A Fast Bit-Vector-Based Conflict Detection Algorithm for Packet Classifiers

Chun-Liang Lee^{a,b,*}, Yu-Cheng Hsu^c, Yaw-Chung Chen^c, Hsiang-Chien Hsieh^a

^aDepartment of Computer Science and Information Engineering, School of Electrical and Computer Engineering, College of Engineering, Chang Gung University, Kwei-Shan, Tao-Yuan 33302, Taiwan

^bStroke Center and Department of Neurology, Chang Gung Memorial Hospital, Linkou, New Taipei 20401, Taiwan

^cDepartment of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan

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

Abstract

To support various network services, routers have to classify packets based on header fields and a set of filters. Each filter contains several fields to specify ranges of header values, and an associated action, which specifies how to handle the packet matching the filter. If two filters overlap and the actions defined in the filters conflict, it may lead to a false classification. In this paper, we propose a fast bit-vector-based algorithm for conflict detection. Experimental results using synthetic filter sets generated by ClassBench indicate that the proposed algorithm reduces the average detection time per filter by up to 77.9% compared to the scalable bit vector (SBV) algorithm. Furthermore, our algorithm does not consume significant extra memory and pre-computation time.

Keywords: packet classification, conflict detection, bit vectors.

1. Introduction

Packet classification enables routers to support various network services, such as firewalls, virtual private networks, and policy-based routing. For each incoming packet, a router has to check the head fields of the packet and determine if these fields match any filters in its database. A filter F with k fields is called a k -dimensional filter, denoted as $F = (f[1], f[2], \dots, f[k])$. Each field in a filter specifies a set of values, which can be represented as an exact value, a binary prefix, or a range. Possible fields in a filter include source and destination IP addresses, source and destination ports, and protocol type. A packet is said to match a filter if every field in the packet header matches the corresponding field defined in the filter. Each filter has an associated action that specifies how to handle the packets matching

Table 1. An example filter database with four filters.

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

this filter.

Problems may arise if a packet matches multiple filters with conflicting actions. For example, let us consider a two-dimensional filter database with 4 filters as shown in Table 1. Assume that the length of each packet header field is 4 bits. In addition, the action field defines whether the matching packets are allowed to pass through the router. A packet $P1 = (1000, 0000)$ matches filter D , and will be denied to pass through. However, another packet $P2 = (0000, 1010)$ 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.

To resolve conflicts, the conflicting filters can be reordered so that the filter with the expected action is stored in front of the other conflicting filters. When performing packet classification, the action of the first matching filter will be executed⁽¹⁾. Hari *et al.*⁽²⁾ pointed out that filter reordering cannot fully solve the problem caused by filter conflicts. They proposed the use of 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 a fast conflict detection algorithm for multi-dimensional packet filters. Our algorithm uses bit vectors and tries for conflict detection.

Table 2. An example filter database with five filters.

Filter	Field 1	Field 2	Group
$R0$	10^*	1^*	$G8$
$R1$	10^*	001^*	$G7$
$R2$	1^*	00^*	$G7$
$R3$	1^*	11^*	$G8$
$R4$	10^*	10^*	$G8$

By dividing filters into different groups, our proposed algorithm can reduce the amount of bits required to read, and thus can detect all conflicts in a filter database faster than other bit-vector-based algorithms.

The remainder of this paper is structured as follows. Section 2 briefly 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.

Among the algorithms mentioned above, the SBV algorithm can be used in multi-dimensional filters. Since this algorithm divides the overlapping-detection problem into k 1-D problem instances, it is possible to implement this algorithm using parallel computing to speed up the conflict detection. Thus, in this paper, we focus on bit-vector-based conflict detection algorithms.

3. Proposed Conflict Detection Algorithm

Since our proposed algorithm was motivated by the

Fig. 1. Two tries and bit vectors constructed using the filter database in Table 2.

SBV algorithm, we first describe how the SBV algorithm operates to detect all conflicts among filters. For each field, the SBV algorithm builds a trie. Each trie node associated with a valid prefix contains two bit vectors. The length of each bit vector is N bits, where N is the number of filters. The first bit vector ($BV1$) indicates the filters that exactly match the prefix associated with the prefix node. The second bit vector ($BV2$) shows the filters in a subtree rooted at the prefix node. Fig. 1 shows two tries constructed using the filter database shown in Table 2. Since there are five filters in the example filter database, the lengths of both $BV1$ and $BV2$ are 5 bits. Field 1 has only two distinct prefixes, and thus there are two valid nodes in trie $T1$. Filters $R0$, $R1$, and $R4$ have the same field 1 (i.e., 10^*), so the $BV1$ of node B in trie $T1$ is 11001. Assume that a filter $F = (10^*, 1^*)$ needs to be checked for conflicts with all filters in the database. Each field of F is used to traverse the corresponding trie. It computes the union of all nodes' $BV1$ from the root node to the longest matching prefix node, then computes the union of the longest matching prefix node's $BV2$. Finally, the intersection of each field's bit vector contains the conflict information. If the k th bit is set in the final bit vector, F conflicts with the k th filter in the database. For the first field of F , i.e., 10^* , we compute the union of the following three bit vectors in $T1$: $T1(A).BV1$, $T1(B).BV1$, and $T1(B).BV2$. Thus, we have $(00110) \text{ OR } (11001) \text{ OR } (11001) = (11111)$. For the second field of F , i.e., 1^* , we compute the union of the following two bit

Group	(Field1, Field2)
G0	(*, *)
G1	(*, 0*)
G2	(*, 1*)
G3	(0*, *)
G4	(0*, 0*)
G5	(0*, 1*)
G6	(1*, *)
G7	(1*, 0*)
G8	(1*, 1*)

Fig. 2. Mapping between group number and prefix pair.

vectors in $T2$: $T2(C).BV1$ and $T2(C).BV2$. Thus, we have $(10000) \text{ OR } (10011) = (10011)$. Finally, we compute the intersection of the results obtained from two tries. Thus, we have $(11111) \text{ AND } (10011) = (10011)$. The final bit vector indicates that F conflicts with three filters, i.e., $R0$, $R3$ and $R4$.

The SBV algorithm can find all conflicts with $O(kNW)$ time complexity, where N is the number of filters, W is the longest prefix length, and k is the number of fields. In addition, the space complexity is $O(kN^2)$. Since the SBV algorithm spends most time to read and process bit vectors, our key idea to improve the SBV algorithm is to reduce the amount of bits required to read when bit vectors are accessed, which, in turn, reduces the time required to perform the union and the intersection operations.

Given two filters $F = (f[1], f[2], \dots, f[k])$ and $G = (g[1], g[2], \dots, g[k])$, if there exists a field i , such that $F[i]$ and $G[i]$ are disjoint, F and G are disjoint. In other words, F does not conflict with G . Based on this observation, we divide filters into several groups based on prefix fields, which are generally used to denote source and destination IP addresses. Fig. 2 shows the mapping between group numbers and prefix field pairs. We can have a conflict matrix to indicate whether two groups are in conflict or not, as shown in Fig. 3. More specifically, if $M[i][j]$ is 1, filters in group i conflict with filters in group j . On the contrary, if $M[i][j]$ is 0, filters in group i are disjoint with filters in group j , which means conflict detection can be skipped for two filters belonging to group i and group j , respectively.

According to the conflict matrix, some bits can be skipped when reading bit vectors. For example, if a filter $R5 = (10^*, 11^*)$ wants to find all conflicting filters in Table 2, the SBV algorithm must read all the entire bit vector when traversing a trie node in Fig. 1. However, according to

	G0	G1	G2	G3	G4	G5	G6	G7	G8
G0	1	1	1	1	1	1	1	1	1
G1	1	1	0	1	1	0	1	1	0
G2	1	0	1	1	0	1	1	0	1
G3	1	1	1	1	1	1	0	0	0
G4	1	1	0	1	1	0	0	0	0
G5	1	0	1	1	0	1	0	0	0
G6	1	1	1	0	0	0	1	1	1
G7	1	1	0	0	0	0	1	1	0
G8	1	0	1	0	0	0	1	0	1

Fig. 3. Conflict Matrix M .

the conflict matrix in Fig. 3, we know that group $G7$ is disjoint with group $G8$. Since filter $R5$ belongs to group $G8$, it is disjoint with those filters which are classified as group $G7$. Therefore, filter $R5$ is disjoint with $R1$ and $R2$, it only needs to read the 1st, 4th, and 5th bit values.

Suppose that a filter F needs to be checked for conflicts with all filters in the database. A trivial design is that we can first determine which group that F belongs to, say G_i . Then, instead of reading the entire bit vectors during traversing the tries, we can only read those bits that denote filters in groups conflicting with G_i . For example, if filter $R5$ performs the SBV algorithm, the final bit vector is $(00110 \text{ OR } 11001 \text{ OR } 11001) \text{ AND } (10000 \text{ OR } 00010 \text{ OR } 00010) = 10010$. With the help of the conflict matrix, only the 1st, 4th, and 5th bits in bit vectors should be read. Thus, the final bit vector is $(010 \text{ OR } 101 \text{ OR } 101) \text{ AND } (100 \text{ OR } 010 \text{ OR } 010) = 110$, which represents the same result as in the SBV algorithm. It is obvious that our proposed algorithm can reduce the amount of bits that needs to be read and processed. However, the trivial design is inefficient. For each bit vector, the bits required to read may not be continuous, and thus the time spent on reading bit vectors may not be shorter than that in the SBV algorithm.

The above trivial method is not feasible because filters belonging to the same group are not adjacent to each other, so we cannot use a range to record the scope of each group. To solve this problem, we can pre-sort filters by group, and the preprocessing time complexity is $O(N \log N)$. In our proposed algorithm, the order of filters within the same group type doesn't affect the bit vector algorithm. It only needs to cluster the filters, and thus the time complexity is $O(N)$. By sorting all filters in the database by group number, we can use two variables to record the beginning and the end locations of a group.

Table 3. Example filter database after sorting by group.

Filter	Field 1	Field 2	Group
R_0	10*	001*	G_7
R_1	1*	00*	G_7
R_2	10*	1*	G_8
R_3	1*	11*	G_8
R_4	10*	10*	G_8

(a) T_1 : trie associated with field 1.

(b) T_2 : trie associated with field 2.

Fig. 4. Two tries and bit vectors constructed using the filter database in Table 3.

Our proposed algorithm retrieves each filter's group, and then obtains each group's reading range. After getting the reading range of each group, the algorithm refers the range to get the required bits. Compared with the trivial design, setting the reading range can reduce the amount of memory access. Table 3 shows the filter database in Table 2 after sorting by group, and the corresponding tries are built as in Fig. 4. Again, suppose that filter R_5 wants to find all conflicting filters in Table 3. According to the conflict matrix we know that G_8 (R_5 's group) is disjoint with G_7 (R_0 and R_1 's group). It only needs to read the 3rd, 4th, and 5th bits in the bit vector. We can see that through sorting filters by group, bits required to access in each bit vector are continuous in this example.

One may find that even after sorting filters by group, bits required to access in each bit vector may still be divided into multiple segments. To reduce the maximum number of segments, we rearrange all groups so that elements with value of 1 in each row of the conflict matrix can be as continuous as possible. Figs. 5 and 6 show the

Group	(Field1, Field2)
G_0	(*, *)
G_1	(*, 0*)
G_2	(0*, 0*)
G_3	(0*, *)
G_4	(0*, 1*)
G_5	(*, 1*)
G_6	(1*, 1*)
G_7	(1*, *)
G_8	(1*, 0*)

Fig. 5. Mapping between group number and prefix pair.

	G_0	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
G_0	1	1	1	1	1	1	1	1	1
G_1	1	1	1	1	0	0	0	1	1
G_2	1	1	1	1	0	0	0	0	0
G_3	1	1	1	1	1	1	0	0	0
G_4	1	0	0	1	1	1	0	0	0
G_5	1	0	0	1	1	1	1	1	0
G_6	1	0	0	0	0	1	1	1	0
G_7	1	1	0	0	0	1	1	1	1
G_8	1	1	0	0	0	0	0	1	1

Fig. 6. Conflict matrix after rearranging groups.

new mapping between group number and prefix pair, and the corresponding conflict matrix, respectively. Under this new conflict matrix, each group has at most two non-disjoint segments. When a filter wants to find all conflicting filters, our proposed algorithm needs at most 4 extra memory accesses and 36 variables for storing the beginning and the end locations of all 9 groups.

4. Experimental Results and Discussion

In this section, we present the experimental results of our proposed algorithm, compared with the SBV algorithm. Both algorithms were implemented in C++ and benchmarked on a 3.2 GHz Intel Core i5-4570 PC. The operating system is Ubuntu 14.04.1. The tested filter sets were synthesized by ClassBench⁽⁷⁾, which is a public available tool and contains 12 seed files to generate three types of filter sets: access control list (ACL), firewall (FW), and IP chain (IPC). For each seed file, we generate one filter set with 30,000 filters. Since our proposed algorithm and the SBV algorithm use the same tries for conflict detection, the time required to create tries was excluded in measuring conflict detection time. All data represent

Fig. 7. Average detection time for filter sets with 30,000 filters.

averages from 100 simulations.

Fig. 7 shows the average detection time for each filter with different filter sets. Compared with the SBV algorithm, our proposed algorithm takes less time to detect all conflicts for all filter sets. In filter set acl5, our proposed algorithm achieves the largest reduction in detection time by 77.9%. Both algorithms have the shortest detection time in filter set acl5. This is because the number of arbitrary ranges in the source port and the destination port fields is relatively small compared with other filter sets. To build tries, a range field has to first convert to multiple prefixes⁽³⁾, which means multiple new filters are generated. As a result, in filter set acl5, fewer new filters are generated due to range-to-prefix conversion. In filter set ipc2, our proposed algorithm achieves the smallest reduction in detection time by 22.6%. Overall, compared with the SBV algorithm, our proposed algorithm reduces the detection time by an average of 44.7%.

5. Conclusions

In this paper, we proposed a fast conflict detection algorithm based on bit vectors. Our proposed algorithm reduces the amount of bits required to access bit vectors, and thus can provide faster detection speed than the SBV algorithm. Experimental results show that our proposed algorithm can reduce the detection time by up to 77.9% for 30K-filter databases. More importantly, our proposed algorithm does not incur noticeable pre-computation time and extra memory space.

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 107-2221-E-182-029 and 108-2221-E-182-010) 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) 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.