A FAST AND SCALABLE IPV6 PACKET CLASSIFICATION

Xiaoju Zhou[1], Xiaohong Huang[1], Qiong Sun[1], Wei Yang[1], Yan Ma[1,2]

[1]Research Institute of Networking Technology, Beijing University of Posts and Telecommunications, Beijing
[2] Beijing Key Laboratory of Intelligent Telecommunications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876
{zhouxj, huangxh, sunq, yangw@buptnet.edu.cn}, mayan@bupt.edu.cn

Abstract

In order to support more value-added services and a larger space for IP addresses, the Internet requries to classify IPv6 packets into flows for specific requirements. High performance IPv6 packet classification (PC) algorithms are therefore in high demand. This paper proposes a new IPv6 PC algorithm based on flow label (FL), called FLIN (Flow Label Initial Noticed). FLIN is fast and scalable with small memory requirements. This paper also suggests an IPv6 PC design method aimed at the features of FL. The experiment results show that FLIN achieves a great improvement in optimizing IPv6 PC performance, especially in the case that a large number of different rules have equivalent FL values at the same time.

Keywords:IPv6, flow label, independent set, packet classification.

1 Introduction

As IPv6 with a FL (flow label) in its header has been increasingly applied all over the world, well performed IPv6 PCs become crucial for ISP to support value-added services in future, such as firewall packet filtering, traffic billing and quality of service (QoS) applications including integrated services (IntServ) and differentiated services (DiffServ) .

Performance metrics for a PC algorithm include speed (measured by times of memory access), memory storage requirements, update speed, and scalability to both the number of the rules and the number of fields included in the rules, among possible others.

In this paper, there are 3 contributions as follows. Firstly, a new IPv6 PC algorithm called FLIN (Flow Label Initial Noticed) is proposed. Besides FLIN, there is only one published IPv6 PC algorithm considered FL [1]. Compared with it, our experimental result shows that FLIN achieves a great improvement in optimizing IPv6 PC performance, especially in the case that equivalent FL values are shared by a large number of rules. Secondly, FLIN will efficiently classify IPv6 packets with small memory consuming. FLIN optimizes a famous PC algorithm PC-SAM （Packet Classification Consuming Small Amount of Memory）[2] while extracting its essence. Its performance is competitive to the best IPv6 PC algorithm at present. Thirdly, an IPv6 PC design method aimed at the features of FL is proposed. It profits explore the advantages of FL in PC while covering probable problems.

The rest of this paper is organized as follows. In Section 2, we highlight performances of some existing PC algorithms in both IPv4 and IPv6. An IPv6 PC design method aimed at the features of FL is proposed in Section 3. In Section 4, some basic definitions are introduced. Next the IPv6 PC problem is defined and the decision tree for FL, the optimization of PC-SAM and the searching and updating details of FLIN are described. In Section 5 and 6, complexities of the algorithm and results of experimental studies are presented. Concluding remarks are made in the last Section.

2 Related work

Surveys on PC algorithms were given in [3]. In this section, we briefly introduce the performance for some of the algorithms.

Algorithms based on decision tree, such as FIS trees[4], EGT[5], are utilized frequently in IPv4 PC while their speed will decline obviously in IPv6 because the depth of decision tree will sharply increase to slow down search.

On the contrast, parallel algorithms have less effect by IPv6, P^2C[6], PC-SAM showed flexible scalability as the length of IP address is raised to 128 bits, but none of them considered FL as a field in IPv6 PC rules. Although FL was defined in RFC2460 [7] as a 20-bit field in IPv6 header as a flow identifier, only one published PC algorithm is specifically designed based on IPv6 FL, Source IP Address (SA) and Destination IP Address (DA)

978-1-4244-4900-2/09/$25.00 ©2009 IEEE

until now. It used hash to search SA and DA. However, hash may suffer from two troubles although itself would be fast in theory. One is that too many hash collisions would severely decrease the search speed. The other is that a big amount of memory will be consumed by a large hash table.

3 Design Methods of IPv6 PC Algorithm Aimed at the FL features

As defined in RFC 3697[8], FL acts as a flow identifier associated with SA and DA to replace traditional 5-field flow identifier (SA, DA, source port, destination port and protocol number). Two important characteristics of FL are defined. One is a FL value should be created randomly and the other is lifecycle of a FL is no more than 120 seconds.

Compared with traditional attributes in 5-field flow identifier, FL is a new field in PC. The differences of features between FL and the other fields require a specific design aimed at FL.

For a rule, a value in FL field is usually provided as one single point. Since the label value for each flow is random, flows which are included in a range of FL field is independent with each other and a limit range to FL in rules seems meaningless. However, for traditional 5 fields, the field in a rule may consist by a single value or a certain range, for example one IPv6 SA with /128 in a rule can be handling all packets from a computer or with /64 to control a whole network.

With lifecycle which is no longer than 120 seconds, IPv6 PC based on FL should be more flexible in updating, or it is probable that a huge number of filters would be expired in a short time and routers hardly bare rules change severely.What's more, 20-bit FL is much shorter than the other IPv6 fields, 128-bit SA and DA, even shorter than 32-bit IPv4 address. So it is possible to arouse some excellent and classical IPv4 PCs to search FL in IPv6.

Combined with the search and update advantage of decision tree, IPv6 PC can use hierarchical structure to construct a FL tree firstly and a data structure in parallel PC based on each FL value.

4 FLIN Algorithm

4.1 Definitions

In our proposed algorithm FLIN, PC-SAM is extracted to construct the basic data structure. PC-SAM performs well in all aspects of the performance measurement in both IPv4 and IPv6 PC. Both PC-SAM and FLIN are based on the concept of the independent sets and global maximum independent sets. The formal definition of them will be introduced in this subsection.

4.1.1 Independent Sets

Definition: Let $C = \{R_1,\ldots, R_N\}$, where R_i (i=1,…,N) is a rule. For index k, $1 \le k \le d$, two rules, $R_i = (F_1^i,\ldots, F_d^i)$ and $R_j = (F_1^j,\ldots, F_d^j)$, are called independent along dimension k, if $F_k^i \cap F_k^j = \varnothing$. For a set S of rules, if any two rules in it are independent along dimension k, we call S an independent set along dimension k, which is denoted as an I_k-set or simply an I-set if no confusion arises.

The number of the elements in a set A is referred to as the size of the set A denoted by $|A|$. For a classifier, consider its all possible independent sets along dimension k. An independent set with the largest size is defined as a maximum independent set along dimension k in C. A classifier C may have more than one maximum independent set. An independent set with the largest size among all independent sets along all dimensions is called a global maximum independent set.

Based on the concept of independent sets, PC-SAM is developed. The general procedure of implementing PC-SAM is described as follows. Given a classifier $C = \{R_1,\ldots, R_N\}$, we try to separate from it a global maximum independent set. Then, from the set of the remaining rules (treated as a new classifier), we separate a global maximum independent set with respect to the new classifier, and continue the process until the set of the remaining rules is empty. More formally, assume that $I[1]$ is a global maximum independent set separated from C. Let $C_1 = C - I[1]$, we then find a global maximum independent set $I[2]$ with respect to C_1. Let $C_2 = C_1 - I[2]$, we continue the process until the resulting classifier is empty. Thus, C is divided into a collection of independent sets $\{I[1],I[2],\ldots,I[S]\}$. Next, we group the independent sets $\{I[1],I[2],\ldots,I[S]\}$ according to the dimension. Two independent sets belong to the same group if they are independent along the same dimension. Let $G_1,G_2,\ldots,G_d$ be the resulting groups, where is the number of dimensions for rules in the classifier C. G_i (i=1,…,d) is the group of independent sets which are independent along dimension . Note that a group may be empty. We search in all nonempty groups of $G_1,G_2,\ldots,G_d$, respectively. The query result can be obtained by comparing all the matches to find the rule with the highest priority. In the following, we will discuss how to find a maximum independent set and how to create a data structure for searching G_i (i=1,…,d).

4.1.2 Basic Data Structure for a Group of Independent Sets

In one dimension d, Let $G = \{I[1],I[2],\ldots,I[S]\}$ be a group of independent sets obtained in the last section, where $I[k]$ (k=1,…,S) is an independent set and S is the number of independent sets in G. Let

$I[k] = \{ r_1^k, \ldots, r_{n_k}^k \}$, where r_i^k (i=1,…,n_k) is a rule and $n_k = |I[k]|$. For each $I[k]$ (k=1,…,S), we mark the begin point and end point of each rule r_{ki} (i = 1,…, n_k) with b_{ki} and e_{ki}. Let start point set $B_k = \{ b_1^k, \ldots, b_{n_k}^k \}$ for each $I[k]$. All points in B_k are different, since rules in $I[k]$ are independent. We assume that all points in B_k are sorted in increasing order. Let X_{ki} stands for the range $[b_{ki}, e_{ki}]$.

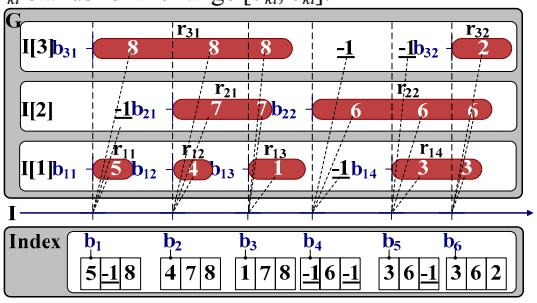

Figure 1. The data structure in PC-SAM

Then, we merge the points in all S sets $B_1,\ldots, B_S$ into a master set $I=\{b_1,\ldots,b_M\}$, $M=|I|$. Apparently, the number of points in I satisfies $|I| \leq \sum |B_i|$ (i=1,…,S), since two different sets B_i and B_j may contain equivalent points. Take Fig.1 as an example. The number in the rectangle (rule) is the index of the corresponding rule. Next, for each b_i (i=1,…,M) in I, we add "virtual index" -1 along b_i in I[k] when b_i is not included in the range X_{ki}.

4.2 Algorithm Description

In this section, FLIN with hierarchical structure will be divided into two parts: a FL structure is designed based on decision tree in the first part. And in the next part, a new remark and reordering method will be introduced in creating data structures to identify a filter which would optimize the performance of searching speed in multiple dimensions. The process of search and update would be presented later.

4.2.1 FL Decision Tree

In FLIN, a red-black tree is constructed based on FLs from rules. FLIN based on a red-black trees will pay a small lookup cost because the tree is not perfectly balanced, but, in return, they get fast, bounded insertion and deletion operations. Using a red-black tree, FLIN can, thus, be indicated in situations where nodes come and go frequently.

Every FL value in rules can be found in this tree as a node. Based on each node, one data structure will be built in PC-SAM way. For example, to all rules with the FL value 0xABCDE, their SAs or DAs will be assembled into a data structure identified by 0xABCDE.

4.2.2 Remark and Reordering

To optimize the search performance of PC-SAM, FLIN firstly remarks all -1 virtual indices considering the order they appeared in an array. As shown in Fig, 2, for example in b_4 , the 1st index is -1, so we also remark it with -1. And the 3rd index is -1, it will be remarked as -3. Remarked indices will help to update a new rule, we will describe it later.

Then reorders rule indices in each b_i by how big their end points are. For example, e_{21} is the end point of rule r_{21}, in b_3, $e_{21}>e_{31}$ and $e_{31}>e_{13}$, so we have b_3->1(r_{13} index), 7(r_{21} index), 8(r_{31} index) will be reordered into b_3-> 1, 8, 7. If a virtual index is in front of a non-virtual index in b_i, replace it with the rest following indices bigger than it. After reordering rules as below, the range of one rule will cover ranges of any other rules with behind it in b_i, because all rules have the same start point b_i.

Reordering rules is an effective method to decrease the access times in memory for search. For arranged by FLIN order, indices in b_i will not completely be used for matching all rules like PC-SAM does when searching a point, unless all indices before the last one match the point in $[b_i, b_{i+1})$. For example, if a point P in $[b_i, b_{i+1})$ doesn't match the 2nd rule, all rules from 3rd rule to Nth rule in b_i will not need to match P. So FLIN will not keep matching the rest rules.

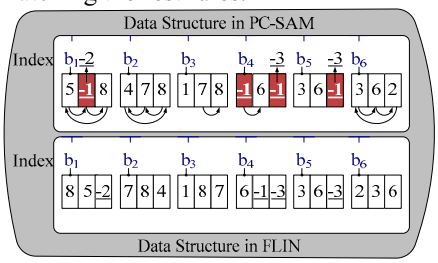

Figure 2. Remark and Reordering

4.2.3 Search

To classify a packet, there are two steps to search a classification for one IPv6 packet. First, FL value in this packet will find a node in the decision tree. Second, SA and DA fields in this packet will be searched in a specific table by one-dimensional process in serial or in parallel and then pick a final result.

The value in the relevant field of the packet is used as the key. An unique flow identifier can be separated into Key F (for FL), Key S (for SA) and Key D (for DA).

In a FL decision tree, Key F can be used to find a node N from the tree. The search process starts from the root node, if root node value $V > F$, then go left. Or if $V<F$, go right. Only if $V=F$, go to data structure V for a one-dimensional search.

In one-dimensional search process, the key (S or D) can be used to find the start point with the value that is the largest one among all which are smaller than or equal to the key. Fetch the first index

(ignore the virtual index) by the start point to get the rule fields by indexing into the classifier array. Then compare the key with both SA and DA fields of the rule. If there is a match, get the priority of this rule and keep searching. If not, take one with the highest priority from all matching rules. Then, results in two dimensions will be compared and a rule with higher priority will be found.

4.2.4 Update

There are two kinds of updates: preprocessing and incremental update. The first one is to create a new tree and the data structure from scratch whenever there is a change while the second one is to modify the tree and its table data structure whenever there is a change. Usually incremental update is faster, however, it may make the table data structure non optimal. We discuss both of them as follows:

4.2.4.1 Incremental Update

For adding a rule, there are three steps in FLIN. The first step is to find or modify a node by the FL value in a rule. The second step is to find a group containing a set of which the new rule is independent. The last step is to make modification of the relevant data structure of the group found. We will illustrate the second and third steps by the following example and then describe them in detail.

As shown in Figure 3, 1st step is to follow the red-black tree update process. In 2nd step, we need to search both SA and DA groups. How to choose a group first can be decided by the real application. In searching a group, we extract the start and end points of the relevant field of the new rule as b_x and e_x. We will try to find a smallest b_j to make $b_j \geq e_x$. If b_j cannot be found, take the last start point as b_{j-1}.There are two main cases when a new rule is coming. Case one: There is an existing $b_i=b_x$. In this case, check all virtual indices in each b_p ($p=i,...,j-1$). If there is a common virtual index $-V$ in all b_p ($p=i,...,j-1$), replace all $-V$ indices with the index of the new rule. Or if there is no common index in all b_p ($p=i,...,j-1$), we need to add a new $I[S+1]$ in G for the new rule and then add the index of the new rule into each b_p ($p=i,...,j-1$) as well as the virtual index $-(S+1)$ into the rest b_i. Finally, it is necessary to do reordering in each b_p ($p=i,...,j-1$).Case two: There is no existing start point equal to b_x. We need to add a new b_x between b_i and b_{i+1} to make $b_x > b_i$ and $b_{i+1} > b_x$. If either b_i or b_{i+1} cannot be found, just ignore it. Rebuild all indices for old rules in both b_i and b_x, then take the new rule as in case one.

To delete a rule, we first find the group in a certain data structure that contains the rule and then check if the deletion of the rule results in the deletion of the corresponding start point in the multi-way range search tree. In order to do that, we just need to compare the begin point of the rule with the begin points of the rules in the same entry. If there is a match, we do not need to remove a start point; otherwise, the corresponding start point is removed. The rest of the work is a modification of entries of indices which is the reverse procedure for adding a rule. If a data structure connected with a FL node is null, we can delete the node in red-black tree way.

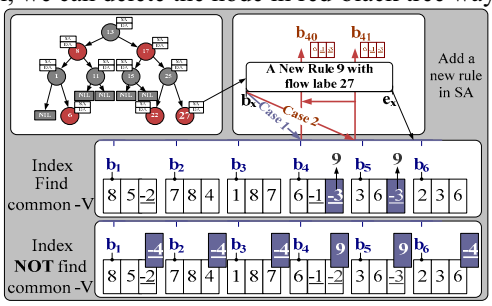

Figure 3. The process of adding a new rule

4.2.4.2 Recreate Data Structure

After many times of incremental update, the data structure may become non-optimal. After the deletion of rules, the part left may be no longer maximum independent sets in this case. We would form a new maximum independent set. In order to avoid this situation, we may recreate the data structure. How often we need to recreate the data structure can be decided by the real application.

5 Complexities of the Algorithm

Globally, an array stores the classifier. The size of the array is linear to the number of rules in the classifier. In addition, a multi-way range search tree in each flow label node will be constructed corresponding to at most two groups of independent sets in SA and DA. Assume that the group G_i consists of S_i independent sets and contains N_i rules. Then the memory storage requirement for the multi-way range search tree corresponding to group G_i is linear to the number of N_i. The number of indices stored in the leaves is at most $N_i * S_i$, since there are at most N_i leaves and each leaf stores S_i indices. Set M as the number of different FL values, the total number of indices is $\sum_{i=1}^{M} S_i * N_i$.

To add all together, memory storage requirement for FLIN is upper bounded by $O(M+S*N)$, where N is the number of rules and S is the number of total independent sets.

The memory access times consist of two parts: One is the times in searching the red-black tree (in FL field) and the multi-way range search trees. The other is the times in fetching the indices plus S accesses to the rules. The performance of FLIN can be adequately explored when there are a large number of different rules with same FL values which determinates the number of independent sets S in each FL. The smaller the number S is, the less memory access times and storage are required.

The time require to perform red-black of FL is $O(\log N_0)$, and PC-SAM of (SA, DA) is $O(I)$, N is the number of different FL values and I is the number of independent sets association with the FL so the total is $O(\log N+I)$.

6 Experimental Results

One experimental scenario with 40000 rules as follows: There are 50 nodes (different FL values) in red-black tree and 800 rules on average for each node. As to a table of FL F, we initiates exactly 800 rules. The memory costs in F will be considered both index cost and rule cost. Firstly, after using FLIN 800 rules were divided into 9 I-sets: three of them are $I1$-sets for field SA and 6 sets are $I2$-sets for field DA. Based on these two groups of I-sets, two one-dimensional range search trees [9] are generated. Based on these 800 rules, the range search tree corresponding to field SA has 283 start points; start points in field DA is 258. In field SA, each start point in $I1$-sets aims to three rule indices and the whole index arrays is no more than 849, which is 3 times of the number of its start points. Since the total number of rules is 800, each index requires at least 10 bits for $2^{10}>800$. And the memory will take no more than 8490 bits/1062 bytes. We can also calculate the memory cost will be less than 1932 bytes in field DA by the same argument. Secondly, we assume each rule needs 288 bits: 128*2 bits for field SA and DA, 7*2 bits for the length of the source and destination prefix, and 18 bits for specifying priority. Therefore, 800 rules take 28800 bytes of memory in all. The experimental study shows that about 32 K bytes are totally needed by each table.

The number of memory accesses for the range search tree is three each. The number of memory accesses for the indices depends on the memory width. We assume that 32 bits memory width is used. Then, three of memory accesses are needed for fetching 9 (3 in SA and 6 in DA) indices and 81 memory access are need for fetching rules for each 288-bit rule need 9 times of memory access. In total 90 memory access are needed. This is the worst case plain implementation. Therefore the total storage in this scenario is 1600K. As obviously shown in Figure 4, FLIN for a big rule set takes less storage for each rule.

Method	Rules	Storage	
		Total	Per rule
This paper	40000	1600K	0.04K
This paper	80000	2950K	0.04K
Other paper[1]	7000	490K	0.07K
Other paper[1]	5000	300K	0.06K

Figure 4. Experimental results

7 Conclusions

We have developed a fast and scalable IPv6 PC algorithm FLIN based on FL features. Extensive experiments have been done, which shows the good performance of the proposed algorithm.
With well performance and the advantages of FL, FLIN can be a candidate among the possible bests for future high performance IPv6 PC.

Acknowledgements

Project 60772111 supported by National Natural Science Foundation of China, Specialized Research Fund for the Doctoral Program of Higher Education(200800130014)

References

[1] Eric C.K. Poh and Hong Tat Ewe "IPv6 Packet Classification based on Flow Label, Source and Destination Addreses ". The Third International Conference on Information Technology and Application(ICITA'05)

[2] Xuehong Sun, Sartaj K. Sahni, Fellow,IEEE, and Yiqiang Q.Zhao, Member,IEEE. "Packet Classification Consuming Small Amount of Memory" IEEE/ACM transactions on networking, VOL 13, NO.5, OCTOBER 2005

[3] P.Gupta and N. McKeown, "Algorithms for packet classification," IEEE Network, vol, 15,no. 2, pp. 24-32, Mar./Apr. 2001.

[4] A.Feldman and S.Muthukrishnan, "Tradeoffs for packet classification", in Proc. IEEE INFOCOM, vol. 3, Mar.2000, pp.1193-2002.

[5] Florin Baboescu, Sumeet Singh and George Varghese, "Packet Classification for Core Routers: Is there an alternative to CAMs?", IEEE INFOCOM 2003, vol 1, pp. 53-63, 30 March-3 April 2003.

[6] Jan van Lunteren and Ton Engbersen, "Fast and Scalable Packet Classification", in IEEE Journal on selected areas in communications,vol. 21, NO.4, May 2003

[7] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC2460, Dec. 1998.

[8] J. Rajahalme, A. Conta, B. Carpenter and S. Deering,"IPv6 Flow Label Specification", RFC3697, Mar. 2004.

[9] X.Sun and Y. Q. Zhao, "An On-Chip IP Address Lookup Algorithm." Carleton Univ., Ottawa, Canada, Tech. Rep., 2003.