Hybrid Tree-Rule Firewall for High Speed Data Transmission

Thawatchai Chomsiri, Xiangjian He, *Senior Member, IEEE*, Priyadarsi Nanda, and Zhiyuan Tan

Abstract—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 incurs extra computational overhead. In order to reduce this overhead, we propose a hybrid Tree-rule firewall in this paper. This hybrid scheme takes advantages of both Tree-rule firewalls and traditional listed-rule firewalls. The GUIs of our Tree-rule firewalls are utilized to provide a means for users to create conflict-free firewall rules, which are organized 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.

Index Terms—Firewall, high speed firewall, network security, computer network, cloud network

1 Introduction

Firewalls were first invented in 1990s [1], and have been developed to operate more securely and faster. Since the first generation firewalls, the commercially used firewalls still perform network traffic regulation based on listed rules. 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' [2] 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 [3].

To address the afore-mentioned problems, we recently proposed a new type of firewall called 'Tree-rule firewall' in [4]. 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 [4]. In our recent follow-up study [5], 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) [6] 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 $1{,}400/(1.4 * 4) = 250$ 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.

- *T. Chomsiri, X. He, and P. Nanda are with the University of Technology Sydney, PO Box 123, Broadway 2007, Sydney, Australia. E-mail: ThawatchaiChomsiri@student.uts.edu.au, {Xiangjian.He, Priyadarsi.Nanda}@uts.edu.au.*
- *Z. Tan is with the University of Twente, PO Box 217 7500AE, Enschede 7522 NB, the Netherlands. E-mail: z.tan@utwente.nl.*

Manuscript received 17 Jan. 2015; revised 22 Feb. 2016; accepted 3 Apr. 2016. Date of publication 0 . 0000; date of current version 0 . 0000. Recommended for acceptance by M. Shamim Hossain, C. Xu, M. Murshed, J.H. Abawajy, and A. El Saddik. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCC.2016.2554548

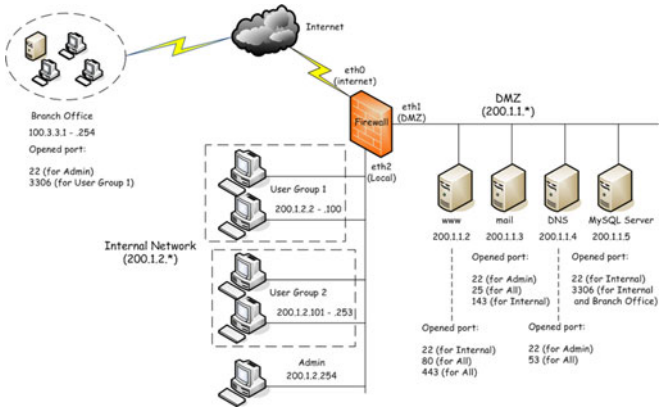

Fig. 1. An example network.

Motivated by the above, the contributions of this paper 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 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 paper is organized as follows. The background and the related work are introduced in Section 2. Our proposed hybrid firewall scheme is then detailed in Section 3. The implementation of our proposed scheme is presented and the experimentation is demonstrated in Section 4. Finally, conclusion is drawn along with the discussion of our future research in Section 5.

2 Background and Related Work

Previous research approaches aiming to enhance functional speed of firewalls can be categorized 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 emphasizes 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 paper.

TABLE 1
A Set of Listed Rules Created for an Example Network in Fig. 1

No.	Source_IP	Dest_IP	Dest_Port	Action
1	100.3.3.*	200.1.1.5	3,306	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.*	3,306	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	3,306	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

2.1 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 1, for example, illustrate how to regulate traffic traversing over the network presented in Fig. 1 in compliance with the network topology

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 [2], [4] and [7], 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 [4]. Redundant rules decrease the processing speed of a firewall [2], [4]. 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 [2]. 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 [8].

The methods proposed in [7] also aim to discover rule conflicts. However, the proposed method in [7] is based on relational algebra techniques. It can discover more rule conflicts in comparison with the method suggested in [2]. The findings highlighted in [2], [7] and [8] suggest potential solutions to remove these problematic rules from a firewall rule set.

In addition, tools such as Binary Decision Diagrams (BDDs) [9], Constraint Logic Programming (CLP) [10] and Fireman Toolkit [11] were proposed to help analyze and remove rule conflicts from the rule set of a listed-rule firewall.

Although these studies [2], [7], [8], [9], [10], [11] 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.

2.2 Enhancing Processing Speed via Hardware Implementation

Fong et al. [12] 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 [12].

Likewise, Erdem and Carus [13] 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. [14] proposed the Massively Parallel Firewall Circuits (MPFC) to generate customized 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 [12], [13], [14] 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.

2.3 Enhancing Processing Speed via Advanced Filtering Mechanisms

Ni et al. [15] 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 [16] 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. [17] proposed an analytical dynamic multi-level early packet filtering mechanism to enhance firewall performance. The proposed mechanism uses statistical splay tree filters that utilize traffic characteristics to minimize 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 [18] 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 minimize 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 [19] 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 [2], [8] can convert firewall rules to a 'policy tree', the 'policy tree' cannot be considered as a tree-based filtering firewall mentioned in this paper. 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 [22], [23], [24] and [25] employ SDN and support centralized 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.

2.4 Background of Tree-Rule Firewall

Chomsiri et al. have further studied firewall rules' problems, and published their interesting findings in [3] and [4]. They proposed a Tree-rule firewall to overcome these problems. The Tree-rule firewall not only organizes firewall rules in a tree structure as shown in Fig. 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

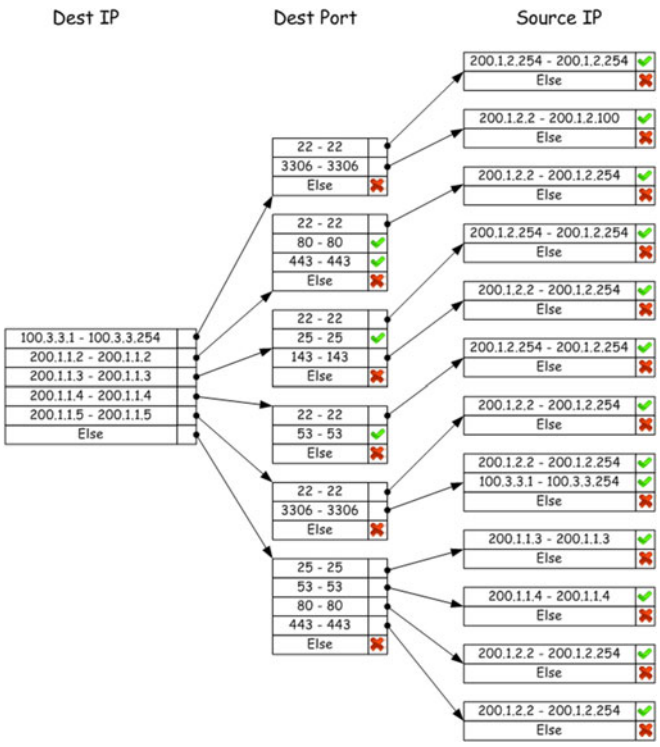

Fig. 2. A Tree rule structure created for an example network in Fig. 1.

fields sequentially against their respective tree nodes at the corre-sponding 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 Fig. 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.

To further improve the processing speed of the Tree-rule firewall [4], we have proposed a stateful mechanism in [5]. However, this mechanism requires hashing calculation [6] at least once per packet. Therefore, the speed of the firewall can be significantly improved if this complex hashing is eliminated. To achieve better speed performance, we propose a new hybrid firewall in this paper. The details of the proposed firewall are presented in Section 3.

3 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 [4] 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 optimize the processing speed of the hybrid firewall.

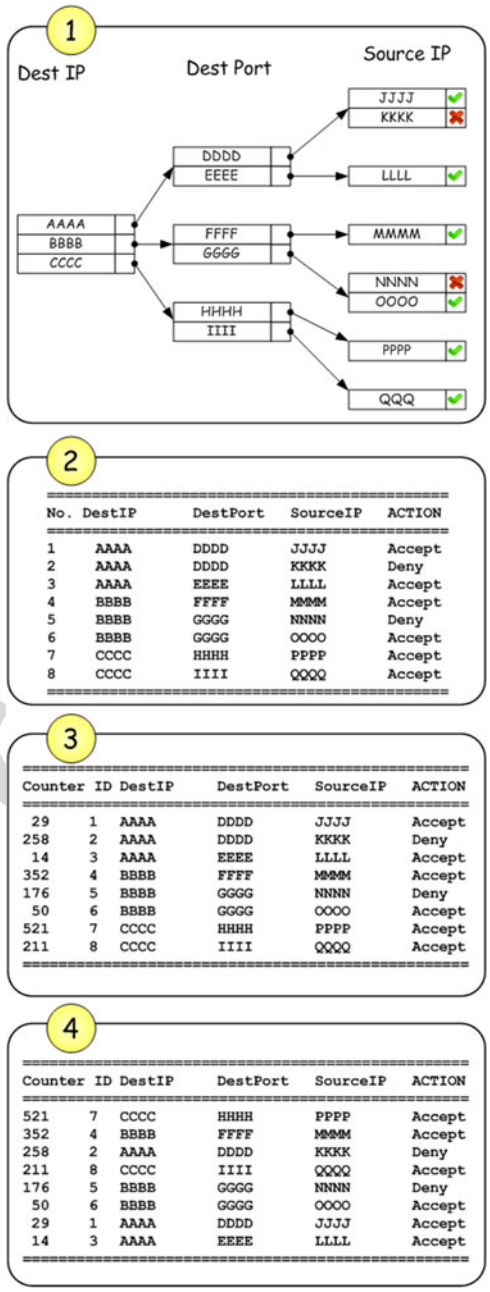

Fig. 3. Four steps of proposed scheme.

3.1 Methodology

As shown in Fig. 3, there are four steps involved in the process of our hybrid approach. In the first step shown in Fig. 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 Fig. 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 Fig. 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 Fig. 3-(4). The counters of all the rules will be reset to 0 when a pre-determined 'Time

TABLE 2
Example of a Listed Rule Transformed from a Rule Path

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

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 Fig. 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

$$([100.3.3.1\text{-}100.3.3.254] --> [22\text{-}22] --> [200.1.2.254\ 200.1.2.254] --> \text{Accept})$$

can be transformed into the listed rule shown in Table 2. The second rule path

$$([100.3.3.1\text{-}100.3.3.254] --> [22\text{-}22] --> [\text{Else}] --> \text{Deny})$$

can be transformed into the listed rule shown in Table 3.

Bearing the same idea in mind, the tree rules shown in Fig. 2 can be transformed to the listed rules shown in Table 4.

After designing and transforming tree rules into listed rules using the GUI, the listed rules shown in Table 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.

3.2 Discussion on Efficiency

Although various methods [2], [7], [8], [10], [11], [19] have been designed to minimize rule conflicts through rearrangement 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 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 re-positioned to the top of

TABLE 3
Example of Two listed Rules Transformed from a Rule Path

RuleNo.	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

TABLE 4
The Listed Rules Transformed from the Tree Rules in Fig. 2

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.-200.1.13	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

TABLE 4
(*Continued*)

RuleNo.	Dest-IP	Dest-Port	Source-IP	ACTION
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	0.0.0.0-200.1.1.3	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

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 4 can be moved to any position independently.

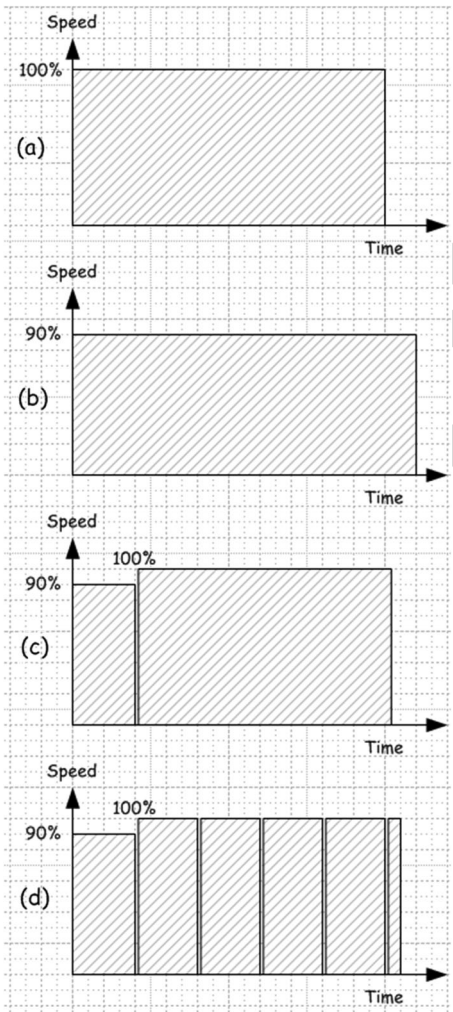

Fig. 4. Transmission speed versus transmission time.

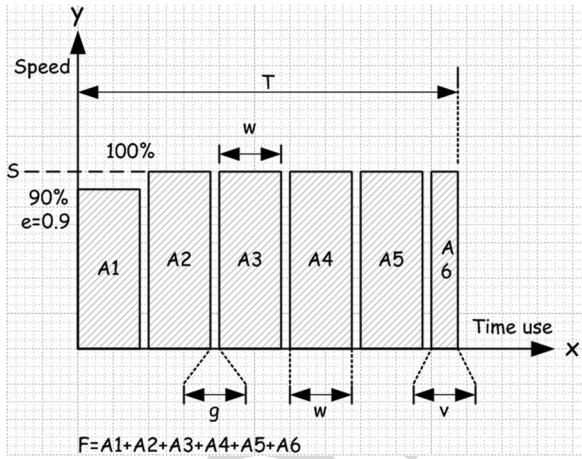

Fig. 5. Time (*T*) used for data transmission and the five main factors (*w*, *F*, *S*, *e* and *g*).

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 the firewalls' time to process unmatched rules before reaching the matched one and allowing/denying packets to pass through. According to our studies, 1,000 redundant rules can reduce data transmission speed by approximately 10 percent. The drop of speed depends on several factors, i.e., type of firewall [20] 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 percent if data transmission speed drops by 10 percent as shown in Fig. 4a and 4b respectively). Moving the matched rule from the bottom of firewall rule list to the top position (e.g., from rule number 1,000 to rule number 1 enhances the data transmission speed and shortens transmission time as illustrated in Fig. 4c. Using our proposed scheme, rule sorting is executed periodically for each specified time interval, such as 1, 3 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 Fig. 4d. The time consumption shown in Fig. 4d is more than that revealed in Fig. 4c but less than that revealed in Fig. 4b.

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*).

Fig. 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 1,000, which decreases transmission speed by roughly 10 percent of the maximum speed. In the first state, transmission speed begins with 90 percent ($e = 0.9$) 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 percent 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 $F = A1 + A2 + A3 + A4 + A5 + A6$.

The efficiency, e, is determined by the number of rules. We have created a special program to measure e with 1,000 rules on a 2.8 GHz CPU computer and 345 Mbps network speed. We found that e was 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 [26] is used for rule sorting in this paper. Thus, g increases slightly while the number of rules increases. We measured g in the same environment where e was done. We found that the value of g was approximately 1 millisecond for 1,000 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 Section 4.

3.3 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 Fig. 5),
- F denote size of data being transmitted (in bits),
- e denote efficiency of transmission speed before sorting the rules, $0 < e < 1$,
- S denote speed of network (in bits per seconds),
- w denote time interval between two rule sortings (in seconds),
- g denote 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 Fig. 5,
- u denote v/w, $0 < u < 1$, and
- T denote the time used for data transmissions.

Then, we have

$$\begin{aligned} F &= eSw + nSw + Sv \\ F - eSw &= nSw + Sv \\ (F - eSw)/S &= nw + v = nw + uw \\ (F - eSw)/Sw &= n + u \end{aligned}$$

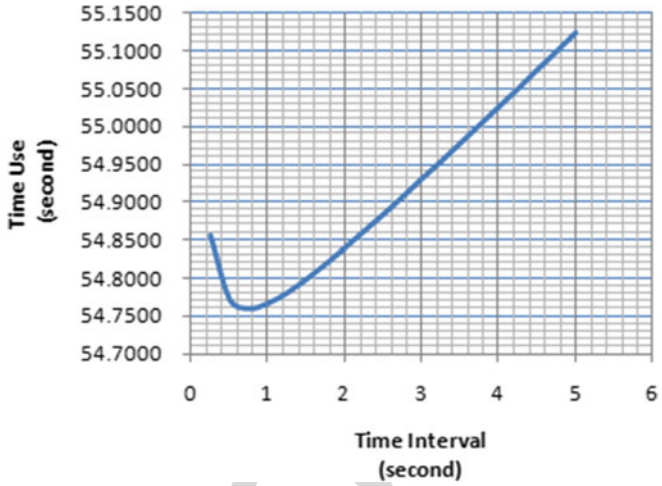

Fig. 6. Relation between Time use (T) and Time Interval (w).

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

The time T used for data transmissions shown in Fig. 5 is defined as,

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

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

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

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 Fig. 6. We specified F = 2,048 MB (16,384 Mbits), S = 300 Mbps, g = 0.001 seconds and $e = 0.9$. We calculated 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. Fig. 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.

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

$$T = \frac{F}{eS}. \quad (4)$$

Thus, in this example, without using our proposed scheme, the firewall will take time: $T = 16{,}384/(0.9 * 300) = 60.6815$ seconds. In contrast, using our proposed scheme, the transferring time can be saved for 9.76 percent for $w = 0.75$, and 9.75, 9.48 and 9.16 percent for $w = 1$, 3 and 5 seconds, respectively.

3.4 Determining Time Interval *w*

To determine the time interval w, we created a special program to measure a time used for sorting 1,000 rules. We found that the sorting took 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 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 in cloud which 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.

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

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

$$\frac{dT}{dw} = 0. \quad (6)$$

From Equation (3) in Section 3, 'T' can be calculated by:

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

Therefore, Equation (6) is equivalent to

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

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

In Fig. 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 we have discussed in Section 3.2. With the same environments and parameters (e.g., the same value of F, S, g and e), we have calculated w using Equation (5), and found that the optimal w, which 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)

4 Implementation and Experimentation

Similar to our previous schemes [4], [5], we implement the proposed schemes based on the Netfilter module [27], [28], [29]. We hook packets' events using a technique presented in [30] by calling the function named 'nf_register_hook' [30]. 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 *))
{
}
```

4.1 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 rule editor GUI using C# on Windows. The firewall rule is created by GUI and is sent to the core firewall running on Linux. The rule structure is modified for handling listed rules and counters information.

We evaluate the firewall on one Giga bits per second link speed LAN with seven standard PCs as shown in Fig. 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 a cloud environments. Within the server, we create five Virtual Machines (i.e., guest OSs) to serve as web servers (as shown in Fig. 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 decided to test on only on ESXi for the proposed work in this paper.

In our experimentation, time used for downloading big size of data (e.g., big files) is measured. To do so, we store a

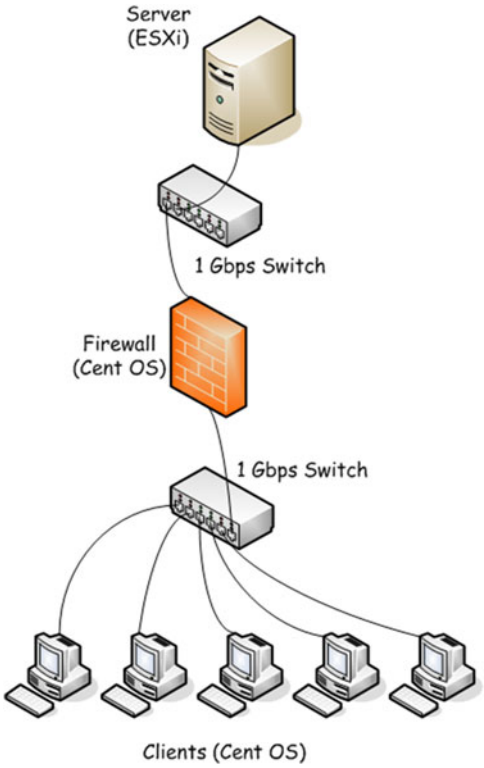

Fig. 7. Experiment with ESXi.

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.

4.2 Experiments

The equation used in Section 3.4 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.

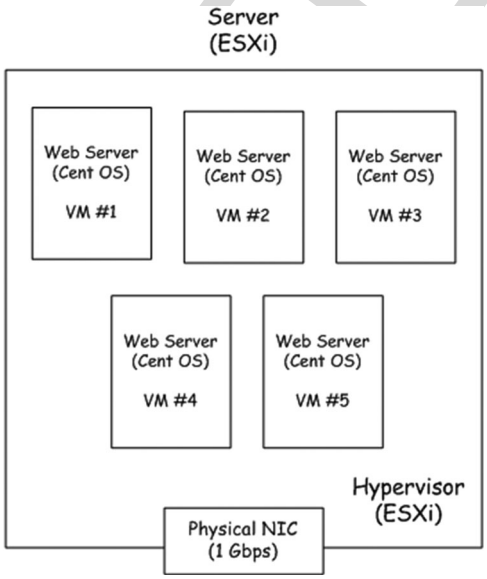

Fig. 8. Five Linux Web Servers within a ESXi Hypervisor.

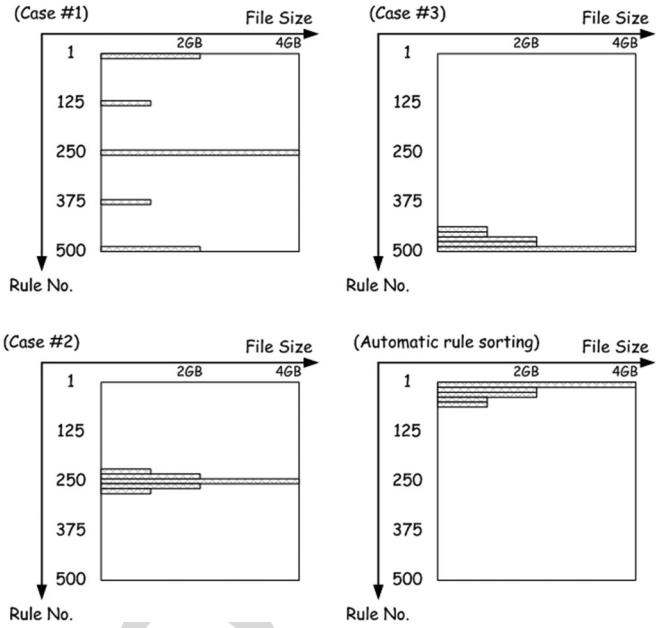

Fig. 9. Three cases of 'non automatic rule sorting' and a case of 'automatic rule sorting'.

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. For example, a computer LAB which is matched with one allowed rule, and open 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 percent 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 which shows percentages 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 three cases with non-automatic rule sorting as shown in Cases #1, #2 and #3 of Fig. 9. We create 500 firewall rules and intentionally make rule #250 match 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 time consumption in 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

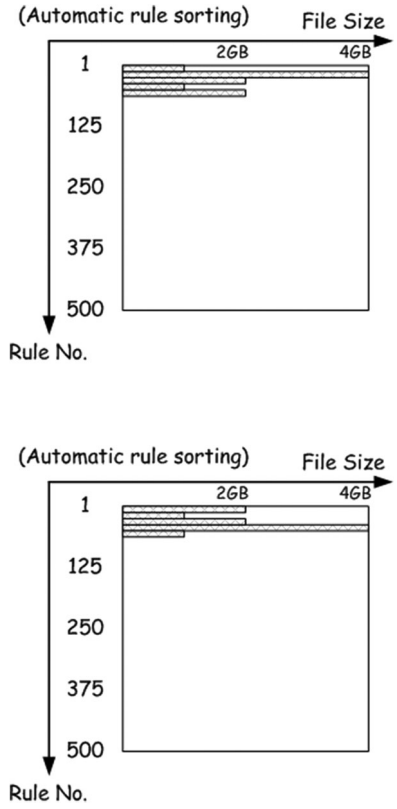

Fig. 10. Sequences of rules in 'automatic rule sorting'.

rules are resorted. Whilst five files are downloaded simultaneously, results of sorting may be different from the right bottom picture of Fig. 9. They may be sorted in many sequences as shown in Fig. 10.

Lastly, we test with 1,000, 2,000 and 4,000 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 experimentation for five times, and the average result numbers are taken and highlighted in Table 5.

Case #1 and Case #2 in Table 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 three cases, the downloading times are longer when the number of rule 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, 1,000, 2,000, and 4,000 rules are slightly different. The percentages of time saving are presented in Table 6. As shown in Table 6, our scheme can reduce the processing time of the firewall with 500 rules by 8.17 percent on average. More time is saved in the cases with bigger rule sizes. For example, the proposed method saves 60.89 percent of the time for the case with 4,000 rules as shown in Table 6.

TABLE 5
Time Consumption for Transferring Files from Servers to Clients (Minutes)

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

TABLE 6
Time Save in Percentage

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 rule having different size. 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 was 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 1,440 bytes packet size. We choose a bigger packet size because HTTP typically uses packet size of 1,400-1,500 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 experimentation but using one matched rule at the bottom of rule list.

We measure speeds of IPTABLES with different rule size, e.g, 100, 250, 500, 1,000, 1,500, 2,000, 2,500, 3,000, 3,500 and 4,000 rules. The 'hping3' command with '—flood' can throttle the firewall to operate with its maximum speed (throughput). With no rule (rule size = 0), IPTABLES can process 30,956 packets per second, as shown in Table 7. In Table 7, the firewall speed was represented in term of packets per second, and mega bytes per second. The data were calculated using 1,440 bytes packet size.

We can see, the speed of IPTABLES drops from 42.51 to 22.25 MB/s (47.66 percent) having 1,000 rules. The percentage of speed drop increases when the firewall processes a bigger rule size.

We also test the proposed firewall with the same condition (as we tested IPTABLES) by disabling the feature 'Automatic rule sorting'. As shown in Table 8, speed of our firewall operating with rule size = 20,000, 30,000, 40,000, 50,000, 60,000, 70,000 and 80,000 indecate that our firewall operates faster than the IPTABLES approximately by 20 times. For rule size = 1,000, speed of our firewall drops only 7.43 percent. In comparison, IPTABLES speed drops

TABLE 7
Speed Achieved Through IPTABLES

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

TABLE 8
Speed of Proposed Firewall without 'Automatic Rule Sorting'

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

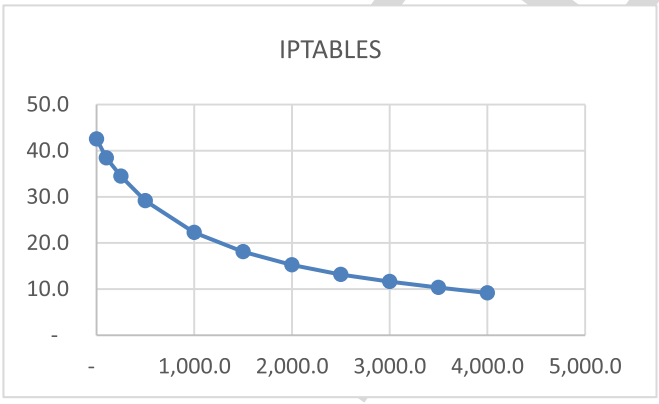

Fig. 11. Speed of IPTABLES (represented in graph).

by 47.66 percent. The two plots as shown through Figs. 11 and 12 translate corresponding data present in Tables 7 and 8. In the two plota, vertical axis of the graph represents speeds of firewall in MByte/sec whereas the horizontal axis represents numbers of rules.

We perform more experiments for the proposed firewall to compare between operations with and without 'Automatic rule sorting'. Experimental results are presented in Table 9 and Fig. 13.

TABLE 9
Speed of Proposed Firewall with 'Automatic Rule Sorting'

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

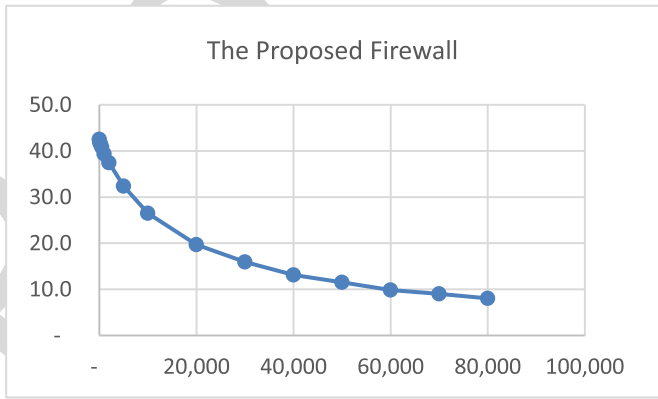

Fig. 12. Speed of proposed firewall without 'automatic rule sorting' (represented in graph).

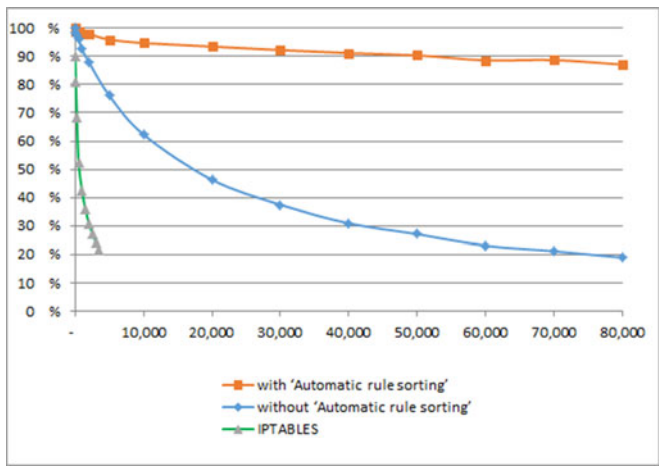

Fig. 13. Comparision of firewalls' speeds.

With rules size = 1,000 in Table 9, the proposed firewall with 'Automatic rule sorting' gives 2.07 percent of speed drop whereas operating without 'Automatic rule sorting' gives 7.43 percent (see Table 8). Fig. 13 shows 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 particulay with large size of rule set.

5 Conclusion and Future Works

In this paper, we have 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 contributes a more secure and faster processing firewall. Counters are introduced to analyze which rules match 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, 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 'time use' for data transferring and other relevant factors, especially '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 a big size of data transferring. The experimental results show that our scheme can reduce firewall processing time significantly. For our future research, we will further improve and test the proposed firewall in other environments.

References

[1] W. Cheswick, S. Bellovin, and A. Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker*. Reading, MA, USA: Addison-Wesley, 2003.

[2] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly detection and rule editing," in *Proc. IEEE/IFIP Integr. Manage.*, 2003, pp. 17–30.

[3] T. Chomsiri, X. He, and P. Nanda, "Limitation of listed-rule firewall and the design of Tree-rule firewall," in *Proc. 5th Int. Conf. Internet Distrib. Comput. Syst.*, 2012, pp. 275–287.

[4] X. He, T. Chomsiri, P. Nanda, and Z. Tan, "Improving cloud network security using the Tree-rule firewall," *Future Gen. Comput. Syst.*, vol. 30, pp. 116–126, 2014.

[5] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "A stateful mechanism for the tree-rule firewall," in *Proc. IEEE 13th Int. Conf. Trust, Security Privacy Comput. Commun.*, 2014, pp. 122–129.

[6] P. Ayuso, *Netfilter's Connection Tracking Syst., LOGIN; The USENIX Mag.*, vol. 32, pp. 34–39, 2006. Q1

[7] C. Pornavalai and T. Chomsiri, "Firewall policy analyzing by relational algebra," in *Proc. Int. Tech. Conf. Circuits/Syst., Comput. Commun.*, 2004, pp. 214–219.

[8] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. Select. Areas Commun.*, vol. 23, no. 10, pp. 2069–2084, Oct. 2005.

[9] S. Hazelhusrt, "Algorithms for analyzing firewall and router access lists," *Dept. Comput. Sci., Univ. Witwatersrand*, Tech. Rep. TR-WitsCS-1999, 1999.

[10] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Proc. 6th Nordic Workshop Secure IT-Syst.*, 2001, pp. 100–107.

[11] L. Yuan, J. Mai, and Z. Su, "FIREMAN: A toolkit for Firewall modeling and analysis," in *Proc. IEEE Symp. Security Privacy*, 2006, pp. 199–213.

[12] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "ParaSplit: A scalable architecture on FPGA for terabit packet classification," in *Proc. IEEE 20th Annu. Symp. High-Perform. Interconnects*, 2012, pp. 1–8.

[13] O. Erdem, and A. Carus. "Multi-pipelined and memory-efficient packet classification engines on FPGAs," *Comput. Commun.*, vol. 67, pp. 75–91, 2015.

[14] S. Hager, F. Winkler, B. Scheuermann, and K. Reinhardt, "MPFC: Massively parallel firewall circuits," in *Proc. IEEE 39th Conf. Local Comput. Netw.*, 2014, pp. 305–313.

[15] C. Ni, G. Jin, and X. Jiang, "A new multi-tree and dual index based firewall optimization algorithm," *TELKOMNIKA Indonesian J. Elect. Eng.*, vol. 11, no. 5, pp. 2387–2393, 2013.

[16] S. Fengjun, P. Yingjun, P. Xuezeng, and B. Bin, "Research on a stochastic distribution multibittrie tree IP classification algorithm," *J. Commun. (Chin.)*, vol. 29, no. 7, pp. 109–117, 2008.

[17] Z. Trabelsi, M. M. Masud, and K. Ghoudi, "Statistical dynamic splay tree filters towards multilevel firewall packet filtering enhancement," *Comput. Security*, vol. 53, pp. 109–131, 2015.

[18] N. M. Hung and V. D. Nhat, "B-tree based two-dimensional early packet rejection technique against DoS traffic targeting firewall default security rule," in *Proc. 7th IEEE Symp.Comput. Intell. Security Defense Appl.*, 2014, pp. 1–6.

[19] A. Liu and M. Gouda, "Diverse firewall design," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 9, pp. 1237–1251, Sep. 2008.

[20] 1000 redundant rules of IPTABLES (TCCSI-2015–01–0032.R1) (2016) [Online]. Available: https://www.youtube.com/results?search_query = TCCSI-2015–01–0032.R1

[21] L. Zhao, A. Shimae, and H. Nagamochi, "Linear-tree rule structure for firewall optimization," in *Proc. 6th IASTED Int. Conf. Commun. Internet Inf. Technol.*, 2007, pp. 67–72.

[22] M. Kang, J. Choi, H. Kwak, I. Kang, M. Shin, and J. Yi, "Formal modeling and verification for SDN firewall application using pACSR," in *Proc. 4th Int. Conf. Electron. Commun. Netw.*, 2014, p. 155.

[23] S. Kumar and R. Perumalraja, "Establishing user-defined firewall in software defined network," *Int. J. Res.*, vol. 2, no. 6, pp. 28–31, 2015.

[24] M. Suh, S. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, 2014, pp. 744–748.

[25] H. Hu, G. Ahn, W. Han, and Z. Zhao, "Towards a reliable sdn firewall," in *Proc. Open Netw. Summit*, 2014. Q2

[26] Java applets centre—University of Canterbury (2015) [Online]. Available: http://www.cosc.canterbury.ac.nz/mukundan/dsal/QSort.html

[27] R. Rosen, *Netfilter, Linux Kernel Netw.*, Apress, 2014, pp. 247–278.

[28] The netfilter.org project (2014) [Online]. Available: http://www.netfilter.org/

[29] P. Ayuso, *Netfilter's Connection Tracking Syst., LOGIN; The USENIX Mag.*, vol, 32, pp. 34–39, 2006.

[30] V. Fidel and J. María, "Mecanismopara el accesopúblico a servidores con direccionamientoprivado," 2011.

Thawatchai Chomsiri is working toward the PhD degree at the Faculty of Engineering and Information Technology, University of Technology, Sydney, Australia. He is also an assistant professor at the Department of Information Technology, Faculty of Informatics, Mahasarakham University, Thailand. He has 20 years of experience in industry, teaching and research. His research interests include computer networking, and computer and network security.

Xiangjian He is a professor of computer science. He is also the Director of Computer Vision and Recognition Laboratory and a co-leader of the Network Security Research group, University of Technology, Sydney. He has been awarded Internationally Registered Technology Specialist by International Technology Institute (ITI). His research interests include network security, image processing, pattern recognition and computer vision. He is a senior member of the IEEE.

Priyadarsi Nanda is a senior lecturer in the School of Computing and Communications, and is a core research member at the Centre for Real-time Information Networks, University of Technology, Sydney. His research interests include network QoS, network securities, assisted health care using sensor networks, and wireless networks. He has over 25 years of experience in teaching and research.

Zhiyuan Tan is a postdoctoral researcher in Services, Cyber security and Safety Research Group, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, the Netherlands. His research interests include network security, pattern recognition, machine learning and distributed computing.

Queries to the Author

Q1. Please provide full bibliography for Ref. [6], [27], [29], [30].
Q2. Please provide page range for Ref. [25].