Implementation of Packet Filter Configurations Anomaly Detection System with SIERRA

Yi Yin, R.S. Bhuvaneswaran, Yoshiaki Katayama, and Naohisa Takahashi

Department of Computer Science and Engineering Graduate School of Engineering,
Nagoya Institute of Technology,
Gokiso, Showa-ku, Nagoya, 466-8555, Japan
{yinyi, bhuvan, katayama, naohisa}@moss.elcom.nitech.ac.jp

Abstract. Packet filtering in a firewall is one of the useful tools for network security. Packet filtering examines network packet and decides whether to accept, or deny it and this decision is determined by a packet filtering configuration developed by the network administrator. An administrator may find hard to understand and maintain a configuration, and this burden will furthermore be increased to find anomalies between two configurations, especially when the size of filters in a configuration increased. This difficulty may leave the administrator with less confidence that the configurations are correctly and completely implemented. This paper presents a system with SIERRA (A systolic filter sieve array) which can detect the anomalies between two configurations. It provides three functions, side-effects analysis function, equality judgment function, and composition analysis function. Experimental results show that the proposed system is suitable for small network and configurations with large number of filters.

Keywords: Network security, Packet filtering, configuration, filter, anomaly detection.

1 Introduction

Network security has gained significant attention in recent years. Firewalls have become important integrated elements in our society. Packet filtering in firewall is used as a tool for improving network security and performance.

According to the configurations developed by the administrator, packet filtering is a decision of acceptance or denial of a packet. It is difficult for an administrator to understand and maintain configurations in different cases, such as to modify filters in configurations, or to replace one configuration with another configuration written in different language, or to find whether there exist redundant filters in hierarchical structure configurations.

The administrator always has to wonder if the configurations are really accomplishing what was intended, or if the configuration has some inadvertent hole in it that the administrator has somehow overlooked.

In this paper, we proposed an anomaly detection system to help administer to yield correct configurations with greater confidence.

S. Qing et al. (Eds.): ICICS 2005, LNCS 3783, pp. 467–480, 2005.

We have used four primitive operations based on SIERRA (A Systolic Filter Sieve Array used for high-speed packet classification[1]) to implement the proposed system. In our paper, we use SIERRA as the data structure mainly to explain and compare two configurations[6]. The anomaly detection system has three functions as follows and we will discuss in section 4 in detail:

1. Side effects Analysis function.
2. Equality Judgment function.
3. Composition Analysis function.

The rest of the paper is organized as follows. In section 2, we introduce some background work of firewall and configuration. In section 3, we present about SIERRA, and explain the implementation steps of SIERRA. In section 4, we describe primitive operations and discuss three functions in detail. In Section 5, we first describe the experimental system, and findings. We discuss the related work in section 6. Finally, in section 7, we show our conclusions.

2 Background

A firewall is an intelligent device based on configurations between two or more networks for security purposes. A firewall configuration is a list of ordered filter-set that define the actions performed on network packets based on specific conditions. A filter is composed of some key fields (we called these key fields as predicates) and an action field.

The predicates of a filter represent the possible values of the corresponding fields in actual network packet that matches this filter. Each predicate could be a single value or range of values. The most commonly used predicates are: protocol type, source IP addresses, destination IP addresses, source port number, destination port number. The actions of a filter are either "accept" or "deny".

The packet is permitted or blocked by a specific filter if the packet header information matches all the predicates of this filter. Otherwise, the following filter is examined and the process is repeated until a matching filter is found, or else, the default filter action is performed. An example of a typical firewall configuration is shown in Fig.1.

	Protocol	SrcIP	DesIP	SrcPort	DesPort	Action
1:	tcp	*	123.4.5.6	>1023	23	Accept
2:	tcp	*	123.4.5.*	>1023	25	Accept
3:	tcp	129.6.48.254	123.4.5.9	>1023	119	Accept
4:	udp	*	123.4.*.*	>1023	123	Accept
5:	tcp	*	*	*	*	Deny

Fig. 1. Configuration example

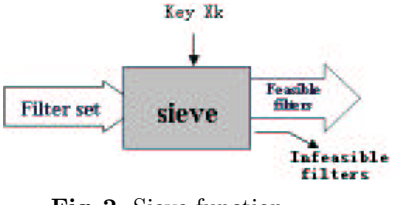

Fig. 2. Sieve function

3 SIERRA

3.1 Problem of Point-Location

SIERRA (A Systolic filter Sieve Array) is used in a high-speed packet classifier. A packet classifier that deals with n key fields (header fields) is modeled as an n-dimensional point location problem in computational geometry. The n-dimensional point location problem is described as follows: given a point in n-dimensional space, and a set of m n-dimensional objects, find the object that the point belongs to. In a packet classifier, a packet corresponds to the point and m filters correspond to the m objects.

3.2 Sieve

Sieve is a function that can reduce an n-dimensional point-location problem to an n-1 dimensional problem[1].

Given a filter-set F, sieve according to k-th (k=1,2,,n) key field value x_k of a packet to remove infeasible filters and return feasible filters. Infeasible filters are filters whose predicates corresponding to the key fields are false, and feasible filters are filters whose predicates corresponding to the key fields are true. Sieve is represented by sieve (F, k, x_k) and the sieve function is shown as in Fig.2.

3.3 Structure of SIERRA

Fig.3 shows that if all the key fields are given as inputs in a pipelined sieve array, a filter-set in which predicates for all key fields are true is derived, that is, the feasible filter-set F' as the output of the final stage of the pipeline. This is represented as:

$F' = F_{n-1}$ provided that
$F_0 = sieve(F, 0, x_0)$,
$F_1 = sieve(F_0, 1, x_1)$, ...,
$F_{n-1} = sieve(F_{n-2}, n-1, x_{n-1})$,

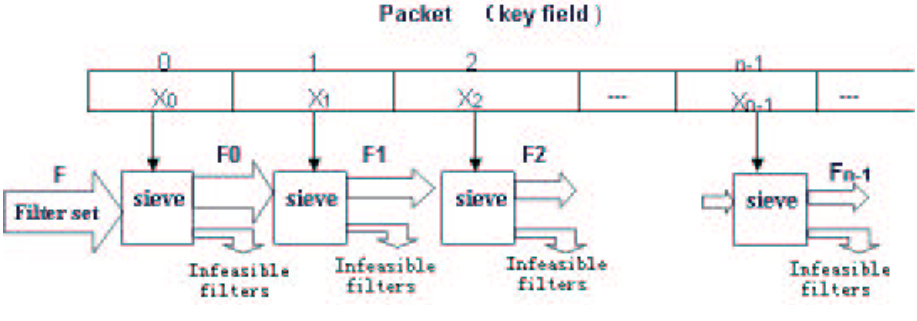

Fig. 3. A sieve array

3.4 Implementation of SIERRA

We use table (that we called sieve table) to implement SIERRA. We consider an abstract filter-set is shown below, F=$\{f_1, f_2, ..., f_m\}$

$f_1 : p_{1,1}\ \ p_{1,2}\ \ \cdots\ \ p_{1,n}$
$f_2 : p_{2,1}\ \ p_{2,2}\ \ \cdots\ \ p_{2,n}$
$\vdots$
$f_f : p_{f,1}\ \ p_{f,2}\ \ \cdots\ \ p_{f,n}$
$\vdots$
$f_m : p_{m,1}\ \ p_{m,2}\ \ \cdots\ \ p_{m,n}$

Here,$f_1, f_2, ..., f_m$ are filter identifiers, and $p_{f,i}$ (f=1,...,m and i=1,...,n) is the i-th predicate of filter. In order to make sieve table, we need several steps shown as follows:

Step1: Represent the i-th predicate $p_{f,i}$ (f=1,...,m and i=1,...,n) of all the filters by 256-bit vectors whose elements are either T(true) or F(false).

In our system, we define that the each predicate value, say x_i is 8-bit long($0 \leq x_i \leq 255$). Any point in this vector whose value changes from "T" to "F" or from "F" to "T" is called a boundary.

Step2: Partition the domain at all boundaries and make domain descriptor at each sub-domain.

The domain of the predicate value ($0 \leq x_i \leq 255$) is partitioned into intervals at all boundaries of all predicates. An order set of values within each interval is called a sub-domain. The set of sub-domains has the following properties.

1. Disjoint: There are no pairs of sub-domains that have common elements.
2. Direct sum: The union of all sub-domains equal to the original domain.
3. Unique: When a sub-domain is determined, the filter-sets for which a "T" predicates is given are uniquely determined.

Each sub-domain is assigned an identifier number, which is called domain descriptor. Whenever there is a boundary, the domain descriptor is increased.

Step3: Find the feasible filters at each sub-domain.

Step4: Check whether i is equal to n; if $i \neq n$, according to feasible filters that get from Step3, execute Step1 through Step4 recursively.

For example, we have a filter-set is shown as Fig.4. For simplicity, the filter-set have two filters and the predicate value of each filter is three bits long, that is the predicate value range is $0 \leq x_i \leq 7$, and can use a 8-bit vectors to present the predicate value. According to step1, we represent the first predicate of all filters into bit vectors. The first predicate $p_{f,1}$ of all filters is shown in Fig.5. The bit vectors of the first predicate of all filters is shown as in Fig.6.

According to step2, we partition the domain of Fig.6 into 5 sub-domains shown in Fig.7 and each domain gives a domain descriptor.

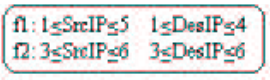

Fig. 4. Example filter-set

Fig. 5. The first predicate of all filters in Fig.4

Fig. 6. Bit vectors of Fig.5

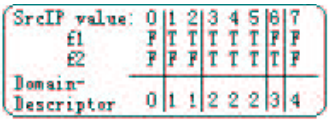

Fig. 7. Domain partition of Fig.6

Fig. 8. Make feasible filters of Fig.7

In step3, according to "T" or "F", we can find the feasible filters of all sub-domains. For example, when the SrcIP value is "3", at this column in Fig.7, the bit vector's element of filter "f1" and "f2" are all "T", so when SrcIP value is "3", the feasible filters are {f1,f2}. In this example, the feasible filter sets of all sub-domains are shown at the last line in Fig.8. According to step1 through step3, we can get sieve table of SrcIP in Table1. In step4, according to the feasible filters, we make the sieve table about next predicate to all the sub-domains. For example, in the third sub-domain (Domain descriptor=2), since the feasible filters are {f1,f2}, so we take the next predicate value of the feasible filters, and shown as follows.

$$\text{f1:} 1 \leq DesIP \leq 4$$
$$\text{f2:} 3 \leq DesIP \leq 6$$

Table 1. Sieve table of SrcIP

SrcIP value	Domain descriptor	Feasible Filters
0	0	{}
1	1	{f1}
2	1	{f1}
3	2	{f1,f2}
4	2	{f1,f2}
5	2	{f1,f2}
6	3	{f2}
7	4	{}

Table 2. Sieve table of DesIP when Domain descriptor=2 in Table1

DesIP value	Domain descriptor	Feasible Filters
0	0	{}
1	1	{f1}
2	1	{f1}
3	2	{f1,f2}
4	2	{f1,f2}
5	3	{f2}
6	3	{f2}
7	4	{}

According to step1 through step3 again, we can get the sieve table of DesIP in Table 2, like this, we make the next predicate's sieve table for all sub-domains, and for the space is limited, we don't list all the sieve tables.

4 Proposed System

The proposed system uses four primitive operations based on SIERRA. In this section, we first introduce the four primitive operations, and then explain three functions in detail.

4.1 Primitive Operations

Create, merge, mark and list-up are the four primitive operations which are discussed below.

Create Operation. Create operation can create sieve table for a given configuration according to the implementation steps of SIERRA discussed in section 3.4.

That is to say, we use sieve table to explain the meaning of configuration.

For example, let us consider two configurations, Configuration A and Configuration B, shown in Fig.9 and Fig.10. For simplicity, each configuration has "DesIP" and "Port" number. A part of sieve table of each configuration is shown in Fig.11 and Fig.12.

	DesIP	Port	Action
A1:	10.0.0.1	53	Accept
A2:	10.0.0.2	*	Deny
A3:	10.0.0.0	123	Accept
A4:	*	*	Deny

Fig. 9. Configuration A

	DesIP	Port	Action
B1:	10.0.0.1	53	Accept
B2:	10.0.0.2	177	Accept
B3:	10.0.0.0	123	Accept
B4:	*	*	Deny

Fig. 10. Configuration B

DesIP	Feasible filters
10.0.0.0	{A3A4}
10.0.0.1	{A1A4}
10.0.0.2	{A2A4}
10.0.0.3~255	{A4}

(a) sieve table of "DesIP"

Port	Feasible filters
0	{A2A4}
...	
255	{A2A4}

(b) sieve table of "Port" when DesIP=10.0.0.2

Fig. 11. Sieve table of Fig.9

Merge Operation. If we have two sieve tables (sieve1 and sieve2), we merge two sieve tables into a new one. A new boundary is made in the merged sieve table where there is a boundary in sieve1 or sieve2. In the merged sieve table, the decision of feasible filters is according to the feasible filters in sieve1 and sieve2. For example, the merged sieve table of the first predicate "DesIP" is shown as in Fig.13 (a).

For example, when DesIP value is 10.0.0.0, the feasible filters in configuration A are {A3,A4}, while in configuration B are {B3,B4}, hence, when DesIP=10.0.0.0 in the merged sieve table, the feasible filters are{{A3,A4} {B3,B4}}.

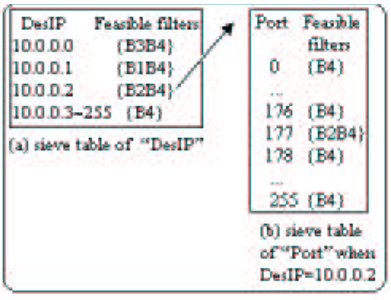

Fig. 12. Sieve table of Fig.10

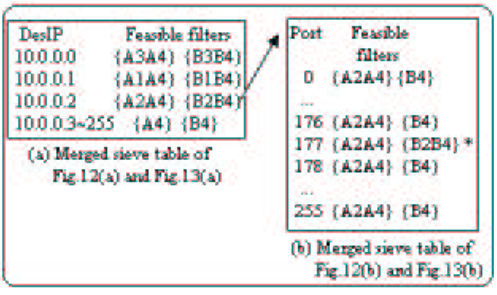

Fig. 13. Merged sieve table of Fig.11 and Fig.12

Mark Operation. In the merged sieve table of the last predicate, to each predicate value, we have feasible filters of two configurations, we take the highest priority filter of each configuration from feasible filters, and then compare the actions of the two filters. If the actions are the same, we do nothing, or else, the place with different actions between two configurations is made a mark.

For example, When the port number is 177, the feasible filters are {A2,A4} and {B2,B4}, the highest priority filter of configuration A is "A2", and the highest priority filter of configuration B is "B2", the action of A2 is "deny", while the action of B2 is "accept", hence, at this place, two configurations have different setting, and we make a mark "*" at the place where the port number is 177 in Fig.13(b).

List-up Operation. The operation that list all marked places from the first predicate value to the last predicate value is called list-up operation.

For example, according to the marked place in Fig.13(b), we can list the predicate value range from the first predicate to the last predicate, it is shown as below.

DesIP	Port
10.0.0.2	177

4.2 Function of Proposed System

The proposed system has three functions which are discussed below:

Side Effects Analysis. When an administrator changed (add, delete, replace, or modify) filters in a configuration, changed filters may cause some potential problems, that is, a changed filter can cause side effects to other filters in configuration.

The side effects analysis function wants to find this kind of potential problems that caused by the changed filters. We present three pieces information as the output of this function.

1. The range of predicate value that influenced by the changed filter.
2. The action setting of each configuration to the range mentioned in 1.
3. Display the changed filters and the influenced filters in the configuration.

According to the result of this function, the administrator is able to master the influence caused by the changed filter, and correctly change the configurations according to the administrator's intents.

For example, we have a configuration in Fig.14, and when the administrator wishes to add a piece of filter (shown as below), the original configuration as file1 and the changed configuration as file2, are considered as input of the proposed system, and the result is shown in Fig.15:

Type	SrcIP	DesIP	SrcPort	DesPort	Action
tcp	*	123.4.5.7	≥ 1023	25	Deny

	Type	SrcIP	DesIP	SrcPort	DesPort	Action
A1	tcp	*	123.4.5.6	>1023	23	Accept
A2	tcp	*	123.4.5.*	>1023	25	Accept
A3	tcp	129.6.48.254	123.4.5.9	>1023	119	Accept
A4	udp	*	123.4.*.*	>1023	123	Accept
A5	tcp	*	*	*	*	Deny

Fig. 14. Configuration

Anomaly :
Range of Influenced Predicate value :

Type	SrcIP	DesIP	SrcPort	DesPort
tcp	*	123.4.5.7	>1023	25

The action setting of influenced range in File1: Accept
The action setting of influenced range in File2: Deny

The influenced filters are:

tcp	*	123.4.5.*	>1023	25	Accept

The changed filters are:

tcp	*	123.4.5.7	>1023	25	Deny

Fig. 15. Result of side effect analysis for the configuration shown in Fig.14

From the result, the administrator noticed that if he wants to add a filter, he should add this filter before A2 in the original configuration, or else, the added filter is no meaning, or the added filter can't take its effect to packet.

We can implement this function using four primitive operations that introduced in section 4.1, and the implementation steps are shown as follows:

Step1: Make sieve table of configuration. (Create operation)
Step2: Make sieve table of the changed configuration. (Create operation)
Step3: Merge two sieve tables get from step1 and step2 and seek feasible filters. (Merge operation)
Step4: Compare two configurations and make a mark at the places where exist different setting. (Mark operation)
Step5: Get the range of predicate value and the filters corresponding to the extracted place as the side effect analysis result. (List-up operation)

Equality Judgment. Configurations can be written in different description languages, for example, the iptables of Linux system, the access-list of cisco router, and etc. If the administrator wants to replace one configuration with

another configuration written in different language, the administrator always wonder whether the two configurations have the same meaning.

The equality judgment function can judge whether two configurations have the same meaning, even if two configurations are written in different description languages.

The equality judgment function can present three pieces of information as follows:

1. The range of predicate value that with different setting between two configurations.
2. The action setting in each configuration to the range mentioned in 1.
3. Display filters that caused the difference between two configurations.

Using the above information the administrator is able to detect the mistakes by the replaced device with different description languages.

For example, when the administrator wants to replace the iptables (Fig.16) by access-list of Cisco router (Fig.17), the proposed system can find whether there exist the different places between two configurations.

```
A1: iptables –A INPUT –p tcp –sport 137:139
    –s 0/0 –d 0/0 –j  ACCEPT
A2: iptables –A OUTPUT –i eth0 –p tcp –tcpflags
    ACK  ACK –j  ACCEPT
A3: iptables –A OUTPUT –i eth0 –p tcp –syn –s 0/0
    –d 0/0 –j  ACCEPT
A4: iptables –A INPUT –j DROP
        ……
```

Fig. 16. Configuration1(iptables)

```
B1: access-list 100 permit tcp any any range 137 140
B2: access-list 200 permit tcp any any established
B3: access-list 200 permit tcp any any syn
B4: access-list 200 deny ip any any
        ……
```

Fig. 17. Configuration2(access-list)

The analysis result is shown in Fig.18:

```
Anomaly :
Range of Predicate value with Different setting :
Type     SrcIP    DesIP    SrcPort    DesPort
 tcp       *        *         140         *
The action setting of this range in File1: Deny
The action setting of this range in File2: Accept
-------------------------------------------------------------
The related filters in File1:
A2: iptables –A INPUT –j DROP
The related filters in File2:
B1: access-list 100 permit tcp any any range 137 140
B2: access-list 200 deny ip any any
```

Fig. 18. Result of equality judgment for configurations shown in Fig16 and Fig17

According to this result, if administrator wants to make configuration2's setting the same as to configuration1, a rule can be added (shown as below, for simplicity we only show the rule with simple style) before B1.

Type	SrcIP	DesIP	SrcPort	DesPort	Action
tcp	*	*	140	*	Deny

And if administrator wants to make configuration1's setting the same as to configuration2, a rule can be added before A2 (shown as below).

Type	SrcIP	DesIP	SrcPort	DesPort	Action
tcp	*	*	140	*	Accept

We can implement this function using four primitive operations, and the implementation steps are shown as follows:

Step1: Make sieve table of one configuration. (Create operation)
Step2: Make sieve table of another configuration. (Create operation)
Step3: Perform merge operation like Step3 in function1. (Merge operation)
Step4: Perform mark operation like Step4 in function1. (Mark operation)
Step5: Get the range of predicate value and filters of extracted place with different settings as an analysis result of equivalence judgment. (List-up operation)

Composition Analysis. In hierarchical structure configurations, each level has a configuration with its own set of filters. Hence, there is a possibility of contradictory or redundant filters among lower and higher levels, which is highly difficult to verify by an administrator.

The composition analysis function is used to find whether contradiction or redundant filters are existing between two configurations that are successively used to filter a packet. This composition analysis function can present three pieces of information as follows:

1. Display the range of predicate value with contradiction setting.
2. Analyze and display the action of contradictive range in each configuration.
3. Display the filter pairs in each configuration that caused the contradiction.

Based on this information, the administrator can able to recognize the unforeseen problem such as the packet that wanted to pass through the lower-class firewall that is abandoned by the upper-class firewall.

For example: We have the upper-class configuration in Fig.19 and the lower-class configuration in Fig.20, the analysis output is shown as in Fig.21.

In this example, according to this result, the administrator can notice that at the range of

Type	SrcIP	DesIP	Port
udp	*	10.0.0.2	117

the action to all the packets in that range in upper-class configuration is "deny", while in lower-class configuration is "accept", and at the same time the administrator can find the filters in upper-class and lower-class configurations that caused the problem.

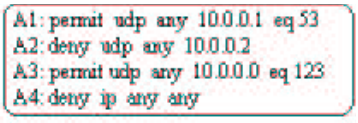

Fig. 19. Upper-class Configuration

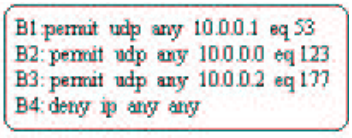

Fig. 20. Lower-class Configuration

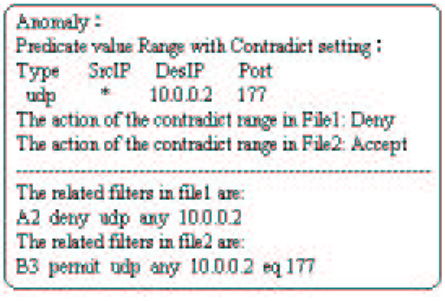

Fig. 21. Result of composition analysis for configuration shown in Fig.19 and Fig20

We can implement this function using four primitive operations, and the implementation steps are shown as below:

Step1 ∼ Step4: The same as Step1 ∼ Step4 in function2.
Step5: From extract division chose the division that the upper-class configuration has "deny" setting while the lower-class configuration has "accept" setting.
Step6: Get the range of predicate value and filters corresponding to the place selected with Step5 as the analysis result of composition analysis. (List-up operation)

5 Experimental Evaluation

We implemented the proposed system with C program on a generic computer. In order to evaluate the feasibility and usability of our proposed system, we measure the synthesis time and memory usage of sieve table when we execute each function.

The synthesis time is the time of taken to create, merge, mark and list up operations. Memory usage of sieve table is the sieve table size that created by each function.

5.1 Experimental Items

We use two experimental items to measure the feasibility and the usability.

1. When the number of predicates increases, and the number of filters is fixed, measure the synthesis time and sieve table size of each function.
2. When the number of filters increases, and the number of predicates is fixed, measure the synthesis time and sieve table size of each function.

5.2 Test Data

We produce configurations with random filters in order to represent the realistic configurations. The filters number in a configuration are range from 5 to 30 for small network firewall, and the filters used in this experiment is that they be stateless.

Each filter can include at most 16 predicates (16-byte) shown as follow:

Source and destination address(es): 8 byte
Source and destination port number: 4 byte
Protocol type: 1 byte
Length: 2 byte
TCP flags: 1 byte

In order to represent all kinds of configurations in practical firewall, in our experiment we select predicate number is equal to 6, 11, 16 to evaluate synthesis time and sieve table size of each function.

5.3 Experimental Results and Consideration

The experiment results are shown as in Table3 to Table 5.

From the result in Table3 to 5, we can see that even in the worst case (in Table4), the synthesis time and sieve table size are acceptable. In the worst case (in Table 4), the experimental system will take $0.97s \sim 4.51s$ and $1.45kb \sim 5.62kb$ memory size to detect anomalies between two configurations with filters from $5 \sim 30$. And in the best case (in Table 5), we take $0.81s \sim 3.71s$ and $1.00kb \sim 3.18kb$ memory size to detect anomalies. Hence, our proposed system is feasible for small network firewall.

Table 3. The synthesis time and Sieve table size of Side effect

Width	6				11				16			
File1_num	5	12	20	30	5	12	20	30	5	12	20	30
File2_num	6	13	21	31	6	13	21	31	6	13	21	31
Time(sec)	0.82	1.39	2.04	2.70	1.54	2.40	2.91	3.27	2.17	3.09	3.42	3.98
Size(KB)	1.25	2.04	2.91	3.59	2.00	3.04	3.66	3.99	2.31	2.93	3.56	4.18

When the number of predicates is fixed, and the number of filters increases, the needed synthesis time and sieve table will increase, this kind of increase is called "increase A".

When the number of predicates increases, and the number filter is fixed, the needed synthesis time and sieve table will increase, this kind of increase is called "increase B".

Through Table3 to 5, we can find that "increase A" is smaller than "increase B". From this result we can confirm that only the predicates number is fixed, and the filter number increases, the synthesis time and sieve table size will not increase very quickly. Hence our proposed system can be usable for configurations with large number of filters.

Table 4. The synthesis time and Sieve table size of Equality Judgment

Width	6				11				16			
File1_num	5	10	20	30	5	10	20	30	5	12	20	30
File2_num	5	10	20	30	5	10	20	30	5	13	20	30
Time(sec)	0.97	1.10	0.99	1.65	1.49	2.35	2.60	3.49	2.07	2.70	4.06	4.51
Size(KB)	1.45	1.53	1.53	2.31	2.00	2.94	3.40	4.50	3.12	3.44	5.15	5.62

Table 5. The synthesis time and Sieve table size of Composition Analysis

Width	6				11				16			
File1_num	5	10	20	30	5	10	20	30	5	12	20	30
File2_num	5	10	20	30	5	10	20	30	5	13	20	30
Time(sec)	0.81	0.93	1.67	2.04	1.27	1.56	2.08	2.52	1.51	2.05	2.80	3.71
Size(KB)	1.00	1.31	1.93	2.56	1.62	1.93	2.56	3.18	1.62	1.93	2.56	3.18

6 Related Work

The similar work of ours has been done by Hazelhurst at et al. [2]. The paper describes a method for transforming a firewall filter specified in a Cisco-like access list language into a BDD(Binary Decision Diagrams), including the handling of issues with overlapping rules. Although this work is capable of answering questions on the types of packets allowed or excluded by a set of firewall rules and even able to display the redundant rules between two configurations, it has many limitations as follows:

1. This tool is insufficient to represent the anomaly detection to various situations of modern firewalls, while in our proposed system, through three functions can provide anomaly detection in detail.
2. This tool has not efficiently implemented in software, while through the experiment result we had evaluate the feasibility and usability of our proposed system.

Another similar work has been done by Pasi Eronen and Jukka Zitting in an expert system for analyzing firewall rules [5]. The expert system is used for verifying the functionality of filtering rules by performing queries.

This work is based on the use of the principle of expert system where firewall access-lists are converted directly into an expert system knowledge base. The resulting knowledge base is then manipulated using the underlying inference engine of readily available Logic Programming language interpreters.

Although this research can address four main areas of problems that is network properties, configuration properties, access-list properties and defining new predicates for higher level queries, this research also has limitations as follows:

1. This tool does not provide comparison function between two configurations, while our proposed system can detect anomaly between two configurations in detail.

2. This tool requires high user expertise to write the proper queries to identify different configuration problems, while our proposed is simple to user to find anomaly easily.

7 Conclusion

This paper presents a system with SIERRA, used to detect anomalies among packet filter configurations.

This proposed system provides three functions, side-effects analysis function when modify a configuration, equality judgment function between two configurations written in different description languages, and the composition analysis function configurations in a hierarchical structure.

Experimental results show that although the synthesis time and sieve table size will increase when the filter number and predicate number increase, the proposed system is suitable for small network and useable for configurations with large number of filters.

Acknowledgment. This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for JSPS Fellows 1604285 and Scientific Research (C) 16500028.

References

1. N.Takahashi, "A Systolic Sieve Array for Real-time Packet Classification," IPSJ Journal, Vol.42, No.2, pp.146-166(2001)
2. Soctt Hazelhurst, Anton Fatti,and Andrew Henwood. Binary decision diagram representations of firewall and router access lists. Technical Report TR-Wits-CS-1998-3, Department of Computer Science, University of the Witwatersrad, South Africa October 1998
3. Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys, 24(3):293-318, 1992.
4. Lakshman,T.V.and Stiliads,D.:High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching, Proc.SIGCOMM 98,pp203-214 (1998).
5. P.Eronen and .J.Zitting. An expert system for analyzing firewall rules. In Proc.6th Nordic Workshop on Secure IT Systems(NordSec 2001), pages 100-107, Nov.2001.
6. Yin Yi, Yosshiaki katayama, Naohisa Takahashi. "A system for Comparing Packet Filter Configuration Files with SIERRA". 2004. Tokai branch rengo conference. (In Japanese)
7. Gupta, P. and Mckeown, N.:Packet classification on multiple fields, Proc. SIGCOMM 99, pp.147-160(1999).
8. Takahashi, N.: Real-time packet classification based on the partial evaluation of filter-sieve functions (in Japanese), Proc. Workshop on Internet Technologies 99, pp.190-197, JSSST (1999).