A Mapping Mechanism for Periodic Filters in a Conflict Detection System for Time-Based Firewall Policies

[1] **Subana Thanasegaran**, [2] **Yuichiro Tateiwa**, [3] **Yoshiaki Katayama**, [4] **Naohisa Takahashi**

[1,2,3,4] Department of Computer Science and Engineering, Nagoya Institute of Technology, Nagoya, Japan

Abstract
Recently, time-based filters are introduced in several practical firewalls like CISCO ACLs and LINUX Iptables to control network traffic in time. It is very handy when a service is required to be available at certain times of a day or at certain days. However, network administrators struggle to maintain time-based firewall policies due to their high-complexity. Conflict is a misconfiguration that occurs when a packet matches two or more filters. It makes the filters either redundant or shadowed, and as a result the network does not reflect the actual configurations of the time-based firewall policies. Even though, conflict detection techniques for time-based filters have been proposed, it takes huge computation time and memory when the conflict detection period is too long due to the enormous repetition of periodic time-based filters. To solve this problem, we have proposed a mapping mechanism to treat the periodic filters and remove the unnecessary repetitions of the periodic filters which reduces the huge computation time and memory. Furthermore, we have evaluated the feasibility and the usefulness of the proposed system by carrying out experiments with the available conflict detection systems with various time-based firewall policies, and have proved the effectiveness of the mapping mechanism.

Keywords:
time-based rules, periodic filters, mis-configuration, time scheduling

1. Introduction

A firewall protects the network from unauthorized access and provides a secure access to the outside world. The packet filtering technique in a firewall provides an initial level of security and operates on the network layer of the OSI model or at the IP layer of the TCP/IP model. It controls the network traffic with a predefined, ordered set of filters called a firewall policy, FP. Every filter, f, has a condition and an action. The condition consists of 'n' predicates and the action can be either accept or deny. Each predicate in the condition is written based on the values in each of the 'n' key fields of a packet header. A packet, P, matches a filter f if and only if the packet header satisfies all the predicates in the condition of the filter. In our research, we adopt the first matching filtering scheme, where the priority of the filter flows from top to bottom like IP firewall in FreeBSD and the action of the first matching filter is executed [1].

The firewall policy management is a notoriously difficult task for any network administrator due to its dynamic and highly interactive filters. For example, while examining 37 firewalls in production enterprise networks, Wool found that all the firewalls are misconfigured and vulnerable [2]. Among many misconfigurations, a conflict is a misconfiguration that frequently occurs in a firewall. It occurs in an FP when a packet matches multiple filters. Conflict is a very common misconfiguration, and Hamed et al. found that there is a high probability of creating conflicts even by expert system administrators [3]. The presence of conflicts in firewall policies either brings malicious traffic into the network or blocks some intended network packets. Therefore, it is necessary to reconfigure the filters to discard the conflicts in the FPs.
Nowadays, time-based filters are widely in use to restrict the network traffic by time [3-20]. They are actively used in many applications like CISCO ACLs, Surf control web filters and Linux Iptables [21]. They are very handy when a service is required to be available only at certain times of a day or even certain days. For example, they can allow better QOS in the accounting department during last three days of a month and as well, can control certain unintended web services in business hours. A firewall policy with a time-field is called as Time-based Firewall Policy, TFP. The time-field specifies the time-constraint, which restricts the access of a filter at specific dates and times. A filter in a TFP can be either periodic or non-periodic. A periodic filter repeats along its own period and the non-periodic filter never repeats. A periodic filter is either active on certain weekdays or certain times of a day and a non-periodic filter is active on certain dates. A packet P matches a time-based filter, f, if and only if the arrival time of the P satisfies the time-constraint of f and the values of the P's key fields satisfy all the conditions of the f.

A conflict occurs in a TFP when a packet matches multiple time-based filters. Many techniques have been developed to detect conflicts in FPs [3-20]. We have proposed a conflict detection system which simultaneously analyzes the time field, 'n' key fields and computes the conflicts in simultaneous basis [5-6]. In the previous approach, initially the time-based filters are mapped in a Conflict Detection Period CDP predefined by the user. The mapped filters are analyzed and the conflicts are computed for the given CDP. However when a user wants to detect

Manuscript received April 5, 2012
Manuscript revised April 20, 2012

conflicts for a longer CDP, the conflict detection system takes huge computation time and memory due to the innumerous repetition of periodic filters in the CDP. Let us demonstrate this problem with an example. Consider a policy TFP1 consisting of f0, f1, f2 where f0, f1 are periodic filters and f2 is a non-periodic filter. The filter f0 is active on everyday from 09AM to 17PM, f1 is active on Monday from 09AM to 12PM and f2 is active on 30/08/2012 from 09AM to 22PM. If a user wants to detect conflicts for a TFP during the year 2012, then the user defines the CDP to be CDP = [01/01/2012, 31/12/2012]. The filters are mapped over the CDP to detect conflicts in the TFPs. During mapping, the filter f0 repeats 366 times, f1 repeats 53 times and f2 never repeats. A same set of filters repeats innumerable times during the mapping and thereby it takes huge computation time and memory to detect conflicts. This problem gets even worse when the number of time-based filters in TFP is large. Therefore it is necessary to tackle the problem of huge computation and memory and by removing the unnecessary repetitions of the periodic filters.

To solve this problem, we have presented a preliminary version of a mapping mechanism to treats the periodic filters which removes the innumerable repetitions of the periodic filters [4]. It analyzes the time field initially and gets the conflicting filters sets of each interval and for each interval the 'n' key fields of the filters are analyzed in iterative basis. In this paper, we have extended the preliminary version and have provided detailed description of the algorithms of the mapping mechanism. Our main contributions in this paper are as follows: (1) A method to determine System Defined CDP which has minimal repetitions of periodic filters. (2) A complete implementation of mapping mechanism with detailed descriptions and examples. (3) A Comparative Analysis of the proposed system with the previous approaches [4-6] and proved the better performance systems.

The rest of the paper is organized as follows. We first describe the formally designed TFP and its conflicts in section 2. In section 3, we present the method of determining the System Defined CDP. We discuss the implementation of mapping mechanism and its algorithms and functions in section 4. In section 5, we present the comparative analysis of various conflict detection systems and evaluate the better performance systems. In section 6, we survey the related work and in section 7, we give the conclusion and some future directions.

2. Conflicts in Time-Based Firewall Policy

2.1. Definition of a TFP

We have formally designed a TFP that consists of an ordered set of 'm' filters, and expressed as follows: TFP: (f0, f1,…,fm-1). Consider the time-based filters fi and fj (i, j ∈ [0, m-1], i<j), where the filter fi is placed before fj in TFP. We adopt the first matching filtering scheme where the priority of the filter decreases from f0 to fm-1. Each filter fi consists of a condition and an action. The condition consists of (n+1) predicates, (pi0, pi1,…,pin), and the fi is expressed as follows:

$$f_i : p_{i0}, p_{i1},\dots,p_{in}, \textit{action}$$

where p_{i0} to p_{in-1} are constraints for the values of the key fields to be used in packet filtering and p_{in} is a predicate which specifies a time-constraint. The commonly used key fields are: source IP addresses (represented as **SrcIP**), destination IP addresses (**DesIP**), source port (**SrcPort**), destination port (**DesPort**), and protocol (**Pro**).

Each predicate p_{ik} ($i \in [0, m-1]$, $k \in [0, n-1]$) can be represented as an exact value, a prefix, a range value or list in many firewall systems. However, in this paper, we use only the range value because of the simplicity and a filter with predicates in the other forms can be easily converted into one or multiple filters with the range values. Each predicate p_{ik} ($i \in [0, m-1]$, $k \in [0, n-1]$) is represented as $a_{ik} \le u_k < b_{ik}$, by using a uniform range value $[a_{ik}, b_{ik})$ and the value in the k^{th} key field of the packet header, u_k. The predicate p_{in} is the time-constraint that the packet arrival time must satisfy to match the filter. It is represented as ***isActive***(f_i, T), which is defined as follows, by using a time-constraint which consists of a time range $[ts_i, te_i)$, a date d_j and a set of days of the week S_i as well as $T = (t, de, dy)$ where t, de and dy are the time, the date and the day of the week when a packet arrives at the firewall.

[**isActive**(f_i, T)]

isActive(f_i, T) = $((ts_i \le t < te_i) \wedge (de = d_i)) \vee ((ts_i \le t < te_i) \wedge (dy \in S_i))$.

Given a time and date T, we say that a filter f_i is **active** if ***isActive***(f_i,T) is TRUE. We also say that a packet P satisfies the time-constraint of a filter f if P arrives at the firewall when the filter f is **active**. Now, we define a predicate ***isMatched***(P, T, f_i) which shows whether a packet with arrival time and date T matches a filter f_i as follows.

[**isMatched**(P, T, f_i)]

isMatched(P, T, f_i) = $(a_{i0} \le u_0 < b_{i0}) \wedge \dots \wedge (a_{in-1} \le u_{n-1} < b_{in-1}) \wedge$ **isActive**(f_i,T).

We represent a filter in a form, called an **internal form,** which includes range values instead of the predicates in (n+1) fields**.** Since we can use a date and days of the week alternatively in the $(n+1)^{th}$ field, i.e. time-field, there are two types of forms to represent a filter as follows:

(1) Type-1: **periodic** filter

f_i: $[a_{i0}, b_{i0}), [a_{i1}, b_{i1}),\dots,[a_{in-1}, b_{in-1}), ([ts_i\ te_i), S_i)$, *action,*

(2) Type-2: **non-periodic** filter

f_i: $[a_{i0}, b_{i0}), [a_{i1}, b_{i1}),\dots,[a_{in-1}, b_{in-1}), ([ts_i\ te_i), d_i)$, *action*
The field of ([ts$_i$ te$_i$), S$_i$) in a type-1 filter and ([ts$_i$ te$_i$), d$_i$) in a type-2 filter are called **ActTIME** in which the subfield, [ts$_i$ te$_i$), is called **TIME** and the subfields, S$_i$ and d$_i$, are called **DAY**. The ts$_i$ and te$_i$ in the sub-field TIME represents the start and stop values, respectively. A time and a date are represented in 24 hours format, hh:mm and in DD/MM/YYYY, respectively. The days of the week is a subset of **S = {Sun, Mon…Sat}**. For example, S$_i$ = {Mon, Fri}.
The Figure 1 shows an example TFP written in the internal form including the 0th field SrcIP, the 1st field DesIP and the time-field ActTIME. In this Figure, a hyphen in the DAY field shows the **S**, i.e. every day. We also represent the predicates and the action by **f$_i$.pname and f$_i$.action** where pname is a predicate name**.** For example, f$_3$.ActTIME.TIME.start=08:00, and f$_1$.action= Accept in a TFP shown in Figure 1.

	Src IP	Des IP	ActTIME		Action
			TIME	DAY	
f_0	[0,3)	[3,7)	08:00-18:00	-	Accept
f_1	[1,5)	[1,5)	16:00-22:00	12/09/2012	Accept
f_2	[3,5)	[1,7)	08:00-18:00	Mon, Wed	Deny
f_3	[0,3)	[3,7)	08:00-20:00	-	Accept
f_4	[3,5)	[1,4)	08:00-12:00	Mon	Deny
f_5	$[0, 2^{32})$	$[0, 2^{32})$	00:00-23:59	-	Deny

Figure 1. An internal form representation of a sample TFP

In the example given in Figure 1, f_0 is **periodic** and becomes **active** from 08:00 to 18:00 on everyday and f_2 is **periodic** and becomes **active** from 08:00 to 18:00 on every Monday and Friday while the default filter f_5 is always **active**. The f_3 is **non-periodic** and becomes **active** from 16:00 to 22:00 on Sep 28, 2011. For example, when a packet *P* arrives on 15:00 Sep 28, 2011, the four filters f_0, f_2, f_4, f_5 becomes **active**. A packet matches a filter *f* from the above filters, if the filter has the values u$_0$ and u$_1$ in SrcIP and DesIP fields such that ($f.a_{i0} \le u_0 < f.b_{i0}$) ∧ ($f.a_{i1} \le u_1 < f.b_{i1}$). The default filter, f_5 has the lowest priority, which denies access to all the packets when no other filter matches the packet.

2.2. Conflict in a TFP

A conflict occurs in a TFP when a packet matches multiple time-based filters. When a condition of a filter overlaps with a condition of another filter, conflict occurs. For example, in Figure 1, we find that the predicate values of the filter f$_0$ overlaps with the predicate values of the filter f$_3$ on everyday from 08:00 to 18:00 hrs and therefore we can say that f$_1$ conflicts with f$_2$. We can broadly classify the conflicts into **errors and warnings**. If a filter f$_i$ completely overlaps with another filter f$_j$, we can say that f$_i$ causes an error to f$_j$ and if f$_i$ partially overlaps with filter f$_j$, we can say that f$_i$ creates a warning to f$_j$.

3. Conflict Detection Period

3.1. Definition of CDP

Conflict Detection period is a period in which the time-based filters are projected and the conflicts are computed for that particular period. In previous conflict detection systems [5-6], the user must define a range of dates for which the conflict detection is performed. For example, if a user wants to detect conflicts for the year 2012, then the user defined CDP is [01/01/2012, 31/12/2012]. The disadvantage is that when the user defined CDP is long, the periodic filters repeats innumerable times in the mapping process resulting in large number of filter sets that takes huge computation time and memory to detect the conflicts in the TFP. This problem becomes even worse when the number of filters in the TFP is large. Therefore, an effective treatment is necessary to handle the periodic filters and to remove the unnecessary repetitions. We have proposed a new method to detect CDP called System defined CDP, SCDP which removes the unnecessary repetitions of the periodic filters.

3.2. System Defined CDP

In this method, the system decides the SCDP with minimum repetitions of periodic filters. It is possible by finding a set of days from the input TFP in which the filters are mapped to detect conflicts in the TFP. In this paper, we have represented the three types of time-based filters as FE, FW and FD where FE is the set of filters which are active in every day, FW is the set of filters which are active in weekday and FD is the set of filters which are active in date. The default filter is not considered for conflict detection as it conflicts with all the remaining filters. For example, for Figure 1, FE = $\{f_0, f_3\}$, FW = $\{f_2, f_4\}$ and FD = $\{f_1\}$. The system defined CDP consists of three sets of days corresponding to each type of time-based filter. The addition of all the three set of days gives the length of SCDP and represented by NCDP. The SCDP consists of the following set of days: (1) A Set of weekdays. (2) A Common day. (3) A Set of dates. The first one corresponds to FW and the number of days is represented by N1, the second one corresponds to FE and the number of days is represented by N2 and the third one corresponds to FD and the number of days is represented by N3. NCDP is the addition of N1, N2 and N3. For each set, its corresponding filters are mapped and the conflicting filter sets are computed. Sometimes the second set is not considered for the calculation of the SCDP when the input TFP consists of the filters which are active in all the seven weekdays. The following steps show the computation of the system defined CDP.

```
Step1: Initialize SCDP = {}, NCDP=0, N1=0, N2=0, N3=0.
Step2: Check the DAY field of the input TFP and add the weekdays to the SCDP.
    For each weekday,
        SCDP = SCDP ∪ {Weekday}.
        N1=N1+1;
Step3: If SCDP consists of all the seven weekdays, then go to step 4. Or otherwise, add a day to SCDP and make N2 to one.
    If N1<7,
        SCDP = SCDP ∪ {A Common day}.
        N2=1;
Step4: Check the DAY field in the input TFP and find the filters which are active in different dates.
    For each date,
        SCDP =SCDP ∪ {Date}.
        N3=1;
Step5: NCDP=N1+N2+N3;
```

For example, the computational steps for the example TFP given in Figure 1 is as follows. The Step1 does the initialization and therefore SCDP = { }, NCDP = 0, N1=0, N2=0, N3=0. The Step2 checks the DAY field and find that there are two weekdays Monday, Wednesday in the example TFP. Add those weekdays to the SCDP and therefore SCDP = {Monday, Wednesday}, N1 = 2. The Step 3 adds a day to SCDP as N1<7 and therefore SCDP = {Monday, Wednesday, A Common Day}, N2=1. The Step4 checks the DAY field in Figure 1 and adds the date to SCDP and therefore SCDP= {Monday, Wednesday, A Common Day, 12/09/2012}, N3=1, and in Step4, NCDP=4. We have found that the length of SCDP is only four and by mapping the filters in only four days, we can compute the conflicting filters of the TFP. The length of SCDP depends upon the input TFP and not the user.

In this paper, we compute the system defined CDP and map the filters in the SCDP and compute the conflicting filters of the TFP. We have developed a mapping mechanism which maps the filters in the system defined CDP.

4. Implementation of Mapping Mechanism

The implementation of mapping mechanism is accomplished through time divisor. The main goal of the mapping mechanism is to extract the essential component of the time field – SFS (Set of filter sets). The set of filter sets SFS is computed by projecting the filters in the SCDP. The computational steps of the extraction of the essential components of the remaining 'n' fields are not explained in this paper as those steps are same as described in [5].

We have introduced the following notations to be used in the implementation of the mapping mechanism. The notation FS_{CD} represents the set of filters that should be mapped in **A Common Day**. FS_{WDAY} represents the set of filters that should be mapped in a weekday WDAY in **A Set of Weekdays**. For example, FS_{SUN} represents the filters that are active on Sunday; FS_{MON} represents the set of filters that are active on Monday and so on. Likewise $FS_{DT(j)}$ represents the set of filters that should be mapped in some jth date in **A Set of Dates**. The variable 'j' represents the number of DATES in a TFP. For example, in Figure1, FS_{DT1} represents the filters that are active on some date 12/09/2012. The notation F (i) returns the ith filter of a filter set F. For example, FE (i) returns the first filter in FE and therefore returns f_0. We have proposed two functions named MakeCDP and GetDay. The MakeCDP (TFP) function computes the system defined CDP, N1, N2, N3, and NCDP for the given input TFP. The GetDay (Date) function finds the day of the week for the given date. The computational steps of SFS are as follows:

```
Step1: (SCDP, N1, N2, N3, NCDP) = MakeCDP (TFP);
Step2: (for i=1, i<=N1, i++)
{
    If (FW (i).ActTIME.DAY=SUN)
        Append FE, FW (i) to FS_SUN;
    If (FW (i).ActTIME.DAY=MON)
        Append FE, FW (i) to FS_MON;
    If (FW (i).ActTIME.DAY=TUE)
        Append FE, FW (i) to FS_TUE;
    If (FW (i).ActTIME.DAY=WED)
        Append FE, FW (i) to FS_WED;
    If (FW (i).ActTIME.DAY=THU)
        Append FE, FW (i) to FS_THU;
    If (FW (i).ActTIME.DAY=FRI)
        Append FE, FW (i) to FS_FRI;
    If (FW (i).ActTIME.DAY=SAT)
        Append FE, FW (i) to FS_SAT;
}
Step3: If (N1<7)
    add FE to FS_CD;
Step4: (for j=1, j<=N3, j++)
    {
        if (FD (j).ActTIME.DAY=DATE)
            Add FD (j) to FS_DT(j);
            WDAY=GetDay(FAD (j).ActTIME.DAY);
            Append FS_WDAY to FS_DT(j);
    }
Step5: Map each set of filters to its corresponding set of days in the system defined CDP according to the values in the time field of the filter, f_i.ActTIME.start and f_i.ActTIME.end.
```

Step6: Decompose each filter in their boundaries mapped on the SCDP and the time intervals such as $\{I_0, I_1\ldots\}$ are created.

Step7: Obtain a set of filters from each interval created in Step6 and makes SFS using the filter sets. In this step the repetitive filters sets are removed from SFS.

For example, the computational steps for the TFP shown in figure 1 are as follows: We have already discussed about the computation of the CDP where N1=2, N2=1, N3=1, NCDP=4. In Step2, it makes FS_{MON} and FS_{WED} and the remaining filter sets such as FS_{SUN} FS_{TUE} are null. In Step2, initially FE is added to FS_{MON} and FS_{WED} and the corresponding filters in FW are added and then $FS_{MON} = \{f_0, f_3, f_2, f_4\}$ and $FS_{WED} = \{f_0, f_3, f_2\}$. In Step3, as N1<7 we found that $FS_{CD} = \{f_0, f_3\}$. Initially in Step4 as N3=1, FS_{DT1} $\{f_1\}$ and then in step4, we found that WDAY of f_1 is Wednesday from the GetDay function and therefore FS_{WED} is added to FS_{DT1} and finally $FS_{DT1} = \{f_1, f_0, f_3, f_2\}$. The Step5 maps each set of filters on their corresponding set of days. Each filter mapped in the time field and the filters are decomposed in their boundaries and intervals, I_0, I_1 are created as shown in the Figure 2. The Step7 gets the set of filters in each interval and removes the repetitive filter sets and makes SFS. For example, the filter set $\{f_0, f_2, f_3\}$ repeats in I_1, I_3, I_7, and therefore the repetitive filter sets are removed and therefore SFS = $\{\{f_0, f_2, f_3, f_4\}, \{f_0, f_2, f_3\}, \{f_3\}, \{f_0, f_3\}\ldots\}$.

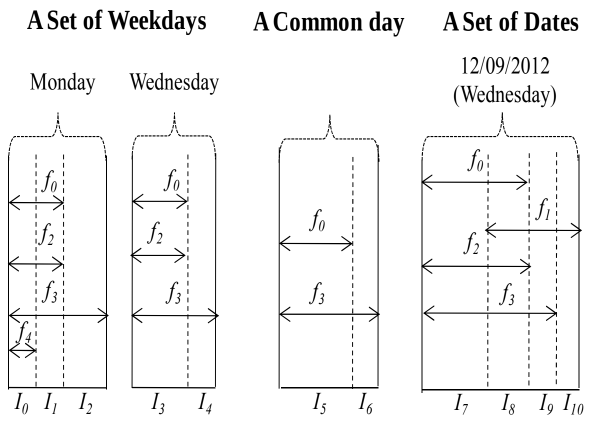


Figure 2. Mapping of filters in the System Defined CDP for the TFP in figure 1

5. Experimental Analysis

5.1. Experiments

We evaluate the feasibility and usefulness of the proposed system by comparative studies with the related works appeared in [4-6]. As we have discussed earlier, the conflict detection system appeared in [4] treats the periodic filters with mapping mechanism as it computes the System Defined CDP and detects the *n*-dimensional TR of the filters for each interval, I_i. For example, if there are p intervals, then the *n*-dimensional TR of the filters are computed for 'p' times as depicted in Figure 3. The disadvantage of the technique is that, the conflict detection results are confined to a single interval and it is necessary to summarize the results of all the intervals to compute the conflicts. The disadvantage of the previous system [4] is overwhelmed by conflict detection system appeared in [5-6]. It computes the conflicts for the whole TFP rather than intervals through its simultaneous approach by finding the *n+1*-dimensional TR of the filters as shown in Figure 4. It simultaneously analyzes the 'n' key fields and the time field and computes the conflicts for the whole TFP. But it detects conflicts for a user defined CDP. Therefore when the user defined CDP is longer, it suffers from the problem of huge computation time and memory due to the unnecessary repetitions of the periodic filters.

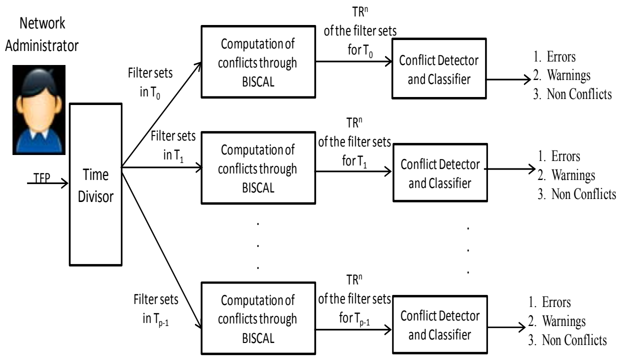

Network Administrator
TFP
Time Divisor
Filter sets in T0
Filter sets in T1
Filter sets in Tp-1
Computation of conflicts through BISCAL
TRn of the filter sets for T0
TRn of the filter sets for T1
TRn of the filter sets for Tp-1
Conflict Detector and Classifier
1. Errors
2. Warnings
3. Non Conflicts

Figure 3. A Conflict detection system through interval basis

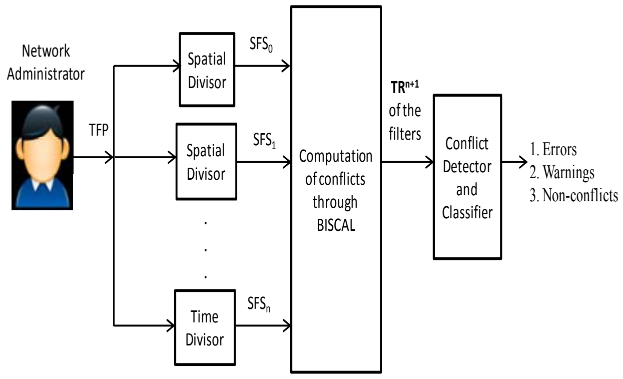


Figure 4. A conflict detection system through simultaneous analysis

As we know that, there are two different approaches to detect the conflicts of TFP and two different methods to find the CDP, therefore we construct four systems by making different combinations of the systems and the methods of CDPs are as follows:

1. **SystemA**: A conflict detection system through interval basis by the user defined CDP.

2. **SystemB**: A conflict detection system through interval basis by the system defined CDP.
3. **SystemC**: A conflict detection system by finding the *n+1*-dimensional TR by the user defined CDP.
4. **System D**: A conflict detection system by finding the *n+1*-dimenisonal TR by the system defined CDP.

We compare the above four systems and finds the system that performs well in terms of computation time. To perform the comparative analysis, we have implemented the above systems in JAVA programming language and the experiments were performed on Intel (R) Core (TM) i5 CPU 750 @ 2.67 GHz 2.67 GHz with 4.00GM RAM running on Windows 7 professional. Due to the unavailability of publicly used TFPs, we have synthesized ample number of policies, TFP (m) which was synthesized from FP (m), which has *m* filters as follows:

(1) **FP (m)**

We have synthesized a firewall policy without time-field, FP(m), by using a firewall policy of 100 filters without the time-field, FP_L, which has five fields, SrcIP, DesIP, SrcPort, DesPort and Pro and is used in our laboratory, as follows: it is a first-m filters of the FP_L in case $m \leq 100$ while it is a combination of the FP_L and new filters which are synthesized by using a randomly selected filter from FP_L in case m>100.

(2) **TFP (m)**

We have converted the FP (m) to TFP (m) by adding the values of TIME and DAY in the ActTIME field as follows: the k^{th} filter in TFP (m) is a combination of the k^{th} filter of FP (m) and generated values of TIME as given by the two time charts shown in figure 5. The time chart shown in Figure 5(a) is created based on controlling certain network services that can be used in industrial environments. The time chart shown in figure 5(b) is an example in which large number of filters conflicts with each other.

Time intervals	
1. Business Hrs	10:00-17:00
2. Morning	08:00-12:00
3. Afternoon	12:00-17:00
4. Lunch	12:00-13:00
5. Full day	00:00-23:59

(a) Case I

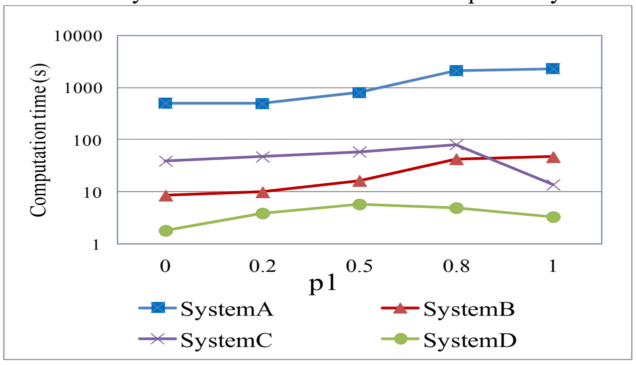
Time intervals	
1.	01:00 – 23:00
2.	02:00 – 22:00
3.	03:00 – 21:00
4.	04:00 – 20:00
5.	05:00 – 19:00
6.	06:00 – 18:00
7.	07:00 – 17:00
8.	08:00 – 16:00
9.	09:00 – 15:00
10.	10:00 – 14:00
11.	11:00 – 13:00

(b) Case II

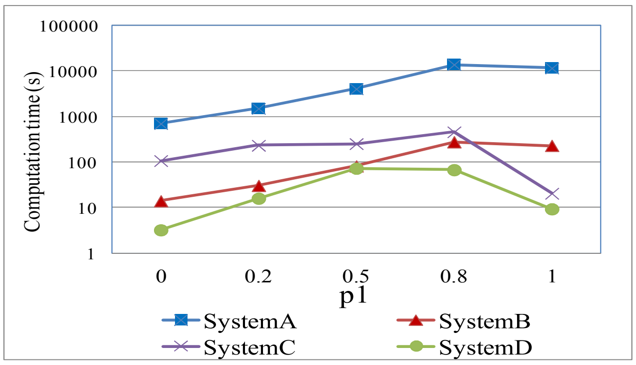
Figure 5 Time Charts

We randomly generated different values of DAY controlled by two parameters, p1-the percentage of FE and p2- the percentage of FW. We varied the percentage of p1 and p2 by maintaining the sum of p1 and p2 as 1. For example, if 20% of p1 filters is added then 80% of p2 filters are added to make the sum as 100%.

The SystemA and SystemC compute the user defined CDP and the system defined CDP is computed for SystemB and SystemD. We have conducted experiments with four different systems and have found the computation time to find the conflicts in TFP and plotted the graph where p1 is plotted in X-axis and the computation time is plotted in Y-axis. The graphs shown in figure 6(a) and (b) shows that the computation time of the all the four systems for case I and Case II respectively.



(a) Case I



(b) Case II

Figure 5. Comparative Analysis of the different systems

We can infer that from the graphs shown in Figure 5 that, when the ratio of p1 is between 0.5 and 0.8, computation time is comparatively higher due to the large number of conflicting filter sets. When the ratio of p1 is low, computation time is lower due to the small number of conflicting filter sets. When the ratio of p1 is between 0.8 and 1, the computation time decreases gradually because of the lower number of days in the System Defined CDP, SCDP and therefore the number of intervals is lower even though the number of conflicting filters is large. When we analyze the graphs we found that SystemC outperforms SystemB when p1 is 1 and SystemB outperforms SystemC when p1 is between 0 and 0.8. Among all the performances, we have found that SystemD performs better the remaining systems. It is because of the advantage of the SCDP and the mapping mechanism and also the advantage of finding the n+1-dimensional TR of the filters. Therefore we conclude the SystemD is efficient among all the systems and it would be helpful for the system administrator to take decisions quickly for a TFP.

6. Related Work

In this section, we briefly discuss about the researches on the management of firewalls [2-20]. Hamed *et al.* proposed an algorithm to detect the conflicts caused between the filters in a firewall policy based on the relations between every two filters and the taxonomy of the different conflicts in a firewall policy [3]. Y. Yin *et al.* proposed a technique to detect the conflicts caused by a combination of filters by analyzing the filters in *n*-dimensional space [7]. Both of them do not recommend how to handle time-based firewall policies.

V. Capretta *et al.* proposed the formalization of conflict detection for firewalls; it defines conflicts for filters only if the actions of the filters are different [15], but they do not provide any methods to deal with TFPs. In this paper, we have explicitly classified the conflicts of the time-based filters for cases where the actions of the filters are same as well as different. Alex *et al.* developed a technique to compress the firewall policy by minimizing the number of filters [16]. M. Yoon *et al.* proposed a technique to minimize the firewall rule set in a multiple firewall environment [17]. These methods dealt with common headers like Source IP and Destination IP, but do not provide any suggestions for firewalls where a time field is included.

Mayer *et al.* developed a firewall analysis tool *Fang* to perform customized queries on a set of filters and operate on a more understandable level of abstraction [11]. Wool *et al.* improved the usability of *Fang* [12]. G. Misherghi *et al.* proposed a general framework for rule-based firewall optimization [18]. These tools and methods help the administrators to verify the correctness of firewall policies manually. However, they have not analyzed the cases where time-based filters are present. K. Matsuda proposed a matrix decomposition model to analyze the filters in FPs with few compression methods [26]. In his work, for some reasons, an administrator can intentionally embed redundant filters, which will conflict with the other filters in a firewall policy. However, time-based firewall policies were not considered in this model.

H. G. Verizon *et al.* proposed a fast and scalable method for resolving the anomalies in firewall policies [19] which can be useful for large-scale firewall policies. H. Hu *et al.* proposed a firewall anomaly management and resolution environment: FAME, and developed a grid-based visualization of the firewall policy [20]. However, both of the above techniques do not provide any methods to detect conflicts in TFPs. The above-discussed techniques provide different approaches to manage firewall policies, but none of the above techniques dealt with time-based firewall policies.

We have proposed a conflict detection system for the TFPs which simultaneously analyzes the time field, 'n' key fields and computes the conflicts in simultaneous basis [5-6]. However when the user defined CDP is long, the conflict detection system takes huge computation time and memory due to the innumerous repetition of periodic filters in the CDP.

We have presented a preliminary version of a mapping mechanism to solve the unnecessary repetitions of the periodic filters appeared in [4]. The drawback of the previous paper is that the proposed system detects conflicts in iterative basis and the conflict detection results are confined only to certain time intervals. In this paper we have developed a conflict detection system which drives away the drawbacks of the previous versions [4-6] and proved the same in the experimental analysis. In this paper, we have solved the conflict detection problems in the time-based filters which was not addressed in any of the related works discussed above.

7. Conclusion and Future Work

In this paper, we have provided methods to determine CDP with minimal number of repetitions and discussed the implementation of the mapping mechanism in detail. We have proved with our experimental analysis that the performance of SystemD is efficient among all the other systems. By adopting System D, the workload of the administrator is considerably reduced due to the effort to reconfigure the TFPs to discard conflicts is reduced. Our future research plan focuses on the detection of conflicts caused by combinations of filters for the TFPs.

Acknowledgments

This research was partially supported by JSPS KAKENHI 23500085 and by the Hori Science and Art Foundation.

References

[1] The FreeBSD Documentation Project, Ipfw, http://freebsd.org/doc/enUS.ISO88591/books/handbook/firewalls-ipfw.html
[2] A. Wool, "A quantitative study of firewall configuration errors," Computer, vol.37, pp.62-67, 2004.
[3] H. Hamed, E. Al-Shaer, "Taxonomy of Conflicts in Network Security Policies," IEEE Communication Magazine, vol.44, no.3, pp.134-141, 2006.
[4] T. Subana, Y. Tateiwa, Y. Katayama, N. Takahashi, "An improved conflict detection system with periodic cycle treatment for time-based firewall policies", Proc. of 19th IEEE ICCCN 2010, pp.1-8, Zurich, Swiss, Aug 2010.

[5] T. Subana, Yuichiro Tateiwa, Yoshiaki Katayama, Naohisa Takahashi, "Design and Implementation of Conflict Detection System for Time-based Firewall Policies," JNIT: International Journal of Next Generation Information Technology, Vol. 2, No. 4, pp. 24-39, Nov 2011.

[6] T. Subana, Y. Yin, Y. Tateiwa, Y. Katayama, N. Takahashi, "Simultaneous analysis of time and space for conflict detection in time-based firewall policies," Proc. of 10th IEEE International Conference on CIT, pp.1015-1021, Bradford, June 2010.

[7] Y. Yin, Y. Katayama, N. Takahashi, "Detection of conflicts caused by a combinations of filters based on spatial relationships," IPSJ Journal, vol.49, pp.3121-3135, Sep 2008.

[8] D. Eppstein, S. Muthukrishnan, "Internet packet filter management and rectangle geometry," Proc. of 12th Annual ACM-SIAM SYM, Washington, pp.827-835, 2001.

[9] K. Golnabi, R.K. Min, L. Khan, E. Al-Shaer, "Analysis of Firewall Policy Filters using Data Mining Techniques," Proc. of IEEE NOMS 2006, pp.305-315, Canada, April 2006.

[10] L. Yuan, J. Mai, Z. Su, H. Chen, P. Mohapatra, "FIREMAN:a toolkit for firewall modeling and analysis," Proc. of IEEE Symposium on Security and Privacy, pp.199-213, Oakland, May 2006.

[11] A. Mayer, A. Wool and E. Ziskind, "FANG: a firewall analysis engine," Proc. of IEEE Symposium on Security and Privacy, pp. 177-187, Oakland, May 2000.

[12] A. Wool, "Architecting the lumeta firewall analyzer," Proc. of 10th Conf. USENIX Security SYM, pp.7-7, USA, Aug 2001.

[13] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, C. Pitcher, "Specifications of a high-level conflict-free firewall policy language for multi-domain networks," Proc. of SACMAT'07, pp.185-194, Sophia Anti polis, France, June 2007.

[14] Hari. A, Suri. S and Parulkar. G, "Detecting and resolving packet filter conflicts," Proc. of IEEE INFOCOM 2000, pp.1203-1212, Israel, Mar 2000.

[15] V. Capretta, B. Stepien, A. Felty and S. Matwin, "Formal correctness of conflict detection for firewalls," Proc. of ACM workshop on Formal Methods in Security Engineering, Virginia, pp.22-30, USA, Nov 2007.

[16] Alex X. Liu, Eric Torng, Chad R. Meiners, "Firewall Compressor: An Algorithm for Minimizing Firewall Policies," Proc. Of 27th IEEE INFOCOM, Phoenix, Arizona, pp.176-180, April 2008.

[17] M. Yoon, S. Chen, Z. Zhang, "Minimizing the Maximum Firewall Rule Set in a Network with Multiple Firewalls," IEEE Transactions on Computers, vol. 59, no.2, pp. 218-230, Feb 2010.

[18] G. Misherghi, L. Yuan, Z. Su, C-N. Chuah, and H. Chen, "A General, Framework for Benchmarking Firewall Optimization Techniques," IEEE Transactions on Network and Service Management, vol. 5, no. 4, pp. 227-238, Dec 2008.

[19] H.G.Verizon, K.A. Ahmat, "Fast and Scalable Method for Resolving Anomalies in Firewall Policies",Proc. of 14th IEEE Global Internet Symposium 2011 at IEEE INFOCOM 2011, pp. 839-844, April 2011.

[20] H.Hu, G.J.Ahn and K.Kulkarni, "FAME: A Firewall Anomaly Management Environment", Proc. of ACM SafeConfig'10, ISBN: 978-1-4503-0093-3, Oct 2010.

[21] http://linux.die.net/man/8/iptables

[22] N. Takahashi, "A Systolic Sieve Array for Real-time Packet Classification," IPSJ Journal, vol. 42, no.2, pp.146-166, 2001.

[23] T. Srinivasan, N. Dhanasekar, M. Nivedita, R. Dhivyakrishnan, A. A. Azeezunnisa, "Scalable and parallel aggregated bitvector packet classification using prefix computation model," Proc. of Int Symposium on PAR ELEC 2006, pp.139-144, Bialystok, 2006.

[24] T. V. Lakshman, "High-speed policy based packet forwarding using efficient multi-dimensional range matching," Proc. of ACM SIGCOMM 98, vol. 28, pp.203-214, Vancouver, Sep 1998.

[25] S. Singh, F. Baboescu, G. Varghese, J. Wang, "Packet classification using multidimensional cutting," Proc. of ACM SIGCOMM 03', pp.213-224, Germany, Feb 2003.

[26] K. Matsuda, "A packet filtering filters compression by decomposing into matrixes," IPSJ Journal, vol.48, no.10, pp.3357-3364, 2007 [in Japanese].

[27] https://www.cisco.com/en/US/docs/security/pix/pix63/release/notes/pixrn634.html

[28] http://www.pcis.com/products/astaro_firewall.html

[29] Max J Egenhofer, "A Formal Definition of Binary Topological Relationships," Proc. of LNCS 367/1989, pp.457-472, USA, 1989.