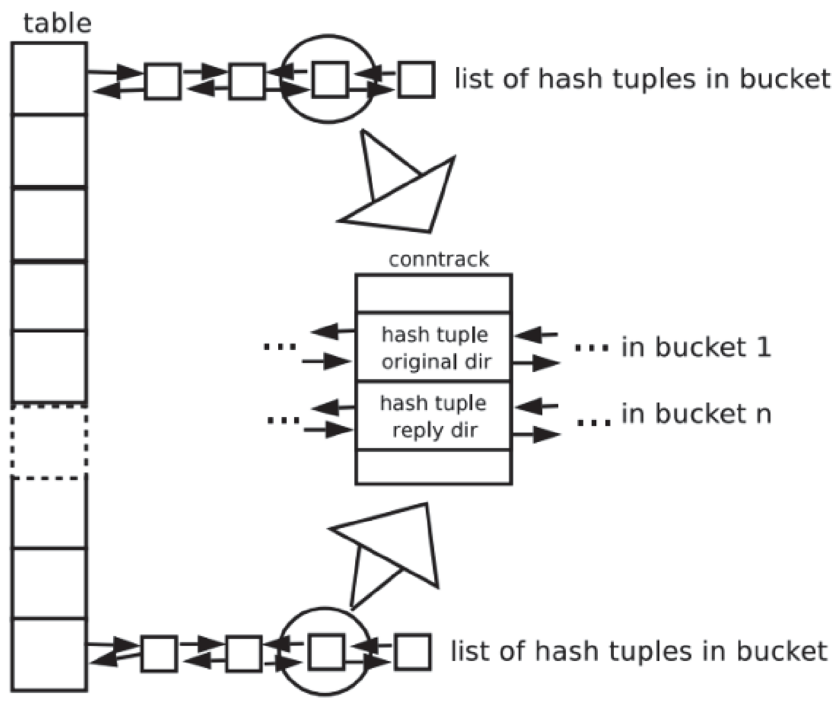


Figure 2.1: A stateful mechanism in IPTABLES firewall [11].

The nodes are used for recording important information of packets such as their source IP address, destination IP address, source port, destination port, direction type (original or reply direction), and connection time out. Information of one packet will be recorded in two nodes because this mechanism wants to verify the packet in both directions.

When the second packet or later packets arrive at the firewall, the packets are examined with only the hashing table (shown in Figure 2.1) instead of the listed rule. The firewall calls the hashing function to make the Entry_number_1 to search for a node, which belongs to the packet. The searching will be performed in only one entry (Entry_number_1) by comparing important information inside the nodes and the packet. If the searching result is found, IPTABLES can identify that the packet is in existing connections so that the packet will be allowed. If not, IPTABLES can identify that the packet is in an invalid connection so that the packet should be dropped. IPTABLES may consider TCP flags in packet's header to determine that the packet is in a TCP's closing state not. If so, it may record an appropriate number in the 'time out' field in the node and create a timer object for deleting the node. The timer object is a trigger for deleting the node when the connection is closed.

2.4 Summary

This chapter has summarised the methods used for identifying the rule conflicts and recap the existing firewall tools presented in the existing work. The hierarchical and tree models proposed by other researchers have been discussed. Moreover, a stateful mechanism used in IPTABLES firewall has been introduced.

To identify rule conflicts occurring in traditional firewalls' rule lists, there has been an introduction of four types of anomalies. These anomalies consist of:

- Shadowing Anomaly
- Correlation Anomaly
- Generalization Anomaly
- Redundancy Anomaly

Several researchers have presented various models, which may look like a tree structure and a hierarchical firewall rule. However, these are different from the Tree-Rule firewall concept proposed in this thesis. For example, the Alex Liu's Diverse Firewall Design uses a tree decision diagram to discover and eradicate the rule conflicts. However, this tool only works on a rule list of traditional firewalls, and there is no modification inside a firewall to process a packet using the tree structure of rule.

A stateful mechanism used in IPTABLES firewall has been discussed. It is based on key components shown as follows.

- Hashing Table
- Nodes of packet header information
- Buckets (linked lists of nodes)

This mechanism can be improved and extended, and applied to the proposed firewall presented in this thesis.

Chapter 3 Limitations of Listed-Rule Firewalls

This chapter will illustrate that firewalls today (Listed-Rule Firewall) have five important limitations which may lead to security problem, speed problem, and "difficult to use" problem. Firstly, the limitation about "Shadowed rules" (the rule that cannot match with any packet because a packet will be matched with other rules above) can lead to security and speed problem. Secondly, the limitation about swapping position between rules can bring a change in firewall policy and cause security problem. The third limitation is about "Redundant rules", which can cause speed problem. Next, the limitation of rule design because firewall administrators have to put "Bigger Rules" only at the bottom or lower positions that can result in a "difficult to use" problem. Lastly, the limitation from sequential computation can lead to speed problem.

3.1 Background and Related Works

Firewalls are important devices that can improve network security. A firewall's security level does not depend on its cost, but rather comes from the secure rules inside it. While configuring the firewall, we should focus on creating accuracy and non-conflicting rule sets. There are many studies about firewall rule conflicts (anomalies) that occur within rule sets. Ehab Al Shaer et al [12] proposed several anomaly definitions including “Shadowing anomaly”. He defined the “Shadowed Rule” as the rule that cannot match with any packet. For example, rule number 4 (see Table 3.1) is a shadowed rule. This type of rule can be removed from the rule list without any change of policy. Moreover, they have applied their definitions and theories for analysing a distributed firewall [23]. In [12][23][24], authors focused on mathematics for analysing firewall rules. Scott Hazelhurst [25] used Binary Decision Diagrams (BDDs) to present and analyse rule sets. Pasi Eronen [26] proposed an Expert System that was based on Constraint Logic Programming (CLP) for users to write higher-level operations to detect common configuration mistakes and find packet matched on each rule.

Table 3.1: An example of rules on the Listed-Rule Firewall

No.	Protocol	Source_IP	Destination_IP	Destination_Port	Action
1	TCP	10.1.1.1	20.1.1.1	80	Accept
2	TCP	10.1.1.2	20.1.1.1	80	Deny
3	TCP	10.1.1.0/24	20.1.1.1	80	Deny
4	TCP	10.1.1.3	20.1.1.1	80	Accept
5	TCP	10.2.2.0/24	20.2.2.5	80	Deny
6	TCP	10.2.2.5	20.2.2.0/24	80	Deny
7	TCP	10.3.3.0/24	20.3.3.9	80	Accept
8	TCP	10.3.3.9	20.3.3.0/24	80	Deny
9	IP	0.0.0.0/0	0.0.0.0/0	0-65535	Deny

3.2 Limitations of Listed-Rule Firewall

In this section, we will illustrate that the existing firewalls (Listed-Rule firewalls) have five critical limitations which lead to security problem, speed problem, and ‘difficult to use’ problem. These limitations are that

1. ‘shadowed rule’ (i.e., a rule that can never be matched by any packet because the packet must have matched with other rules above) can lead to security and speed problem;

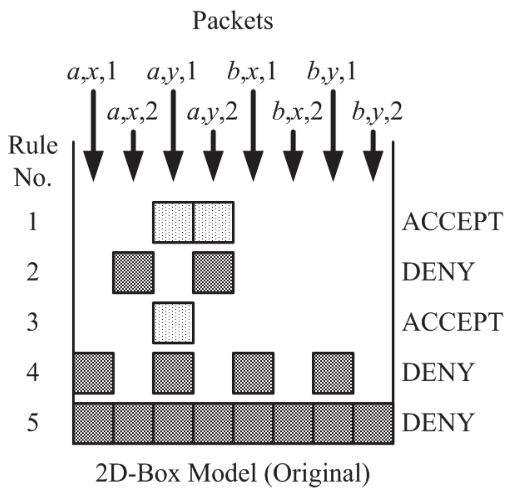
2. swapping position between rules changes the firewall policy and hence causes a security problem;

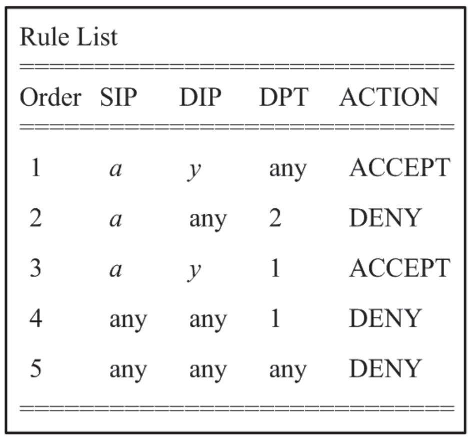
3. ‘redundant rules’ can cause speed problem;

4. firewall administrators have to locate ‘bigger rules’ only after ‘smaller rules’, and this results in a ‘difficult to use’ problem; and

5. sequential rule searching can lead to a speed problem.

In the following, we design a model to reveal the five limitations of the Listed-Rule firewall. This newly designed model is called the ‘2D-Box Model’ [27][28] for explaining a matching between packets and firewall rules as shown in Figure 3.1. Suppose that there are two source IP addresses (‘a’ and ‘b’), two destination IP addresses (‘x’ and ‘y’), and two port numbers (‘1’ and ‘2’) in the system. We do not include other attributes (such as source ports and protocol types) for easier understanding.




Rule List

Order	SIP	DIP	DPT	ACTION
1	*a*	*y*	any	ACCEPT
2	*a*	any	2	DENY
3	*a*	*y*	1	ACCEPT
4	any	any	1	DENY
5	any	any	any	DENY

Figure 3.1: The 2D-Box Model (left) and the rules of a Listed-Rule firewall (right) [27][28]

In Figure 3.1, SIP stands for the set of Source IP addresses of a rule entry (corresponding to a rule order number), DIP stands for the set of Destination IP addresses, and DPT stands for the set of Destination Port numbers.

Referring to the 2D-Box Model (see Figure 3.1), the incoming packets will be matched with Rule-1 (the 1st rule in the list) first. In this case, Rule-1 will 'accept' two packets. The remaining packets will continue falling down to Rule-2 that has a 'deny' action. After that, the remaining packets will continue falling down to other rules below until they reach the last rule or match with some rules.

Given a rule entry with its SIP, DIP and DPT, let

$$\begin{aligned}\text{SIP} \times \text{DIP} \times \text{DPT} = \{(i, j, k) \mid\ & i = \text{source IP address,}\\ & j = \text{destination IP address and}\\ & k = \text{destination port number}\}. \end{aligned} \qquad (1)$$

Note that the action for either 'Accept' or 'Deny' is excluded from the above representation, where ''×'' is an operator for computing the 'Cartesian product' [27]. The result of the Cartesian products represented in Eq. (1) is called the relation corresponding to the rule entry [27]. For example (see Figure 3.1), for Rule-1, we have the relation (denoted by R_1),

$$R_1 = a \times y \times \text{any} = \{(a, y, 1), (a, y, 2)\},$$

and for Rule-4, we have the relation (denoted by R_4),

$$R_4 = \text{any} \times \text{any} \times 1 = \{(a, x, 1), (a, y, 1), (b, x, 1), (b, y, 1)\}.$$

In general, for a given rule entry order number *i*, let Rule-*i* denote Rule *i*, and R_i denote the relation corresponding to Rule-*i* (without including any action).

For example, suppose that Rule-*x* is:

```
==================================================================
Rule No.     Source IP       Dest IP          Dest Port     ACTION
==================================================================
   X         10.1.1.1        20.2.2.0/30      80-81         Accept
==================================================================
```

Then, R_x is

```
{ ( 10.1.1.1, 20.2.2.0, 80 ),
  ( 10.1.1.1, 20.2.2.0, 81 ),
  ( 10.1.1.1, 20.2.2.1, 80 ),
  ( 10.1.1.1, 20.2.2.1, 81 ),
  ( 10.1.1.1, 20.2.2.2, 80 ),
  ( 10.1.1.1, 20.2.2.2, 81 ),
  ( 10.1.1.1, 20.2.2.3, 80 ),
  ( 10.1.1.1, 20.2.2.3, 81 ) }
```

We can apply the 2D-Box Model to the firewall rules. For example, we can define a range of IP addresses = {0.0.0.0–255.255.255.255}, or a range of port numbers = {0–65535} in IPv4 as illustrated in the above examples. Moreover, the 2D-Box Model can also be extended and applied on IPv6.

3.2.1 Limitations on the shadowed rule

Recall that a 'shadowed rule' is a rule that can never be matched by any packet because the packet should have been matched with one of the rules before this rule. For example, Rule-4 in Table 3.1 is a shadowed rule because any packet that matches this rule has already matched Rule-3 in order of precedence. Another example is Rule-3 on the right side of Figure 3.1. This limitation can cause security and speed problems.

Security problems are likely to occur, especially in an enterprise network that has a large number of rules in the firewall. For example, suppose that a new worm is sending packets to attack the network. After this attack is detected, the firewall administrator will add a new firewall rule for protection against such an attack. If this added rule is shadowed by old rules above, which allow attacking packets to go through, then the security problem definitely occur.

Speed problem can occur because many shadowed rules can waste the firewall processing time on these useless rules. Because most of packets will be matched with the last rule (i.e., the rule that denies all packets), the shadowed rules will have to be processed for matching the packets before the last rule. This can generate low throughput to the firewall.

In the following, we prove that shadowed rules are not necessary for firewall and can be deleted without any change to firewall policy.

Theorem 1. If Rule-*i* (i.e, the *i*-th rule) in a Listed-Rule firewall is a shadowed rule, then we can remove Rule-*i* without any changes to firewall rule policy.

Proof. Suppose that *p* matching a packet is an entry of the relation (i.e, a set represented by the Cartesian product defined in Eq. (1)) corresponding to Rule-*i*. Then, *p* should have also been an entry of the relation corresponding to one of the rules above Rule-*i*, according to the definition of a shadowed rule described at the beginning of this section. Therefore, we have

$$p \in R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1$$

This implies that

$$R_i - (R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1) = \phi.$$

Therefore,

$$R_i \cup R_{i-1} \cup \ldots \cup R_1 = R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1.$$

This concludes that

$$R_{i+1} - (R_i \cup R_{i-1} \cup \ldots \cup R_1) = R_{i+1} - (R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1),$$

i.e., deleting Rule-*i* or not will not affect the rules after Rule-*i* and hence does not change the rule policy of the firewall.

3.2.2 Limitation about swapping position between rules

Swapping the positions of two rules on a Listed-Rule firewall can cause policy changes on the firewall if the two rules have different actions, and both of them can be matched with the same packet. For example, swapping between Rule-7 and Rule-8 (see Table 3.1) will change the action on the packet (with Source IP = 10.3.3.9, Destination IP = 20.3.3.9, Destination Port = 80) from being accepted to being denied. Changing the packet action from being accepted to being denied can also be a security problem. For example, if packets that send/receive between clients and servers are blocked, it can be deemed as another security problem because of the lack of Availability (ready to use). Moreover, security problems are likely to occur if dangerous packets that must be denied are accepted because of the rule swapping. The above limitations are further described in the following theorem.

Theorem 2. Let Rule-*x* and Rule-*y* ($y > x$) be two rule entries on a Listed-Rule firewall and have different actions, and *K* be $R_{x-1} \bigcup R_{x-2} \bigcup \ldots \bigcup R_1$. Then, swapping the positions of Rule-*x* and Rule-*y* will cause rule policy changes on the firewall if $(R_x \bigcap R_y) - K \neq \phi$.

Proof. Note that

$$(R_x \bigcap R_y) - K \neq \phi.$$

Thus, we have that

$$(R_x \bigcap R_y) - K = (R_x - K) \bigcap (R_y - K) \neq \phi. \tag{2}$$

Equation 2 indicates that there is a *relation* entry, *p*, that will fall into both $R_x - K$ and $R_y - K$. Let *P* be a packet that matches *p*, then before swapping the positions of Rule-*x* and Rule-*y*,

$$p \in R_x - K, \tag{3}$$

and *P* is taken the action defined for Rule-*x*. After swapping the positions of Rule-*x* and Rule-*y*, Rule-*y* becomes the first rule after Rule-(*x*-1),

$$p \in R_y - K, \tag{4}$$

and *P* is taken the action defined for Rule-*y*. By the assumption of this theorem, the actions taken before and after swapping the positions of Rule-*x* and Rule-*y* (corresponding to Equations 3 and 4 respectively) are different, so the swapping operation changes the rule policy of the firewall.

3.2.3 Limitation about redundant rules

A redundant rule is a rule that is redundant to (or has been implied in) another rule below it with the same action. For example, Rule-8 in Table 3.1 is redundant to Rule-9. As another example, Rule-4 in Figure 3.1 is redundant to Rule-5. If we remove a redundant rule, a firewall policy should not change. Redundant rules can cause a speed problem because many redundant rules can waste the firewall processing time. We prove below that redundant rules are not necessary and can be deleted without any change to firewall policy.

Theorem 3. Suppose that Rule-*i*, Rule-(*i*+1), Rule-(*i*+2), …, Rule-(*i*+*n*) on a firewall have the same action (where '*n*' is a positive integer). If

$$R_i \subset R_{i+1} \cup R_{i+2} \cup \ldots \cup R_{i+n},$$

Then, removing Rule-*i* will not make any change of policy on the firewall.

Proof. Let

$$R_A = R_{i+1} \cup R_{i+2} \cup \ldots \cup R_{i+n}.$$

Before removing Rule-*i*, let

$$p \in R_i - (R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1).$$

Note that $R_i \subset R_A$, so

$$R_i - (R_{i-1} \cup R_{i-2} \cup \ldots \cup R_1) \subset R_i \subset R_A .$$

Therefore, after removing Rule-*i*,

$$p \in R_{i+1} \cup R_{i+2} \cup \ldots \cup R_{i+n},$$

and packets that used to match *p* will fall down to match with Rule-(*i*+1), Rule-(*i*+2), ..., or Rule-(*i*+*n*) in order. Because all of Rule-*i*, Rule-(*i*+1), ..., and Rule-(*i*+*n*) have the same action, there are no changes to the policy after removing Rule-*i*.

3.2.4 Limitation of rule design

In the rule design process of a Listed-Rule firewall, firewall administrators have to put ‘bigger rules’ after ‘smaller rules’.

Note that

– a ‘bigger rule’ is a rule that can be mapped into a ‘bigger relation’, where

– a ‘bigger relation’ is a relation that is bigger than some other relations (i.e., a superset of other relations).

An example of bigger rule is the last rule in a Listed-Rule firewall (e.g., Rule-18 in Table 3.2), which is bigger than every other rule above. We cannot move this rule to the first position because it can shadow all other rules if we do.

Table 3.2: An example of rules on a medium size network [28]

No.	Source_IP	Dest_IP	Dest_Port	Action
1	200.1.2.99	200.1.1.3	22	Accept
2	200.1.2.99	200.1.1.4	22	Accept
3	200.1.2.*	200.1.1.2	22	Accept
4	200.1.2.*	200.1.1.5	22	Accept
5	200.1.2.*	200.1.1.3	110	Accept
6	200.1.2.*	200.1.1.3	143	Accept
7	200.1.2.*	200.1.1.5	3306	Accept
8	*	200.1.1.1	22	Accept
9	*	200.1.1.1	80	Accept
10	*	200.1.1.2	80	Accept
11	*	200.1.1.2	443	Accept
12	*	200.1.1.3	25	Accept
13	*	200.1.1.4	53	Accept
14	200.1.2.*	200.1.1.*	*	Deny
15	200.1.2.*	*	*	Accept
16	200.1.1.3	*	25	Accept
17	200.1.1.4	*	53	Accept
18	*	*	*	Deny

As another example of bigger rule, if we want to prevent normal users from attacking the servers in a DMZ (see Figure 3.2) but allow admin people to manage servers through port 22, we have to prevent access of normal users using Rule-14 (in Table 3.2) but allow access of admin people using Rule-1 and Rule-2. As a result, Rule-14 is a bigger rule compared with Rule-1 (and Rule-2), and we have to locate Rule-14 after Rule-1 (and Rule-2). With this limitation, it is difficult to design rules on a Listed-Rule firewall because rule positions are not independent. Moreover, in a Listed-Rule firewall, the biggest rule (e.g., Rule-18 in Table 3.2) cannot be moved upward to other positions. This also causes a speed problem because the packets that only match this biggest rule (i.e., the last rule) will have to go through all other rules first to find any possible matching.

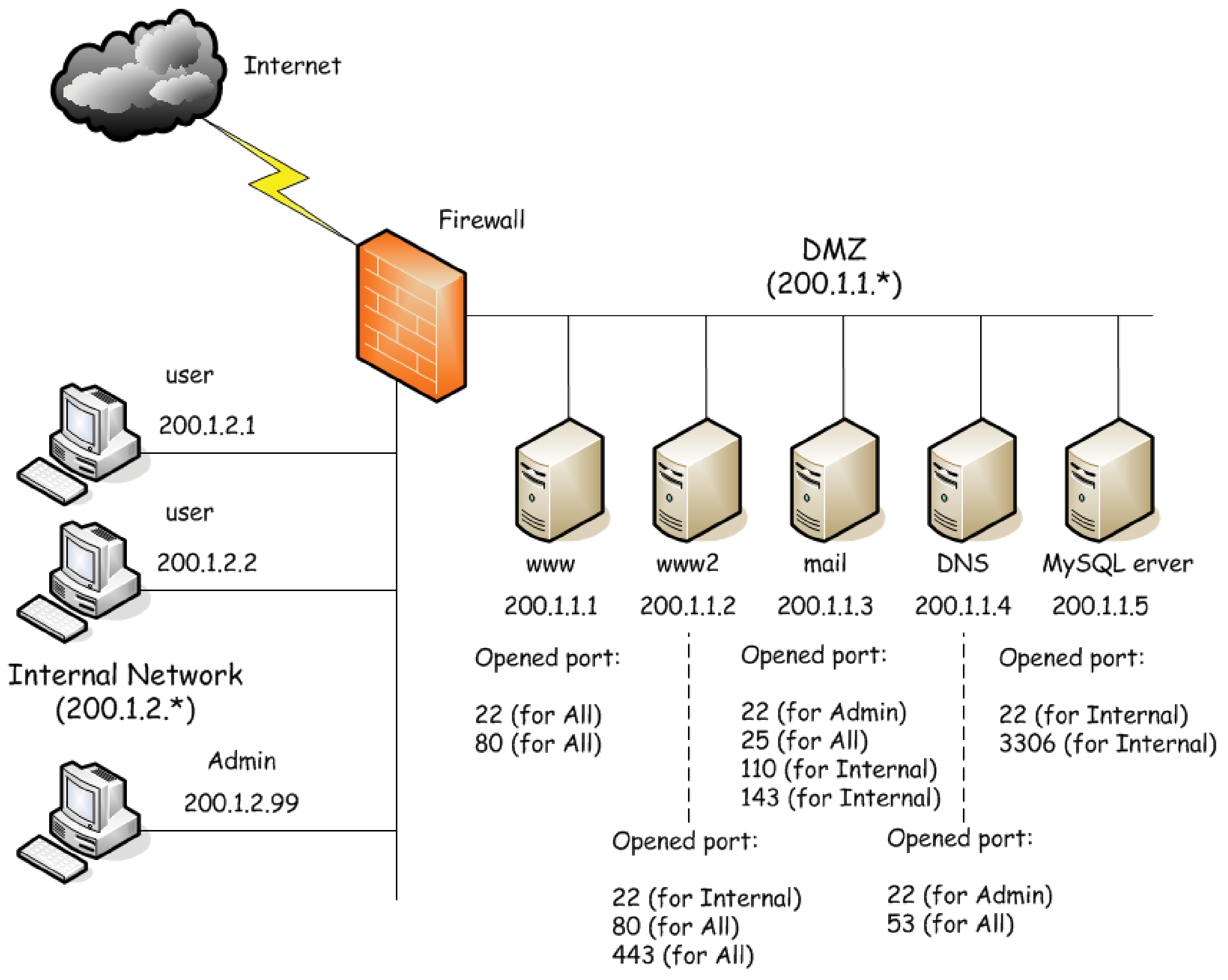


Figure 3.2: A medium size network with a DMZ [28]

3.2.5 Limitation from sequential computation

Rule computation for packet decision on a Listed-Rule firewall is a sequential process. Consequently, it may cause a speed problem. Especially, a firewall that has a large number of rules may work with a slow speed. The time used for rule computation would depend on the number of rules of the firewall.

Let the number of rules in a Listed-Rule firewall be *N*. Then, the number of rules (on average) that will be compared with a packet is *N*/2, and the time to compute each packet's matching is $t \in O(N)$.

3.3 Conclusion

In this chapter, we identified five important limitations on Listed-Rule Firewall which may lead to security problem, speed problem, and "difficult to use" problem. These limitations consist of (1) limitation about "Shadowed rules" which can lead to security and speed problem, (2) limitation about swapping position between rules which cause security problem, (3) limitation about "Redundant rules" which can cause speed problem, (4) limitation of rule design that can result in a "difficult to use" problem, and (5) limitation from sequential computation that can lead to speed problem. We presented various theories and their proof to validate our arguments for the above limitations.

Chapter 4 Tree-Rule Firewall

This chapter proposes a new model of firewall called the 'Tree-Rule Firewall', which offers various benefits and is applicable for large networks such as 'cloud' networks. In a Tree-Rule firewall, the rule positioning is based on a tree structure instead of traditional rule listing. To manage firewall rules, we implement a Tree-Rule firewall on the Linux platform and test it on a regular network and under a cloud environment respectively to show its performance. It is demonstrated that the Tree-Rule firewall offers better network security and functional speed than the Listed-Rule firewall. Compared to the Listed-Rule firewall, rules of the Tree-Rule firewall are easier to be created, especially on a large network such as a cloud network.

4.1 Background and Related Works

Nowadays, 'cloud' as an architectural structure has been broadly employed so that users can use cloud computing with optimal benefits including fast processing and network speed, effective data distribution and low cost. However, cloud computing always encounters new problems such as problematic network security. Technically, cloud computing normally functions on the virtual system of a network connected to various organizations mostly working on the same physical network or sometimes on the same physical machine. At the point where the network security is critical, security devices, such as firewalls [29] and Intrusion Detection Systems (IDSs) [30], are applied in a cloud network.

4.1.1 Firewall background

There have been many studies about firewall rule conflicts (anomalies) that occur within rule sets (e.g., the rule set shown in Table 4.1 for a traditional Listed-Rule firewall).

Table 4.1: An example of rules on the Listed-Rule Firewall [28]

No.	Protocol	Source_IP	Destination_IP	Destination_Port	Action
1	TCP	10.1.1.1	20.1.1.1	80	Accept
2	TCP	10.1.1.2	20.1.1.1	80	Deny
3	TCP	10.1.1.0/24	20.1.1.1	80	Deny
4	TCP	10.1.1.3	20.1.1.1	80	Accept
5	TCP	10.2.2.0/24	20.2.2.5	80	Deny
6	TCP	10.2.2.5	20.2.2.0/24	80	Deny
7	TCP	10.3.3.0/24	20.3.3.9	80	Accept
8	TCP	10.3.3.9	20.3.3.0/24	80	Deny
9	IP	0.0.0.0/0	0.0.0.0/0	0-65535	Deny

E-hab Al Shaer et al. [12] proposed several anomaly definitions including ‘shadowing anomaly’. They defined ‘Shadowed Rule’ as a rule that cannot be matched by any packet. Therefore, this type of rules should be removed from the rule list without any changes to a firewall policy. Moreover, they also applied their definitions and theories for analysing a distributed firewall [23]. The authors of [12][23][24] focused on mathematically analysing firewall rules. To get rid of the rule conflicts in a firewall, Lihua Yuan et al. proposed the Fireman Toolkit [31], which could help administrators to design and analyse firewall rules. However, their toolkit only mitigated some problems of traditional firewall, and the Fireman Toolkit was not a new type of firewall compared with the traditional Listed-Rule firewall (see, for example, Table 4.1). Liang Zhao et al. [21] proposed to use ‘goto’ function inside Listed-Rule firewalls (e.g., a ‘jump’ command in IPTABLES). Although their rule structure looks like a tree structure, their sub-rules (or nodes) contain Listed-Rules. Therefore, their firewalls are still deemed as Listed-Rule firewalls and are time consuming when performing linear and sequential rule searching. Our research in this chapter proposes a new type of firewall, which has different mechanisms. We use a tree-shape and hierarchical rule set. Although the phrase 'hierarchical rule set' [32] has appeared in the manuals of CSS (Cisco Services Switch), those rules are relevant to load-balancing devices in a network rather than on the firewall and merely describe which content (e.g., an html file) is accessible by visitors to the Website. Alex Liu and Mohamed Gouda proposed 'Diverse Firewall Design' [13] using tree structure rules translated from a rule list to discover and eradicate some of the rule conflictions. However, their work was still based on the traditional firewall design.

4.1.2 Firewall on cloud environment

Dimitrios Zissis et al. [33] addressed cloud computing security and focused on cryptography. Seoksoo Kima et al. [34] proposed an enterprise security management system with reinforced internal security. However, the work presented in both [33][34] did not focus on the firewall directly. A firewall can be implemented on cloud environment using hardware or software. If a software firewall (e.g., IPTABLES installed on Guest OS) is applied, the firewall position is as shown in Figure 4.1(a) [35]. On the other hand, if a hardware firewall is applied, the firewall position differs from what is shown in Figure 4.1(a) [36] and the firewall is not in the hypervisor (i.e., a piece of computer software, firmware or hardware that creates and runs virtual machines) [37] although the levels of network security remain the same. The firewall positioning in Figure 4.1(a) has its own benefits because it does not consume much resource due to only a single firewall computer required. However, this positioning still faces security problems because a virtual machine (VM) behind the firewall may be attacked by other VMs situated in the same domain.

Therefore, to upgrade the security level, one proposal is to reposition the firewalls as shown in Figure 4.1(b) [35].

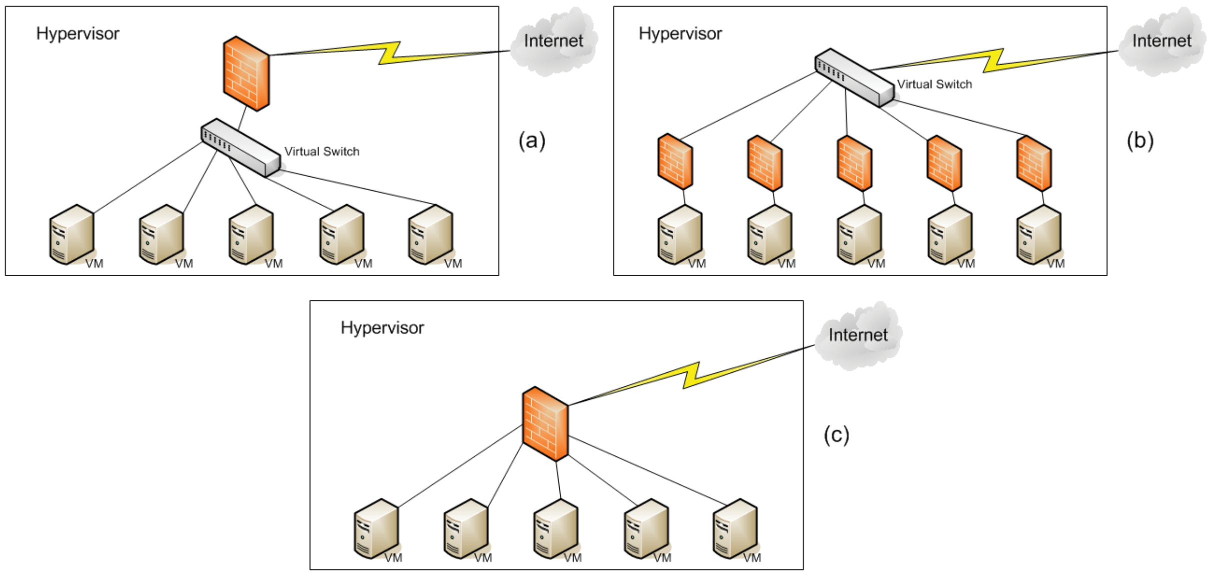


Figure 4.1: Firewall models in cloud environment [35][38]

However, this positioning consumes much more resource (e.g., disk space, RAM, and hypervisor's CPU). To resolve this problem, it is proposed that the firewalls are repositioned as shown in Figure 4.1(c) [35] and this proposal consumes less resource but requests more rules in the firewall than the one shown in Figure 4.1(a). The added rules are the ones for preventing the attacks between VMs. To overcome the problems, we will present our Tree-Rule firewalls to support the third model (Figure 4.1(c)) so that the firewall will consume less resource, process rapidly, and show no rule conflicts.

4.1.3 Chapter organization

The aim of this chapter is to describe the existing work related to our work. In Section 4.2, we will propose a new firewall model called the Tree-Rule firewall. We will implement and perform our newly designed the Tree-Rule firewall on the Linux platform. The experimental results will be shown in Section 4.3. We will test our proposed Tree-Rule firewall using a regular network and under a cloud environment to compare its performance efficiency under such scenarios. This chapter will be concluded in Section 4.4.

4.2 Design and implementation of the Tree-Rule firewall

This section presents the design and implementation of the newly proposed Tree-Rule firewall. The basic design [28][38] is presented and illustrated in subsection 4.2.1. We will analyse the benefits of the proposed firewall. In subsection 4.2.2, we will improve the basic design and analyse the time complexity. The implementation of Tree-Rule firewall is shown in subsection 4.2.3.

4.2.1 Basic design

The design of our proposed 'Tree-Rule firewall' is shown in Figure 4.2. This design can avoid the five limitations of Listed-Rule firewall as presented in the previous chapter.

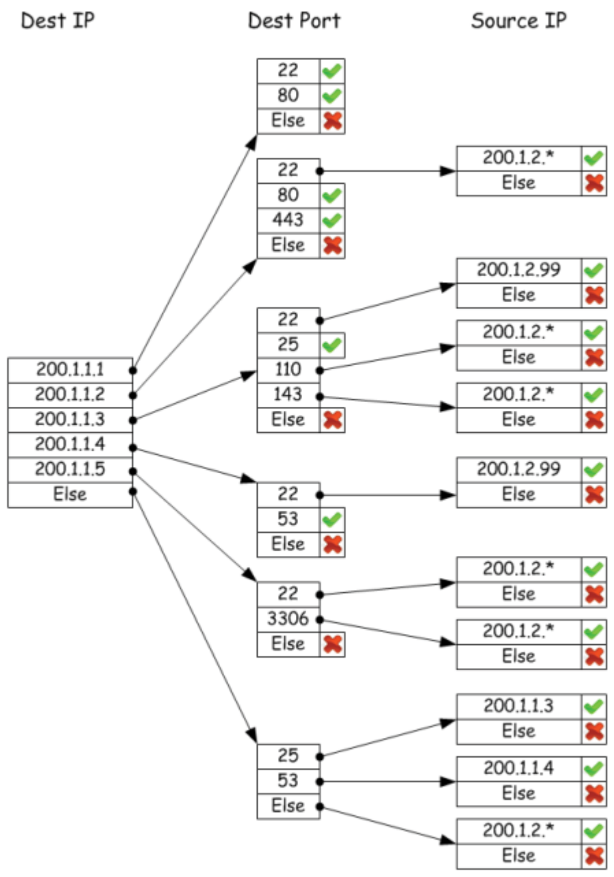


Figure 4.2: A basic Tree-Rule firewall structure [28][38].

In this subsection, we will explain the advantages of Tree-Rule firewall including:

– no shadowed rules,

– no need of rule swapping because all rules will be sorted automatically,

– no redundant rules,

– ease of rule design (with independent rule paths), and

– high speed for packet access decision.

The Tree-Rule firewall is a new kind of firewall in which the rules are presented in a tree form (see Figure 4.2) instead of a list of lines (i.e., rules). This firewall will read attribute information from packet header and compare packet's first attribute with the data in the root nodes in the rule tree. After that, the firewall will check packet's other attributes in order by searching only on relevant nodes at the corresponding levels. As a result, the packet will be decided quickly with a specific action. For example, as shown in Figure 4.2, when packets arrive at the Tree-Rule firewall, the firewall will consider Dest IP (destination IP address), Dest Port (destination port), and Source IP respectively in order until packets' access decisions are made by predefined actions.

Actually, an attribute within the root node can be Source IP, Destination Port, or any attribute suitable to work with the firewall rules. Users can select attributes that they want for each column before creating tree rules. For example, if we focus on the protection for servers inside our network, we should select Destination IP to be the root node. This is because we can easily imagine that targeted servers are important sources of information and their IP addresses are most significant to block any misused information from them. On the other hand, if we focus on permissions for users, we should specify Source IP to be the root node to allow where (destination IP addresses) they (Source IP addresses) want to go. We have created a Graphic User Interface (GUI), a rule editor, where users can specify attributes for each column easily. In this study, we use Destination IP for the root node.

As we can see in Figure 4.2, the Tree-Rule firewall has no security problem because the users do not need to swap rule positions. Also, the Tree-Rule firewall has no rule numbers. Instead, we call each path of the tree a 'Rule Path'. Data in each node will be sorted in the ascending order. Rule designers are not necessary to have any skills. They need only basic concepts for Tree-Rule firewall design. This means that Tree-Rule firewall's rules are easy to design.

Moreover, the rules (or rule paths) that will be matched by almost all packets (such as the rule on the bottom path that shows 'Else → Else → Else → Deny') in Figure 4.2 will take the same length of time to make packet access decision (Accept or Deny) as other rules (or rule paths).

4.2.1.1 Time complexity of the basic design

With regard to performance, the time complexity for making a packet access decision in a Listed-Rule firewall is O(*N*), where *N* is the number of rules. The time complexity in a Tree-Rule firewall is in the order of log(*N*).

Table 4.2: An example of rules on a medium size network [28]

No.	Source_IP	Dest_IP	Dest_Port	Action
1	200.1.2.99	200.1.1.3	22	Accept
2	200.1.2.99	200.1.1.4	22	Accept
3	200.1.2.*	200.1.1.2	22	Accept
4	200.1.2.*	200.1.1.5	22	Accept
5	200.1.2.*	200.1.1.3	110	Accept
6	200.1.2.*	200.1.1.3	143	Accept
7	200.1.2.*	200.1.1.5	3306	Accept
8	*	200.1.1.1	22	Accept
9	*	200.1.1.1	80	Accept
10	*	200.1.1.2	80	Accept
11	*	200.1.1.2	443	Accept
12	*	200.1.1.3	25	Accept
13	*	200.1.1.4	53	Accept
14	200.1.2.*	200.1.1.*	*	Deny
15	200.1.2.*	*	*	Accept
16	200.1.1.3	*	25	Accept
17	200.1.1.4	*	53	Accept
18	*	*	*	Deny

For example, for the Listed-Rule firewall rules shown in Table 4.2, if we assume that the chances for rules to be matched by packets are equal. Then, it will take (18/2) × 3 × C = 27C (where C is the time interval that is used for comparing between 'one attribute of packet header' and 'one attribute of rule'). On the other hand, the Tree-Rule firewall in Figure 4.2 (where Dest IP has 6 lines, Dest Port has 4 lines (on average), and Source IP has 2 lines (on average)) will take a time less than C × Log 8 + C × Log 4 + C × Log 2 = C × (3 + 2 + 1) = 6C. Note that all 'Log' values above are of base 2.

We use an enterprise network as another example to compare the computation complexity using a Listed-Rule firewall and a Tree-Rule firewall. Assume that the enterprise network consists of approximately 100 servers, and each server opens about 20 ports and has approximately 5 groups of users requesting access to the DMZ and the Internet. Then, there are approximately 100 × 20 × 5 = 10,000 rules on the Listed-Rule firewall on the network to control the access to the DMZs and the Internet. Therefore, on average, it will take about (10,000/2)×3×C = 15,000C to make the access decision for each packet and require 10,000 × 3 × C = 30,000C in the worst case. In contrast, using a Tree-Rule firewall, it takes less than C × Log 128 + C × Log 32 + C × Log 8 = C × (7 + 5 + 3) = 15C to make the access decision for any packet in the worst case.

Note that, in the above description,

128 is the number of destination IP addresses rounded up from 100,

32 is the number of destination Ports rounded up from 20, and

8 is the number of source IP addresses rounded up from 5.

As we can see, the above time complexity comparison between a Tree-Rule firewall and a Listed-Rule firewall is similar to the comparison between 'Binary Search Tree' [39] and 'Linear Search' [40] in an Array.

4.2.1.2 Additional benefits of the basic design

On the 'security' aspect, a Listed-Rule firewall on an enterprise network (having many rules) is likely to encounter with confliction of rules. Such conflictions include shadowed and redundant rules. In contrast, a Tree-Rule firewall creates no rule conflictions because the Tree-Rule firewall can never contain any shadowed rules or redundant rules.

On the 'easy to use' aspect, it is very difficult to design rules for a Listed-Rule firewall following the policy of an organization. With many lines of rules, it is difficult to check and test how each rule works. On the contrary, a Tree-Rule firewall can be designed easily because every rule sentence (i.e., rule path) of Tree-Rule firewall takes a separate path.

4.2.2 Improvement of basic design

In most cases, many computers need to be protected with the same policy. In the basic design of Tree-Rule firewall (see Figure 4.2), the root node (column of 'Dest IP') has the number of lines equal to the number of user's computers. Each line is linked to some sub-trees with repeated (the same) data. To get rid of the above-mentioned problems and to improve the basic design, the 'Single IP Address' design (as shown in Figure 4.2) is replaced by 'IP Address Range' design (see Figure 4.3). In corresponding to the changes, those single destination ports (i.e., Dest Ports) shown in Figure 4.2 are replaced by the port ranges shown in Figure 4.3.

In Figure 4.3, it can be seen that the IP Addresses between 200.1.2.1–200.1.2.254 apply the same policies that allow 200.1.1.1–200.1.1.254 to remotely access to port number 22. Using the improved design, 253 lines can be saved on the root node, and the memory spaces that were previously requested in the basic design for the 253 sets of the repeated data (sub-trees) can now be saved.

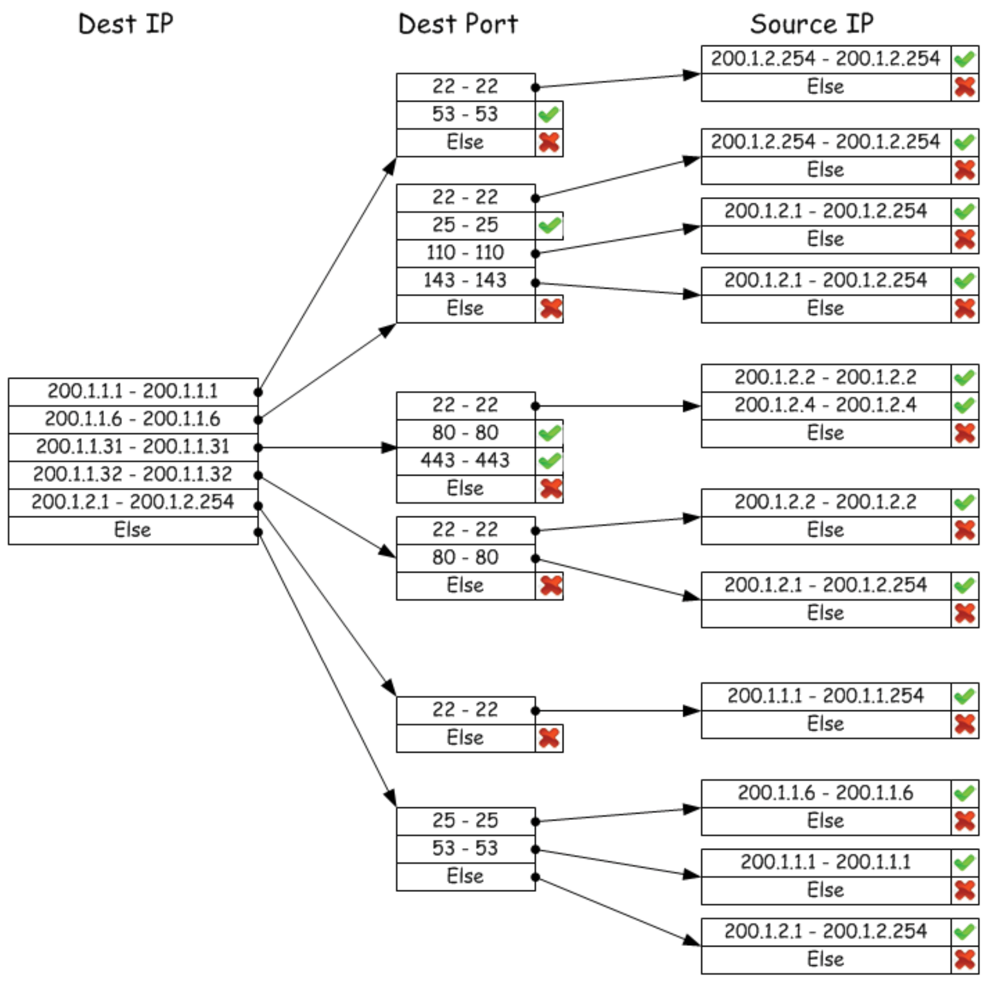


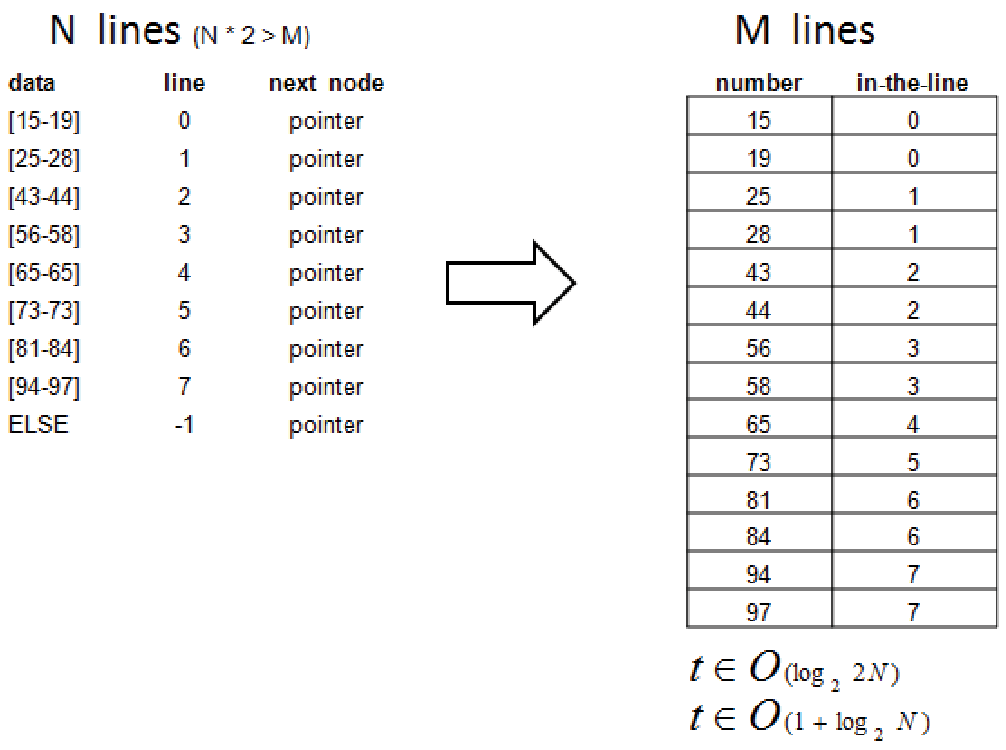
Figure 4.3: Improved design of Tree-Rule firewall using IP address ranges and port ranges [38]

4.2.2.1 Time complexity of improved design

We have shown that the improved design can save memory spaces. We now show that the rule searching time increases only a bit because of the change from a single number design to a range design. It is found that the searching time within the node slightly increased as shown below.

$$t \in O(\log_2 N) \text{ is changed to } t \in O(1 + \log_2 N)$$

as shown in Figure 4.4.



N lines (N * 2 > M)

data	line	next node
[15-19]	0	pointer
[25-28]	1	pointer
[43-44]	2	pointer
[56-58]	3	pointer
[65-65]	4	pointer
[73-73]	5	pointer
[81-84]	6	pointer
[94-97]	7	pointer
ELSE	-1	pointer

M lines

number	in-the-line
15	0
19	0
25	1
28	1
43	2
44	2
56	3
58	3
65	4
73	5
81	6
84	6
94	7
97	7

$$t \in O(\log_2 2N)$$
$$t \in O(1 + \log_2 N)$$

Figure 4.4: Slightly increased *t* when a range of number is applied

As shown in Figure 4.4, a new attribute called ‘in-the-line’ is added to the data structure of a node to help search the data using a Binary Search algorithm (in array). The searching results as shown on the right part of Figure 4.4 can be that

1. the number is certainly found (e.g., searching for number 25),
2. the number is not found but still on the range (e.g., searching for number 17), and
3. the number is not found and not on the range (e.g., searching for number 20).

In the first two cases, the firewall will check ‘Dest Port’ (see Figure 4.3) based on the link indicated in the ‘in-the-line’ (see Figure 4.4). For example, when searching for number 17, the firewall will check the ‘Dest Port’ on link 0. To search for numbers within a node, we need only use a regular Binary Search algorithm, e.g., the algorithm presented on http://www.cosc.canterbury.ac.nz/mukundan/dsal/BSearch.html, and utilise the ‘first’, ‘mid’ and ‘last’ parameters to see that the searching result belongs to which of those three cases listed above.

Regarding time consuming in a node, it can be concluded that

$$t \in O(1 + \log_2 N)$$

where N is the row number within a node (see the left figure of Figure 4.4). The data searching on the nodes will not be more than 3 times per packet and they will only occur in the root node, ‘Dest Port’ node and ‘Source IP’ node. If the searching times of the three nodes are t_1, t_2, and t_3, respectively, the access decision time per packet will be $t = t_1 + t_2 + t_3$. If the numbers of the lines within the three columns are N_1, N_2, and N_3, respectively, the decision time per packet is estimated as:

$$t \approx k_1(1 + \log_2 N_1) + k_2(1 + \log_2 N_2) + k_3(1 + \log_2 N_3)$$

where k_1, k_2, and k_3 are all constants.

4.2.3 Implementation

The firewall implementation is conducted on Cent OS Linux with the Tree-Rule firewall functioned on Netfilter (see Figure 4.5).

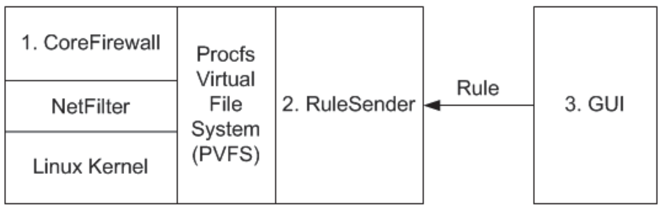


Figure 4.5: Implementation of Tree-Rule firewall

Similar to IPTABLES, we focus on the network firewall that verifies the packet forwarding between the network interfaces. Our algorithm, NF_IP_FORWARD, includes three programs as follows.

1. Core Firewall. It is written with C language on Linux in order to detect the packets and make a decision on tree rule regulation for whether the packets should be accepted or dropped. This program runs on the Kernel Space with a file type of ‘‘.ko’’.

2. Rule Sender. This is written with C language on Linux in order to receive tree rules from GUI (running on the user's Windows XP/7/8). Rule Sender would send the tree rules to Core Firewall through ‘procfs Virtual File System’, a specific memory for the data exchange between the regular software and the software functioning on Kernel. This Rule Sender runs on the User Space (not the Kernel Space).

3. GUI. It is written with C# language on Windows in order to communicate with users so that each user could create a graphical tree rule. After creating and editing a tree rule, the user can either save or send/apply the rule to the firewall so that the rule can be functioned. GUI will communicate with the ‘Rule Sender’ on the firewall. To install and run a Tree-Rule firewall, users do not need to uninstall the IPTABLES software from the system but run both of them together. Before running the Tree-Rule firewall, users must launch a command ‘service iptables stop’. Meanwhile, running the Tree-Rule firewall can be commanded with ‘insmod CoreFirewall.ko’ and ‘./RuleSender’, respectively.

4.3 Experimental results

The firewall is tested on both regular networks (LANs) and cloud environment. The results are presented in subsections 4.3.1 and 4.3.2, respectively.

4.3.1 Testing in LAN

We conduct the performance test to compare between Tree-Rule firewall and IPTABLES (a free firewall popularly used on Linux) on the same computer and OS.

For testing, we use three computers (Intel 2.8 GHz CPU with 4GB RAM) according to their roles as the Firewall, the Target, and the Attacker. The Firewall has two network cards (100 mbps speed) to be directly connected with the target and the attacker. The firewall computer is located in the middle. Testing is conducted to assess the CPU load and throughput when the number of the rules is increased. At an early stage, we add a similar number of firewall rules that users normally used, and the number is between 100 and 200 rules or rule paths. In the Tree-Rule firewall, the number of the rule paths is calculated based on the total number of the actions as illustrated in Figure 4.3. We generate the rules by randomising the Source IP addresses, Destination IP Addresses and Destination Ports, and define the actions as ‘Accept’ to allow all packets passing through the firewall (in order to obtain accurate throughput). Besides, we boot the Attacker computer with Back Track 5 R3 and generate the packets with hping3 command to create the packets with randomised IP addresses and ports before sending them to the Target as many as possible with the maximum rapidity by using ‘--flood’ parameter. The testing results are shown in Figures 4.6 and 4.7.

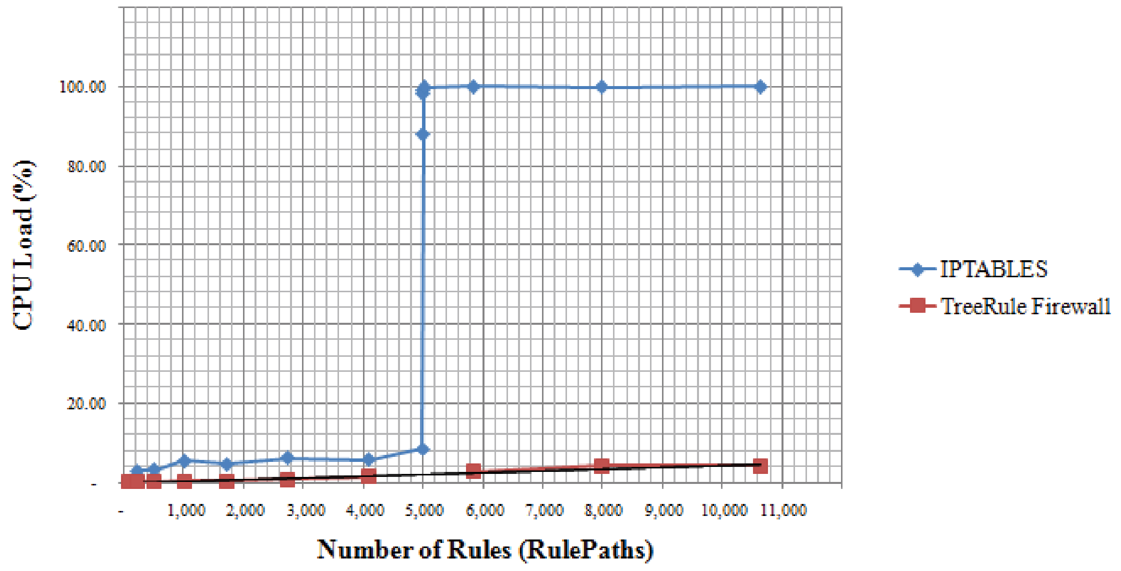


Figure 4.6: CPU load of firewalls

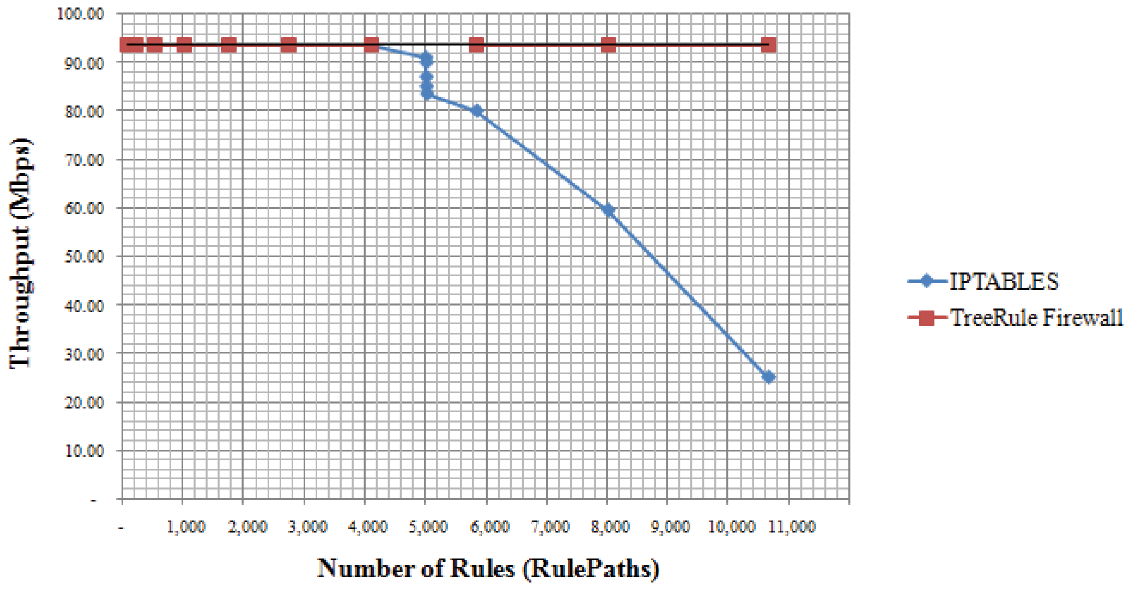


Figure 4.7: Throughput of firewalls

From the results shown above, it is found that using 200–4000 rules or rule paths requires a small space of CPU (3%–5% for IPTABLES and 0.4%–1.0% for Tree-Rule firewall); meanwhile, the throughput using either firewall is close to 100% (93.6% for both IPTABLES and Tree-Rule firewalls). Then, the rule number is constantly enlarged. We discover that if the rule number is increased to more than 5000 rules, the computer with IPTABLES needs more space on the CPU and the CPU usage is close to 100%, whereas the computer with Tree-Rule firewall requires only up to 4.3% of CPU. In terms of throughput, if there are more than 5000 rules, the throughput of the computer with IPTABLES will significantly decrease. However, the throughput of the computer with the Tree-Rule firewall remains regularly high.

4.3.2 Testing on cloud environment

The most popular four hypervisors are ESXi (from VMware), KVM, Hyper-V (from Microsoft) and Xen Server [41-48]. In this research, we pay attention to ESXi and Hyper-V only because Hyper-V is easy to use (it is a Microsoft Windows based server) and ESXi is very reliable (noting that VMware Company is the pioneer of Virtual Machines). The Tree-Rule firewall is tested on both operating systems. Typically, ESXi has its own firewall built in but Hyper-V has not. However, there is a firewall called the '5Nine vFirewall' which has been developed to suit functioning on Hyper-V and is recently popular. Both the ESXi firewall and 5Nine vFirewall are Listed-Rule firewalls. In this subsection, we compare the CPU loads and throughputs between the Tree-Rule firewall and IPTABLES on ESXi and Hyper-V respectively. We also indicate the disadvantages of ESXi's own firewall and '5Nine vFirewall' on Hyper-V compared with the Tree-Rule firewall.

4.3.2.1 Testing on ESXi

From the test, it is found that ESXi firewall protects only ESXi itself (i.e., the Hypervisor) but is unable to protect the internal virtual machines. Therefore, to keep the internal virtual machines protected, users require additional firewalls.

Moreover, we test both Tree-Rule firewall and IPTABLES on ESXi with 2^3, 4^3, 6^3, ..., and 16^3 rules (i.e., 8, 64, 216, ..., and 4096 rules) respectively. The results showing the throughput of these firewalls are demonstrated in Table 4.3 and Figure 4.8.

Table 4.3: Throughput comparison between Tree-Rule firewall and IPTABLES on ESXi and Hyper-V

	On ESXI		On Hyper-V	
No. of rules	Tree-Rule Firewall	IPTABLES	Tree-Rule Firewall	IPTABLES
8	96.45	99.13	93.12	98.59
64	95.47	98.43	92.14	93.19
216	96.33	95.61	89.04	85.72
512	95.16	89.62	87.22	75.20
1,000	96.58	80.34	84.79	62.66
1,728	93.54	71.33	84.49	46.48
2,744	92.30	59.82	87.70	35.61
4,096	94.02	47.11	88.12	24.85

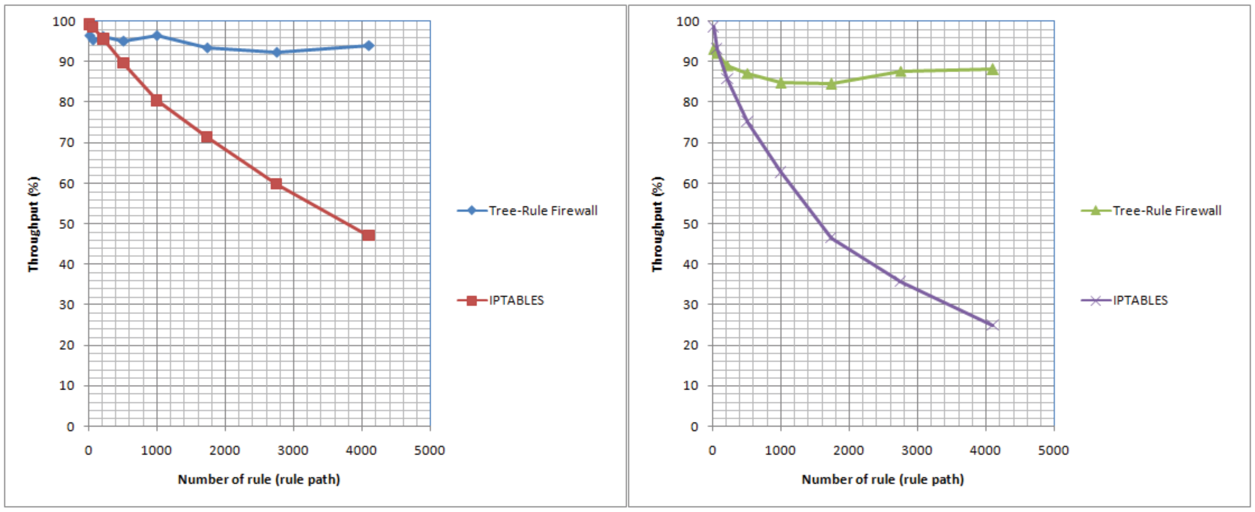


Figure 4.8: Throughput of firewalls experimented on ESXi (left) and Hyper-V (right)

4.3.2.2 Testing on Hyper-V

Recall that 5Nine vFirewall is a firewall developed for functioning on Hyper-V. The benefit of 5Nine vFirewall, compared with ESXi, is that 5Nine vFirewall can protect the virtual machines from the external attacks and prevent the attacking between the virtual machines [49][50]. However, one disadvantage of vFirewall is that it is required to install agents inside the virtual machines and this is not feasible for the virtual machines with Linux OS since they do not have an agent. Another disadvantage of vFirewall is the instability. For example, with a huge number of rules, the firewall will set a virtual machine into a PUASE state.

We also test both Tree-Rule firewall and IPTABLES on Hyper-V with 2^3, 4^3, 6^3, ..., and 16^3 rules (i.e., 8, 64, 216, ..., and 4096 rules) respectively. The results showing the throughput of these firewalls are also demonstrated in Table 4.3 and Figure 4.8.

4.3.2.3 Testing analysis

In Table 4.3, the percentage for the throughput is obtained by dividing the actual throughput (in Mbps) by the maximum throughput (i.e., the forwarding rate without applying any firewall), and then multiplying the result by 100. In our experiments, the maximum throughput of the firewall computer is 615Mbps when no firewall rules are applied. This number depends on CPU clock speed, OS version, quality of hardware, NICs, etc.

We only compare the throughput of IPTABLES and Tree-Rule firewall because thousands of IPTABLES rules can be created using shell scripts, and the rules for a Tree-Rule firewall under a cloud environment can easily be produced by modifying our source code written for the GUI of Tree-Rule firewall creation (see subsection 4.2.3). Because the proposed Tree-Rule firewall has three attributes (i.e., Source IP, Dest IP and Dest Port), its rules can be easily produced using a three layer programming loop.

On the other hand, it is not easy to create thousands of 5Nine vFirewall rules or ESXi's rules using the corresponding GUI. As a result, we are not able to test the speeds of the ESXi firewall and 5Nine vFirewall when the sizes of their rule lists are large. Nevertheless, we have investigated and found that when the positions of rules are swapped, these rules create all types of the rule conflicts that we have mentioned in the previous chapter, because these two firewalls are Listed-Rule firewalls.

From the three types of the firewall positioning (as previously mentioned in subsection 4.1.2), the conclusion and analysis are made as follows.

A. Positioning a single firewall to prevent the external attacks (Figure 4.1(a))

• The ESXi firewall is capable, but only protects the ESXi itself, not the internal virtual machines.

• The 5Nine vFirewall is capable and can protect the internal virtual machines.

• The Tree-Rule firewall and IPTABLES are capable. If there are too many virtual machines, the number of the rules will increase and the IPTABLES will encounter some problems (some Cloud Service Providers may contain more than 40,000 of the internal virtual machines). Besides, with too many rules, the rule creation/modification may simply cause redundant rules and conflict rules, which mostly occur with the firewall employing a listed rule (e.g., IPTABLES, Cisco ACL and other firewalls recently available).

B. Positioning the firewalls based on the number of VMs or Cloud networks/Company networks (Figure 4.1(b))

• The ESXi firewall is incapable since it is unable to protect the internal virtual machines.

• The 5Nine vFirewall is capable and gives the same results as the topology does. It needs neither installing additional software nor rearranging the virtual network within the hypervisor. Although it is flexible, it can only run on Microsoft platforms.

• The Tree-Rule firewall and IPTABLES are capable. Compared with the 5Nine vFirewall, these two firewalls require much more disk spaces and RAMs, since they need to be installed on every position of virtual machines (or cloud networks/company networks), which is time consuming. Moreover, the virtual network inside the hypervisor needs to be rearranged. Differently, when comparing between the Tree-Rule firewall and IPTABLES, both of them require equal disk spaces and RAMs since they consume less resource compared to the size of their OS (Linux).

C. Positioning a single firewall with various interfaces (Figure 4.1(c))

• The ESXi firewall is incapable.

• The 5Nine vFirewall is capable for this topology, with neither additional software nor rearranging the virtual network inside the hypervisor. This firewall is flexible since it can communicate directly with the hypervisor. However, it can only work on Microsoft platforms.

• The Tree-Rule firewall and IPTABLES are capable, and the down time during an appending of the virtual NIC (Network Interface Card) and the virtual network inside the hypervisor need to be rearranged. Moreover, if there are too many virtual NICs (because of many virtual machines), the number of the rules will increase and the IPTABLES may encounter problems. With a huge number of rules, the rule creation/modification may simply cause redundant rules and conflict rules.

Based on subsections 4.3.2.1–4.3.2.3, the additional conclusion is shown in Table 4.4. The 5Nine vFirewall is flexible with the increase of VMs (since it directly communicates with the hypervisor via API), and both the Tree-Rule firewall and IPTABLES need the installation of an additional instance (Guest OS) or more virtual NICs to protect the newly added virtual machines. However, the 5Nine vFirewall has four disadvantages. First, the 5Nine vFirewall can only be used on Microsoft platforms. Second, users have to install its agents on all VM clients with Windows XP SP3 (or higher version) Operating System. Third, the 5Nine vFirewall is difficult to manage rules since the user must select a VM (virtual machine) to manage its rule. The rules that protect each VM are on different pages. On the contrary, rules of Tree-Rule firewall and IPTABLES are on the same page (see Figure 4.1(c) for the case of using various virtual NICs to connect with their VMs). Last, the rules of the 5Nine vFirewall possibly cause redundancy and confliction (as its rules are based on the Listed-rule model). Redundancies and conflictions, however, can never occur in any Tree-Rule firewall.

Table 4.4: Capability comparison among the firewalls on cloud environment

	ESXi Firewall	**5Nine vFirewall**	**IPTABLES**	**Tree-Rule Firewall**
can protect Hypervisor	Yes	Yes	Yes	Yes
can protect internal VMs	No	Yes	Yes	Yes
can prevent attacking between VMs	No	Yes	Yes	Yes
stable	Yes	No	Yes	Yes
fast packet decision	No	No	No	Yes
support more than 5,000 rules	No	No	No	Yes
no shadowed/redundant/confliction rules	No	No	No	Yes
no down time for adding new VMs	No	Yes (only on Hyper-V)	No	No

4.4 Conclusion

In this study, we have proposed the Tree-Rule firewall that demonstrates none of the limitations presented in Chapter 3. The Tree-Rule firewall utilises rules in a tree data structure, and forwarding decision of an input packet based on tree rules will follow the tree structure so that the decision on the packet becomes faster. The Tree-Rule firewall has been tested and compared with IPTABLES on LANs and we have found that the Tree-Rule firewall gives better performance. Moreover, the Tree-Rule firewall has been tested on a cloud environment and we have found it more suitable than the Listed-Rule firewall for a cloud network, which is a large network that requires a number of computers and large rule size, and found it rapid for forward decision on packets. We have made a capability comparison among the proposed Tree-Rule firewall, the IPTABLES and two state-of-the-art firewalls under a cloud environment. The advantages of using the Tree-Rule firewall have been demonstrated.

Chapter 5 A Stateful Mechanism for the Tree-Rule Firewall

In this chapter, we propose a novel connection tracking mechanism for Tree-Rule firewall which essentially organises firewall rules in a designated Tree structure. A new firewall model based on the proposed connection tracking mechanism is then developed and extended from the basic model of Netfilter's ConnTrack module, which has been used by many early generation commercial and open source firewalls including IPTABLES, the most popular firewall. To reduce the consumption of memory space and processing time, our proposed model uses one node per connection instead of using two nodes as appeared in Netfilter model. This can reduce memory space and processing time. In addition, we introduce an extended hash table with more hashing bits in our firewall model in order to accommodate more concurrent connections. Moreover, our model also applies sophisticated techniques (such as using static information nodes, and avoiding timer objects and memory management tasks) to improve its processing speed. Finally, we implement this model on Linux Cent OS 6.3 and evaluate its speed. The experimental results show that our model performs more efficiently in comparison with the Netfilter/IPTABLES.

The first generation of firewall was proposed in1990s [3] and applies packet filtering mechanism to detect unwanted packets based on packet header information (e.g., source IP address, destination IP address, and destination port) without considering their connection states. In the later development, a new type of firewall called 'Stateful firewall', which is the second generation of firewalls was suggested. This type of firewall considers connection states, such as start of a new connection, being part of an existing connection and not being part of any connection. The stateful firewalls operate faster and more secure in comparison with the packet filtering firewalls (stateless firewalls). Nowadays, almost all firewalls are stateful firewalls [19][51]. However, these traditional firewalls, including packet filtering and stateful firewalls, have a flat structure in terms of listing of firewall policy rules. These firewalls have to process packets by comparing packet header information against the listed rules sequentially. The process will be done if a match is found or the last rule is reached. This method gives a low performance compared to other searching algorithms. Apart from the speed problem, the traditional firewalls suffer rule conflicts, caused by shadowed rules and redundant rules [12] [28][38][52], especially the shadowed rules. Recall that a shadowed rule is a rule which cannot be matched with any packets because all packets are already matched with other rules above it. This can cause a security problem because some rules that are important in protecting against hackers may be shadowed and not be used by a firewall at all. Moreover, it is quite hard for firewall administrators to design listed rules for a traditional firewall and to protect a large network which requires a large number of rules without any rule conflicts.

In our recent studies [28][38], as shown in the previous chapters, we proposed a new type of firewall to solve the aforementioned problems, and it achieves a high processing speed and fewer rule conflicts in large networks. In [28] and [38], we redesigned the firewall model to support two goals mentioned above. However, our redesigned model works only as a packet filtering firewall but not as a stateful firewall maintaining several connection states as suggested previously. In particular, the stateful (connection tracking) function includes an algorithm that can identify new and existing connections, as well as invalid ones which otherwise may raise a security concern. Thus, in this chapter, we propose a novel connection tracking mechanism which will be used in the Tree-Rule firewall [28][38].

The rest of this chapter is organised as follows. Section 5.1 discusses the most relevant background and previous research works to this study. In particular, the problems with traditional List-rule firewalls are recapped in Section 5.1, and the corresponding solution suggested in the Tree-Rule firewalls are reviewed in Section 5.2. The design and analysis of a novel connection tracking mechanism are presented in Section 5.2. The implementation of the proposed stateful mechanism and experimentations are detailed in Section 5.3. Finally, a conclusion is drawn in Section 5.4.

5.1 Background and Previous Works

In this section, we present some of the problems with traditional firewalls and recap corresponding solutions used in our Tree-Rule firewall model to avoid such problems. Below, the background information on packet filtering firewalls and stateful firewalls are first introduced.

5.1.1 Traditional firewall (Listed-rule firewall)

The traditional firewall (Listed-rule firewall) filters packets by comparing their header information against a set of pre-defined firewall rules. This process operates rule by rule until a specific condition is reached. The rules in traditional firewalls are a list of condition statements followed by actions which are shown in Table 5.1 below.

Table 5.1: An example of rule in Listed-rule firewall

Rule	Protocol	Source_IP	Destination_IP	Destination_Port	Action
1	TCP	192.168.1.1	54.251.1.1	80	ACCEPT
2	TCP	192.168.1.2	54.251.1.1	80	DENY
3	TCP	192.168.1.*	54.251.1.1	80	DENY
4	TCP	192.168.1.3	54.251.1.1	80	ACCEPT
5	TCP	192.168.2.*	54.251.2.3	80	DENY
6	TCP	192.168.2.4	54.251.2.*	80	DENY
7	TCP	192.168.3.*	54.251.3.5	80	ACCEPT
8	TCP	192.168.3.6	54.251.3.*	80	DENY
9	Any	Any	Any	Any	DENY

We have identified three key issues with the traditional firewalls in our previous studies [28][38]. They are

1. Security problem, caused by potential shadowed rules [28] and the change of meaning of the rule policy due to rule repositioning;
2. Functional speed problem, raised by shadowed rules [28], redundant rules [28] and sequential rule matching; and
3. Difficulty in rule design, in which one needs to carefully choose the proper positions for the firewall rules in order to avoid listing 'bigger rules' [38] before 'smaller rules' [38].

To overcome the aforementioned problems, we have proposed a 'Tree-Rule firewall' [38], where firewall rules are presented and operated in a tree data structure. A forwarding decision of an input packet follows the tree structure, so that the decision on the packet becomes faster.

5.1.2 The Tree-Rule firewall

In this section, we recall the Tree-Rule firewall [38] and present its design in Figure 5.1. As discussed in [38], the advantages of the Tree-Rule firewall include:

- No shadowed rules,
- No redundant rules,
- No need of rule swapping because all rules will be sorted automatically,
- Ease of rule design with independent 'rule paths' [38], and
- High speed for packet decision.

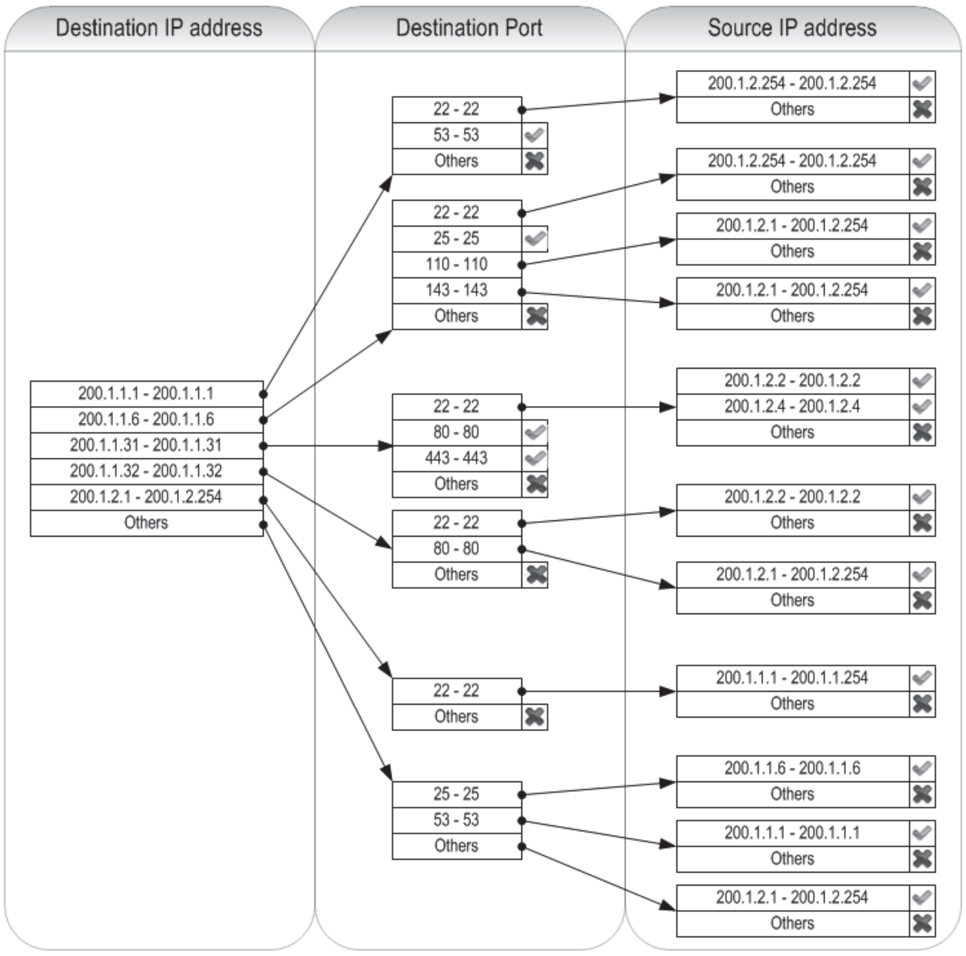


Figure 5.1: The design of Tree-Rule firewall [38]

With the Tree-Rule firewall, network administrators can design a firewall rule easily using Graphical User Interface (GUI) that comes with the firewall. The firewall rules are organised in a tree shape as shown in Figure 5.1. They can define attributes for each column, and can add more attribute columns such as 'protocol type' and 'source port' columns. The attribute within the root node (the left column) can be Source IP address, Destination Port, or any attribute suitable to work with their networks. Ranges of numbers in each node will be sorted automatically without overlapping. Thus, administrators do not need to specify the position of a numbers so that this can avoid problems caused by rule swapping. The pointers between nodes can distinguish the 'rule paths' [38]. Therefore, no packet can match with two or more rule paths. This technique can avoid shadowing anomaly and redundancy anomaly. Apart from the users' view, the firewall also processes the rule on a tree structure as well. The firewall will take a corresponding attribute within packet header (i.e., Destination IP address) for searching in the root node first. Then, the firewall will verify packet's other attributes in order, by searching only on relevant nodes at the corresponding levels. As a result, the packet will be decided quickly with a specific action.

The next subsection will provide background knowledge on Packet filtering (Stateless) and Stateful firewalls.

5.1.3 Packet filtering and Stateful firewalls

Packet Filtering firewalls (Stateless firewalls) have been designed and created in the first decade of firewall age. These firewalls use an uncomplicated operation to examine incoming and outgoing packets. With this operation, each line of a set of firewall rules can regulate packet flow in one direction only [9]. In the context of Packet Filtering firewalls, Johansson and Riley [9] highlighted that a rule, which allows outbound access, requires a mirror rule [9] for allowing inbound access to permit replies to enter the network. Even though some Packet Filtering firewalls can add mirror rules automatically, most of these firewalls need firewall administrators to perform this task manually [10]. More importantly, Packet Filtering firewalls have a critical security issue because the mirror rule generated by the firewalls or even administrators can cause a big loophole, which allows attackers connecting into protected ports on protected computers of internal networks.

Similar to Packet Filtering firewalls, Stateful firewalls would examine packets' header (i.e., IP, TCP and UDP header). However, Stateful firewalls are little smarter because they understand TCP and UDP connection status by watching all connection states [9]. If clients behind a Stateful firewall create a new connection to communicate with web servers on the outside, the firewall will create inbound filters to allow relevant reply packets to travel from the web server to the clients. The Stateful firewall will keep tracking the state of a network connection which the packets travel across. The firewall may check TCP sequence numbers for observing the connection status. Some examples of Stateful firewalls are Check Point Firewall-1 [7][53], Juniper Net Screen [54][55], and IPTABLES [56][57].

The IPTABLES is one of the most popular open-source Stateful firewalls. There are many sources of documents which can guide one to understand the mechanism inside the Stateful firewalls. The article entitled "Netfilter's Connection Tracking System" by Pablo Neira Ayuso [11] gives useful information about a stateful mechanism within the Netfilter used by IPTABLES. This article reveals that the Netfilter uses a hashing algorithm to digest important packet header information (i.e., source IP address, destination IP address, source port, and destination port), and then push the hashed result into a hash table. To prevent hashing collision, the Netfilter uses buckets of linked lists to carry the hashed results. If a new connection is created, a firewall will compute a hashing result and find a location for it. If a reply packet reaches back to the firewall, the firewall will verify whether the packet belongs to an existing connection or not. Finally, the entry of a hash will be destroyed when the relevant connection is disconnected or expired. Thus, the first packet of connection will be examined against the firewall's rule list while other packets will be examined against the hash tables and buckets.

Considering the merits of stateful firewall models in configuration and security, we intend to enhance our Tree-based firewall via an integration of the mechanism of stateful firewall models into its system design. The details of the new stateful Tree-based firewall will be presented in the next section. The connection tracking model in our new stateful Tree-based firewall outperforms the one of Netfilter.

5.2 Design and Analysis

Our stateful Tree-Rule firewall is designed based on the connection tracking model of Netfilter [11][56][57][58][59] with improvements. Similar to the connection tracking model of Netfilter, a 'Hashing Table' is involved in the connection tracking model in our stateful Tree-Rule firewall. Comparatively, this newly designed model consumes less memory space (RAM) and operates in a faster manner. The details of this new model are discussed in subsections 5.2.1 to 5.2.6.

5.2.1 Using one node per connection

Netfilter uses two nodes per connection for storing packet information (i.e., Source IP address, Destination IP address, Source Port, Destination Port, and Protocol Type), which is called 'tuple'. The first node (node 'x') as shown in Figure 5.2(a) is used for recording information of packets transmitting from the 'original direction' (i.e., packets coming from the point that started the connection), while the second node (node 'y') is used for the 'reply direction' (i.e., reply packets going to the point that started the connection).

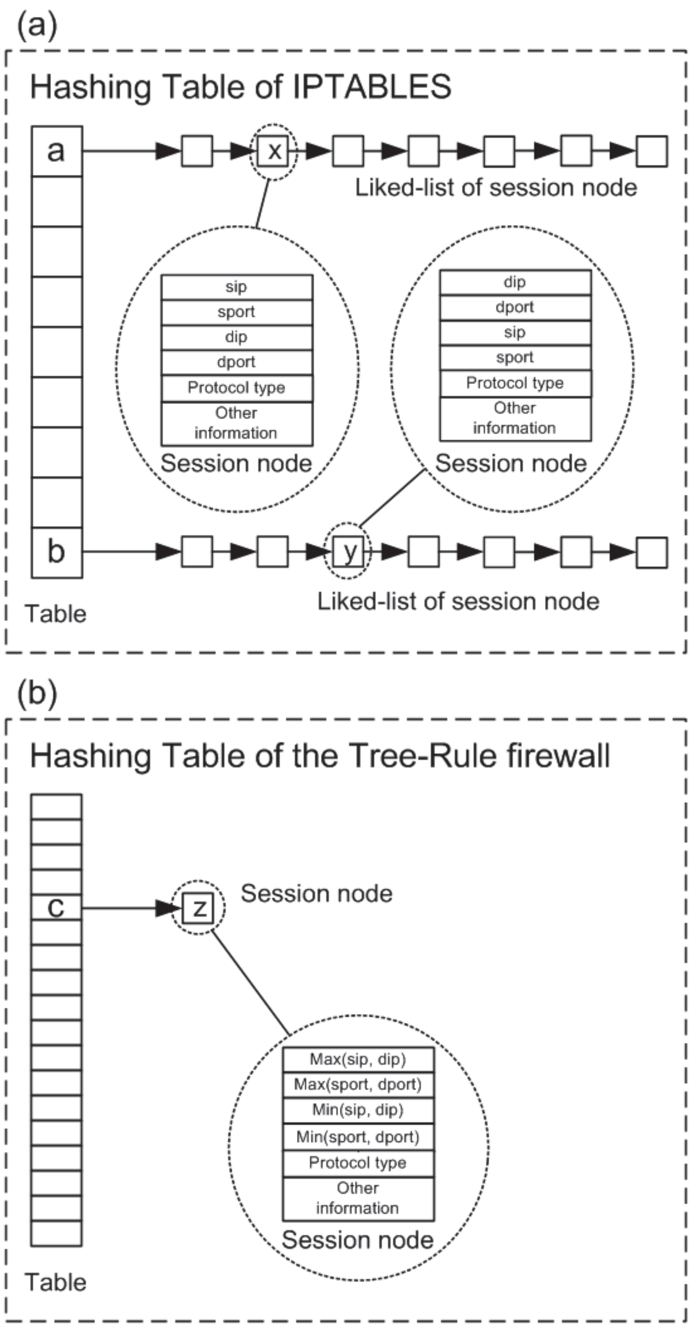


Figure 5.2: Hashing Tables of IPTABLES and the Tree-Rule firewall [60]

In the Netfilter's model, the five important attributes of packet information are stored into nodes and are digested (hashed) using hashing function to obtain entry numbers (i.e., 'a' and 'b' in Figure 5.2 (a)). These numbers then indicate which buckets the nodes are located in. Thus, the two nodes related to the same connections are located on different entries. For example, Entry1 and Entry2 are entry numbers for the two nodes related to the same connection. They are calculated using the functions as shown below.

Entry1 = Hash(Source_IP, Destination_IP, Source_Port, Destination_Port, Protocol_Type, Key),

and

Entry2 = Hash(Destination_IP, Source _IP, Destination_Port, Source_Port, Protocol_Type, Key),

where 'Key' is a secret number generated by the firewall randomly.

Differently, in our model, only a single node is used per connection using Max() and Min() functions. Consequently, we need only one entry number (i.e., 'c' in the Figure 5.2(b)) which is obtained using the function below:

Entry = Hash(Max(Source _IP, Destination_IP), Min(Source_IP, Destination_IP), Max(Source_Port, Destination_Port), Min(Source_Port, Destination_Port),
Protocol_Type, Key).

Note: Jenkins' hash (jhash) is used in both the connection tracking model of Netfilter and that of our stateful Tree-Rule firewall. This is due to the reason that jhash provides fast operation and is capable of distributing its 32 output bits randomly.

By using our proposed scheme, approximately 50 per cent of the memory space can be saved. This allows our firewall to increase the number of concurrent connections up to two times more. In the Netfilter's model, the first packet in a new connection, as shown in Figure 5.3, needs to be calculated twice. The first calculation is for the 'original direction' node and the second calculation is for the 'reply direction' node (as shown in Figure 5.2(a). Moreover, it has to allocate memory two times for the two nodes as well. In contrast, our model needs only one hashing calculation because we use only one node per connection. With the same reason, to terminate a connection, it needs two operations to delete the two nodes from memory, while our model takes only one node away. Thus, our model can reduce the processing time significantly.

Additionally, in our model, the number of connections handled by our stateful Tree-Rule firewall can reach the maximum value. This is different from the Netfilter's model, which has a limitation on the use of pair nodes. Given a pair nodes in the Netfilter's model with a new connection (as shown in Figure 5.3), after calculating Entry1 and Entry2, the Netfilter looks for spaces to place the two nodes by checking bucket lengths. If the relevant buckets are close to be full or already full, then the length of the two buckets can be categorised into three cases (A, B, and C):

A. 7 and 7

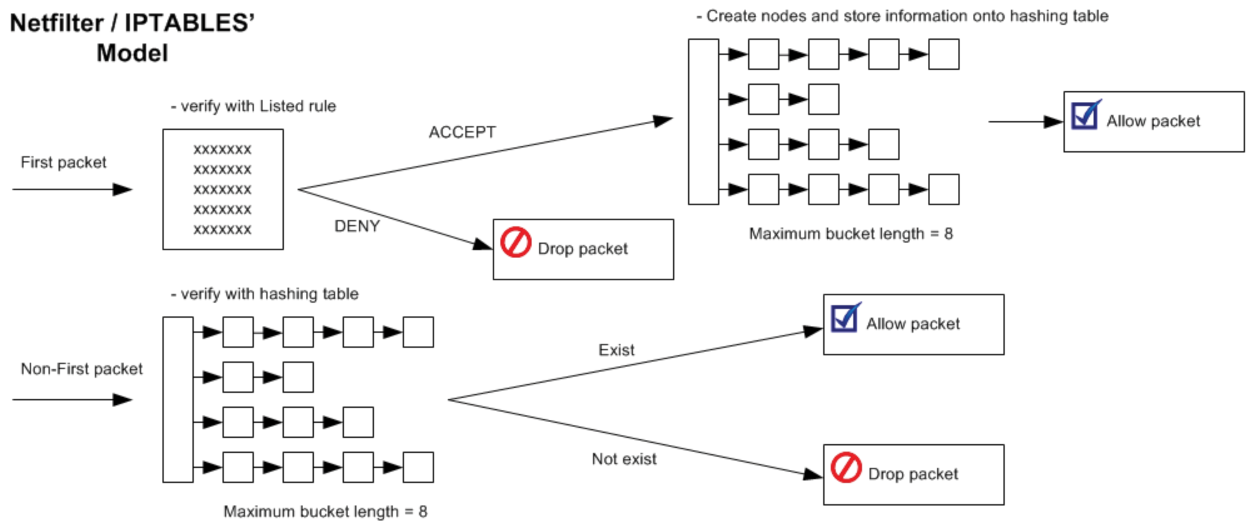
B. 8 and 8

C. 7 and 8

Note:

- The maximum bucket length of Netfilter's model is 8, and

- Assuming that Jenkins' hash is used in Netfilter's model and our model to provide random output bits.



Netfilter / IPTABLES' Model
First packet
- verify with Listed rule
xxxxxxx
ACCEPT
DENY
Drop packet
- Create nodes and store information onto hashing table
Maximum bucket length = 8
Allow packet
Non-First packet
- verify with hashing table
Maximum bucket length = 8
Exist
Not exist
Allow packet
Drop packet

Figure 5.3: Steps of Netfilter's connection tracking [11][60]

In case A, luckily, there are memory spaces for the two nodes. If the two nodes are allocated to the tails of the buckets, the lengths of the both buckets will now be 8s exactly. In case B, unluckily, the two relevant buckets are already full. No more nodes can be appended to the tails of these buckets. In case C, there is only a space for one of the pairs but not enough for two. In this case, Netfilter's model is not able to use the memory space efficiently because maximum bucket length in average, for this case, is 7.5 but not 8.

One may argue that our model has to calculate Max and Min for the Source IP address, Destination IP address, Source Port, and Destination Port, for every packet. As such, these operations may consume additional CPU load. To respond to this argument, we have created a small program for measuring time consumption of Jenkins' hash function and the Max/Min functions. It is found that the computation of Jenkins' hash applied in Netfilter's model takes approximately 1,000 times longer than those of the Max and Min functions. Our model requires hashing calculation once only and Max and Min calculations for four times, whiles Netfilter's model makes the hashing calculations twice and no Max and Min calculations. Overall, our model achieves a better performance in terms of speed.

5.2.2 Expanding Hashing Table size vertically

In this section, we propose to improve Netfilter's model by expanding the size of Hashing table vertically (as shown in Figure 5.4). This method can reduce hashing collisions and further improve the speed of operation. We decrease 'maximum bucket length' and then increase the number of entries in Hashing table with the same memory space. For example, if we double the size of Hashing table vertically, we need to decrease maximum bucket length by half of the size (i.e., from 8 to be 4). Hence, by increasing the size of Hashing table by two times, the collision will be decreased approximately by 50%. This percentage is reasonable for decreasing maximum bucket length to half. In our model, in fact, we use one node per bucket and expand the Hashing table vertically to 8 times or more. However, the size of a hashing table can be expanded to 16, 32, 64 times or more because our model requires a smaller node size than that used in the Netfilter's model.

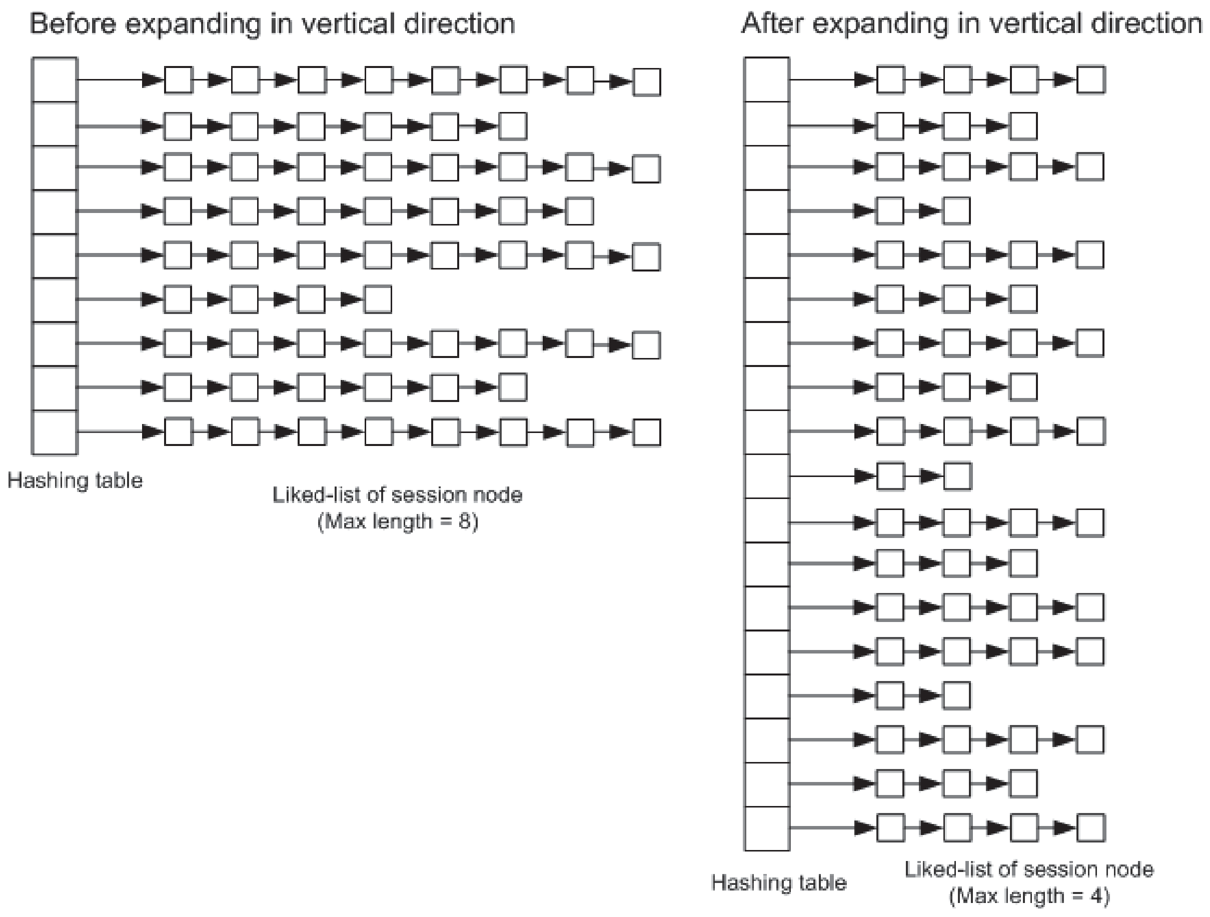

Before expanding in vertical direction
Hashing table
Liked-list of session node (Max length = 8)
After expanding in vertical direction
Hashing table
Liked-list of session node (Max length = 4)

Figure 5.4: Expanding hashing tables in vertical direction [60]

It is worth emphasising that a collision may still occur because there will be a chance to have two different packets associated with the same hashed output. However, expanding the size of the Hashing table in vertical direction as proposed in this section can reduce probability of collisions generated by a hashing function. This method can handle more concurrent connections and allows shorter bucket lengths for the same memory space, and the same chance of collisions. Consequently, time consumption for a sequential search within buckets will be decreased automatically.

Considering systems A and B which have different sizes of Hashing tables (the size of A is bigger than the size of B). If distribution of nodes of the two systems is random, and increasing rates of nodes in the two systems are the same, we have found that the bucket length in A will be shorter than the bucket length in B. Consequently, searching for nodes in system A will be faster than B. Thus, we can conclude that expanding size of Hashing table vertically can reduce searching time within the buckets.

Although expanding the size of a Hashing table vertically requires more spaces for the pointer of each expanded entry because each entry must point to its first node, these pointers require little memory space compared to the size of nodes.

In the Netfilter's model, one node requires 350 bytes [61], while one pointer requires only 8 bytes. In our model, one node requires 40 bytes while one pointer requires also only 8 bytes. The number of bytes used for each node has been reduced by 8.75 times.

5.2.3 Using one node per bucket

It is apparently that a short bucket length requires less time for a sequential search in comparison with a long bucket length (as shown in Figure 5.4). Thus, we decide to use one node per bucket in our model (as shown in Figure 5.2(b)) instead of using 8 nodes per bucket as presented in the Netfilter's model. However, to mitigate the collision problem, we will expand the size of Hashing table vertically by at least eight times.

With this scheme, we can avoid the sequential search within buckets, and provide a better speed performance. This scheme also results in less memory consumption because no left and right pointers are required to be maintained in a node to work as doubly linked list (i.e., in the

Netfilter's model [11]). Consequently, the node size can be reduced. Moreover, this decrease of complexity provides an easy way to implement or create an efficient firewall.

Some users may have negative thinking about the collision problem on our model (maximum node per bucket = 1) because they may believe that collisions can be mitigated using only buckets. In fact, we have already mitigated the collision problem by expanding the size of Hashing table vertically. Reducing a bucket length from 8 to 1 can be compensated by extending the Hashing table by 8 times. Moreover, we can extend it up to 64 times with the same amount of memory space used in Netfilter because our nodes are smaller than Netfilter's nodes (noting that our node size = 40 bytes, Netfilter's node size = 350 bytes).

5.2.4 Verifying non-first packets using Tree Rule before Hashing Table

Generally, in firewalls including Netfilter, the 'rule set' created by firewall administrators will be used only for the first packet of a connection (as shown in Figure 5.3.). The first packet of a connection is verified with the 'rule set'. If the result is ‘ACCEPT’, then the firewall will create corresponding nodes within the 'Hashing table'. Then, the subsequent packets, namely the second packet, third packet and so on, will be verified with the entries in the 'Hashing table' instead of the 'rule set'. Therefore, it can be seen that verifying packets using Hashing table will be done more often than verifying packets using the rule set. From our study, we have found that hashing calculation takes approximately 1,000 times in comparing two numbers used in a verifying rule set or Tree rule. This is because hashing function is very complex and involves a complicated computing procedure. In contrast, verifying packets using rule set does not take much time because it requires only comparison between a pair of numbers.

However, there are some attack packets that are deliberately designed as non-first packets and injected into host connections. A good example of such type of attack is the 'Reset Flood Attack' [62][63][64], where packets are generated by worms randomly and contain no SYN flag. These attack packets will be verified by using the Hashing Table. If these packets are verified by using a rule set (or Tree rule) and are dropped, it can reduce CPU time because verifying a packet with a rule set is similar to a comparison between numbers which is faster than using hashing function. We can drop packets which are denied by a rule set (or Tree rule) without using the Hashing Table again because if the first packets are denied by the rule set, packets' information will have no chance to appear in the Hashing Table. In this method, we will consider packets from both directions.

Therefore, in our model, we will verify non-first packets using Tree Rule first for both directions (as shown in Figure 5.5). If at least one of results of this test is ACCEPT, we will then verify the packets using the Hashing Table.

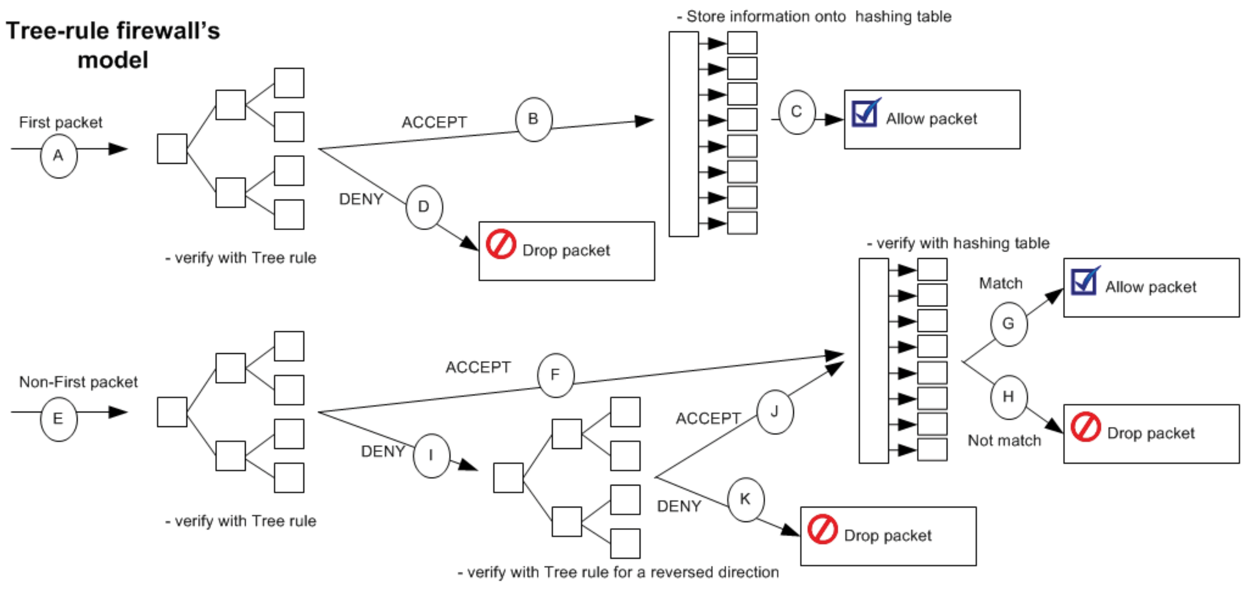


Figure 5.5: Verifying non-first packets using Tree Rule before Hashing Table [60]

As such, the proposed firewall can operate faster in the case of an attack using random packets because approximately 50% of randomly generated packets will be the non-first packets which will be denied by firewall rule as it can be seen in the bottom flow of packets in Figure 5.5 (e.g., the path E→I→K). In this case, by eliminating the unnecessary hashing calculation, the overhead of our proposed stateful Tree-Rule firewall in packet filtering is further reduced.

In the case of attacks where the number of non-first packets is greater than the number of first packets, if the verified result of a Tree rule is 'ACCEPT' (e.g., the path E→F or the path E→I→J shown in Figure 5.5), the proposed firewall will need only little extra time to verify packets using Tree rules before verifying packets using the Hashing table. However, the overall time consumption is only slightly increased because verifying packets using Tree rules takes very little time and can be ignored compared to verifying packets using Hashing table.

5.2.5 Use of Static Node and Label to identify free nodes

In the Netfilter's model, if a new connection is created, the firewall must prepare memory spaces for relevant nodes (i.e., using the 'kmalloc()' function in C language). Also, if the connection is terminated, the firewall must return these memory spaces back to OS (i.e., using the 'kfree()' function). These operations involve a time factor performing these operations. Considering a normal cycle for the HTTP request and reply, which use TCP connection with approximately five relevant packets (in average), the firewall called the 'kmalloc()' function and 'kfree()' function twice generating a total of four memory related operations, which is high compared to the number of relevant packets (five packets). To resolve this problem, we develop our model using a static memory. The node, which is used for recording packet information, should be a static node and will be created when the firewall is loaded and executed. If the connection is created, packet information will be stored in the node, and the 'Label' of this node will be marked to be 'Unavailable'. If the connection is terminated, the Label of this node will be marked to be 'Available' without destroying the node from memory. If the next connection, which has the same entry number, is created, the firewall can use this node immediately (without requesting further memory space from OS for the new node) by overwriting the information of new packets to this area of memory, and then change the Label to be 'Unavailable'. With this method, verifying packets can be done easily by considering the Label and packet information within the node. In implementing the above scheme, the variable type for the Label is a one byte variable (actually it requires only one bit '0' if the node is 'Available' and '1' if the node is 'Unavailable').

In the context of computer programming, each function including memory management function affects the run time. With our scheme, the firewall does not need to request and return memory space from and to the OS for nodes (i.e., calling kmalloc() and kfree()). As a result, the proposed firewall can operate faster.

Using our scheme, firewall administrators must prepare a fixed but a very small size of memory for the firewall software, and this memory space cannot be shared with other processes although the firewall is working with only few packets.

Based on our calculation, a 768 MB RAM can handle 16.7 million connections. This provides approximately 46 bytes per connection for our firewall model. According to the information presented on the Netfilter's website [57], the default value for maximum connection of IPTABLES is 8192, which requires 128 MB of RAM. Therefore, it requests 16,384 bytes per connection for the Netfilter's model. Hence, our scheme can carry a total of 16,384 / 46 = 356 times more connections compared to the Netfilter's model.

5.2.6 Using the Label for Time Out instead of Timer Object

In the Netfilter's model, if a connection is terminated by FIN or RST flags, firewalls will create a 'Timer Object' to schedule and specify the time when the relevant nodes will be deleted. For example, the Timer Object will delete a node in the next 30 seconds since the FIN flag was detected. This operation uses event driven techniques. Creating, deleting and managing Timer Objects consumes OS resources and CPU time. These resources and time consumptions are significant when compared with a short connection (e.g., normal use of web browsing / HTTP protocol). Thus, our model removes this drawback by using a Label for expired nodes instead of using the Timer Objects. With this method, if a firewall finds FIN or RST flags, it will calculate expiration time for the node and store it into the Label. A variable type for this Label is the 'struct timespec' (in C language), which requires 16 bytes, and can record date and time in seconds, and even nanoseconds. To verify the node, the firewall will check the timer's Label in the node before using information from the node. If the current time is new and greater than the time stored in the Label, it means that this node is expired (the node does not exist). In contrast, if the current time is less than the time stored in the Label, it means that this node is still active, and the firewall can use information from this node.

Firewalls can work faster because they do not necessarily manage timer objects such as calling the 'new_timer()' and 'delete_timer()' functions. Moreover, this scheme can save OS resources and easy to implement.

5.3 Firewall Implementation and Experimental Analysis

We implement the Tree-Rule firewall using C language on Linux Cent OS 6.3, and conduct experiments on real network environments. In our previous research [28][38], it has been shown that the Tree-Rule firewall provides higher security in comparison with traditional firewalls. In this chapter, however, we focus on speed issue along with our stateful (connection tracking) mechanism. We emphasise on the following four cases of packets in TCP connections (as shown in Figure 5.6.):

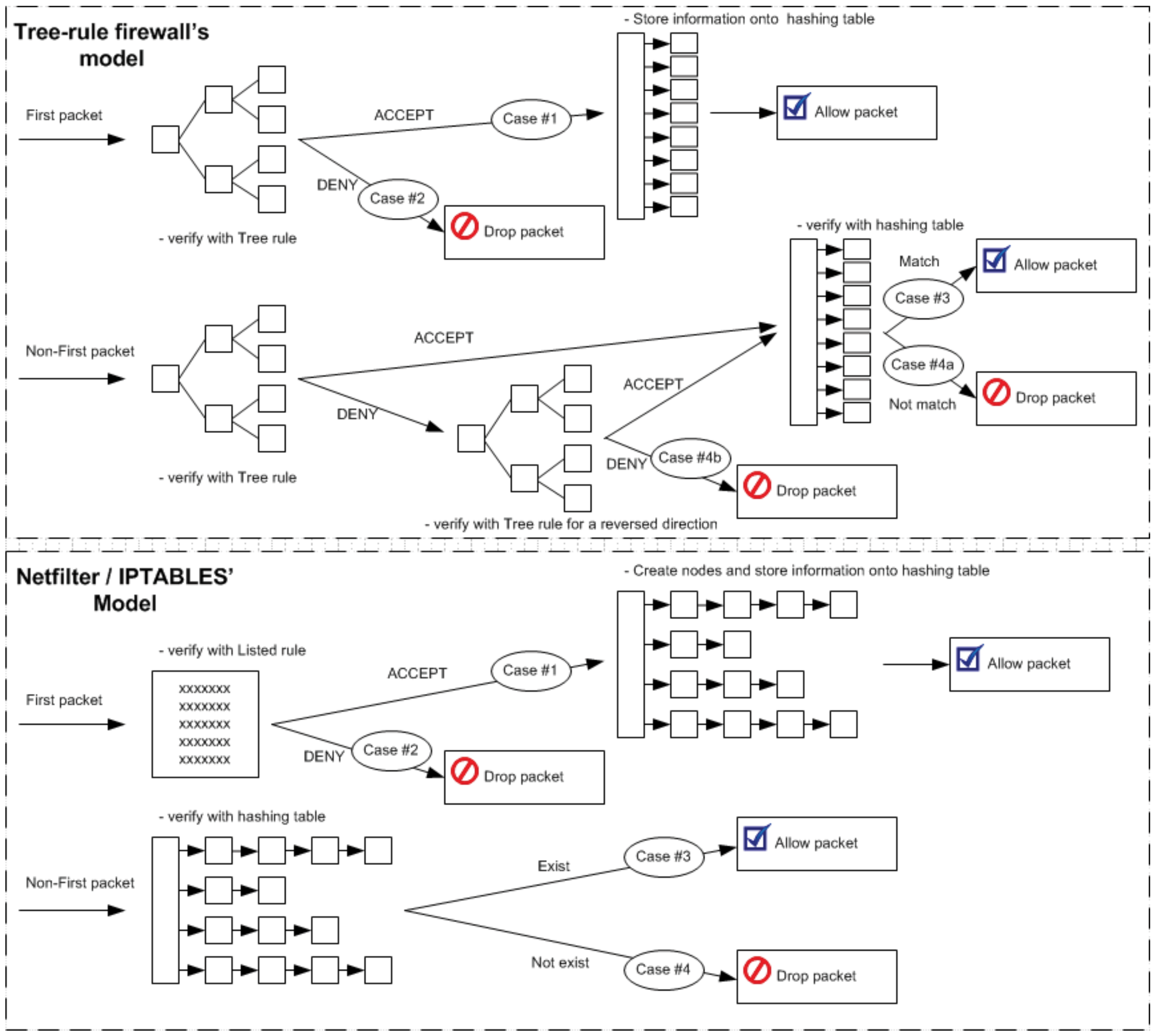


Figure 5.6: The four cases which was evaluated on speed issue [60]

On client machines, apart from a normal Internet browsing, we use BackTrack 5 R3 live boot flash to generate packets and use 'hping3' command to generate the first packet of connection with '-S' parameter. The detailed command is given below.

```
# hping3 192.168.22.2 -a 192.168.11.2 -p 333 -S -s +1 -d 1440 -i u1000
```

This command can be used for Cases #1 and #2 because the '-s +1' parameters will generate packets of which their source ports will be starting from '1', and increased by one, for each packet. In addition, we use '-d 1440' to specify packet size of 1440 bytes (not including layer-2 header and tail size) because this number is a regular size for the HTTP protocol. With this tool, we can create and send packets as much as possible using the '--flood' parameter, and can specify time interval for each packets using the '-i' parameter. For generating packets related to Cases #3 and #4, we will use this command without '-S' parameter to turn off the SYN flag.

We compare the processing speed of the connection tracking module in our stateful Tree-Rule firewall with that of the connection tracking module in Netfilter. Figure 5.6 shows the four cases on the Tree-rule and IPTABLES/Netfilter. We also measure the processing time on the packet decision process for one packet. We start to record packet arrival time on the first line of the packet hooking function (i.e., the function 'hook_func()' of the link [65]). We then record the end time before the line 'return NF_ACCEPT' and 'return NF_DROP' for all cases. Programmed codes between start time and end time for each case, give different time consumptions because some cases need to compute hashing function while other cases do not. The recorded time in nanosecond will be written on the file '/var/log/messages' using the 'PRINTK' function. Likewise, we also measure the computation time of packets on Netfilter/IPTABLES' module by using this technique. We modify the file '/kernel/net/netfilter/nf_conntrack_core.c' of the Netfilter's

ConnTrack module, and recompile it (including relevant files) to create the new 'nf_conntrack.ko' file.

We reload the modified modules to operate with IPTABLES. Information about starting time and ending time for the four cases of our prototype software and Netfilter/IPTABLES are collected and their average is computed. We test them on a real network environment with more than 50,000 packets before taking average for every case. Our experiments use approximately 50 client PCs in a university network LAB. These computers (firewall and clients) use Intel Core i5 with 2.30 GHz of CPU, and 4GB of RAM. We test with 50 rules for IPTABLES and approximately 50 rule paths for Tree-Rule firewall. The experimental results are presented in Table 5.2 and Figure 5.7.

Table 5.2: Time consumption of packet decision on Tree-Rule firewall and Netfilter / IPTABLES [60]

Firewall	**Processing Time for one packet (nanosecond)**				
	Case 1	**Case 2**	**Case 3 (average)**	**Case 4a (average)**	**Case 4b**
Tree-Rule firewall	481.13	164.44	497.69	455.12	239.94
Netfilter/IPTABLES	3,722.82	1,259.94	3,319.55	3,193.41	3,193.41

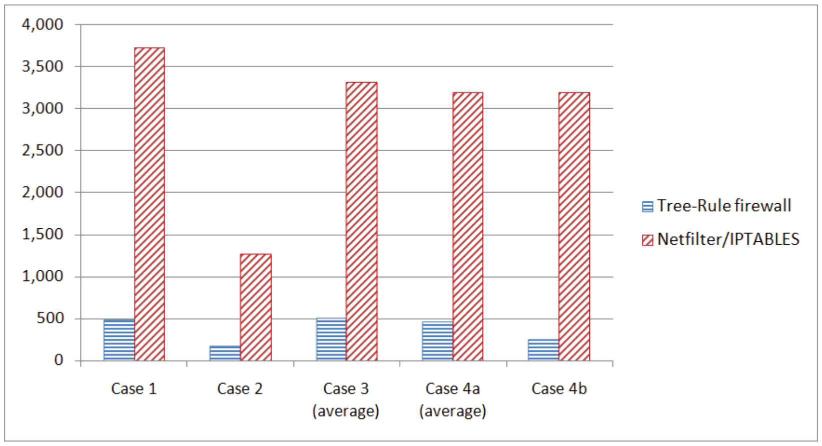


Figure 5.7: Representation of time consumption on Tree-Rule firewall and Netfilter/IPTABLES

The experimental results of the Tree-Rule firewall shown in Table 5.2 and Figure 5.7 reveal that Case #3 takes little more time than Case #1, because Case #3 needs to verify packet header with a tree rule for one or two times before verifying the packet information with the hashing table. This slight time difference is due to the fact that verifying packets with the tree rule is an easy job in comparison with calculating a hashing function. Case #2 and Case #4b of the Tree-Rule firewall take less time compared to other cases because they do not need to perform the hashing function. Theoretically, Cases #3 and #4 of Netfilter/IPTABLES should take an equal time because they use same algorithms. However, in practical, they may encounter different times of computations. The largest time used by Netfilter/IPTABLES is the time in Case #1, where the firewall has to perform the hashing task two times (for two directions), and also requests memory spaces from OS to create two nodes for storing connection information for both original and reply directions.

Apart from time consumption on packet decision, we also measure memory consumption of both the Tree-Rule firewall and Netfilter/IPTABLES. Theoretically, Netfilter requires 350 bytes per connection [61], while our model (Tree-Rule firewall) requires only 40 bytes per connection. The Tree-Rule firewall uses less memory because it does not use timer objects and memory spaces for reply-direction nodes. Consistently, our experimental results show that Netfilter takes approximately 400-430 bytes per connection, whereas our model takes approximately 40-60 bytes per connection. These numbers are based on 10,000 concurrent connections with normal traffic load on real network.

5.4 Conclusion

In this chapter, we have proposed a stateful mechanism which is used on the Tree-Rule firewall. We design and develop our model from the basic connection tracking model of Netfilter. Our proposed connection tracking model requires a low memory space, while it can handle more concurrent connections by using one node per connection. Our scheme extends the hashing table vertically and uses one-node bucket length. Moreover, this model has a low time consumption, which benefits from the avoidance of hashing computation by considering the Tree rule first. If packets are not corresponding to the Tree rule (for both directions), the firewall will not perform hashing calculation. We have also got rid of the timer objects used for closing connections. Instead, we use the Label which can tell the firewall which connections are expired by reading only 16 bytes of time information from the memory. Moreover, this model does not necessarily create new nodes to store packets' information because we use static memory for all nodes created after the firewall is loaded into the memory and executed. With this scheme, the firewall only checks whether the relevant node is free or not by using the marked Label. Our experimentation has been conducted on real network environments, and the results have shown that our stateful Tree-Rule firewall operates faster than Netfilter/IPTABLES. Our stateful Tree-Rule firewall also requires less memory owing to the fact that the size of a node has been reduced and the number of nodes used per connection has been decreased from two to one.

Chapter 6 Hybrid Tree-rule Firewall for High Speed Data Transmission

Traditional firewalls employ listed rules in both configuration and process phases to regulate network traffic. However, configuring a firewall with listed rules may create rule conflicts, and slows down the firewall. To overcome this problem, we have proposed a Tree-Rule firewall in our previous study. Although the Tree-Rule firewall guarantees no conflicts within its rule set and operates faster than traditional firewalls, keeping track of the state of network connections using hashing functions, as shown in the previous chapter, incurs extra computational overhead. In order to reduce this overhead, we propose a hybrid Tree-Rule firewall in this chapter. This hybrid scheme takes advantages of both Tree-Rule firewalls and traditional listed-rule firewalls. The GUIs of our Tree-Rule firewalls are utilised to provide a means for users to create conflict-free firewall rules, which are organised in a tree structure and called 'tree rules'. These tree rules are later converted into listed rules that share the merit of being conflict-free. Finally, in decision making, the listed rules are used to verify against packet header information. The rules which have matched with most packets are moved up to the top positions by the core firewall. The mechanism applied in this hybrid scheme can significantly improve the functional speed of a firewall.

Firewalls were first invented in 1990s [3], and have been developed to operate more securely and faster. Since the first generation firewalls, the commercially used fire-walls still perform network traffic regulation based on listed rules. As described in the previous chapters, the listed rules are a set of rule sequences which consist of conditions and actions. If information carried in the header fields (e.g., Source IP, Destination IP and Destination Port) of an incoming packet is matched with the condition of a rule, the packet will be accepted or denied in accordance with the action specified in the rule. However, in the listed-rule set of a traditional firewall, there may be 'shadowed rules' [12] and/or redundant rules. On one hand, shadowed rules may cause security problems because protection rules could be shadowed by other rules listed ahead. On the other hand, redundant rules cause latency in traffic processing and lower the throughput of a network due to the undesirable waste of time on verifying against these rules. The detailed discussion of these problems can be found in our previous work published in [28] and in the previous chapters.

To address the afore-mentioned problems, we recently proposed a new type of firewall called 'Tree-Rule firewall' in [38] and shown in the previous chapters. It has been proved that the Tree-Rule firewall guarantees no conflicts (e.g., no shadowed rules and no redundant rules) in rule sets, and is more efficient in traffic processing in comparison with traditional listed-rule firewalls [38]. In our recent follow-up study [60] shown in the previous chapter, a new stateful mechanism was proposed to further improve the Tree-Rule firewall with the capability of tracking the states of network connections. In comparison with IPTABLES, the most popular open source firewall, the stateful Tree-Rule firewall is more advanced in terms of processing speed.

However, complex hashing computations are involved in the stateful mechanisms used in the Tree-Rule firewall and the IPTABLES. A hashing function has to be invoked at least once in either the stateful Tree-Rule firewall or the IPTABLES in stateful mode to verify each single packet travelling through the firewall. It takes approximately 1,400 nanoseconds to compute the Jenkins hash (jhash) [11] used in these two firewalls running on a standard PC with a Pentium 2.4 GHz CPU. Whereas, comparing two variables takes only 1.4 nanoseconds with the same setup.

On contrary, if an incoming packet matches with the first rule in a stateless firewall (e.g., IPTABLES in stateless mode), then the firewall needs to conduct comparisons between four packet header fields (i.e., Source IP address, Destination IP address, Source Port and Destination Port) and the respective conditions specified in the rule. This rule matching is approximately 1400/(1.4*4) = 250 times faster than that of a stateful firewall.

Although the traditional stateless firewalls (e.g., IPTABLES in stateless mode) can operate fast, the rule conflict problem is still the main obstacle for improving firewall speed using the rule sequence tuning. In a firewall rule list, there may be many frequently matched rules which are positioned at the bottom of the list. These rules, especially the last rule which was created to deny all packets, cannot be moved up to the top positions because rule conflicts may cause the change of firewall policy if they are moved up. However, if frequently matched rules in a firewall can be moved up to top positions, the firewall, especially a firewall working in a large network with a huge number of rules, will operate faster.

Motivated by the above, the contributions of this chapter are shown as follows.

- We propose a hybrid firewall which takes advantages of both the Tree-rule and stateless mechanism in design. This scheme ensures no rule conflicts and high traffic processing speed in nature. More frequently matched rule will be moved to higher positions in the rule list automatically.
- We derive a mathematical model for measuring the time consumption in the hybrid firewall and a mathematical model for measuring the efficiency of data transmission. The experimental results show a great improvement in terms of efficiency on the proposed firewall.

- The proposed firewall is implemented under a cloud environment. The experimental results show that the proposed hybrid firewall using 'automatic rule sorting' outperform the ones with 'non-automatic rule sorting' modes.

The rest of this chapter is organised as follows. The background and the related work are introduced in Section 6.1. Our proposed hybrid firewall scheme is then detailed in Section 6.2. The implementation of our proposed scheme is presented and the experimentation is demonstrated in Section 6.3. Finally, the conclusion is drawn in Section 6.4.

6.1 Background and Related Works

Previous research approaches aiming to enhance functional speed of firewalls can be categorised into three types. The first type focuses on discovery and elimination of rule conflicts, especially redundant rules, to reduce the rule size of a firewall. This can reduce memory space consumption and processing time on a firewall. The second type emphasises on developing firewalls with high performance hardware, such as implementing a firewall on Field Programmable Gate Array (FPGA). Whereas, research of the third type focuses on filtering mechanisms of firewalls, for instance, converting firewall rules into a tree structure which can process packets faster than a traditional sequential rule list.

In this section, we first conduct a review on the recent advances in the afore-discussed research focuses. Then, we present the achievements from our previous studies on Tree-Rule firewall. These achievements are the underlying infrastructure of the new hybrid firewall proposed in this chapter.

6.1.1 Enhancing processing speed via rule conflict elimination

Rule conflicts have come into focus of many researches on traditional firewalls. These firewalls use their listed rules to filter packets. The listed rules shown in Table 6.1, for example, illustrate how to regulate traffic traversing over the network presented in Figure 6.1 in compliance with the network topology.

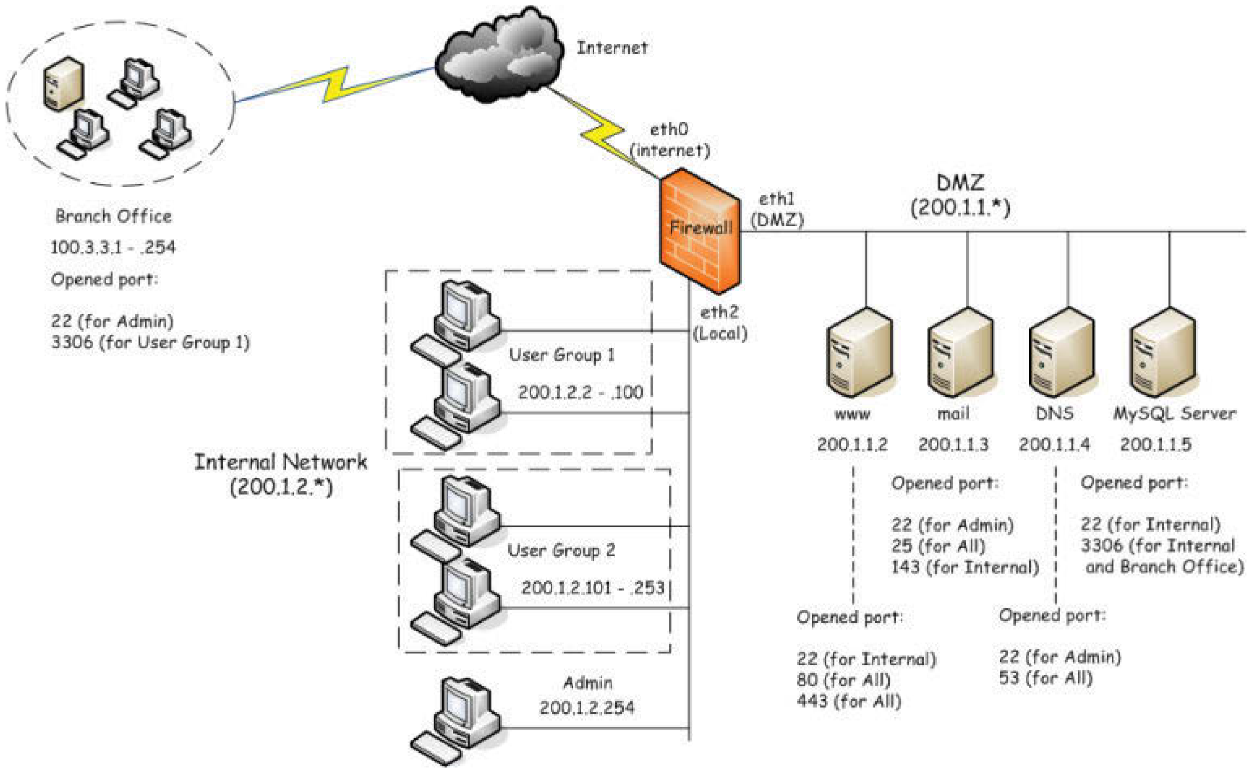


Figure 6.1: An example network

Table 6.1: A set of listed rules created for an example network in Figure 6.1

No.	Source_IP	Dest_IP	Dest_Port	Action
1	100.3.3.*	200.1.1.5	3306	Accept
2	100.3.3.*	200.1.1.2	80	Accept
3	100.3.3.*	200.1.1.2	443	Accept
4	100.3.3.*	200.1.1.3	25	Accept
5	100.3.3.*	200.1.1.4	53	Accept
6	100.3.3.*	*	*	Deny
7	200.1.1.*	100.3.3.*	*	Deny
8	200.1.1.*	200.1.2.*	*	Deny
9	200.1.1.*	*	*	Accept
10	200.1.2.2-100	100.3.3.*	3306	Accept
11	200.1.2.254	100.3.3.*	22	Accept
12	200.1.2.*	100.3.3.*	*	Deny
13	200.1.2.*	200.1.1.2	22	Accept
14	200.1.2.*	200.1.1.2	80	Accept
15	200.1.2.*	200.1.1.2	443	Accept
16	200.1.2.254	200.1.1.3	22	Accept
17	200.1.2.*	200.1.1.3	25	Accept
18	200.1.2.*	200.1.1.3	143	Accept
19	200.1.2.254	200.1.1.4	22	Accept
20	200.1.2.*	200.1.1.4	53	Accept
21	200.1.2.*	200.1.1.5	22	Accept
22	200.1.2.*	200.1.1.5	3306	Accept
23	200.1.2.*	200.1.1.*	*	Deny
24	200.1.2.*	*	*	Accept
25	*	200.1.1.2	80	Accept
26	*	200.1.1.2	443	Accept
27	*	200.1.1.3	25	Accept
28	*	200.1.1.4	53	Accept
29	*	*	*	Deny

In the context of firewall, rule conflicts can be classified into two categories, the ones causing speed issues and the ones causing security problems, respectively. As discussed in [12], [27][28] and [38], these rule conflicts result from shadowed rules and redundant rules, and they present critical impact on the performance of traditional firewalls.

Specifically, shadowed rules result in security problems on a traditional firewall. Rules blocking attack packets can be shadowed by some other rules with higher priorities (i.e., positioned ahead of them) and may not be used by the firewall at all. This, consequently, causes security problems and weakens the firewall [28][38]. Redundant rules decrease the processing speed of a firewall [12][28][38]. This is because they are redundant to other rules and waste the firewall's time to process them. Therefore, shadowed rules and redundant rules should be cleaned from a firewall rule set to improve the functional speed of a firewall.

To detect these rule conflicts, Al-Shaer and Hamed applied the set theory in their work published in [12]. Their approach is to map the original listed rules to a 'policy tree'. The conflicting rules and the types of the conflicts are reported after detection is completed. The authors further extended their methods to discover anomalies inside distributed networks [13].

The methods proposed in [27] also aim to discover rule conflicts. However, the proposed method in [27] is based on relational algebra techniques. It can discover more rule conflicts in comparison with the method suggested in [12]. The findings highlighted in [12], [23] and [27] suggest potential solutions to remove these problematic rules from a firewall rule set.

In addition, tools such as Binary Decision Diagrams (BDDs) [25], Constraint Logic Programming (CLP) [26] and Fireman Toolkit [31] were proposed to help analyse and remove rule conflicts from the rule set of a listed-rule firewall.

Although these studies [12][23][25][26][27][31] have introduced several schemes to deal with rule conflicts, their solutions are not satisfactory to this problem yet because listed rules are still in favor of all these proposed schemes.

6.1.2 Enhancing processing speed via hardware implementation

Fong et al. [66] implemented their firewall on FPGA devices to achieve a Terabit per second throughput for large and complex rule sets. They presented a scalable parallel architecture, named ParaSplit, for high-performance packet classification. Moreover, a rule set partitioning algorithm based on range-point conversion was proposed to reduce the overall memory requirement [66].

Likewise, Erdem and Carus [67] proposed a multi-pipelined and memory-efficient firewall to classify packets. They designed high throughput SRAM-based parallel and pipelined architectures on FPGAs. Hager et al. [68] proposed the Massively Parallel Firewall Circuits

(MPFC) to generate customised firewall circuits in the form of synthesizable VHDL code for FPGA configuration. They claimed that MPFC circuits were highly parallel and could achieve a deterministic throughput of one packet per clock cycle.

However, the high speed performance achieved by the above-mentioned firewalls [66][67][68] was relied on special hardware (i.e., the FPGA) rather than on the design of a rule set architecture or development of a filtering algorithm.

6.1.3 Enhancing processing speed via advanced filtering mechanisms

Ni et al. [69] applied statistical analysis on two Transport layer protocol header fields of packets (i.e., Protocol and IP Address) based on the extracted features and the characteristics of multi-tree and dual-index strategy to decrease the firewall preprocessing time. This research used the 'data storage structure and search diagram' to filter packets. This structure is considered as a tree structure. However, the tree consists of only the fields of Protocol and IP address. It has no Port and Action fields in their tree. Moreover, firewall administrators still create firewall rules in a form of listed rule. Their approach compares the performance of their algorithm with Stochastic Distribution Multibit-trie (SDMTrie) algorithm [70] only. They claimed that their scheme was better than traditional firewalls and firewalls working with the SDMTrie algorithm. However, performance comparison with standard firewalls (e.g., IPTABLES, Cisco ACL) and any well-known firewall algorithm is not presented.

Trabelsi et al. [71] proposed an analytical dynamic multilevel early packet filtering mechanism to enhance firewall performance. The proposed mechanism uses statistical splay tree filters that utilise traffic characteristics to minimise packet filtering time. The statistical splay tree filters are reordered according to the network traffic divergence upon certain threshold qualification (Chi–Square Test). They claimed that this method was faster than traditional methods because unwanted packets were rejected as early as possible, and the proposed mechanism could also be considered as a device protection mechanism against Denial-of-Service (DoS) attacks.

Hung et al. used B-Tree [72] to improve the speed of classifying and processing packets on firewall. They proposed a new two-dimensional early packet rejection technique based on the B-Tree. They defined a core firewall process as the 'Original Filter', and created their new scheme called 'Early rejected filter'. Their work focused on preventing unwanted packets and applied the 'Original Filter' to minimise packets traversing to the core firewall process. Their scheme can reduce firewall processing time under DoS attacks. However, under normal network operations (without DoS attack), their 'Early rejected filter' scheme may slightly increase firewall processing time.

Liu and Gouda [13] proposed 'Diverse Firewall Design' using tree-structured rules, which are converted from a rule list, to discover and eradicate rule conflicts. However, their work was still based on listed rules of traditional firewalls.

Zhao et al. [21] proposed to use 'goto' function inside listed-rule firewalls (e.g., a 'jump' command in IPTABLES). Although their rule structure looks like a tree structure, their sub-rules (or nodes) contain listed rules. Therefore, their firewalls are still deemed as Listed-rule firewalls and are time consuming when performing linear and sequential rule searching.

Likewise, although the methods proposed in [12][23] can convert firewall rules to a 'policy tree', the 'policy tree' cannot be considered as a tree-based filtering firewall mentioned in these papers. This is because the 'policy tree' is used only for rule conflicts discovery but not for filtering packets.

Apart from the afore-discussed three types of approaches, recent research has been investigating to develop a new generation firewall based on Software Defined Networking (SDN). For example, the firewalls proposed in [73], [74], [75] and [76] employ SDN and support centralised management like SDN switches and SDN router do. However, this SDN-based approach focuses on connectivity and compatibility with other SDN devices instead of firewall rule optimization.

6.1.4 Background of Tree-Rule firewall

Chomsiri et al. have further studied firewall rules' problems, and published their interesting findings in [28] and [38] as shown in the previous chapters. They proposed a Tree-Rule firewall to overcome these problems. The Tree-Rule firewall not only organises firewall rules in a tree structure as shown in Figure 6.2 but also filters out unwanted packets in accordance with tree-structured rules. To inspect a packet, the Tree-Rule firewall first reads the relevant header fields from the packet. Then, the value of the first header field is compared with a firewall sub-rule stored in the root node of the tree. Afterwards, the firewall checks the other header fields sequentially against their respective tree nodes at the corresponding levels. Finally, a consequent action, such as an approval or a denial of access to the network, is taken on the packet. As shown in Figure 6.2, packet header fields including Destination IP address (Dest IP), Destination Port (Dest Port), and Source IP address (Source IP) are taken into account in the example Tree rule. This tree structure eases the design of firewall rules and makes sure that they are conflict free, namely non-shadowed and non-redundant rules.

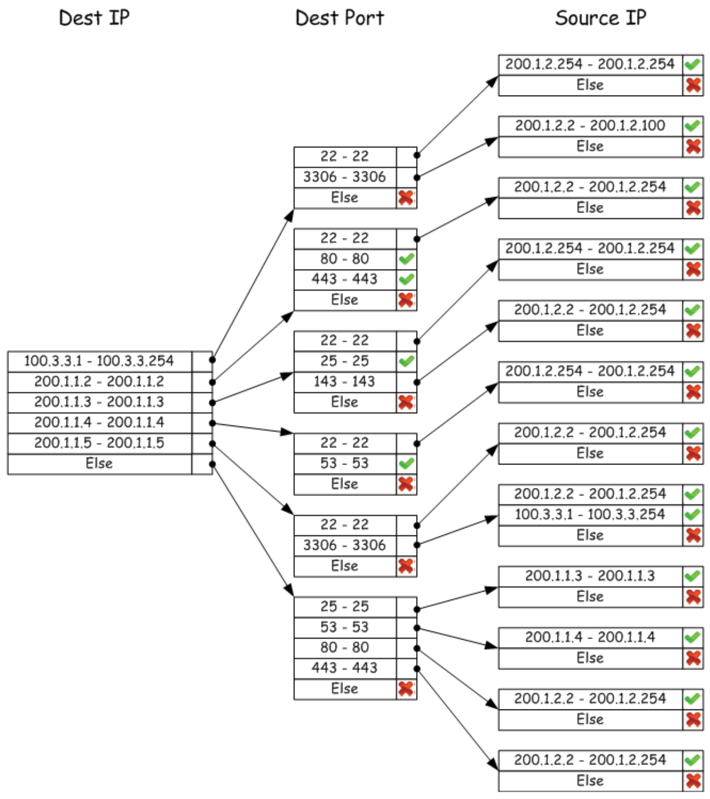


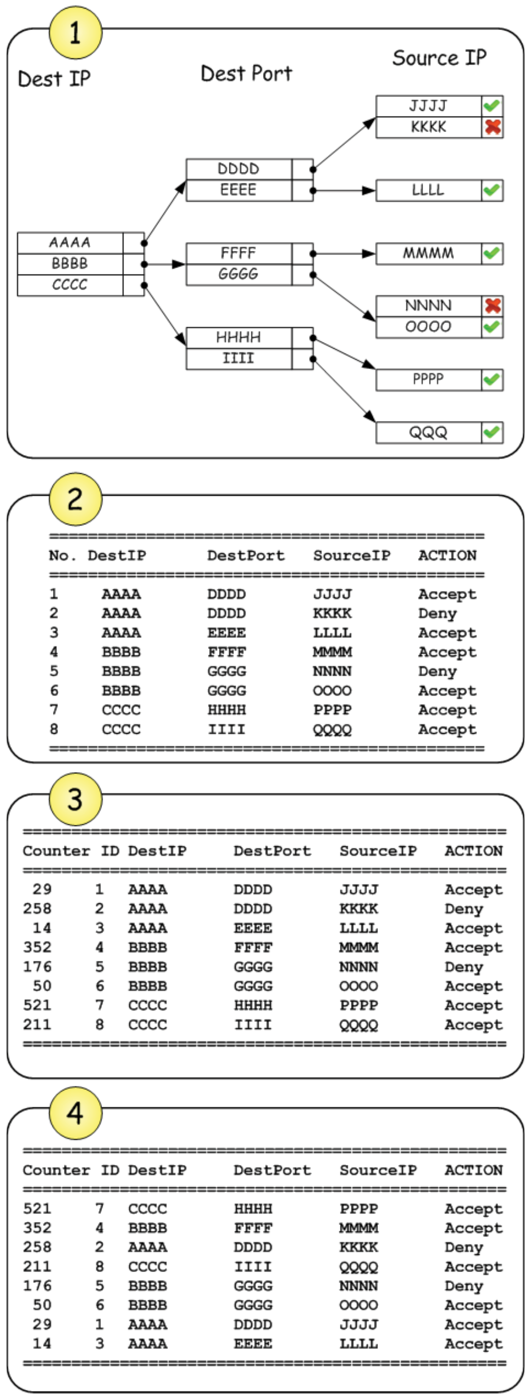
Figure 6.2: A Tree rule structure created for an example network in Figure 6.1 [38]

6.2 Our Approach

In this section, we propose a hybrid firewall which is a combination of a Tree-Rule firewall and a traditional firewall. A Tree-Rule firewall's GUI presented in our previous work [38] is used in the configuration phase to create tree rules, which are then converted to traditional conflict-free listed rules. During decision making, an incoming packet is verified against the listed rules sequentially until a match is found. Unlike the traditional firewalls, our hybrid firewall periodically re-arranges a sequence of rules. Each rule is independently moved to its suitable position in accordance with the number of matches with the incoming packets. For example, the rule matching with most packets is moved up to the top of the list in order to optimise the processing speed of the hybrid firewall.

6.2.1 Methodology

As shown in Figure 6.3, there are four steps involved in the process of our hybrid approach.



```
============================================
No. DestIP      DestPort   SourceIP   ACTION
============================================
1    AAAA       DDDD       JJJJ       Accept
2    AAAA       DDDD       KKKK       Deny
3    AAAA       EEEE       LLLL       Accept
4    BBBB       FFFF       MMMM       Accept
5    BBBB       GGGG       NNNN       Deny
6    BBBB       GGGG       OOOO       Accept
7    CCCC       HHHH       PPPP       Accept
8    CCCC       IIII       QQQQ       Accept
============================================
```

```
================================================
Counter  ID DestIP      DestPort   SourceIP   ACTION
================================================
 29      1   AAAA       DDDD       JJJJ       Accept
258      2   AAAA       DDDD       KKKK       Deny
 14      3   AAAA       EEEE       LLLL       Accept
352      4   BBBB       FFFF       MMMM       Accept
176      5   BBBB       GGGG       NNNN       Deny
 50      6   BBBB       GGGG       OOOO       Accept
521      7   CCCC       HHHH       PPPP       Accept
211      8   CCCC       IIII       QQQQ       Accept
================================================
```

```
================================================
Counter  ID DestIP      DestPort   SourceIP   ACTION
================================================
521      7   CCCC       HHHH       PPPP       Accept
352      4   BBBB       FFFF       MMMM       Accept
258      2   AAAA       DDDD       KKKK       Deny
211      8   CCCC       IIII       QQQQ       Accept
176      5   BBBB       GGGG       NNNN       Deny
 50      6   BBBB       GGGG       OOOO       Accept
 29      1   AAAA       DDDD       JJJJ       Accept
 14      3   AAAA       EEEE       LLLL       Accept
================================================
```

Figure 6.3: Four steps of proposed scheme [77]

In the first step shown in Figure 6.3-(1), a tree rule is created using the GUI by a firewall rule designer. The created tree rule is then converted into listed rules as shown in Figure 6.3-(2). The listed rule is then used in a core firewall for verifying against the header fields of an incoming packet. 'Counter' field shown in Figure 6.3-(3) records the number of packets matched with each rule and is initially set to 0 for each rule. The 'Counter' of a rule will increase by one when a match between an incoming packet and the rule is confirmed. The counter determines which rule is most frequently matched. To reduce the computational time, the most frequently matched rule is relocated in the top of the list as shown in Figure 6.3-(4). The counters of all the rules will be reset to 0 when a pre-determined 'Time interval' (e.g., 3 seconds) is reached. The 'Time interval' is specified by a firewall administrator.

When putting into practice, a range of IP addresses and a range of ports are applied in each line within nodes. The root node shown on the left-hand side of Figure 6.2 consists of six lines. The range of numbers in each line does not overlap with the ranges of numbers in other lines within a same node. For example, the range [100.3.3.1-100.3.3.254] does not overlap with the range [200.1.1.2-200.1.1.2]. Likewise, the ranges of numbers in lines within a node (e.g., the first node of 'Dest Port' column) do not overlap with each other as well. These non-overlapping ranges allow us to transform a tree rule into a set of conflict-free listed rules.

Transforming a tree rule into a listed rule can be done for one rule path at a time. For example, the first rule path

([100.3.3.1-100.3.3.254]-->[22-22]-->[200.1.2.254 200.1.2.254]-->Accept)

can be transformed into the listed rule shown in Table 6.2. The second rule path

([100.3.3.1-100.3.3.254]-->[22-22]-->[Else]-->Deny)

can be transformed into the listed rule shown in Table 6.3.

Table 6.2: Example of a listed rule transformed from a rule path

Rule No.	Dest-IP	Dest-Port	Source-IP	ACTION
1	100.3.3.1-100.3.3.254	22-22	200.1.2.254-200.1.2.254	Accept

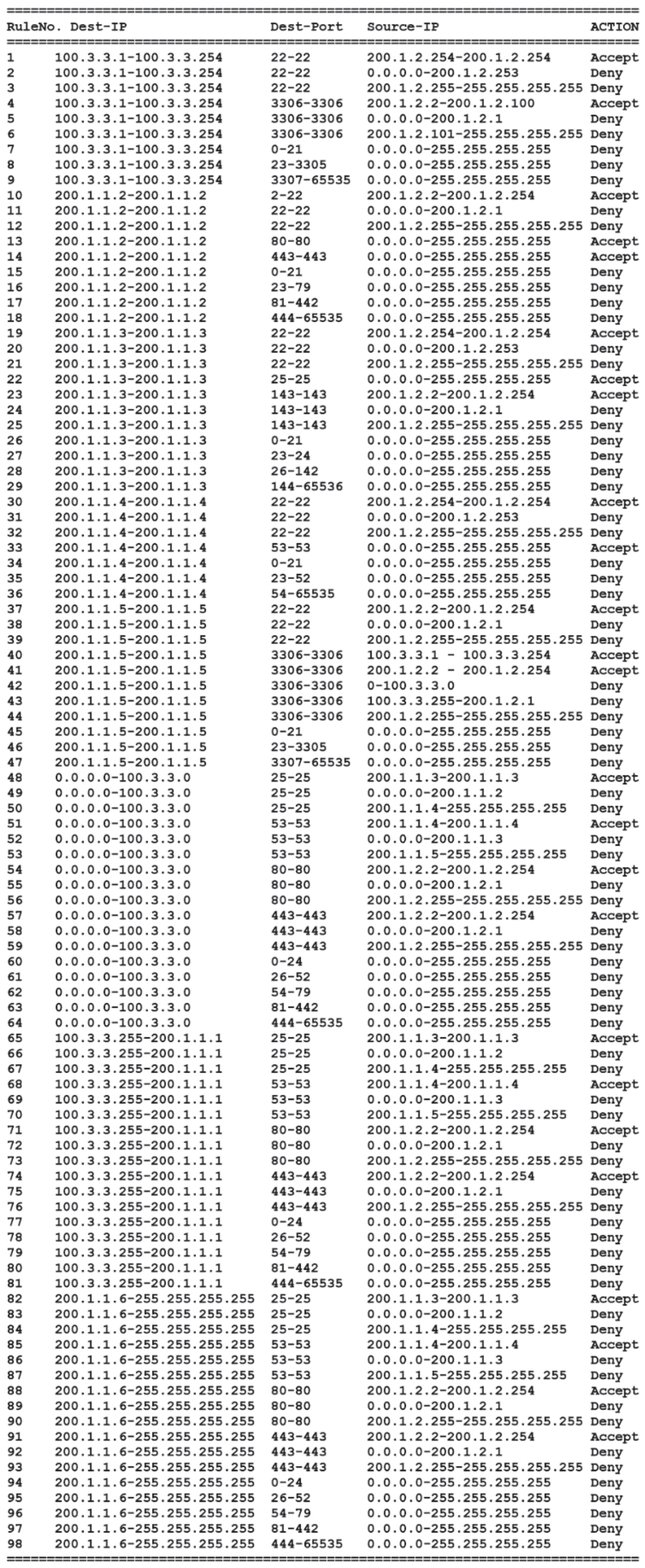
Table 6.3: Examples of two listed rules transformed from a rule path

Rule No.	Dest-IP	Dest-Port	Source-IP	ACTION
2	100.3.3.1-100.3.3.254	22-22	0.0.0.0-200.1.2.253	Deny
3	100.3.3.1-100.3.3.254	22-22	200.1.2.255-255.255.255.255	Deny

Bearing the same idea in mind, the tree rules shown in Figure 6.2 can be transformed to the listed rules shown in Table 6.4.

After designing and transforming tree rules into listed rules using the GUI, the listed rules shown in Table 6.4 are loaded into the memory of the core firewall for verifying against incoming packets. The counter of each rule will be increased individually when a packet is matched with a rule. All rules are sorted in descending order according to the value of a counter.

Table 6.4: The listed rules transformed from the tree rules in Figure 6.2 [77]



RuleNo.	Dest-IP	Dest-Port	Source-IP	ACTION
1	100.3.3.1-100.3.3.254	22-22	200.1.2.254-200.1.2.254	Accept
2	100.3.3.1-100.3.3.254	22-22	0.0.0.0-200.1.2.253	Deny
3	100.3.3.1-100.3.3.254	22-22	200.1.2.255-255.255.255.255	Deny
4	100.3.3.1-100.3.3.254	3306-3306	200.1.2.2-200.1.2.100	Accept
5	100.3.3.1-100.3.3.254	3306-3306	0.0.0.0-200.1.2.1	Deny
6	100.3.3.1-100.3.3.254	3306-3306	200.1.2.101-255.255.255.255	Deny
7	100.3.3.1-100.3.3.254	0-21	0.0.0.0-255.255.255.255	Deny
8	100.3.3.1-100.3.3.254	23-3305	0.0.0.0-255.255.255.255	Deny
9	100.3.3.1-100.3.3.254	3307-65535	0.0.0.0-255.255.255.255	Deny
10	200.1.1.2-200.1.1.2	2-22	200.1.2.2-200.1.2.254	Accept
11	200.1.1.2-200.1.1.2	22-22	0.0.0.0-200.1.2.1	Deny
12	200.1.1.2-200.1.1.2	22-22	200.1.2.255-255.255.255.255	Deny
13	200.1.1.2-200.1.1.2	80-80	0.0.0.0-255.255.255.255	Accept
14	200.1.1.2-200.1.1.2	443-443	0.0.0.0-255.255.255.255	Accept
15	200.1.1.2-200.1.1.2	0-21	0.0.0.0-255.255.255.255	Deny
16	200.1.1.2-200.1.1.2	23-79	0.0.0.0-255.255.255.255	Deny
17	200.1.1.2-200.1.1.2	81-442	0.0.0.0-255.255.255.255	Deny
18	200.1.1.2-200.1.1.2	444-65535	0.0.0.0-255.255.255.255	Deny
19	200.1.1.3-200.1.1.3	22-22	200.1.2.254-200.1.2.254	Accept
20	200.1.1.3-200.1.1.3	22-22	0.0.0.0-200.1.2.253	Deny
21	200.1.1.3-200.1.1.3	22-22	200.1.2.255-255.255.255.255	Deny
22	200.1.1.3-200.1.1.3	25-25	0.0.0.0-255.255.255.255	Accept
23	200.1.1.3-200.1.1.3	143-143	200.1.2.2-200.1.2.254	Accept
24	200.1.1.3-200.1.1.3	143-143	0.0.0.0-200.1.2.1	Deny
25	200.1.1.3-200.1.1.3	143-143	200.1.2.255-255.255.255.255	Deny
26	200.1.1.3-200.1.1.3	0-21	0.0.0.0-255.255.255.255	Deny
27	200.1.1.3-200.1.1.3	23-24	0.0.0.0-255.255.255.255	Deny
28	200.1.1.3-200.1.1.3	26-142	0.0.0.0-255.255.255.255	Deny
29	200.1.1.3-200.1.1.3	144-65536	0.0.0.0-255.255.255.255	Deny
30	200.1.1.4-200.1.1.4	22-22	200.1.2.254-200.1.2.254	Accept
31	200.1.1.4-200.1.1.4	22-22	0.0.0.0-200.1.2.253	Deny
32	200.1.1.4-200.1.1.4	22-22	200.1.2.255-255.255.255.255	Deny
33	200.1.1.4-200.1.1.4	53-53	0.0.0.0-255.255.255.255	Accept
34	200.1.1.4-200.1.1.4	0-21	0.0.0.0-255.255.255.255	Deny
35	200.1.1.4-200.1.1.4	23-52	0.0.0.0-255.255.255.255	Deny
36	200.1.1.4-200.1.1.4	54-65535	0.0.0.0-255.255.255.255	Deny
37	200.1.1.5-200.1.1.5	22-22	200.1.2.2-200.1.2.254	Accept
38	200.1.1.5-200.1.1.5	22-22	0.0.0.0-200.1.2.1	Deny
39	200.1.1.5-200.1.1.5	22-22	200.1.2.255-255.255.255.255	Deny
40	200.1.1.5-200.1.1.5	3306-3306	100.3.3.1 - 100.3.3.254	Accept
41	200.1.1.5-200.1.1.5	3306-3306	200.1.2.2 - 200.1.2.254	Accept
42	200.1.1.5-200.1.1.5	3306-3306	0-100.3.3.0	Deny
43	200.1.1.5-200.1.1.5	3306-3306	100.3.3.255-200.1.2.1	Deny
44	200.1.1.5-200.1.1.5	3306-3306	200.1.2.255-255.255.255.255	Deny
45	200.1.1.5-200.1.1.5	0-21	0.0.0.0-255.255.255.255	Deny
46	200.1.1.5-200.1.1.5	23-3305	0.0.0.0-255.255.255.255	Deny
47	200.1.1.5-200.1.1.5	3307-65535	0.0.0.0-255.255.255.255	Deny
48	0.0.0.0-100.3.3.0	25-25	200.1.1.3-200.1.1.3	Accept
49	0.0.0.0-100.3.3.0	25-25	0.0.0.0-200.1.1.2	Deny
50	0.0.0.0-100.3.3.0	25-25	200.1.1.4-255.255.255.255	Deny
51	0.0.0.0-100.3.3.0	53-53	200.1.1.4-200.1.1.4	Accept
52	0.0.0.0-100.3.3.0	53-53	0.0.0.0-200.1.1.3	Deny
53	0.0.0.0-100.3.3.0	53-53	200.1.1.5-255.255.255.255	Deny
54	0.0.0.0-100.3.3.0	80-80	200.1.2.2-200.1.2.254	Accept
55	0.0.0.0-100.3.3.0	80-80	0.0.0.0-200.1.2.1	Deny
56	0.0.0.0-100.3.3.0	80-80	200.1.2.255-255.255.255.255	Deny
57	0.0.0.0-100.3.3.0	443-443	200.1.2.2-200.1.2.254	Accept
58	0.0.0.0-100.3.3.0	443-443	0.0.0.0-200.1.2.1	Deny
59	0.0.0.0-100.3.3.0	443-443	200.1.2.255-255.255.255.255	Deny
60	0.0.0.0-100.3.3.0	0-24	0.0.0.0-255.255.255.255	Deny
61	0.0.0.0-100.3.3.0	26-52	0.0.0.0-255.255.255.255	Deny
62	0.0.0.0-100.3.3.0	54-79	0.0.0.0-255.255.255.255	Deny
63	0.0.0.0-100.3.3.0	81-442	0.0.0.0-255.255.255.255	Deny
64	0.0.0.0-100.3.3.0	444-65535	0.0.0.0-255.255.255.255	Deny
65	100.3.3.255-200.1.1.1	25-25	200.1.1.3-200.1.1.3	Accept
66	100.3.3.255-200.1.1.1	25-25	0.0.0.0-200.1.1.2	Deny
67	100.3.3.255-200.1.1.1	25-25	200.1.1.4-255.255.255.255	Deny
68	100.3.3.255-200.1.1.1	53-53	200.1.1.4-200.1.1.4	Accept
69	100.3.3.255-200.1.1.1	53-53	0.0.0.0-200.1.1.3	Deny
70	100.3.3.255-200.1.1.1	53-53	200.1.1.5-255.255.255.255	Deny
71	100.3.3.255-200.1.1.1	80-80	200.1.2.2-200.1.2.254	Accept
72	100.3.3.255-200.1.1.1	80-80	0.0.0.0-200.1.2.1	Deny
73	100.3.3.255-200.1.1.1	80-80	200.1.2.255-255.255.255.255	Deny
74	100.3.3.255-200.1.1.1	443-443	200.1.2.2-200.1.2.254	Accept
75	100.3.3.255-200.1.1.1	443-443	0.0.0.0-200.1.2.1	Deny
76	100.3.3.255-200.1.1.1	443-443	200.1.2.255-255.255.255.255	Deny
77	100.3.3.255-200.1.1.1	0-24	0.0.0.0-255.255.255.255	Deny
78	100.3.3.255-200.1.1.1	26-52	0.0.0.0-255.255.255.255	Deny
79	100.3.3.255-200.1.1.1	54-79	0.0.0.0-255.255.255.255	Deny
80	100.3.3.255-200.1.1.1	81-442	0.0.0.0-255.255.255.255	Deny
81	100.3.3.255-200.1.1.1	444-65535	0.0.0.0-255.255.255.255	Deny
82	200.1.1.6-255.255.255.255	25-25	200.1.1.3-200.1.1.3	Accept
83	200.1.1.6-255.255.255.255	25-25	0.0.0.0-200.1.1.2	Deny
84	200.1.1.6-255.255.255.255	25-25	200.1.1.4-255.255.255.255	Deny
85	200.1.1.6-255.255.255.255	53-53	200.1.1.4-200.1.1.4	Accept
86	200.1.1.6-255.255.255.255	53-53	0.0.0.0-200.1.1.3	Deny
87	200.1.1.6-255.255.255.255	53-53	200.1.1.5-255.255.255.255	Deny
88	200.1.1.6-255.255.255.255	80-80	200.1.2.2-200.1.2.254	Accept
89	200.1.1.6-255.255.255.255	80-80	0.0.0.0-200.1.2.1	Deny
90	200.1.1.6-255.255.255.255	80-80	200.1.2.255-255.255.255.255	Deny
91	200.1.1.6-255.255.255.255	443-443	200.1.2.2-200.1.2.254	Accept
92	200.1.1.6-255.255.255.255	443-443	0.0.0.0-200.1.2.1	Deny
93	200.1.1.6-255.255.255.255	443-443	200.1.2.255-255.255.255.255	Deny
94	200.1.1.6-255.255.255.255	0-24	0.0.0.0-255.255.255.255	Deny
95	200.1.1.6-255.255.255.255	26-52	0.0.0.0-255.255.255.255	Deny
96	200.1.1.6-255.255.255.255	54-79	0.0.0.0-255.255.255.255	Deny
97	200.1.1.6-255.255.255.255	81-442	0.0.0.0-255.255.255.255	Deny
98	200.1.1.6-255.255.255.255	444-65535	0.0.0.0-255.255.255.255	Deny

6.2.2 Discussion on efficiency

Although various methods [12][13][23][26][27][31] have been designed to minimise rule conflicts through re-arrangement of those frequent matched rules to the top positions in a rule list, they do not guarantee that a conflict-free rule list can be reached.

Let us take the rule list illustrated in Table 6.1 as an example. When the network is under attack of worms, the last rule will be the most frequently matched rule within the list and is applied to drop those attack packets. Therefore, the last rule, namely Rule-29, will be repositioned to the top of the list. This creates an undesirable consequence that all following incoming packets are blocked by Rule-29 even though they may be allowed by the other rules below. In contrast, individual listed rules created by our proposed scheme as shown in Table 6.4 can be moved to any position independently.

Moreover, given that the most frequently matched rules are listed at the bottom of a rule list, data transmission overhead of the aforementioned firewalls increase along with the expansion of their rule lists. This is because that it takes a firewall's time to process unmatched rules before reaching the matched one and allowing/denying packets to pass through. According to our studies, 1000 redundant rules can reduce data transmission speed by approximately 10%. The drop of speed depends on several factors, e.g., type of firewall [78] and CPU speed of the machine running the firewall.

The decrease of data transmission speed prolongs data transmission time of a system (e.g., time consumption for downloading the data increases 10% if data transmission speed drops by 10% as shown in Figures 6.4-(a) and (b) respectively). Moving the matched rule from the bottom of firewall rule list to the top position (e.g., from rule number 1000 to rule number 1 enhances the data transmission speed and shortens transmission time as illustrated in Figure 6.4-(c).

Using our proposed scheme, rule sorting is executed periodically for each specified time interval, such as 1 second, 3 seconds or 5 seconds. Sorting the firewall rules takes less time in comparison with rule matching. Time consumption for data transmission using our proposed scheme can be found in Figures 6.4-(d). The time consumption shown in Figures 6.4-(d) is more than that revealed in Figures 6.4-(c) but less than that revealed in Figures 6.4-(b).

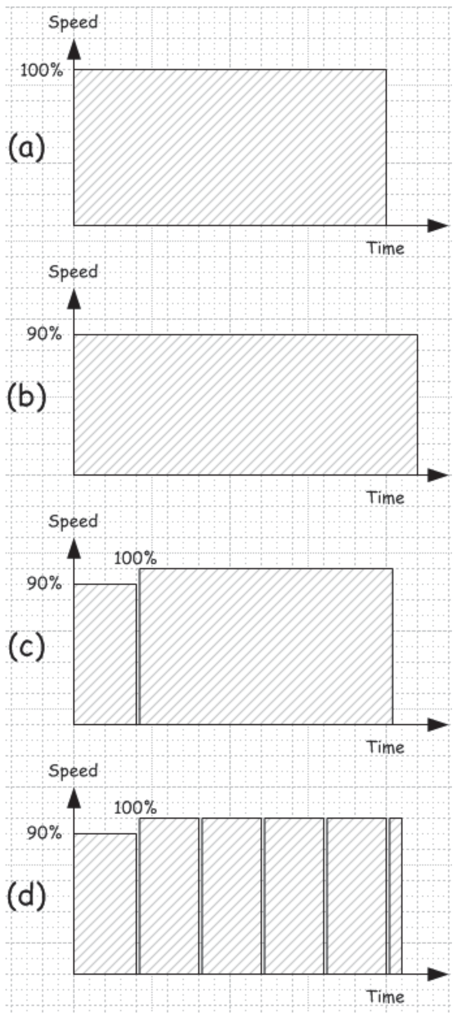


Figure 6.4: Transmission speed versus transmission time [77]

In summary, there are five main factors determining time consumption, *T*, for data transmission and they are shown as follows.

- Time interval (*w*)

- Data size (*F*)

- Network speed (*S*)

- Efficiency of transmission speed before rule sorting (*e*)

- Time for sorting rules (*g*)

Figure 6.5 illustrates the time (*T*) used for transmitting data and the five main factors. *x* axis and *y* axis denote transmission time and transmission speed respectively. The figure reveals the relation between time *T* used for data transmission and the five important factors (i.e., *w*, *F*, *S*, *e* and *g*). In this example, we assume that the matched rule is at the bottom position of a rule list. The size of the rule list is 1000, which decreases transmission speed by roughly 10% of the maximum speed. In the first state, transmission speed begins with 90% ($e = 0.9$) until the time reach the Time Interval (*w*). Then, the firewall takes time *g* to sort its rules. We assume that the transmission speed during this period of time is 0 because the firewall is sorting its rules and not processing any packets. At this moment, data which have been transmitted is denoted as A1. After the rule sorting is complete, the firewall continues to process packets with its sorted rules. The transmission speed can peak at 100% because the matched rule has been moved up to the top position. When time interval *w* ends, the firewall takes time *g* to re-sort its rules again. This process repeats until the transmission of the last block of data (A6) is complete. The time *v* used to transmit the last block may be smaller than *w*. The total amount of data (*F*) transmitted is $F=A1+A2+A3+A4+A5+A6$.

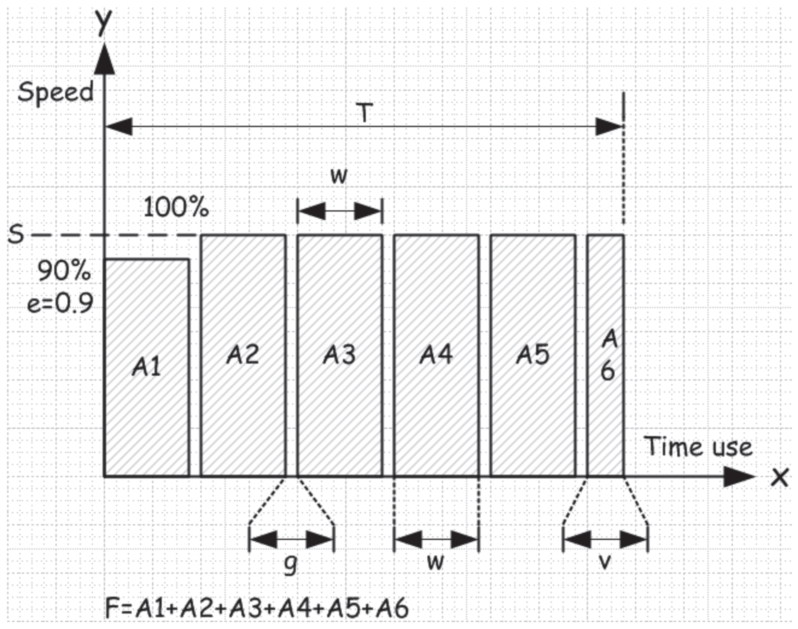


Figure 6.5: Time (*T*) used for data transmission and the five main factors (*w*, *F*, *S*, *e* and *g*).

The efficiency, *e*, is determined by the number of rules. We have created a special program to measure *e* with 1000 rules on a 2.8 GHz CPU computer and 345 Mbps network speed. We find that *e* is approximately 0.9. However, the value of *e* may vary in different environments because it is influenced by multiple factors. Like *e*, *g* is also determined by the number of rules. However, it equals to the base 2 logarithm of the number of rules because the Quick Sort [79] is used for rule sorting in this chapter. Thus, *g* increases slightly while the number of rules increases. We measure *g* in the same environment where *e* is computed. We find that the value of *g* is approximately 1 millisecond for 1000 rules. The *w* is a free parameter and assigned by firewall administrators. It can be 1, 3 or 5 seconds. However, transmission time may be longer than usual if *w* is specified inappropriately. The details of *w* will be discussed later in subsection 6.2.4.

6.2.3 A mathematical model for measuring time consumption

Let

- n denote the number of data blocks that do not include the first and the last data block (e.g., $n = 4$ in Figure 6.5),
- F denote the size of data being transmitted (in bits),
- e denote the efficiency of transmission speed before sorting the rules, $0 < e < 1$,
- S denote the speed of network (in bits per seconds),
- w denote the time interval between two rule sortings (in seconds),
- g denote the time used for rules sorting (in seconds),
- v denote the time span of transmitting the last block (in seconds), e.g., the time span of *A6* in Figure 6.5,
- u denote v/w, $0 < u < 1$, and
- T denote the time used for data transmissions.

Then, we have

$$F = eSw + nSw + Sv$$

$$F - eSw = nSw + Sv$$

$$(F - eSw)/S = nw + v = nw + uw$$

$$(F - eSw)/Sw = n + u$$

$$\begin{aligned} & n+u \\ & =(F-eSw)/Sw \\ & =(F/Sw)-e \end{aligned} \qquad (1)$$

The time T used for data transmissions shown in Figure 6.5 is defined as,

$$\begin{aligned} & T \\ & =(w+g)+n(w+g)+v \\ & =(w+g)+n(w+g)+uw \\ & =(w+g)+n(w+g)+u(w+g)-ug \\ & =(w+g)+(n+u)\times(w+g)-ug \end{aligned} \qquad (2)$$

Substituting Equation (1) into Equation (2), we have that

$$\begin{aligned} & T \\ & =(w+g)+((F/Sw)-e)\times(w+g)-ug \\ & =(w+g)-e(w+g)+(\frac{(w+g)F}{Sw})-ug \\ & =(1-e)(w+g)+\frac{F}{S}(1+\frac{g}{w})-ug \end{aligned} \qquad (3)$$

Equation (3) reveals that the larger the data size F is, the longer time it takes a system to transmit data. Similarly, the higher the network speed S is, the shorter time the system will take to transmit data. Moreover, g, w, and e also play important roles in determining the time used for data transmission.

We have conducted a simple testing using this formula on Microsoft Excel and given some input data for observing the result and output graphs. The results are shown in Figure 6.6. We specify $F = 2048$ MB (16384 Mbits), $S = 300$ Mbps, $g = 0.001$ seconds, and $e = 0.9$. We calculated the consumption time T for w = 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 3.75, 4.00, 4.25, 4.50, 4.75, and 5.00 respectively. Figure 6.6 shows a relation between the consumption time T in the vertical axis and Time Interval w in the horizontal axis.

The curve of graph tells us that there is the optimal value of w which can give the minimum consumption time T for data transferring. In this case, w = 0.75 causes T = 54.7603, which is better than the values T = 54.7673, T = 54.9313 and T = 55.1243 when w = 1.00, 3.00, and 5.00, respectively.

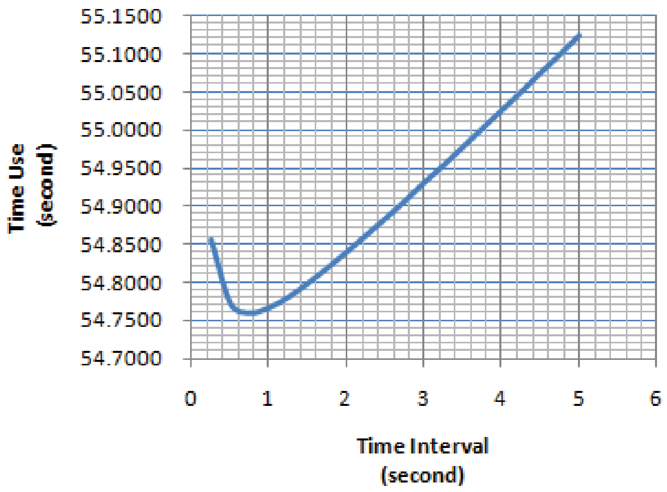


Figure 6.6: Relation between Time use (T) and Time Interval (w) [77]

Regarding the operation without using our proposed scheme, the firewall will take the time calculated using Equation (4) below for data transferring.

$$T = \frac{F}{eS} \tag{4}$$

Thus, in this example, without using our proposed scheme, the firewall will take time: T = 16384/(0.9*300) = 60.6815 seconds. In contrast, using our proposed scheme, the transferring time can be saved for 9.76% for w = 0.75, and 9.75%, 9.48% and 9.16% for w = 1, 3 and 5 seconds, respectively.

6.2.4 Determining time interval *w*

To determine the time interval *w*, we create a special program to measure a time used for sorting 1000 rules. We find that the sorting takes less than 1 millisecond. Taking a four minute data transmission as an example, the sorting function is executed 80 (= 4*60/3) times if rules are sorted every 3 seconds. The overall time taken for rule sorting is merely 80 milliseconds, which is very small in comparison with 4 minutes for the whole process. In the networks that have a small size of data transmission, setting the Time Interval to 3 seconds or 5 seconds may not be suitable because a time use *T* of the firewall applying the proposed scheme may be bigger than a time use *T* of the firewall without applying the proposed scheme (noting that the proposed scheme may waste firewall processing times due to the sorting time *g* as shown in Equation (3)). Firewall administrators should calculate and set a good value of Time Interval *w* to the firewall before using it. The proposed scheme focuses on a cloud, which is mostly working with big size of data transferring. Thus, we can set the Time Interval *w* to any value (e.g., 3 or 5 seconds) as long as the *T* calculated from Equation (3) is less than the *T* calculated from Equation (4).
We have found that the optimal Time Interval can be accurately estimated using Equation (5) below.

$$w = \sqrt{\frac{Fg}{S(1-e)}} \tag{5}$$

We have derived Equation (5) based on Calculus from a function represented as $T = f(w)$, showing the relationship between the time used (T) and the time interval (w). The optimal w occurs at the minimum point on the curve represented by this relation function (see Figure 6.6) and can be obtained by differentiating T with respect to w as shown in Equation (6) below.

$$\frac{dT}{dw} = 0 \tag{6}$$

From Equation (3) in subsection 6.2.3, *T* can be calculated by:

$$T = \frac{F}{S}(\frac{g}{w} + 1) + (1 - e)(w + g) - ug.$$

Therefore, Equation (6) is equivalent to

$$\begin{aligned}
&\frac{d}{dw}\left(\frac{F}{S}(\frac{g}{w} + 1) + (1 - e)(w + g) - ug\right) \\
&= \frac{d}{dw}\left(\frac{F}{S}(\frac{g}{w} + 1) + (1 - e)(w + g)\right) \\
&= \frac{d}{dw}\left(\frac{Fgw^{-1}}{S} + \frac{F}{S} + (1 - e)w + (1 - e)g\right) \\
&= \frac{d}{dw}\left(\frac{Fgw^{-1}}{S} + (1 - e)w\right) = -\frac{Fg}{Sw^2} + (1 - e) = 0.
\end{aligned}$$

Thus, $w = \sqrt{\frac{Fg}{S(1 - e)}}$ that proves Equation (5).

In Figure 6.6, we have calculated the time use (*T*) for *w* = 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00, 3.25, 3.50, 3.75, 4.00, 4.25, 4.50, 4.75, and 5.00, respectively, using Microsoft Excel. We have found that the optimal *w* is 0.75 as discussed in subsection 6.2.2. With the same environments and parameters (i.e., the same values of *F*, *S*, *g* and *e*), we have calculated *w* using Equation (5) and found that the optimal *w* is 0.739008. Therefore, it can be concluded that the optimal *w* can be estimated by either of the two methods as follows.

- Using Equation (3) to find the minimum *T* for various input values of *w*
- Directly using Equation (5)

6.3 Implementation and Experimentation

Similar to our previous schemes [38][60], we implement the proposed schemes based on the Netfilter module [56][57][11]. We hook packets' events using a technique presented in [80] by calling the function named 'nf_register_hook' [80]. Before calling this function, the hooking function must be declared first, as such in the line: 'nfho.hook = hook_func'. When packets arrive at the firewall, the 'hook_func' will be called. It will receive several important parameters as shown below:

```
unsigned int hook_func(unsigned int hooknum,
        struct sk_buff *skb,
        const struct net_device *in,
        const struct net_device *out,
        int (*okfn)(struct sk_buff *))
{
}
```

6.3.1 Experimental setup and environment

We create the Tree-Rule firewall using C on Cent OS 6.3 Linux. It operates as a kernel module and runs in a kernel level. Our original firewall source code, 'firewall.c', is compiled into the 'firewall.ko' and can be executed by the command '# insmod firewall.ko'. We develop a rule editor GUI using C# on Windows. The firewall rule is created by the GUI and is sent to the core firewall running on Linux. The rule structure is modified for handling listed rules and counter's information.

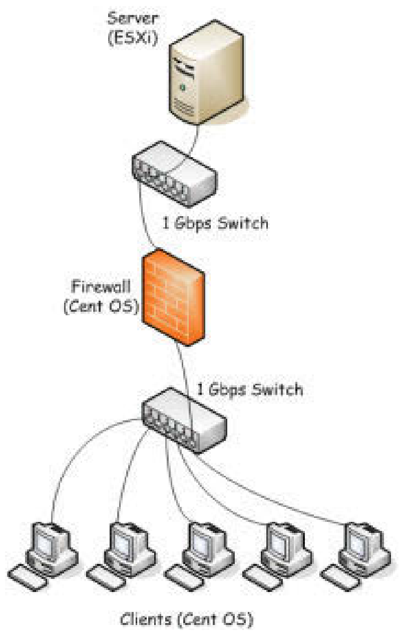


Figure 6.7: Experiment with ESXi

We evaluate the firewall on one Giga bits per second link speed LAN with seven standard PCs as shown in Figure 6.7. The five clients and the firewall machine in this testbed are equipped with a 2.4 GHz CPU and 4 GB RAM as well as a Cent OS 6.3. The server is equipped with a 2.8 GHz CPU and 8 GB RAM as well as an ESXi (by VM Ware company) as OS/Hypervisor in cloud environments. Within the server, we create five Virtual Machines (i.e., guest OSs) to serve as web servers (as shown in Figure 6.8). Each Virtual Machine (VM) runs a Cent OS 6.3. All Ethernet links operate on 1 Gbps speed including network switches. Based on our experience, the performance on different hypervisors, such as VMW, ESXi, Microsoft Hyper-V etc., are almost the same. Therefore, we decide to test only on ESXi for the proposed work in this chapter.

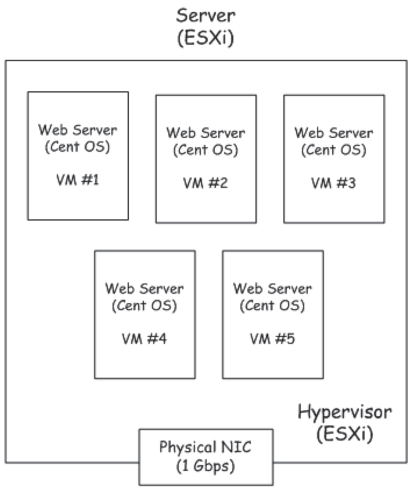


Figure 6.8: Five Linux Web Servers in an ESXi Hypervisor

In our experiments, the time used for downloading big size of data (e.g., big files) is measured. To do so, we store a 4 GB file on VM #1 (Server #1), and 2 GB files on VM #2 and VM #3, respectively. We also place 1 GB files on VM #4 and VM #5, respectively. During evaluation, client #1 downloads a file from VM #1 only. Likewise, client #i downloads a file from VM #i only. We measure the downloading times on both 'automatic rule sorting' and 'non-automatic rule sorting' modes.

6.3.2 Experiments

The equation used in subsection 6.2.4 for finding optimal *w* considers a single file containing firewall rules. However, in a real network, multiple files are simultaneously transmitted and each file may be matched with a different rule as well. Moreover, the size of each transmitting file may vary as well. Thus, finding the optimal "*w*" with multiple files is difficult. The selected *w* of 3 makes administrators easy to manage the network and takes a little time for rule-sorting. In the following example, a computer LAB is matched with one allowed rule, and opens 3 hours for users to use it. Assume that *w* is set to be 3 seconds on a firewall. In this case, the firewall will sort its rules 3*60*60/3 = 3,600 times. If one round of rule sorting takes 0.002 seconds, the total sorting time will be 3600*0.002=7.2 seconds, which is 0.067% in comparison to the 3 hours. This selected *w* leads to a little sorting time in total. The firewall application developed using the proposed scheme can display information in its monitor screen to inform administrator which rules are the frequently matched rules. It is similar to the 'top' command in Linux that shows the percentage of CPU used by each process. If we specify a too small *w* (e.g., 0.5 or 1 seconds), it is hard for administrators to read the information within such a short time window. In contrast, specifying a too big value of *w* (e.g., 5 or 10 seconds) will result in slow reaction to apply administrators' preferences. Hence, the *w* selected in our experiments is set to 3 seconds.

To begin with, we test on 3 cases with non-automatic rule sorting as shown in Cases #1, #2 and #3 of Figure 6.9. We create 500 firewall rules and intentionally make rule #250 matches with the 4 GB file. In this case, the first rule and the last rule will match 2 GB files, while rules #125 and #375 match with 1 GB files. This is for measuring the time consumption in the average case.

Case #2 is another average case for which five rules are in almost middle position. These files are matched with rules #248, #249, #250, #251 and #252, respectively. In case #3, we want to simulate the worst case by creating matched rules in positions 496, 497, 498, 499 and 500.

Secondly, we test with automatic rule sorting. We use a 3-second time interval (*w*), i.e., all rules are resorted every 3 seconds and a counter of each rule is reset to zero after all rules are resorted. Whilst five files are downloaded simultaneously, results of sorting may be different from the right bottom picture of Figure 6.9. They may be sorted in many sequences as shown in Figure 6.10.

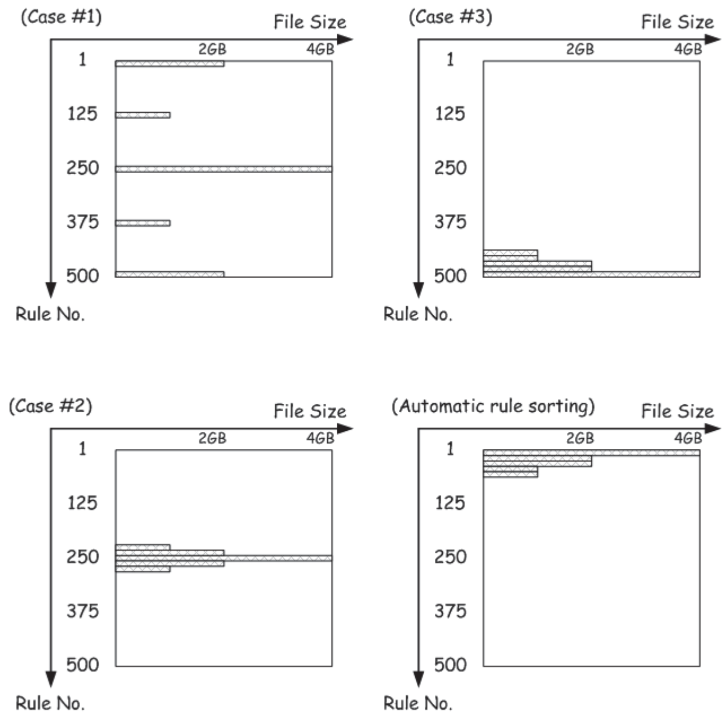


Figure 6.9: Three cases of 'non automatic rule sorting' and a case of 'automatic rule sorting'

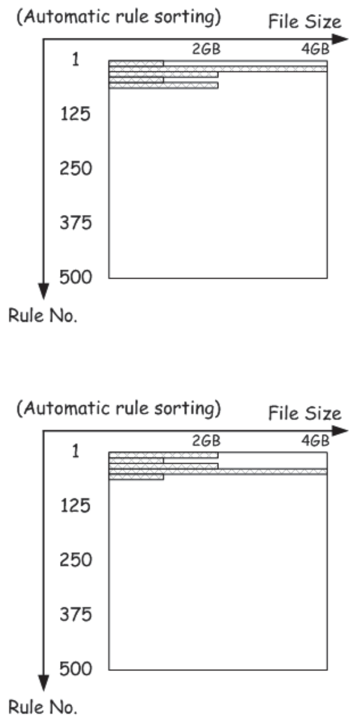
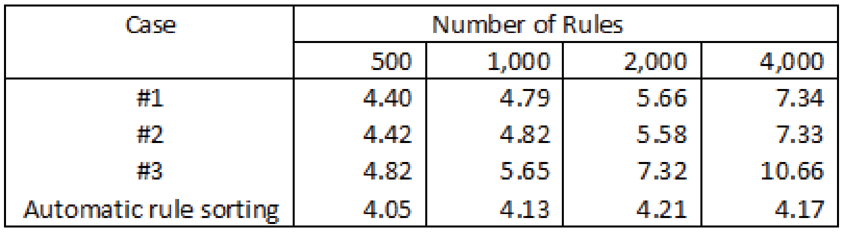


Figure 6.10: Sequences of rules in 'automatic rule sorting'

Lastly, we test with 1000, 2000 and 4000 rules, respectively. Five files start to transfer at the same time. We start a timer at this point. All packets of files travel through the firewall rules. We stop the timer when the transfer of the last file is complete. In each case, we conduct the experiments for five times, and the average result numbers are taken and highlighted in Table 6.5.

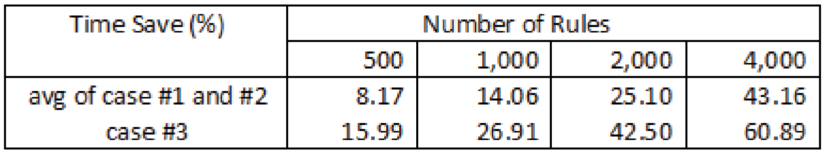
Table 6.5: Time consumption for transferring files from servers to clients (minutes) [77]



Case	Number of Rules			
	500	1,000	2,000	4,000
#1	4.40	4.79	5.66	7.34
#2	4.42	4.82	5.58	7.33
#3	4.82	5.65	7.32	10.66
Automatic rule sorting	4.05	4.13	4.21	4.17

Case #1 and Case #2 in Table 6.5 are average cases, whose results are very similar. Case #3 is the worst case that takes a longer time in comparison with Case #1 and Case #2. In the three cases, the downloading times are longer when the number of rules is increased. In the case of 'automatic rule sorting', firewall rules are sorted every 3 seconds so that five rules matching with fives active connections are moved to the top five positions. In other words, these rules are moved to rules with numbers 1, 2, 3, 4 and 5. The firewall has to verify packets against only the first five rules and is not necessary to process the remaining unmatched rules. Consequently, time consumption in this case is the smallest in comparison with the other cases. Moreover, the time consumptions for 500, 1000, 2000, and 4000 rules are slightly different. The percentages of time savings are presented in Table 6.6. As shown in Table 6.6, our scheme can reduce the processing time of the firewall with 500 rules by 8.17% on average. More time is saved in the cases with bigger rule sizes. For example, the proposed method saves 60.89% of the time for the case with 4,000 rules as shown in Table 6.6.

Table 6.6: Time saved in percentage [77]



Time Save (%)	Number of Rules			
	500	1,000	2,000	4,000
avg of case #1 and #2	8.17	14.06	25.10	43.16
case #3	15.99	26.91	42.50	60.89

Apart from testing on ESXi Hypervisor, we also conduct experiments setting up a small LAN with four servers, four clients and our Tree-Rule firewall in the perimeter. We compare the performance of our proposed firewall with IPTABLES, the most popular open-source firewall, using multiple sets of rules of different sizes. All computers including the firewall machine in this testbed are equipped with a 2.2 GHz CPU and 8 GB RAM. The firewall’s OS is Cent OS 6.3 while the Back Track 5 R3 is used as OS for servers and clients. The servers generate packets using 'hping3' command with '--flood' parameter to create and send the packets as fast as possible. This test uses 1440 bytes packet size. We choose a bigger packet size because HTTP typically uses packet size of 1400-1500 bytes.

The worst cases (when all packets are matched with the last rules) can be tested by creating one matched rule at the bottom position of firewall rule list. Apart from the last rule, other rules are considered unmatched rules. This condition is similar to case #3 of the previous experiments but using one matched rule at the bottom of rule list.

We measure the speeds of IPTABLES with different rule sizes of 100, 250, 500, 1000, 1500, 2000, 2500, 3000, 3500 and 4000 rules. The 'hping3' command with '--flood' can throttle the firewall to operate with its maximum speed (throughput). With no rules (rule size = 0), IPTABLES can process 30956 packets per second, as shown in Table 6.7. In Table 6.7, the firewall speed is represented in term of packets per second, and megabytes per second. The data are calculated using 1440 bytes packet size.

Table 6.7: Speed achieved through IPTABLES [77]

Number of rules	Speed		Drop (%)
	Packets/sec	Mega Bytes/sec	
-	30,956	42.51	-
100	27,964	38.40	9.67
250	25,099	34.47	18.92
500	21,226	29.15	31.43
1,000	16,202	22.25	47.66
1,500	13,172	18.09	57.45
2,000	11,103	15.25	64.13
2,500	9,580	13.16	69.05
3,000	8,471	11.63	72.64
3,500	7,526	10.34	75.69
4,000	6,653	9.14	78.51

We can see that the speed of IPTABLES drops from 42.51 MB/s to 22.25 MB/s (47.66%) with 1000 rules. The percentage of the speed drop increases when the firewall processes with a bigger rule size.

We also test the proposed firewall with the same condition (as tested for IPTABLES) by disabling the feature 'Automatic rule sorting'. As shown in Table 6.8, the speed of our firewall operating with rule size = 20000, 30000, 40000, 50000, 60000, 70000 and 80000 indicates that our firewall operates faster than the IPTABLES approximately by 20 times. For rule size = 1000, the speed of our firewall drops only 7.43%. In comparison, the IPTABLES speed drops by 47.66%. The two plots as shown in Figure 6.11 and Figure 6.12 translate the corresponding data presented in Table 6.7 and Table 6.8. In the two plots, the vertical axis of the graph represents the speeds of the firewall in MBytes/sec whereas the horizontal axis represents the numbers of rules.

Table 6.8: Speed of proposed firewall without 'Automatic rule sorting' [77]

Number of rules	Speed		Drop (%)
	Packets/sec	Mega Bytes/sec	
-	30,956	42.51	-
100	30,475	41.85	1.55
250	30,254	41.55	2.27
500	29,755	40.86	3.88
1,000	28,658	39.36	7.43
2,000	27,251	37.42	11.97
5,000	23,563	32.36	23.88
10,000	19,288	26.49	37.69
20,000	14,338	19.69	53.68
30,000	11,602	15.93	62.52
40,000	9,556	13.12	69.13
50,000	8,384	11.51	72.92
60,000	7,156	9.83	76.88
70,000	6,556	9.00	78.82
80,000	5,860	8.05	81.07

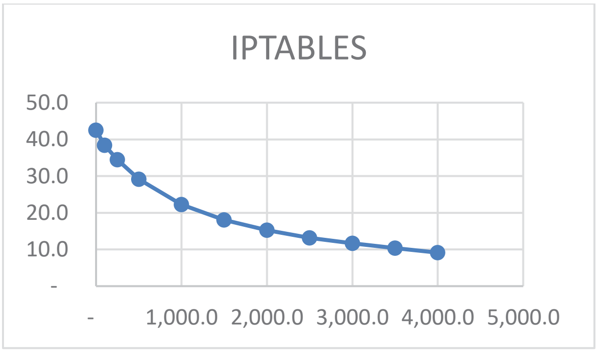


Figure 6.11: Speed of IPTABLES (represented in graph) [77]

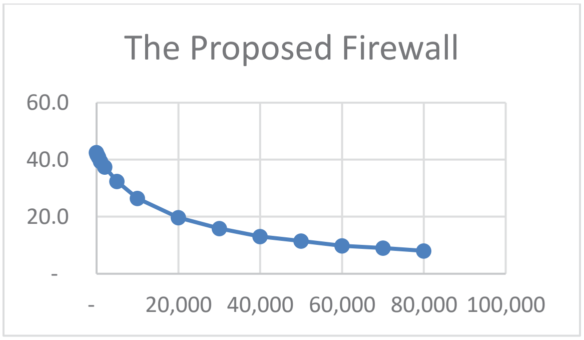


Figure 6.12: Speed of Proposed Firewall without 'Automatic rule sorting' (in graph) [77]

We perform more experiments for the proposed fire-wall to compare between operations with and without 'Automatic rule sorting'. Experimental results are presented in Table 6.9 and Figure 6.13.

Table 6.9: Speed of proposed firewall with 'Automatic rule sorting' [77]

Number of rules	Speed		Drop (%)
	Packets/sec	Mega Bytes/sec	
-	30,948	42.50	-
100	30,672	42.12	0.92
250	30,663	42.11	0.95
500	30,412	41.76	1.76
1,000	30,316	41.63	2.07
2,000	30,060	41.28	2.90
5,000	29,821	40.95	3.67
10,000	29,487	40.49	4.75
20,000	29,043	39.88	6.18
30,000	28,488	39.12	7.97
40,000	28,272	38.83	8.67
50,000	27,696	38.03	10.53
60,000	27,741	38.10	10.39
70,000	27,444	37.69	11.34
80,000	26,936	36.99	12.99

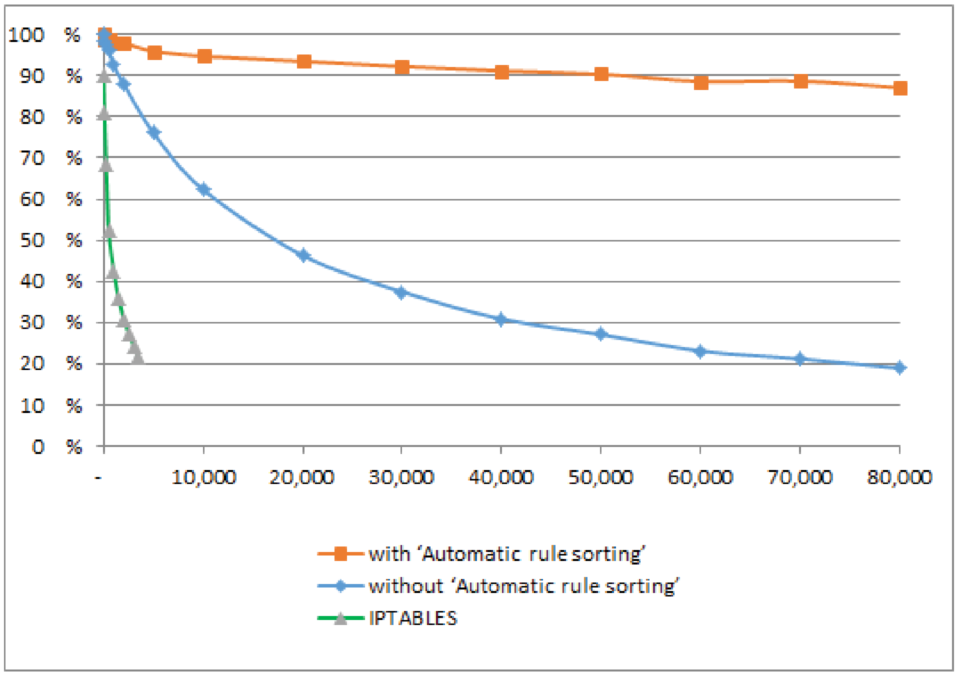


Figure 6.13: Comparison of Firewalls' speeds [77]

With rules size = 1000 in Table 6.9, the proposed firewall with 'Automatic rule sorting' gives 2.07% of speed drop whereas operating without 'Automatic rule sorting' gives 7.43% (see Table 6.8). Figure 6.13 shows the speed comparison for three firewalls, i.e., (1) the proposed firewall operating with 'Automatic rule sorting', (2) the proposed firewall operating without 'Automatic rule sorting', and (3) IPTABLES. The results shown through these graphs confirm that our proposed firewall with 'Automatic rule sorting' operates faster than IPTABLES significantly, and particularly with large size of rule set.

6.4 Conclusion

In this chapter, we have proposed a hybrid Tree-rule fire-wall which reduces the processing time in verifying packets. The proposed firewall applies the concepts of Tree-Rule firewalls in designing conflict-free rules and the concepts of traditional firewalls in decision making. Verifying incoming network packets against conflict-free listed rules contributes to a more secure and faster processing firewall. Counters have been introduced to analyse which rules matched with the most packets. The rules are sorted according to the counters periodically, and the most frequently matched rules are moved to the top positions. As such, the time spent in rule matching can be further reduced because a match can most possibly be found in the first few rules.

We have also proposed a mathematical model to illustrate a relation between the 'time use' for data transferring and other relevant factors, especially the 'time interval'. Moreover, we have proposed an equation for calculating an optimal 'time interval' with a mathematical proof based on Calculus.

Experiments have been conducted using our implemented testbed for evaluating the performance of our proposed hybrid firewall on transferring a big size of data. The experimental results have shown that our scheme can reduce the firewall processing time significantly.

Chapter 7 Conclusion

A firewall is an important network device placed between a network that we want to protect (e.g., company network) and a network that we cannot trust (e.g., an unknown network in the Internet), for regulating the packets travelling between networks. A firewall can be used to be a gateway to link two or more networks together, and it can verify packets travelling across networks using its rules pre-defined by firewall administrators.

Traditional firewalls, including packet filtering and stateful firewalls, are based on the list of condition lines. These firewalls have to process packets by comparing the packet header information with the listed rules. An operation using listed rule causes firewalls to have problems with rule conflicts, e.g. shadowing and redundancy anomalies. Shadowing anomaly can cause security problems while redundancy anomaly can cause a slow speed problem. Moreover, the slow speed problem can be intensified by the sequential processing for a firewall rule. Apart from the slow speed problem and security problems. Firewall administrators also face to difficulty problems when creating error-less firewall rules.

To overcome the aforementioned problems, in this thesis, we have conducted in-depth research on firewalls and developed an effective novel firewall to operate without rule conflicts, process packets faster, and be easy to use. A summary of the research conducted for this thesis is provided in Section 7.1, and potential future work is discussed in Section 7.2.

7.1 Summary

A review of traditional firewalls and rule-conflict detection schemes have been conducted in Chapter 2, followed by a review of hierarchical tree models in the existing work. Chapter 2 has also reviewed the stateful mechanism used by IPTABLES, the most popular firewall.

Chapter 3 has identified five important limitations of Listed-Rule Firewalls which can lead to security problems, speed problems, and 'difficult-to-use' problems. These limitations consist of:

(1) The limitation about "Shadowed rules" which can lead to security and speed problems,

(2) The limitation about swapping position between rules which can cause security problems,

(3) The limitation about "Redundant rules" which can cause speed problems,

(4) The limitation of rule design that can result in "difficult to use" problems, and

(5) The limitation from sequential computation that can lead to speed problems.

Chapter 3 has also presented various theories and their proofs to validate the arguments on the above limitations.

Chapter 4 has proposed the Tree-Rule firewall that demonstrates none of the limitations presented in Chapter 3. The Tree-Rule Firewall has been developed and discussed. Its advantages are listed below:

- No shadowed rule
- No swapping rule (the rule will be sorted automatically)
- No redundant rule
- Easy to design its rules (with independent rule path)
- High speed for packet decision

The Tree-Rule firewall utilises the rules in a tree data structure, and the forwarding decision of an input packet based on tree rules will follow the tree structure so that the decision on the packet becomes faster than the Listed-Rule firewalls. Chapter 4 has also presented a basic design and the improvement of the basic design of a Tree-Rule firewall.

The Tree-Rule firewall uses nodes and links between nodes to crate the tree rules. Each line in nodes may link to other nodes or link to an action. The rule sentence of a tree rule is called a 'rule path'. The GUI editor of the Tree-Rule firewall helps administrator design a tree rule, and maintains rule constraints. This can avoid all of rule conflicts. The packet processing using the tree rule operates like searching data in a tree data structure. The searching algorithm in the tree data structure is faster than a sequential searching in an array. The GUI editor also helps firewall administrators to create a firewall rule easily without worrying about rule positions.

In Chapter 4, the Tree-Rule firewall has been implemented on CentOS Linux. There are three components which are GUI, rule sender, and core firewall. The Tree-Rule firewall operates on the top of NetFilter, at the same level of IPTABLES.

The Tree-Rule firewall has been tested and compared with IPTABLES on LANs. Experimental results have shown that the Tree-Rule firewall can give better performance. Moreover, the Tree-Rule firewall has been tested on a cloud environment. The results have shown that the Tree-Rule firewall is more suitable than the Listed-Rule firewall on the cloud network, which is a large network that requires a number of computers and a large rule size.

Chapter 4 has also made a capability comparison among the proposed Tree-Rule firewall, the IPTABLES and two state-of-the-art firewalls under a cloud environment. The advantages of using the Tree-Rule firewall have been demonstrated.

Chapter 5 has proposed a ‘stateful mechanism’, which can be used on the Tree-Rule firewall. The design and the development of a model, which is used for the stateful mechanism, are based on the basic model of Connection Tracking module used by Netfilter/IPTABLES (the most popular firewall).

The proposed model requires a low memory space, while it can handle more concurrent connections by using one node per connection. The proposed scheme extends the hashing table vertically and uses one-node bucket length. Moreover, the proposed scheme requires lower time consumption in comparison to Netfilter’s scheme. The proposed scheme got benefits from the avoidance of a hashing computation by considering the Tree rule first. If the packets do not match to the Tree rules (for both directions), the firewall will not perform the hashing calculation. The proposed scheme gets rid of the timer objects used for closing connections. Instead, it uses the Label which can tell the firewall which connections are expired by reading only 16 bytes of time information from the memory. Moreover, the proposed scheme does not need to create new nodes to store packets' information because it uses static memory for all nodes created after the firewall is loaded into the memory and executed. With this scheme, the firewall can check only whether the relevant node is free or not by using the marked Label.

An implementation for the proposed model has been conducted on Linux Cent OS 6.3. Experiments have been conducted on real network environments to evaluate a firewall speed. Experimental results have shown that the stateful Tree-Rule firewall implemented using the proposed scheme can operate faster than Netfilter/IPTABLES. The novel stateful Tree-Rule firewall also requires less memory spaces. This is because the size of a node has been reduced, and the number of nodes used per connection, has been decreased from two nodes to one node.

Chapter 6 has proposed a hybrid Tree-Rule firewall which reduces processing time in verifying packets. The proposed firewall applies the concepts of Tree-Rule firewall in designing conflict-free rules and the concepts of traditional firewall in decision making. Verifying incoming network packets against conflict-free listed rules offers a more secure and faster processing firewall. Counters have been introduced to analyse which rules match with most packets. The rules are sorted according to the counters periodically, and the most frequently matched rules are moved to the top positions. As such, time spent in rule matching can be further reduced because a match can most possibly be found in the first few rules.

Chapter 6 has also proposed a mathematical model to illustrate a relation between the 'time use' for data transfer and other relevant factors, especially the 'time interval'. Moreover, it has proposed an equation for calculating an optimal 'time interval' with a mathematical proof based on Calculus.

Experiments have been conducted using an implemented testbed for evaluating the performance of the proposed hybrid firewall on a big size of data transfer. The experimental results have shown that the proposed scheme can reduce the firewall processing time significantly.

7.2 Future Work

In the next step, the Tree-Rule firewall will be extended to be applicable with NAT (Network Address Translation), IPv6, and VPN (Virtual Private Network), and be able to communicate directly with a hypervisor (as a component of the hypervisor). Furthermore, the Tree-Rule firewall will be improved to be become more flexible than the existing firewalls and the firewalls that we have proposed in this thesis.

To operate with a text environment in Linux, command line features should be added to the Tree-Rule firewall. This can allow firewall administrators to add new rules to the Tree-Rule firewall using batch files or shell scripts. Furthermore, this can provide more flexibility for the network security management using automatic programs.

Additionally, the Tree-Rule firewall may be applied to be one type of SDN firewalls. This may help to improve security and functional speed of SDN networks.

Bibliography

[1] C. Stoll, Stalking the wily hacker, Communications of the ACM, Vol. 31, No. 5, 1988, pp. 484–497.

[2] B. Cheswick, The design of a secure Internet gateway, USENIX 1990 Summer Conference, USENIX Association, Berkeley, CA, 1990. URL: http://www.cheswick.com/ches/papers/gateway.ps, Accessed 2002 Feb 20.

[3] W. R. Cheswick, S.M. Bellovin, A.D. Rubin, Firewalls and Internet Security: repelling the wily hacker, 2003.

[4] K. Ingham, S. Forrest, A History and Survey of Network Firewalls, Technical Report 2002-37, University of New Mexico Computer Science Department, 2002. Available at: http://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf

[5] H. Orman, The Morris worm: a fifteen-year perspective, IEEE Security & Privacy, Vol. 1, No. 5, 2003, pp. 35-43.

[6] R. Conway, Code Hacking: A Developer's Guide to Network Security, Hingham, Massachusetts, Charles River Media, 2004, ISBN 1-58450-314-9.

[7] Check Point, 2007, 'Administration Guide - Check Point', http://updates.checkpoint.com/ID/CheckPoint_UTM-1_AdminGuide.pdf.

[8] Juniper, 2007, 'NetScreen-Security Manager: Configuring Firewall/VPN Devices Guide', http://www.juniper.net/techpubs/software/management/security-manager/nsm2007_1/nsm2007_1_device_config.pdf.

[9] J. Johansson, S. Riley, Protect Your Windows Network: From Perimeter to Data (Microsoft Technology), Addison-Wesley Professional, 2005.

[10] What are the risks associated with relying on IPSec IP Filtering?, 2014, http://security.stackexchange.com/questions/3909/what-are-the-risks-associated-with-relying-on-ipsec-ip-filtering.

[11] P. Ayuso, Netfilter's Connection Tracking System LOGIN, the USENIX magazine, 32 (2006) 34-39.

[12] E. Al-Shaer, H. Hamed, Firewall policy advisor for anomaly detection and rule editing, in: Proceedings of the IEEE/IFIP Integrated Management, IM, 2003, pp. 17–30.

[13] A. Liu, M. Gouda, Diverse firewall design, IEEE Transaction on Parallel and Distributed Systems, Vol. 19, No. 9, 2008, pp. 1237–1251.

[14] A. Liu, M. Gouda, Complete Redundancy Detection in Firewalls, Proceedings of 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, 2005, Storrs, CT, USA.

[15] S. Thanasegaran, Y. Tateiwa, Y. Katayama, N. Takahashi, A Mapping Mechanism for Periodic Filters in a Conflict Detection System for Time-Based Firewall Policies, International Journal of Computer Science and Network Security, 2012, Vol.12, No.4.

[16] M. Ali, E. Al-Shaer, H. Khan, S. Khayam, Automated Anomaly Detector Adaptation using Adaptive Threshold Tuning, ACM Transactions on Information and System Security (TISSEC), Vol. 15. No. 4, 2013.

[17] F. Chen, A. Liu, J. Hwang, T. Xie, First step towards automatic correction of firewall policy faults, ACM Transactions on Autonomous and Adaptive Systems (TAAS), Vol. 7, No. 2, 2012, pp. 1-15.

[18] F. Mansmann, T. Gobel, W. Cheswick, Visual analysis of complex firewall configurations, Proceedings of the Ninth International Symposium on Visualization for Cyber Security (VizSec 2012), Seattle, USA, 2012, pp. 1-8.

[19] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, S. Martinez, J. Cabot, Management of stateful firewall misconfiguration. Computers & Security, Elsevier, 39 (2013) 64-85.

[20] H. Sylvain, N. Lunaud, V. Roger, Distributed firewall anomaly detection through LTL model checking, IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, 2013, pp. 194 - 201.

[21] L. Zhao, A. Shimae, H. Nagamochi, Linear-tree rule structure for firewall optimization, In: Proceedings of Communications Internet and Information Technology, 2007, pp. 67-72.

[22] Bob Jenkins, A Hash Function for Hash Table Lookup, URL: http://burtleburtle.net/bob/hash/doobs.html.

[23] E. Al-Shaer, H. Hamed, R. Boutaba, M. Hasan, Conflict classification and analysis of distributed firewall policies, IEEE Journal on Selected Areas in Communications 23 (10) (2005) 2069-2084.

[24] H. Haded, E. Al-Shaer, Taxonomy of conflicts in network security policies, IEEE Communications Magazine, Vol. 44, No. 3, 2006, pp. 134–141.

[25] S. Hazelhusrt, Algorithms for Analyzing Firewall and Router Access Lists, Technical Report TR-WitsCS-1999, Department of Computer Science, University of the Witwatersrand, 1999.

[26] P. Eronen, J. Zitting, An Expert System for Analyzing Firewall Rules, In: Proceedings of the 6th Nordic Workshop on Secure IT-Systems (NordSec), 2001, pp. 100-107.

[27] C. Pornavalai, T. Chomsiri, Firewall Policy Analyzing by Relational Algebra. In: Proceeding of the 2004 International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), 2004, pp. 214–219.

[28] T. Chomsiri, X. He, P. Nanda, Limitation of listed-rule firewall and the design of tree-rule firewall, in: Proceedings of the 5th International Conference on Internet and Distributed Computing Systems, China, 2012, pp. 275–287.

[29] A. Shebanow, R. Perez, C. Howard, The Effect of Firewall Testing Types on Cloud Security Policies, International Journal of Strategic Information Technology and Applications, Vol. 3, No. 3, 2012, pp. 60-68.

[30] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, M. Rajarajan, A survey of intrusion detection techniques in Cloud, Journal of Network and Computer Applications, Vol. 36, No. 1, 2013, pp. 42–57.

[31] L. Yuan, J. Mai, Z. Su, FIREMAN: A toolkit for Firewall modeling and analysis, In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, 2006, pp. 199-213.

[32] Cisco Content Services Switch Basic Configuration Guide, 2012, http://www.cisco.com/en/US/docs/app_ntwk_services/data_center_app_services/css11500series/v7.10/configuration/basic/guide/basicgd.pdf.

[33] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Generation Computer Systems, Vol. 28, No. 3, 2012, pp. 583–592.

[34] S. Kima, S. Kimb, G. Leea, Structure design and test of enterprise security management system with advanced internal security, Future Generation Computer Systems, Vol. 25, No. 3. 2009, pp. 358–363.

[35] Virtual firewall appliances: trust misplaced, 2012. http://blog.cloudpassage.com/2012/01/24/virtual-firewall-appliances-trust-misplaced/.

[36] VMware virtual switch isolation, 2013. http://serverfault.com/questions/186735/VMware-virtual-switch-isolation.

[37] R.P. Goldberg, Architectural Principles for Virtual Computer Systems, Harvard University, 2010, pp. 22–26.

[38] X. He, T. Chomsiri, P. Nanda, Z. Tan, Improving cloud network security using the Tree-Rule firewall, Future Generation Computer Systems, Elsevier, Vol. 30, 2014, pp. 116-126.

[39] Binary search tree, 2013, http://en.wikipedia.org/wiki/Binary_search_tree.

[40] Linear search, 2013, http://en.wikipedia.org/wiki/Linear_search.

[41] Top 10 hypervisors: choosing the best hypervisor technology, 2011. http://searchservervirtualization.techtarget.com/tip/Top-10-hypervisors-Choosing-the-best-hypervisor-technology.

[42] Virtualization wars: VMware vs. Hyper-V vs. XenServer vs. KVM, 2011. http://www.networkworld.com/news/2011/103111-tech-argumentvirtualization-252559.html.

[43] M.A. Bamiah, S.N. Brohi, S. Chuprat, Using virtual machine monitors to overcome the challenges of monitoring and managing virtualized cloud infrastructures, in: Proceedings of the fourth International Conference on Machine Vision, ICMV, 2011.

[44] I. Voras, M. Orli, B. Mihaljevi, An early comparison of commercial and open source cloud platforms for scientific environments, in: Proceedings of 6th KES International Conference, KES-AMSTA, 2012, pp. 164–173.

[45] H. Mousannif, I. Khalil, G. Kotsis, Collaborative learning in the clouds, Information Systems Frontiers 15 (2) (2013) 159–165.

[46] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, A. Brinkmann, Non-intrusive virtualization management using libvirt, in: Proceedings of the Conference on Design, Automation and Test in Europe, DATE, 2010, 574–579.

[47] Initial findings with VMware ESXi and Hyper-V comparison, 2013. http://redmutt.com/?p=228.

[48] Price: VMware (ESXi) vs. Hyper-V (Windows server 2012), 2012. http://www.insideris.com/price-vmware-esxi-vs-hyper-v-windows-server-2012/.

[49] 5nine virtual Firewall 2.1 for Microsoft Hyper-V, 2011. http://www.5nine.com/Docs/vFirewall2_data_sheet.pdf.

[50] 5nine virtual Firewall 2.0 (v-Firewall, Beta), 2009. http://www.5nine.com/Docs/VFW2_QSG_07.pdf.

[51] S. Paul, R. Jain, M. Samaka, J. Pan, Application delivery in multi-cloud environments using software defined networking, Computer Networks, Elsevier, 2014, Vol. 68, pp. 166-186.

[52] A. Ashfaq, S. Rizvi, M. Javed, S. Khayam, M. Q. Ali, E. Al-Shaer, Information theoretic feature space slicing for statistical anomaly detection, Journal of Network and Computer Applications, Elsevier, 2014, Vol. 41 , pp. 473–487.

[53] F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, T. Moataz, X. Rimasson, Handling stateful firewall anomalies, Information Security and Privacy Research, Springer Berlin Heidelberg, 2012, pp. 174-186.

[54] S. Kumar, R. Gade, Experimental Evaluation of Juniper Network's Netscreen-5GT Security Device against Layer4 Flood Attacks, Journal of Information Security, Springer Berlin Heidelberg, Vol. 2, No. 1, 2011, pp. 50-50.

[55] NetScreen-Security Manager: Configuring Firewall/VPN Devices Guide, 2007, http://www.juniper.net/techpubs/software/management/security-manager/nsm2007_1/nsm2007_1_device_config.pdf.

[56] R. Rosen, Netfilter, Linux Kernel Networking, Apress, 2014, pp. 247-278.

[57] The netfilter.org project, 2015, http://www.netfilter.org/.

[58] Q. M. AL-Musawi , MITIGATING DoS/DDoS ATTACKS USING IPTABLES, International Journal Of Engineering & Technology, IJENS Publishers, 2012, Vol. 3, pp. 12-12.

[59] Q. X. Wu, The Research and Application of Firewall based on Netfilter. Physics Procedia, Elsevier, 2012, Vol. 25, pp. 1231-1235.

[60] T. Chomsiri, X. He, P. Nanda, Z. Tan, "A Stateful Mechanism for the Tree-Rule Firewall", 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom.2014), 2014, pp. 122-129.

[61] Netfilter/IPTABLES FAQ: Problems at runtime, 2014, http://www.netfilter.org/documentation/FAQ/netfilter-faq-3.html

[62] A. Anand, B. Patel, An Overview on Intrusion Detection System and Types of Attacks It Can Detect Considering Different Protocols, International Journal of Advanced Research in Computer Science and Software Engineering, IJARCSSE, 2012, No. 8, pp. 94-98.

[63] Malicious Packets - Packet-Craft.net, 2014, http://www.packet-craft.net/Malicious/.

[64] A. Srivastava, B. B. Gupta, A. Tyagi, A. Sharma, A. Mishra, A recent survey on DDoS attacks and defense mechanisms, In Advances in Parallel Distributed Computing, Springer, 2011, pp. 570-580.

[65] P. Kiddie, Creating a simple 'hello world' Netfilter model, 2009, http://www.paulkiddie.com/2009/10/creating-a-simple-hello-world-netfilter-model.

[66] Fong, Jeffrey, Xiang Wang, Yaxuan Qi, Jun Li, and Weirong Jiang. "ParaSplit: a scalable architecture on FPGA for terabit packet classification." In High-Performance Interconnects (HOTI), 2012 IEEE 20th Annual Symposium on, pp. 1-8. IEEE, 2012.

[67] Erdem, Oguzhan, and AydinCarus. "Multi-pipelined and memory-efficient packet classification engines on FPGAs." Computer Communications (2015).

[68] Hager, Sven, Frank Winkler, Bjorn Scheuermann, and Klaus Reinhardt. "MPFC: Massively Parallel Firewall Circuits." In Local Computer Networks (LCN), 2014 IEEE 39th Conference on, pp. 305-313. IEEE, 2014.

[69] Ni, Cuixia, Guang Jin, and Xianliang Jiang. "A New Multi-tree and Dual Index based Firewall Optimization Algorithm." TELKOMNIKA Indonesian Journal of Electrical Engineering 11, no. 5 (2013): 2387-2393.

[70] Fengjun S, Yingjun P, Xuezeng P, Bin B. Research on a Stochastic Distribution MultibitTrie Tree IP Classification Algorithm. Journal of Communications (in Chinese). 2008; 29(7): 109-117.

[71] Trabelsi, Zouheir, Mohammad M. Masud, and KilaniGhoudi. "Statistical dynamic splay tree filters towards multilevel firewall packet filtering enhancement." Computers & Security 53 (2015): 109-131.

[72] Hung, Nguyen Manh, and Vu Duy Nhat. "B-tree based two-dimensional early packet rejection technique against DoS traffic targeting firewall default security rule." In Computational Intelligence for Security and Defense Applications (CISDA), 2014 Seventh IEEE Symposium on, pp. 1-6. IEEE, 2014.

[73] M. Kang, J. Choi, H. Kwak, I. Kang, M. Shin, J. Yi, Formal modeling and verification for SDN firewall application using pACSR, In Electronics, Communications and Networks IV: Proceedings of the 4th International Conference on Electronics, Communications and Networks (CECNET IV), Beijing, China, 2014, pp. 155.

[74] S. Kumar, R. Perumalraja, Establishing User Defined Firewall in Software Defined Network, International Journal of Research 2(6)(2015), pp. 28-31.

[75] M. Suh, S. Park, B. Lee, S. Yang, Building firewall over the software defined network controller, In Advanced Communication Technology (ICACT), 2014 16th International Conference, 2014, pp. 744-748.

[76] H. Hu, G. Ahn, W. Han, Z. Zhao, Towards a reliable SDN firewall, Presented as part of the Open Networking Summit (ONS 2014), 2014.

[77] T. Chomsiri, X. He, P. Nanda, Z. Tan, "Hybrid Tree-rule Firewall for High Speed Data Transmission", IEEE Transactions on Cloud Computing, 2016, no. 1, pp. 1, PrePrints, doi:10.1109/TCC.2016.2554548.

[78] "1000 redundant rules of IPTABLES (TCCSI-2015-01-0032.R1)", https://www.youtube.com/results?search_query=TCCSI-2015-01-0032.R1, 2016

[79] "Java Applets Centre - University of Canterbury", http://www.cosc.canterbury.ac.nz/mukundan/dsal/QSort.html, 2015

[80] Fidel, Vidal, and José María. "Mecanismopara el acceso-público a servidores con direccionamientoprivado." (2011).