Fast Conflict Detection for Two-Dimensional Packet Filters

Chun-Liang Lee^{a,*}, Chung-Yuan Huang^a, Tung-Yi Chen^a

^aDepartment of Computer Science and Information Engineering, School of Electrical and Computer Engineering,
College of Engineering, Chang Gung University,
No. 259, Wen-Hwa 1st Road, Kwei-Shan, Tao-Yuan 33302, Taiwan

*Corresponding Author: cllee@mail.cgu.edu.tw

Abstract

Advanced network services such as firewall, differentiated services, and virtual private networks require network devices to perform packet classification. According to pre-defined rules (or filters), network devices classify incoming packets into different flows and execute the operation specified by the matching filter. However, two filters may overlap and leads to a conflict if the actions specified in the filters conflict. Filter conflicts may cause security breaches or quality-of-service failures. In this paper, we propose an algorithm that can efficiently detect and resolve filter conflicts. By analyzing the conditions that two conflicting filters must hold, our proposed algorithm can detect conflicts without complicated data structures. The time complexity of our proposed algorithm is $O(n^2)$, and the space complexity is $O(n)$, where n is the number of filters. Experimental results show that our proposed algorithm can reduce the average detection time per filter by 30% to 56% compared to the scalable bit vector (SBV) algorithm. Moreover, the memory required by our proposed algorithm is much smaller than that by the SBV algorithm.

Keywords: packet classification, conflict detection.

1. Introduction

Today's Internet offers not only best-effort services, but also advanced network services such as differentiated services, firewalls, virtual private networks, and policy-based routing. All these advanced services require routers to perform packet classification—that is, classifying incoming packets into different flows based on one or more fields in the packet header and a set of rules (or filters). A filter $F = (f[1], f[2], \dots, f[k])$ is called k -dimensional if the filter consists of k fields, where each field can be a variable

Table 1. An example filter database.

Filter	Field 1	Field 2	Action
A	1_b^*	111_b^*	Allow
B	0_b^*	10_b^*	Deny
C	*	101_b^*	Allow
D	100_b^*	*	Deny

length prefix, a range, an exact value or wildcard. Possible fields in a filter include network source address, network destination address, source port, destination port, and protocol type. A packet P is said to match a filter if every field in the packet header satisfies the field defined in the filter. Each filter has an associated action that specifies how to treat packets matching this filter.

It is possible that a packet matches more than one filter. In this case, ambiguity may arise if the actions of matching filters conflict. Depending on the implemented services, filter conflicts may cause different problems such as security breach, violation of service level agreement, and routing problem. For example, Table 1 gives a two-dimensional filter database with 4 filters. Suppose that the maximum length of each field is 4 bits. A packet $P1 = (1000_b, 0000_b)$ matches filter D , and will be denied to pass through if this device is a firewall. However, another packet $P2 = (0000_b, 1010_b)$ matches both filters B and C . The conflicting actions of filters B and C cause a security problem if packets that should be blocked are allowed to pass through.

One approach to resolve conflicts is to select the first matching filter⁽¹⁾. Through reordering the filters, the expected matching filter can be moved before other matching filters. However, filter reordering cannot fully solve the problem caused by filter conflicts. Hari *et al.*⁽²⁾ used resolve filters to solve this problem. A resolve filter is a filter that matches the packets in the overlap region of two

conflicting filters. However, it has been shown that finding the minimum number of resolve filters in a filter database is an NP-hard problem⁽²⁾. Therefore, it is a challenging task to design an algorithm that can efficiently detect and resolve filter conflicts. In this paper, we propose an efficient conflict detection algorithm for two-dimensional packet filters. By analyzing the conditions that two conflicting filters must hold, our proposed algorithm can quickly detect all conflicts in a filter database without generating complicated data structures. Therefore, the memory required by our algorithm is much smaller than other algorithms.

The rest of this paper is organized as follows. Section 2 reviews existing conflict detection algorithms. Section 3 describes our proposed algorithm. Section 4 presents the experimental setup and results. Finally, Section 5 concludes the paper.

2. Related Work

Hari *et al.*⁽²⁾ proposed the use of two grid-of-tries (GoT)⁽³⁾ to detect all conflicts in two-dimensional (2-D) prefix filters. The time complexity of their algorithm is $O(nW+s)$, where n is the number of filters, W is the length of the longest prefix and s is the number of pairs of conflicting filters. Based on the bit vector scheme and the aggregated bit vector scheme⁽⁴⁾, Baboescu and Varghese⁽⁵⁾ proposed a conflict detection algorithm called the scalable bit vector (SBV) algorithm. The time complexity of the SBV algorithm is $O(n^2)$, and the space complexity is also $O(n^2)$. Lu and Sahni⁽⁶⁾ proposed a plane-sweep algorithm that improves the performance of both time and space. The key idea behind the plane-sweep algorithm is to treat each filter as a rectangle with four line segments in the space, and then find all overlap regions (i.e., conflicts) by finding orthogonal line segment intersections. The plane-sweep algorithm runs in $O(n \log n + s)$ time and requires $O(n)$ space. Lai and Wang⁽⁷⁾ pointed out that the SBV algorithm cannot support filters with arbitrary ranges, and proposed several new algorithms based on the SBV algorithm. To achieve fast detection speed, their algorithm required nodes in a trie to store extra bit vectors called boundary bit vectors (BBV). The major drawback of these algorithms based on bit vectors is that a large amount of memory is required to store bit vectors. Lee *et al.*⁽⁸⁾ proposed a tuple-based conflict detection algorithm (TCDA). By exploiting the relationship between the tuple space and filter conflicts, TCDA is not only faster than other compared algorithms,

but also requires less memory space.

3. Proposed Conflict Detection Algorithm

Before we describe our proposed algorithm, we first define the possible relationships between two fields and two filters.

Definition 1: Given two fields f and g , if $f \cap g = \emptyset$, f and g are disjoint. Otherwise, f and g are overlapping, and the possible relationships between f and g include:

- (a) If $f \supset g$, the relationship between f and g is an enclosure.
- (b) If $f \subset g$, the relationship between f and g is a containment.
- (c) If $f = g$, the relationship between f and g is an equality.
- (d) If $f \cap g \neq f$ and $f \cap g \neq g$, the relationship between f and g is a cross-intersection.

Definition 2: Given two filters F and G , if there exists a field i , such that $F[i]$ and $G[i]$ are disjoint, F and G are disjoint. Otherwise, F and G are overlapping, and the possible relationships between F and G include:

- (a) For every field i , the relationship between $F[i]$ and $G[i]$ is an equality or an enclosure. In addition, there exists a field j , such that the relationship between $F[j]$ and $G[j]$ is an enclosure. Then, the relationship between F and G is an enclosure.
- (b) For every field i , the relationship between $F[i]$ and $G[i]$ is an equality or a containment. In addition, there exists a field j , such that the relationship between $F[j]$ and $G[j]$ is a containment. Then, the relationship between F and G is a containment.
- (c) For every field i , the relationship between $F[i]$ and $G[i]$ is an equality. Then, the relationship between F and G is an equality.
- (d) If F and G are not disjoint, and the relationship between them is not an enclosure, a containment, or an equality, then the relationship between F and G is a cross-intersection.

As mentioned before, the value of a field in a filter can be a prefix, a range of values, or an exact. Since a prefix can be represented as a range of values, we can treat each field as a range of values. Let f be a field. We use f_{min} and f_{max} to denote the smallest value and the largest value of f , respectively.

Theorem 1: Two fields f and g are overlapping if and only if $f_{min} \leq g_{max}$ and $g_{min} \leq f_{max}$.

Proof: This theorem can be proved directly using

Fig. 1. Five line segments.

Algorithm 1: Determine if two prefixes overlap.

Input: Two prefixes P and Q

Output: Boolean value indicating if P and Q overlap

```

// For an IPv4 address, the maximum length of a prefix is 32.
1  $P_{min} \leftarrow P \ll (32 - \text{length}(P));$ 
2  $P_{max} \leftarrow P_{min} + 2^{32 - \text{length}(P)} - 1;$ 
3  $Q_{min} \leftarrow Q \ll (32 - \text{length}(Q));$ 
4  $Q_{max} \leftarrow Q_{min} + 2^{32 - \text{length}(Q)} - 1;$ 
5 if ( $P_{min} \leq Q_{max}$ ) and ( $Q_{min} \leq P_{max}$ ) then
6   return true;
7 else
8   return false;
9 end

```

Fig. 2. Pseudo code to determine if two prefixes overlap.

Definition 1.

Fig. 1 shows five fields, and the range of values for each field is represented as a line segment. These line segments include $L1=[4, 8]$, $L2=[1, 2]$, $L3=[10, 11]$, $L4=[4, 5]$, and $L5=[8, 11]$. Theorem 1 can be used to determine if two fields are overlapping. For example, since $L1_{min} > L2_{max}$, $L1$ and $L2$ are not overlapping. On the contrary, $L1$ and $L4$ are overlapping since $L1_{min} \leq L4_{max}$ and $L4_{min} \leq L1_{max}$.

Theorem 2: If two fields f and g can be represented in prefix format, the relationship between f and g cannot be a cross-intersection.

Proof: If f and g are disjoint, we have proved this theorem. Otherwise, f and g are overlapping. Since f and g can be represented in prefix format, either f is a prefix of g , or g is a prefix of f . Otherwise, f and g are disjoint. Therefore, the relationship of f and g must be one of Definition (a)-(c).

We now describe our proposed conflict detection algorithm. Given a 2-D filter $F = (f[1], f[2])$, where both fields are represented in prefix format. The range of values represented by a prefix can be calculated as shown in lines 1-2 in Fig. 2. For example, if the maximum length of a prefix is 4 bits. Given a prefix $P = 11_b$, P_{min} is $P \ll 2$, which is 1100_b . P_{max} is $1100_b + 2^{4-2} - 1$, which is 1111_b . Given two prefixes P and Q , we can determine if P and Q are overlapping by first converting each of them to a range of values. Then, we can use Theorem 1 to determine if they are overlapping. Fig. 2 shows the detailed algorithm.

Once two prefixes can be determined if they are overlapping, we can determine the relationship between two filters. Fig. 3 shows the detailed algorithm. First,

Algorithm 2: Determine the relationship between two filters.

Input: Two filters F and G

Output: The relationship between F and G

```

1 if  $\text{NotOverlap}(F.\text{SrcIP}, G.\text{SrcIP})$  or  $\text{NotOverlap}(F.\text{DstIP}, G.\text{DstIP})$  then
2   return DISJOINT;
3 end
4 if  $\text{length}(F.\text{SrcIP}) < \text{length}(G.\text{SrcIP})$  then
5    $\text{SrcIPRelationship} \leftarrow \text{CONTAINMENT};$ 
6 else if  $\text{length}(F.\text{SrcIP}) > \text{length}(G.\text{SrcIP})$  then
7    $\text{SrcIPRelationship} \leftarrow \text{ENCLOSURE};$ 
8 else
9    $\text{SrcIPRelationship} \leftarrow \text{EQUAL};$ 
10 end
11 if  $\text{length}(F.\text{DstIP}) < \text{length}(G.\text{DstIP})$  then
12    $\text{DstIPRelationship} \leftarrow \text{CONTAINMENT};$ 
13 else if  $\text{length}(F.\text{DstIP}) > \text{length}(G.\text{DstIP})$  then
14    $\text{DstIPRelationship} \leftarrow \text{ENCLOSURE};$ 
15 else
16    $\text{DstIPRelationship} \leftarrow \text{EQUAL};$ 
17 end
18 if  $\text{SrcIPRelationship} = \text{EQUAL}$  and  $\text{DstIPRelationship} = \text{EQUAL}$  then
19   return EQUAL;
20 else if  $\text{SrcIPRelationship} \neq \text{ENCLOSURE}$  and  $\text{DstIPRelationship} \neq \text{ENCLOSURE}$  then
21   return CONTAINMENT;
22 else if  $\text{SrcIPRelationship} \neq \text{CONTAINMENT}$  and  $\text{DstIPRelationship} \neq \text{CONTAINMENT}$  then
23   return ENCLOSURE;
24 else
25   return CROSS-INTERSECTION;
26 end

```

Fig. 3. Pseudo code to determine the relationship between two filters.

according to Definition 2, if either the first fields (i.e., source IP address) or the second fields (destination IP address) in two filters are not overlapping, two filters are disjoint (lines 1-3). Then, the relationship of the source IP address field between two filters can be determined by the prefix lengths (lines 4-10), and so as the destination IP address field. Finally, the relationship between two filters can be determined by the relationships between two fields using Definition 2 (lines 18-26).

For example, the relationship between B and C in Table 1 can be determined as follows. Recall that the maximum length of each field is 4 bits. $B[1]_{min}$ is $0 \ll (4-1) = 0$, and $B[1]_{max}$ is $0 + 2^{4-1} - 1 = 7$. Similarly, we can obtain $C[1]_{min} = 0$ and $C[1]_{max} = 15$. Since $B[1]_{min} \leq C[1]_{max}$ and $C[1]_{min} \leq B[1]_{max}$, the first fields in B and C are overlapping. Similarly, we can find that the second fields in B and C are also overlapping. Since the length of $B[1]$ (i.e., 1) is larger than the length of $C[1]$ (i.e., 0), the relationship between the first fields of B and C is an enclosure. Similarly, the relationship between the second fields of B and C is a containment. Finally, we can determine that the relationship between B and C is a cross-intersection.

Using the algorithm shown in Fig. 3, all conflicts in a filter database can be detected by checking the relationship between every pair of filters in the filter database.

4. Experimental Results

This section compares our proposed algorithm with the SBV algorithm. All algorithms were implemented in C++,

Table 2. Statistics of filter databases.

Filter DB	16K		32K	
	Number of Filters	Number of Conflicts	Number of Filters	Number of Conflicts
ACL1	15,914	7,744	31,857	19,194
ACL2	15,088	1,624,941	29,853	5,977,111
ACL3	15,298	1,933,808	31,339	9,446,057
ACL4	15,059	993,585	31,324	5,440,082
ACL5	11,668	49	23,069	69
FW1	14,901	23,343,641	29,528	93,206,984
FW2	15,424	15,858,371	30,787	61,102,176
FW3	14,303	24,489,885	28,315	93,326,576
FW4	13,959	11,853,301	27,508	47,154,685
FW5	13,992	26,348,951	27,623	102,514,687
IPC1	15,432	1,646,482	31,712	7,757,509
IPC2	16,000	7,849,806	32,000	30,597,960

and benchmarked on a 2.1 GHz Intel Xeon PC. The tested filter databases were synthesized by ClassBench⁽⁹⁾. This tool includes 12 seed files derived from real filter sets to reflect the characteristics of filters of different applications, including three categories: the access control list (ACL), firewall (FW), and IP chain (IPC). For each seed file, we generate filter sets with 16K and 32K filters. Since the filters generated by ClassBench are 5-D, the number of filters for each file generated is slightly less than the specified number after removing duplicate filters by only considering source and destination address fields. Filter database statistics, including the number of filters and the number of conflicts, are shown in Table 2.

Figs. 4 and 5 show the average detection time for 16K and 32K filter databases, respectively. Compared with the SBV algorithm, our proposed algorithm can reduce the detection time by 30% to 40% for 16K-filter databases, and 37% to 56% for 32K-filter databases. One thing worth noting is that the parameter files exerts little impact on the detection time for our proposed algorithm. This is because our proposed algorithm detects conflicts based on whether two fields are overlapping using value comparison. On the contrary, the SBV algorithm has to traverse two tries and perform bitwise operations for bit vectors. Since the number of traversed nodes depends on the length of the prefix being processed, the detection time of the SBV algorithm is sensitive to the characteristics of parameter files.

As for the required storage, our proposed algorithm does not need to generate extra data structures for

Fig. 4. Average detection time for filter databases with 16K filters.

Fig. 5. Average detection time for filter databases with 32K filters.

accelerating the conflict detection process. For each field, our algorithm only needs two 32-bit integers to store the minimum and the maximum values of the range represented by a prefix. In addition, one extra byte is needed to store the length of the prefix. Therefore, our algorithm needs 18 bytes to store a filter. As for the SBV algorithm, one trie is required for each field. In addition, every node in a trie has to store one or two bit vectors. The length of a bit vector is the number of filters. As a result, the average storage size required by a filter for a 16K-filter database in the SBV algorithm varies between several KBs to tens KBs. Table 3 summarizes the average storage size required by a filter for the SBV algorithm and our proposed algorithm. As the number of filters increases, the SBV algorithm requires more memory space to store a filter, while our proposed algorithm does not.

Table 3. Required storage size (in bytes).

Filter DB	16K		32K	
	SBV	Our	SBV	Our
ACL1	27,849	18	136,863	18
ACL2	60,543	18	79,365	18
ACL3	2,958	18	19,021	18
ACL4	5,923	18	15,370	18
ACL5	3,502	18	5,286	18
FW1	75,710	18	137,590	18
FW2	60,415	18	109,803	18
FW3	65,588	18	120,356	18
FW4	65,108	18	111,390	18
FW5	6,608	18	122,037	18
IPC1	7,944	18	72,433	18
IPC2	9,766	18	183,789	18

5. Conclusions

Conflict detection is important for the correctness of packet classification. In this paper, we described our proposal for a conflict detection algorithm to resolve conflicts in two-dimensional packet filters. By analyzing the rules to determine the relationship between two prefixes and the relationship between two filters, our proposed algorithm does not need to generate extra data structures for accelerating the conflict detection process. Compared with the SBV algorithm, our algorithm can reduce the detection time by 30% to 56% for 16K-filter and 32K-filter databases. The space complexity of our algorithm is $O(n)$ and that of the SBV algorithm is $O(n^2)$, where n is the number of filters.

Acknowledgment

This work was supported in part by the High Speed Intelligent Communication (HSIC) Research Center of Chang Gung University, Taiwan, and by grants from the Ministry of Science and Technology of Taiwan (MOST-104-2221-E-182-005 and MOST-105-2221-E-182-069) and Chang Gung Memorial Hospital (BMRP 942).

References

- (1) H. Hu, G.-J. Ahn, and K. Kulkarni : “Detecting and Resolving Firewall Policy Anomalies”, IEEE Transactions Dependable Secure Computing, Vol. 9, No. 3, pp. 318-331, 2012.
- (2) A. Hari, S. Suri, and G. Parulkar : “Detecting and Resolving Packet Filter Conflicts”, in Proceedings of INFOCOM, pp. 1203-1212, 2000.
- (3) V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel : “Fast and Scalable Layer Four Switching”, in Proceedings of ACM SIGCOMM, pp. 191-202, 1998.
- (4) F. Baboescu, and G. Varghese : “Scalable Packet Classification”, in Proceedings of ACM SIGCOMM, pp. 199-210, 2001.
- (5) F. Baboescu, and G. Varghes : “Fast and Scalable Conflict Detection for Packet Classifier”, Journal of Computer Networks, Vol. 42, No. 6, pp. 717-735, 2003.
- (6) H. Lu, and S. Sahni : “Conflict Detection and Resolution in Two-dimensional Prefix Router Tables”, IEEE/ACM Transactions on Networking, Vol. 13, No. 6, pp. 1353-1363, 2005.
- (7) C.-Y. Lai, and P.-C. Wang : “Fast and Complete Conflict Detection for Packet Classifiers”, accepted and to appear in IEEE Systems Journal, 2016.
- (8) C.-L. Lee, G.-Y. Lin, and Y.-C. Chen : “An Efficient Conflict Detection Algorithm for Packet Filters”, IEICE Transactions Information Systems, Vol. 95, No. 2, pp. 472-479, 2012.
- (9) D. E. Taylor, and J. S. Turner : “Classbench: A Packet Classification Benchmark”, IEEE/ACM Transaction Networking, Vol. 15, No. 3, pp. 499-511, June 2007.