Towards a New Design of Firewall: Anomaly Elimination and Fast Verifying of Firewall Rules

Suchart Khummanee, Atipong Khumseela, Somnuk Puangpronpitag

Faculty of Informatics, Mahasarkham University
Maha Sarakham Province, Thailand
[1] suchart.k@msu.ac.th,
[2] atipong.khumseela@gmail.com
[3] *somnuk.p@msu.ac.th*

***Abstract*— Network security is usually protected by a firewall, which checks in-out packets against a set of defined policies or rules. Hence, the overall performance of the firewall generally depends on its rule management. For example, the performance can be decreased when there are firewall rule anomalies. The anomalies may happen when two sets of firewall rules are overlapped or their decision parts are both an acceptance and a denial simultaneously. In this paper, we propose a new paradigm of the firewall design, consisting of two parts: (1) Single Domain Decision firewall (SDD) -- a new firewall rule management policy that is certainly not conflicts, and (2) the Binary Tree Firewall (BTF) -- a data structure and an algorithm to fast check the firewall rules. Experimental results have indicated that the new design can fix conflicting anomaly and increase the speed of firewall rule checking from $O(N^2)$ to $O(\log_2 N)$.**

***Keywords*— Firewall rule optimization, Anomaly, Single Domain Decision firewall (SDD), Binary Tree Firewall rule (BTF)**

I. Introduction

Firewall is an important tool for protecting the security of the network from attacking between trusted and untrusted networking. It detects and filters the packets that pass through itself. However, the security levels depend on verification measures which usually follow by organization policies or rules. If some organizations want a strict usability to access information on the untrusted network, the firewall rules are therefore complicated by the policies. Basically, the policy usually consists of a group of six conditions, and is divided into two parts, namely predicate and decision as follows:

$$\langle predicate\rangle \rightarrow \langle decision\rangle \text{ --------- (1)}$$

The $\langle predicate\rangle$ of a rule denotes a Boolean expression over some packet fields, such as source IP address, destination IP address, source port number, destination port number, and protocol type. The $\langle decision\rangle$ of a rule can be *accept*, *deny*, or *discard*. While packets arrive at the firewall, they are firstly verified with predicate part, if some packets match all the fields in the predicate, the packets are then accepted or denied depending on the decision part. However, the rules in the firewall are often conflict, unnecessarily complex, difficult to understand, improper alignment, and incorrect due to administrator's ignorance. These factors result in a poor overall performance of firewall. In particular, allowing more than two rules to overlap can create a serious problem. We will explain in the further in section II. Several studies have tried to survey these problems, for example verifying rules' conflicts [1-3], reducing and collapsing overlap or duplicate of rules [4, 5], analysing the meaning, relationship and vulnerability of rules [6], sorting out rules by user's behaviours [7, 8], improving the friendly user interfaces [9], and so on. Nevertheless, the original cause of the problems is still not fixed yet. We will also extensively describe the cause in section II. This paper aims to eliminate the root cause of rule's conflicts or anomalies, and increases speed of verifying firewall rules. The rest of the paper proceeds as follows: background and related work, and problems are discussed in Section II. Our ideas and designs are explained in Section III. In Section IV, we show the experimental results for eliminate rule's conflicts, and performance. Finally, we give conclusions and future work in Section V.

II. Background and Related Work

A. The firewall

Firewall precisely checks all of the packets that flow through its input-output interfaces by strictly following the predefined rules. Let *Fwr* denotes a firewall rule, hence firewall rules can be represented by Fwr_1, Fwr_2, Fwr_3,…, Fwr_n respectively, and *n* means the set of positive integers ($\mathbf{Z}^+$). We can define the firewall rules in the format:

Fwr_1: $\langle predicate\rangle_1 \rightarrow \langle decision\rangle_1$
Fwr_2: $\langle predicate\rangle_2 \rightarrow \langle decision\rangle_2$
...
Fwr_n: $\langle predicate\rangle_n \rightarrow \langle decision\rangle_n$

Each rule consists of two parts are: $\langle predicate\rangle$, and $\langle decision\rangle$. In the $\langle predicate\rangle$ consists of five fields are: (1) Source IP Address (*SA*) – 32 bits, (2) Destination IP Address (*DA*) – 32 bits, (3) Source Port (*SP*) – 16 bits, (4) Destination Port (*DP*) – 16 bits, (5) Protocol (*PRO*) – 1 bit (0 = TCP and 1 = UDP), and (6) decision or action (*ACT*) – 1 bit (0 = *deny* or *discard*, and 1 = *accept* or *allow*). We replaced the fields in (1) as follows:

$$\langle SA, DA, SP, DP, PRO\rangle \rightarrow \langle ACT\rangle \text{ --------- (2)}$$

Let $SA=\{x|x\in[0, 2^{32}-1]\}$, $DA=\{x|x\in[0, 2^{32}-1]\}$, $SP=\{x|x\in[0, 2^{16}-1]\}$, $DP=\{x|x\in[0, 2^{16}-1]\}$, $PRO=\{x|x\in\{0, 1\}$ by $0=TCP$ and $1=UDP\}$, and $ACT=\{x|x\in\{0, 1\}$ by $0=deny$ and $1=accept\}$, and $SA, DA, SP, DP, PRO \notin \phi$

Firewall checks the packets with five fields in ⟨*predicate*⟩, if they are true then follows in field ⟨*ACT*⟩ as:

$$\langle (P_{(SA)} \in SA) \wedge (P_{(DA)} \in DA) \wedge (P_{(SP)} \in SP) \wedge (P_{(DP)} \in DP) \wedge (P_{(PRO)} \in PRO) \rangle \rightarrow \langle ACT \rangle \text{ --------- } (3)$$

Given P is an input packet, $P_{(SA)}$ is a set of source IP addresses of the packet, $P_{(DA)}$ is a set of destination IP addresses, $P_{(SP)}$ is a source port number, $P_{(DP)}$ is a destination port number, and $P_{(PRO)}$ is a protocol.

B. *Current problems of firewall rule management*

The most serious problem of firewall rule management is anomaly such as shadowing, correlation, generalization and redundancy [2]. It occurs from several reasons. For instances, IT executives have not a good plan for IT management. Some companies with complex IT activities may generate inconsistent rules. These rules may come from carelessly adding some exception rules for some urgent activities, misunderstanding or an ignorance of organizational network policies. In the real situation, for example, ABC Company permits everyone in their company to access public websites on the Internet. Thereafter, some websites may publish illegal contents. The policy must be changed to block the websites. There have been several studies to solve this problem by various techniques such as protecting and detecting the error rules [10], the inferences [2], the verifying simulation [5], rule collapsing [4], artificial intelligence techniques [2], other techniques [8, 9] and so forth. We have found that even these techniques are good for correcting the anomalies. Nevertheless, the root cause of problems has not been eliminated. The root cause is, "*having more than two sets of firewall rule are overlapped, and different decision at the same time*". Let firewall rule 1 is Fwr_1, and rule 2 is Fwr_2, the rules are conflicted if $Fwr_1 \cap Fwr_2 \rightarrow \langle decision \rangle_1 \neq \langle decision \rangle_2$. For example,

$\boldsymbol{Fwr_1}$:$\langle (SA \in [0,\ 2^8 - 1]) \wedge (DA \in all) \wedge (SP \in all) \wedge (DP \in \{80\}) \wedge (PRO \in \{0\}) \rangle \rightarrow \langle \boldsymbol{accept} \rangle$, and $\boldsymbol{Fwr_2}$: $\langle (SA \in [2^6,\ 2^{14} - 1]) \wedge (DA \in all) \wedge (SP \in all) \wedge (DP \in \{80\}) \wedge (PRO \in \{0\}) \rangle \rightarrow \langle \boldsymbol{deny} \rangle$.

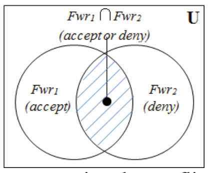


Fig. 1 Venn diagram presenting the conflict of Fwr_1 and Fwr_2

The Venn diagram in Fig. 1 represents the conflict of two sets between Fwr_1 and Fwr_2. The shaded area that is within both the circles represents the sets Fwr_1 and Fwr_2. This area represents the conflict decision between accepting and denying. By allowing the conflict occurs, it leads to the firewall cannot decide what it would react (Accept or Deny). If the firewall believes that the Fwr_1 is right, in the intersection of Fwr_2 will be not processed or the Fwr_2 is valid. So, an intersection area of Fwr_1 is not operated either. Furthermore, a previous study [11] has tried to decide whether Fwr_1 or Fwr_2 is more satisfaction. In fact, the overlapping rules of any two rule sets will not conflict, if the decision field is in the same direction (either accept or deny). According to this concept, we present the situation to support the idea.

Let $(Fwr_2 \subseteq Fwr_1) \wedge (\langle decision \rangle_2 = \langle decision \rangle_1) = \neg$*conflict firewall rule*, for example:

Given $\boldsymbol{Fwr_1}$: $\langle (SA \in all) \wedge (DA \in all) \wedge (SP \in all) \wedge (DP \in \{80\}) \wedge (PRO \in \{0\}) \rangle \rightarrow \langle accept|deny \rangle$, and $\boldsymbol{Fwr_2}$: $\langle (SA \in [2^{14},\ 2^{20} - 1]) \wedge (DA \in all) \wedge (SP \in all) \wedge (DP \in \{80\}) \wedge (PRO \in \{0\}) \rangle \rightarrow \langle accept|deny \rangle$. We can draw Venn diagram to represent this concept as shown in the Fig. 2 below.

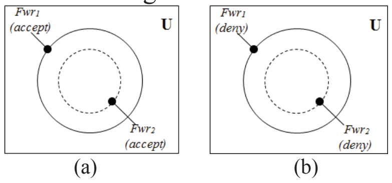


Fig. 2 Venn of $Fwr_2 \subseteq Fwr_1$, which do not occur the rule conflict, (a) both firewall rules are an acceptance decision, (b) denial decision

From Fig. 2 (a) denotes that $Fwr_2 \subseteq Fwr_1$ and both decisions of the rules are permitted, and (b) have both denying decisions. In this situation, firewall does always not conflict between the rules. Yet, a redundancy may remain. However, we coud also swap the rules between Fwr_1 and Fwr_2 freely. To solve the redundancy, we merge two rules into one rule as shown in Fig. 3.

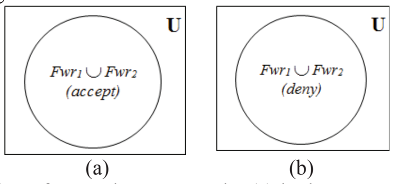


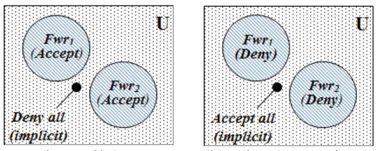
Fig. 3 merging of two rules to one rule, (a) both Fwr_1 and Fwr_2 have the same decision be accepting, (b) denying

III. The Single Domain Decision and Fast Verifying

In this section, we propose the Single Domain Decision concept (SDD) to solve rule anomalies, and the data structure for firewall rule (Binary Tree Firewall: BTF) to increase the speed of rule verification.

A. *A principle of SDD and designing*

In the basic of single domain decision, there are two types: a Close Firewall System (CFS), and an Open Firewall System (OFS). Firstly, CFS is an implicit denying for all services at the beginning of the start-up system. This system leads to all packets cannot pass through the firewall. The packets are always dropped. After that, an administrator can allow some necessary services. This system is quite reasonable for safety system. In the other hand, OFS always opens for all services to pass through the firewall at the start-up time. After working for a while, there may be some harmful activities, such as viruses, worms, denial of service, and so on. The administrator should check, and then block these services or activities. Both systems can be illustrated in Fig. 4. A rectangle indicates the universal set U, which is the set of all firewall rules. Inside this rectangle, we define denying all for CFS, and accepting all for OFS.



(a) The Close Firewall System (b) The Open Firewall System

Fig. 4 Venn for a close and an open firewall system concept (CFS/OFS)

In the SDD concept, we form only accepted decision rules on the CFS, and only denied decision rules on OFS (as shown in Fig. 4). That is, creating firewall rules on CFS will have only firewall rules that decision is particularly acceptance only. For applications or services that do not permit, they will be forced to implicitly deny. In contrast, OFS has only denying rules on the system. For permitted services, they are forced to implicitly accept automatically. The highlight of this concept is to completely protect conflicting problems on previous firewall design by disallowing the duplicated decision of intersect rules. The SDD concept does not only correct conflicting rules. It can also solve other restrictions. For example, firewall is able to re-order rules freely without changing its meaning. Merging several rules to single rule to reduce the redundancy can also be possible. Moreover, the firewall rules can be easily understood, because there are only acceptance rules (for CFS). However, in case that admin requires denying some members in firewall rule on CFS type, we will remove the members to implicit deny by $\forall x\,[\,x \in Fwr_1 \wedge x \notin Fwr_2] \rightarrow \langle accept\rangle$ as shown in Fig. 5.

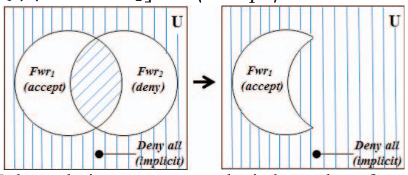


Fig. 5 the technique to remove denied members from CFS

In conclusions of SDD concept, in the CFS has only acceptance rules, and OFS has only denial rules.

For creating the rules of firewall with this concept, we start checking the conflicted rules, when the first rule is generated immediately by disallowing duplicated ⟨*predicate*⟩ and different ⟨*decision*⟩ decisively. We will show each case to create firewall rules (using only the CFS for the rest of this paper) as follows:

Scenario 1: Assume that, ABC company allows everyone in their company to access the generic websites (port number = 80), and secure websites (443). So, the first rule is:

$\boldsymbol{Fwr_1}$:$\langle(SA \in \{all\}) \wedge (DA \in \{all\}) \wedge (SP \in \{all\}) \wedge (DP \in \{80{,}443\}) \wedge (PRO \in \{0\})\rangle \rightarrow \langle accept\rangle$

To obviously clear this Fwr_1, we illustrate by using Tetris game for Firewall Box (TFB) model in Fig. 6.

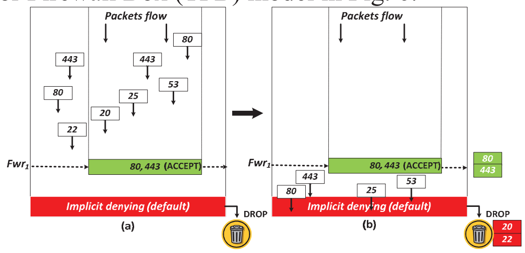


Fig. 6 TFB model while packets flow through Fwr_1, and implicit denying on the CFS system

From TFB model in the Fig. 6, now we have been added one rule in the firewall, named Fwr_1 allowing for generic websites, and secure websites. Supposing that packets have arrived at the firewall (packets flowing from top to bottom in the TFB), these packets are verified with Fwr_1, if a packet matches with this rule (particularly considering in both dash lines from top to bottom of Fwr_1). After that, it can pass through the firewall (presenting by using arrow with a dash line from left to right of Fwr_1). Otherwise, it is automatically dropped with the default rule (representing in a red box and a trashcan). This action is called implicit denial.

Scenario 2: The ABC Company wants to add Fwr_2 permitting to access special destination addresses between 2^5 - 2^8 (224 addresses), and destination ports from 1024 – 1039 (15 ports). The second rule is:

$\boldsymbol{Fwr_2}$:$\langle(SA \in \{all\}) \wedge (DA \in [2^5,\ 2^8]) \wedge (SP \in \{all\}) \wedge (DP \in [1024{,}1039]) \wedge (PRO \in \{0\})\rangle \rightarrow \langle accept\rangle$

Fwr_2 has shown in Fig. 7.

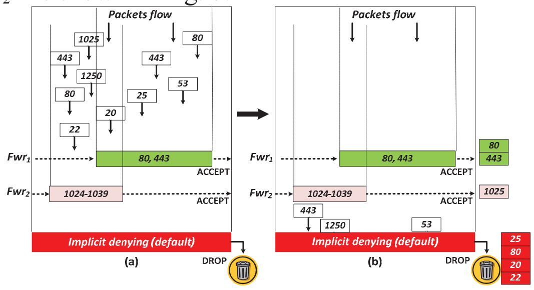


Fig. 7 the packets match and mismatch with Fwr_1, and Fwr_2 on CFS

In the Fig. 7, a pair of dash lines represents a boundary of each firewall rule. Fig. 7 (a) demonstrates packets arrived from external to internal networking, which the packets are firstly filtered by Fwr_1 of the firewall, then filtered by Fwr_2 respectively. While packets touch Fwr_1 in Fig. 7 (b), only packet which is generic websites, or secure websites in a pair of dash lines of Fwr_1 that can pass, the another packets will be dropped with the default rule. For the Fwr_2, it particularly verifies packets that use the port number from 1024 to 1039 and are a member of Fwr_2 only. Both Fwr_1 and Fwr_2 are not conflicted, although they are overlap. We can check the rule's overlapping between Fwr_1 and Fwr_2 by the set operations like: $Fwr_2 \in Fwr_1 \Leftrightarrow \exists x\,[\,x \in Fwr_1 \Rightarrow x \in Fwr_2]$

Scenario 3: In this situation, we create Fwr_3, which will block some source addresses on the Fwr_2 but have a different decision (deny). For the Fwr_3 is:

$\boldsymbol{Fwr_3}$:$\langle(SA \in [0{,}100]) \wedge (DA \in [2^6,\ 2^7]) \wedge (SP \in \{all\}) \wedge (DP \in [1024,\ 1039]) \wedge (PRO \in \{0\})\rangle \rightarrow \langle \boldsymbol{deny}\rangle$

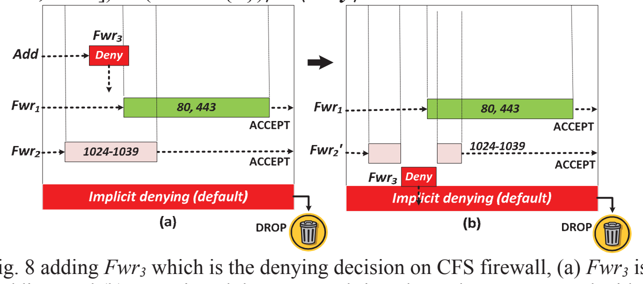


Fig. 8 adding Fwr_3 which is the denying decision on CFS firewall, (a) Fwr_3 is adding, and (b) Fwr_3 ripped the Fwr_2, and ripped members are merged with implicit denying

Fwr_3 is a denial decision. It is inserted in the CFS firewall as shown in Fig. 8 (a). $Fwr_3 \in Fwr_2$ but $Fwr_3 \notin Fwr_1$. In the first step, Fwr_3 is compared with Fwr_1, in Fig. 9 (b). However, Fwr_1 is not modified because it is not a subnet of Fwr_3. Secondly, Fwr_3 *is* compared with Fwr_2, which is ripped due to $Fwr_3 \subseteq Fwr_2$. So, Fwr_2 is separated into two parts, and still remains acceptance decisions for both parts. The ripped members are merged with implicit denying automatically.

Let Fwr_2' denotes the new rule of firewall that occurs from $Fwr_2 - Fwr_3$, that is.

$\boldsymbol{Fwr_2'} = Fwr_2 - Fwr_3 = \forall x[x \in Fwr_2 - Fwr_3 \Leftrightarrow x \in Fwr_2 \notin x\ Fwr_3]$

B. Single Domain Decision Implementation

SDD concept can completely eliminate anomalies on the firewall. However, it leads to an increase of rules when administrator needs to block members in rule with acceptance decision (in case of CFS). For solving this problem, we propose algorithms and the data structure for increase the speed of checking and verifying the firewall rules, namely the Binary Tree Firewall rules (BTF). In this structure, we use the properties of binary tree for sorting and searching, which consume the time complexity be L x $O(\log_2 N) \cong O(\log_2 N)$. L means the depth of the BTF tree that is constant number, it is 5 only, and N is the number of members that must be checked. In case of the generic firewall, a time complexity is $O(N^2)$. The Fig. 9 illustrates BTF data structure, consisting of source addresses (SA) at the first level of the tree (root level). The second level is destination addresses (DA), the third level - source ports (SP), the fourth level - destination ports (DP), and final level is protocol (PRO) respectively.

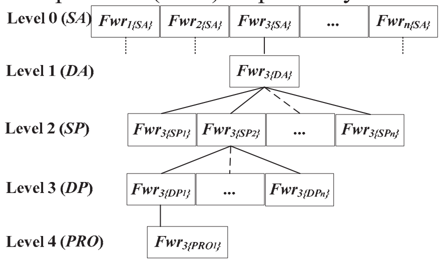


Fig. 9 BTF data structure

To clarify the BTF, we firstly set four rules to explain the structure as follows:

$\boldsymbol{Fwr_1}$:$\langle(SA \in \{\mathbf{192.168.1.1}\}) \wedge (DA \in \{all\}) \wedge (SP \in \{all\}) \wedge (DP \in \{\mathbf{80, 443}\}) \wedge (PRO \in \{\mathbf{0}\})\rangle \rightarrow \langle accept\rangle$

$\boldsymbol{Fwr_2}$:$\langle(SA \in [\mathbf{192.168.2.0/24}]) \wedge (DA \in [\mathbf{172.16.0.0/20}]) \wedge (SP \in \{all\}) \wedge (DP \in \{80, 443\}) \wedge (PRO \in \{0\})\rangle \rightarrow \langle accept\rangle$

$\boldsymbol{Fwr_3}$:$\langle(SA \in [192.168.2.0/24]) \wedge (DA \in [172.16.0.0/20]) \wedge (SP \in \{all\}) \wedge (DP \in \{\mathbf{1863}\}) \wedge (PRO \in \{0, 1\})\rangle \rightarrow \langle accept\rangle$

$\boldsymbol{Fwr_4}$:$\langle(SA \in [\mathbf{192.168.2.0/24}]) \wedge (DA \in [\mathbf{172.16.5.0/22}]) \wedge (SP \in [\mathbf{1024, 1034}]) \wedge (DP \in [\mathbf{1200, 1250}]) \wedge (PRO \in \{\mathbf{0, 1}\})\rangle \rightarrow \langle accept\rangle$

Fwr_1 permits a single source IP address (192.168.1.1) to access both normal and secure websites by using TCP protocol. Fwr_2 allows a group of source IP addresses (256 IP) to access a group of destination IP addresses (4,096 IP). Fwr_3 only opens an internet messenger service (chat). Fwr_4 opens connections between 192.168.2.0 – 192.168.2.255 (256 IP) and 172.16.4.0 – 172.16.7.255 (1024 IP), on 10 source ports (1024 - 1034) to 50 destination ports (1200 - 1250) by using both TCP and UDP protocols. To create the node in BTF tree in Fig. 11, we need to convert the source and destination addresses to positive integers by using this formula.

An address = $SA_{\text{(Octet 4)}}$ x 2^{24} + $SA_{\text{(Octet 3)}}$ x 2^{16} + $SA_{\text{(Octet 2)}}$ x 2^{8} + $SA_{\text{(Octet 1)}}$ x 2^{0}

For example, an IP address, e.g., 192.168.1.1, can be calculated as follows.

192.168.1.1 = (192 x 16,777,216) + (168 x 65,536) + (1 x 256) + (1 x 1) = 3,232,235,777

In case of addresses and port range, we just only calculate start address and stop address, e.g., 192.168.2.0/24, the starting address is 192.168.2.0 (3,232,236,032), and stopping address is 192.168.5.255 (3,232,236,287).

The data structure shows in Fig. 10. It presents that there are overlaps among Fwr_2, Fwr_3 and Fwr_4 but no conflict. This is because the decisions are in the same direction (accept all). Hence, rules are merged into a single rule only, that is $Fwr_2' = Fwr_2 \cup Fwr_3 \cup Fwr_4$. Due to the SDD in Fig.11, it is not necessary to design the tree structure for the decisions, since every rules in the firewall has only a single decision.

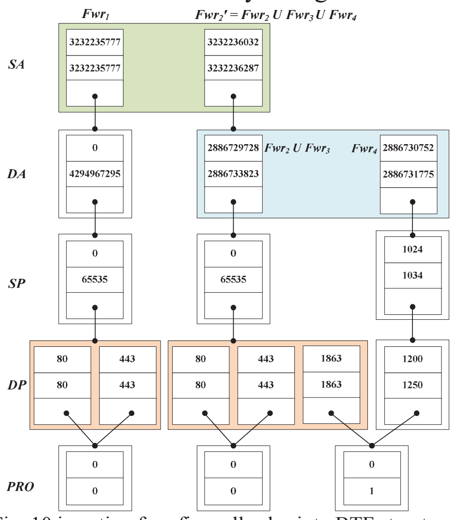


Fig. 10 inserting four firewall rules into BTF structure

Supposing that firewall has worked for a while, after that an administrator would like to add Fwr_5 to disallow some addresses that are members of Fwr_2', in the company to access a service, e.g., the internet messenger application (port 1,863). In this point, Fwr_5 has an effect to split the Fwr_2' to several rules, we called them Fwr_2'' which have an expression below, for the data structure showed in Fig. 11.

Given Fwr_5 is:

$\boldsymbol{Fwr_5}$:$\langle(SA \in \{192.168.2.13\}) \wedge (DA \in [172.16.0.100]) \wedge (SP \in \{all\}) \wedge (DP \in \{1863\}) \wedge (PRO \in \{0\})\rangle \rightarrow \langle \boldsymbol{deny}\rangle$, so

$\boldsymbol{Fwr_2''} = Fwr_2 \cup Fwr_3 \cup Fwr_4 - Fwr_5$

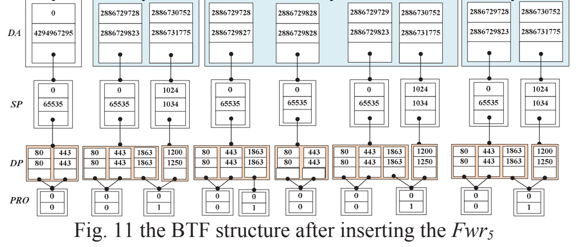


Fig. 11 the BTF structure after inserting the Fwr_5

After we have added the Fwr_5 into the firewall rules, the Fwr_5 affects with Fwr_2 and Fwr_3 directly. This is because Fwr_5 is a subset of both. As a result, the firewall must split Fwr_2' to several parts, and the firewall rules are unavoidably increased. However, the overall performance is still $O(\log_2 N)$. We will explain more in the section IV.

C. *Insert, update and delete algorithm for SDD*

In this section, we introduce algorithms for building the BTF structure for SDD firewall, consisting of inserting (Algorithm 1) and the deleting algorithm (2) respectively. In case of updating, we also use an inserting algorithm.

Algorithm 1: *SDD inserting*

```
1: SET X = Firewall Rule /*set of fields that want to insert*/
2: SET Fwr = Insert Rule (Rule) /*any firewall rule*/
3: LOOP 1: for every element i in X
4: IF (INTERSECT (X(i), Fwr)) THEN
5:     IF (X(i) != Fwr) THEN
6:         $S1_{data1} \leftarrow \min(X(i)_{data1}, Fwr_{data1})$
7:         IF ($X(i)_{data1}$ == $FA_{data1}$) THEN
8:             $S1_{data2} \leftarrow \min(X(i)_{data2}, Fwr_{data2}) - 1$
9:             $S2_{data1} \leftarrow \min(X(i)_{data2}, Fwr_{data2})$
10:        ELSE THEN
11:            $S1_{data2} \leftarrow \min(\max(X(i)_{data1}, Fwr_{data1})) - 1$
12:            $S2_{data1} \leftarrow \min(\max(X(i)_{data1}, Fwr_{data1}))$
13:        END
14:        $S2_{data2} \leftarrow \max(X(i)_{data2}, Fwr_{data2}))$
15:            IF (INTERSECT (X(i),S1)) AND (INTERSECT (X(i),S2)) THEN
16:                $X(i)_{data1} \leftarrow S1_{data2}$
17:                Add S2 in X
18:            ELSE IF( INTERSECT (S2, X(i))) THEN
19:                Add S1 in X
20:            END
21:            IF ($S2_{data2} - S1_{data2}$ != 0) THEN
22:                $Fwr_{data1} \leftarrow \min(X(i)_{data2}, Fwr_{data2}) + 1$
23:                $Fwr_{data2} \leftarrow \max(X(i)_{data2}, Fwr_{data2})$
24:            ELSE STOP
25:        END
26:    END
27: END
28:            IF (Fwr is not Empty) THEN
29:                ADD Fwr in X
30:            END
31: SORT (X)
```

Algorithm *INTERSECT* (set1, set2)

```
1: IF (($set1_{data1}$ >= $set2_{data1}$) AND ($set1_{data1}$ <= $set2_{data2}$)) OR
   (($set1_{data2}$ >= $set2_{data1}$) AND ($set1_{data2}$ <= $set2_{data2}$)) THEN RETURN TRUE
2: ELSE RETURN FALSE
3: END
```

Algorithm 2: *SDD deleting*

```
1: SET X = Firewall Rule
2: SET Fwr = Delete Rule (Rule)
3: FA is Sub-member of X
4: IF (X(i)!=Fwr) THEN
5:     IF ($X(i)_{data1}$== $Fwr_{data1}$) THEN
6:         $X(i)_{data1} \leftarrow Fwr_{data2} + 1$
7:     ELSE IF ($X(i)_{data1}$ == $Fwr_{data2}$) THEN
8:         $X(i)_{data2} \leftarrow Fwr_{data1} - 1$
9:     ELSE THEN
10:        $S1_{data2} \leftarrow Temp$
11:        $X(i)_{data2} \leftarrow Fwr_{data1} - 1$
12:        $S1_{data1} \leftarrow Fwr_{data2} + 1$
13:        Add S1 in X
14:    END
15: ELSE
16:    Remove Fwr from X
17: SORT (X)
```

IV. PERFORMANCE EVALUATION AND EXPERIMENTAL RESULTS

In this section, we have evaluated the efficiency of our firewall by two perspectives. We first prove elimination of anomalies and conflict rules by using SDD concept; then propose the average execution time versus memory space. Furthermore, we have developed a simulation software for testing our firewall by JAVA language v.7, running on MS Windows 7, CPU Quad-core 2.0 GHz, and 8 GB memory. We generate firewall rules that the total number of rules is from 1 – 16,384 rules, and also chose up random all fields of input packets. We have also modeled the test structure to two environments (general, and SDD firewall). After that, we have used *Firewall's overall performance* equation to evaluate them in section B.

A. *Eliminating anomalies on firewall rule*

To test for eliminating anomalies, we refer to Ehab S. Al-Shaer [10]'s four anomalies (Shadowing, Correlation, Generalization and Redundancy). In conclusion, we can completely remove all anomalies and conflicts from our firewall. However, rules may be increased more than the general firewall. As a result, we need to split some rules that have two decisions to only single decision. Nevertheless, the speed of verifying rules is not decreased.

B. *Performance evaluation*

Overall performance = time complexity + space complexity

Let *time complexity = time for building structure + time verifying*

The time for building structure consists of the time of creating and sorting rules. So, we can revise this equation to:

Overall performance = (time for building structure + time verifying) + space complexity.

In Table I, we show the metrics for compute the firewall's performance.

TABLE II

THE FIREWALL'S METRICS

Performance Metrics	**Firewall Type**	
	General	*SSD (BTF)*
Time for building structure	*C*	*C+O(L log N)* Timsort
Time verifying (searching)	*O(N^2)* Sequential	*O(M log N)* Binary
Space complexity	*O(N)*	*O(2N-1)*

Note: C = constant, L and M = deep level of the tree

From Table I, we can compute the time complexity for both firewall types as follows.

Time complexity of general firewall $= C + N^2 \cong N^2$ and

Time complexity of SDD $= C + L \log N + M \log N \cong \log N$

In the testing phase, we have generated rules from 1 to 16,384, and inputted packets by the random technique, and recorded time and space complexity in every epoch, shown in Table II.

TABLE II

FIREWALL'S OVERALL PERFORMANCE

Rules number	**General firewall**			**SDD firewall**		
	Build (msec)	Verify (msec)	Space (Byte)	Build (msec)	Verify (msec)	Space (Byte)
8	3	6	100	3	13	182
16	4	8	200	5	13	376
32	8	13	400	9	15	764
64	16	21	800	22	15	1540
96	20	29	1200	37	16	2316
128	23	37	1600	57	16	3092
192	29	53	2400	79	16	4644
256	35	69	3200	84	18	6196
348	42	91	4350	90	18	8427

512	50	134	6400	111	18	12404
768	62	213	9600	144	18	18612
1024	71	280	12800	208	19	24820
1536	91	423	19200	387	21	37236
2048	106	561	25600	695	21	49652
3072	129	834	38400	1685	22	74484
4096	143	1149	51200	3228	23	99316
6144	165	1723	76800	5873	23	148980
8192	189	2327	102400	9365	25	198644
12288	231	3470	153600	15531	25	297972
16384	253	4742	204800	27964	28	397300

Note: Build = *time for building structure,* Verify = *time verifying rules with packets,* Space = *space complexity, and msec = millisecond*

From Table II, to clarify the trends of consuming of time and space complexity, we summarize and plot to line charts in Fig. 13, 14, and 15 respectively.

Fig. 12 shows the average execution time of verifying rules between SDD and general firewall. The horizontal axis indicates the total of rules which vary from 1 to 16,384. The vertical axis indicates the average verifying time during 0 to 5,000 milliseconds. The number of rules is varied from 8 to 32. The average verifying times of both firewalls are similar around 11 milliseconds approximately. For the number of rules of 64 onwards, the average verifying time of general firewall rapidly grows to around 3,400 milliseconds at 12,000 rules. In contrast, SDD remained constant, and slightly increased from 1,500 to 12,000 rules.

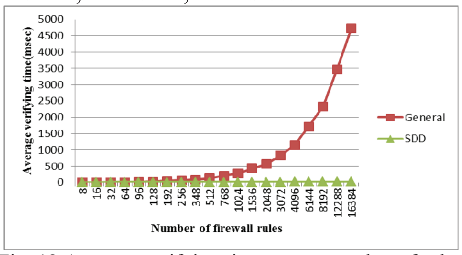


Fig. 12 Average verifying time versus number of rules

Although, the verifying time of SDD is quite better than general firewall. However, the average time of building the rule structure for SDD immediately enlarged from 400 to around 700 milliseconds for 2,000 rules; whereas the general firewall is almost the same, while the number of rules is very large, shown in Fig. 13.

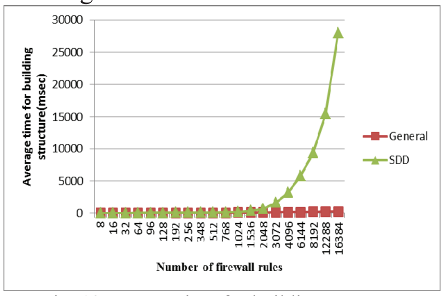


Fig. 13 average time for building structure

In fact, the efficiency of the firewall is counted from the speed of packet verifying through the firewall rules. So, the good firewall should give this as the first priority to improve the firewall. Moreover, firewall is usually not frequently building structure for firewall rules. It mainly builts rule structure at the startup time or the time of inserting or deleting or updating rule only. So, the rule managing activities are generally infrequently. For the space complexity, both firewall consume more or less the memory, and the memory usage grown up while there are more rules, as shown in Fig. 14.

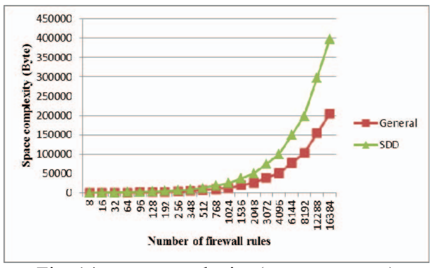


Fig. 14 space complexity (memory usage)

V. Conclusions and Future Work

In this paper, we make a number of contributions in this paper. Firstly, we introduced a new paradigm to completely eliminate rule anomalies, namely SDD. SDD forces the firewall rules to be only single domain decision. Secondly, we present the binary tree firewall for improving the speed of verifying packets. This firewall structure is called BTF. This structure supports the SDD concept and increases the speed from $O(N^2)$ of the general firewall to $O(\log N)$. Thirdly, we have proposed algorithms for inserting, deleting and updating BTF structure. At last, we have evaluated the efficiency of time and space complexities of both firewalls. In Conclusion, the results from experiments show that the speed of SDD is better than the general firewall. Both consume more or less the same amount of memory.

In the future, there are several factors to be considered more on our experiments e.g., bandwidth, traffic, real world policies, etc. We also plan to implement this SDD firewall on the real world network environment.

References

[1] A. X. Liu, "Formal Verification of Firewall Policies," in *Communications, 2008. ICC '08. IEEE International Conference* 2008, pp. 1494 - 1498.

[2] Korosh Golnabi, *et al.*, "Analysis of Firewall Policy Rules Using Data Mining Techniques," in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, 2006, pp. 305 - 315

[3] A. T. Bouhoula, Z. , "Handling Anomalies in Distributed Firewalls," in *Innovations in Information Technology, 2006* 2006, pp. 1 - 5

[4] A. X. Liu, *et al.*, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," in *The 27th Conference on Computer Communications, INFOCOM 2008* 2008, pp. 176 - 180

[5] L. Y. Ghassan Misherghi, Zhendong Su, Chen-Nee Chuah, and Hao Chen, "A General Framework for Benchmarking Firewall Optimization Techniques," in *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT*, 2008, pp. 227 - 238

[6] B. K. Khan, M.K.; Mahmud, M.; Alghathbar, K.S. , "Security Analysis of Firewall Rule Sets in Computer Networks " in *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference*, 2010, pp. 51 - 56

[7] H. E.-A. Hamed, A.; Al-Shaer, E. , "On Dynamic Optimization of Packet Matching in High-Speed Firewalls," *Selected Areas in Communications, IEEE Journal,* vol. 24, pp. 1817 - 1830 Oct 2006.

[8] T. P. Katic, P. , "Optimization of Firewall Rules," in *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference*, Cavtat, 2007, pp. 685 - 690

[9] K. B. P. Liu, "Towards Database Firewall: Mining the Damage Spreading Patterns " in *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual* 2006, pp. 449 - 462.

[10] E. S. H. Al-Shaer, H.H., "Modeling and Management of Firewall Policies," *Network and Service Management, IEEE Transactions,* vol. 1, pp. 2 - 10 April 2004.

[11] R. B. a. W. Noisanguan, "An Axiomatic Approach to Firewall Rule Update," in *The 6th International Joint Conference on Computer Science and Software Engineering (JCSSE 2009)*, Phuket, Thailand, 2009.