PCHA: A Fast Packet Classification Algorithm For IPv6 Based On Hash And AVL Tree

Yu-yan Zhang
School of Network Education,
Beijing University of Posts and Telecommunications
Beijing, P. R. China
zhangyuyan007@163.com

Xing-xing Chen
School of Network Education,
Beijing University of Posts and Telecommunications
Beijing, P. R. China
chenxx@bupt.edu.cn

Xu Zhang, Member, IEEE
School of Cyberspace Security,
National Engineering Laboratory for Mobile Network Security,
Beijing University of Posts and Telecommunications
Beijing, P. R. China
selina_zhangx@bupt.edu.cn

***Abstract*—As the core infrastructure of cloud data operation, exchange and storage, data centerneeds to ensure its security and reliability, which are the important prerequisites for the development of cloud computing. Due to various illegal accesses, attacks, viruses and other security threats, it is necessary to protect the boundary of cloud data center through security gateway. Since the traffic growing up to gigabyte level, the secure gateway must ensure high transmission efficiency and different network services to support the cloud services. In addition, data center is gradually evolving from IPv4 to IPv6 due to excessive consumption of IP addresses. Packet classification algorithm,which can divide packets into different specific streams, is very important for QoS, real-time data stream application and firewall. Therefore, it is necessary to design a high performance IPv6 packet classification algorithm suitable for security gateway.AsIPv6 has a128-bitIP address and a different packet structure compared with IPv4, the traditional IPv4 packet classification algorithm is not suitable properly for IPv6 situations. This paper proposes a fast packet classification algorithm for IPv6 - PCHA (packet classification based on hash andAdelson-Velsky-Landis Tree). It adopts the three flow classification fields of source IPaddress(SA), destination IPaddress(DA) and flow label(FL) in the IPv6 packet defined by RFC3697 to implement fast three-tuple matching of IPv6 packet.It is through hash matching of variable length IPv6 address and tree matching of shorter flow label. Analysis and testing show that the algorithm has a time complexity close to *O(1)* in the acceptable range of space complexity, which meets the requirements of fast classification of IPv6 packetsand can adapt well to the changes in the size of rule sets, supporting fast preprocessing of rule sets. Our algorithm supports the storage of 500,000 3-tuple rules on the gateway device and can maintain 75% of the performance of throughput for small packets of 78 bytes.**

Keywords-IPv6; hash; AVL tree; packet classification;high performance

This work was supported in part by the National Key Research and Development Program under Grant 2018YFB1800702, the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2012BAH45B01.

I. INTRODUCTION

As the infrastructure of cloud computing, cloud data center provides server resource allocation, bandwidth allocation, business support capability, traffic protection and cleaning capability. However, cloud data center also faces many security risks, such as service availability threat, user information leakage, denial of service attack threat, etc. Therefore, it is necessary to deploy security gateway at the boundary of cloud data center.It would be benefitto provide rich security protection for virtual network, including resource access control, data privacy protection, virtualization security and so on. In addition, the cloud data center, as the infrastructure architecture for hosting applications, is gradually shifting from IPv4 to IPv6 due to the high consumption of IP addresses such as virtualization and heavy use of containers. In the process of IPv6 for user services, the upper layer of cloud data center applications needs to complete the end-to-end adaptation of IPv6 from layer 3 to layer 7. This requires the security gateway to have the ability to partition the IPv6 data flow to support the processing requirements of the upper application for different services.

In IPv4, traffic flowcan only be divided by the type of service field, which is difficult to meet the current complex network service needs. In IPv6, RFC3697[1] defines a way of dividing flows.A data stream is determined by the 20-bit stream label field in the packet header, togetherwith the source address and destination address. This packet classification method can divide IPv6 packets belonging to different hosts into corresponding data streams. IPv4 networks typically use the five-tuple fields of the data header which consists of source IP address(SA), destination IP address(DA), source port, destination port, and transport layer protocol number for stream partitioning. Since the five fields are scattered in the network layer and the transport layer, the processing of flows needs to cross two layers to complete, which may lead to layer conflicts.Layer conflicts inevitably reduce the performance of flow processing. The definition of flow label(FL) is intended to simplify stream-based packet classification. Using a single shorter field reduces the overhead of extracting a combination of port numbers and protocol numbers from different headers.

In consideration of the algorithm data structure, this paper selects two data structures based on the characteristics of IPv6 address and flow label. For IPv6 addresses, the 128-bit length provides a larger address space, but it also expands the range of address matching. Common data structures such as linked lists, Trie trees, etc. cannot meet the requirements of fast matching, while the hash table is a constant-level data search structure based on key-value pairs, which is based on a certain space consumption in exchange for lower time complexity. The method to solve the hash conflict can be applied to the rapid matching of IPv6 addresses in this scenario. For the flow label, the length of 20-bits defines the search range, and it has shorter survival time. For this field, a data structure that can be easily accessed to the IP address hash table and supports quick search, insertion, and deletion isconsidered.Therefore, Binary tree is a more appropriate choice. Considering the high-speed matching scenario in this paper, the matching times of packet search are far greater than the number of regular rule additions and deletions. This paper chose AVL tree which is a strictly balanced binary tree, as the data structure of the flow label. The self-balancing characteristic brings time costs in the pre-processing period, but it will pay back in the later matching period.

The main contributions of this paper are as follows:

1. PCHA, a 3-tuple packet classification algorithm based on hashandAdelson-Velsky-Landis (AVL) Tree is proposed. The PCHA algorithmselects the flow label, source IP and destination IP in the IPv6 packet as the matching fields and selects the appropriate data structure for storage and lookup according to the characteristics of each field. Experimental results show that PCHA algorithm is suitable for fast classification of IPv6 packets.

2. The algorithm supports large rule sets. We hope to implement the application of the algorithm on the gateway by supporting the large-scale rule set. If so, the algorithm can solve the problem of rule set expansion caused by the complexity of gateway aggregation traffic and the huge address space of IPv6.

3. Experiments are based on real equipment and data by using physical gateway machines and professional flow generator. Different from some experiments using simulation or simulated data, this paper adopts hardware gateway which can be applied to the actual network and aSpirent flow generator to carry out experimental verification. The experimental results can reflect the actual application effect of the algorithm and ensure the practicability of the algorithm.

The structure of this paper is as follows: section II investigates and analyzes related packet classification algorithms, including IPv4 and IPv6, and describes the design process of packet classification algorithms for flow labels. In section III, PCHA algorithm is introduced, including rule set establishment process and packet matching process, and hash conflict processing is described in detail. In addition, this chapter also analyzes the complexity of the algorithm. Section IV is the experimental verification results of the algorithm. The conclusion is in section V.

II. Related Work

Although this paper studies the IPv6based 3-tuple packet classification algorithm, the existing IPv4 and IPv6 packet classification algorithm still has certain research significance. A basic introduction to packet classification algorithm is given in [2].

There are many classic packet classification algorithms in IPv4 that strike a balance between simple storage structures and fast matching performance. The simplest algorithm is to perform a linear search for all fields through a rule library of size n. This approach has both *O(n)* memory and time complexity and is not suitable for large rule sets. Decision-tree based algorithms are often considered in IPv4 packet classification algorithm design, such as FIS trees[3], EGT[4], Hicuts[5], etc. Hicuts algorithm is a classical package classification algorithm. Its main idea is to divide rules in multiple dimensions and construct classification decision tree. However,if it is applied to IPv6, the depth of the decision tree increases dramatically, resulting in slower search speed and lower performance.CutSplit[6] integrates equal-sized cutting and equal-dense cutting to optimize decision trees,but its performanceonthroughputstillneeds tobetestedinreal network scenario.

There are performance-based package classification algorithms. Hash based package classification algorithms usually have good search time performance, such as Bloom Filter[7], but the problem is that they are prone to hash conflicts and are not suitable for dealing with wildcard or prefix field matching. The dimension decomposition algorithm based on RFC[8] transforms the multidimensional problem of packet into the matching problem of single dimension by splitting the multidimensional information of packet. It belongs to the packet classification algorithm with better matching performance, but the problem is that memory explosion will occur with the increase of rule amount.NeuroCuts[9] is an algorithmic solution that applies deep reinforcement learning to generate efficient decision trees, with the capability to incorporate and improve on existing heuristics.

There are also packet classification algorithms based on special hardware support. The hardware-based three-state content-addressable memory algorithm[10] usually has the support of special hardware such as TCAM, and the matching speed is fast. However, the corresponding hardware is usuallyexpensive, consumes more power, requires more storage space and is difficult to apply to large rule sets. By taking advantage of the inherent parallelism provided by FPGA and other ideal functions, a two-dimensional multi-pipeline architecture based on decision tree[11] was proposed for the next generation of packet classification. The search of individual fields and the search of the final combination of these algorithms require off-chip memory access. The problem with these package classification algorithms based on hardware support is their high cost and lack of universality.

The packet classification of IPv6 is not the same as that of IPv4. Based on the definition in [1], the flow label field consists of 20 bits and is used primarily by the source node to mark packets that belong to a class. FL acts as the stream identifier associated with SA and DA in place of the traditional five-tuple stream identifiers (SA, DA, source port, destination port, and protocol number). The values for the flow label range from 0 to FFFFF(hexadecimal). Each source node that uses a flow label should have the following abilities: 1) Assign random values to the flow label in the packet so that all the flow labels can be used as hash keys. 2) For unrelated transmission connections and application data streams, different flow label values should be selected. 3) Itcan provide the means to set the lifetime of flow label over 120 seconds for application and transmission protocol.

Based on the packet characteristics of IPv6, [12] proposed an IPv6 3-tuple packet classification algorithm based on H-Trie structure. The algorithm establishes the H-Trie tree structure through the flow label field, and then stores the source IP and destination IP after hashing. The H-Trie structure is highly scalable and deployable. The problem, however, is that Trie-based lookups couldn'tperform consistently in the worst case. In addition, in [12], the analysis and solutions to hash conflicts that may occur when hashing is used for IP address calculation are simplified, and actual data analysis is lacking.

Flowlabels have a default lifetime of no more than 120 seconds, which requires that IPv6 packet classification algorithms based on flowlabels be adaptable to flexible updates. The length of the flowlabel is only 20 bits, much shorter than the 128-bit source and destination addresses in IPv6, and even shorter than the 32-bit addresses in IPv4. Therefore, tree structure is an ideal lookup structure for flow tags. In addition, for IPv6 addresses, its length is 128 bits, which is four times the length of IPv4 addresses. Hashing is the fastest way to match a fast lookup in an address space of 2^128. Because the hash algorithm will bring the hash collision problem, how to solve the hash conflict is also one of the research contents of this paper.

In general, the algorithms mentioned above aim to minimize the search scope and reduce the number of searches for the rule sets. For IPv6, under the premise of supporting large rule sets, the algorithm should be designed in a gentle and reasonable way in terms of search time and space.

III. Design Of Pcha

A. Problem Definition

The packet classification rule set$F$ contains rulesof amount n, $F=\{f_1,f_2... , f_n\}$. Each rule f contains three fields $f=\{d_1,d_2,d_3\}$, where d_1 and d_2 represent SA and DA with prefix lengths and d_3 represent FL, the flow label. For a certain rule, for example, *f = {3001:9de:6f2e:d85d::/64, 6768:664e:7ec2:8000::/64,0xA23B1}*is a rule that has a source address sectionof *3001:9de:6f2e:d85d::/64*, destination address section of *6768:664e:7ec2:8000::/64*, and flow label value of*0xA23B1*. For a packet *P*, if the fields are the same to the rule fields, the packet is considered to match the rule. Since *SA* and *DA* differ greatly from *FL* in length, and all of them are clearly defined, the (*SA,DA*) address pair in *f* can be defined as the flow direction, i.e.a flow from *SA* to *DA*.FLcan be defined as flow type. Thus, the problem of packet classification is transformed into the problem of how to classify the flow of *P*by flow direction and flow type.

B. Construction of Hash Table

In a rule set, rules for IPv6 source and destination addresses are usually given in the form of prefixes. Therefore, when matching IP fields in IPv6 packets, prefix matching is required. According to RFC2460, IPv6 addresses consist of a 64-bit prefix and a 64-bit interface ID. In the IPv6 address block already allocated, address allocation agencies such as APNIC will assign the IPv6 address block of /32 mask to each ISP, and then the ISP will assign the /48 prefix from the /32 prefix to each IPv6 site as the site prefix. The address space of the /48 prefix can be further divided into the /64 subnet. Therefore, this paper takes the three most commonly used prefix lengths /32, /48, /64 in consideration as the possible prefix lengths in the IPv6 rule set.

According to the flow direction defined above, the IP address pair (*SA,DA*) is processed together during the processing of the IP address in rule *f*.The*Key* of the hash function is *(SA,DA)*, and the *Value* of the hash result is calculated as:

$$Value = Hash(Key) \qquad (1)$$

Hash is the hash function. *Value* is the index of the storage location of rule *f* in the hash table. The *SA* and *DA* of rule *f* are stored in the location of *Value*, and the *FL* is inserted into the AVL tree.

C. Resolution of Hash Conflicts

Since the hashing algorithm may calculate data far beyond the calculated result range, there will be different data calculated getting the same value, which is also called hash conflict. Generally, the solutions to hash conflicts can be divided into minimizing the probability of hash conflicts and remediation after the occurrence of hash conflicts. Here we use the quadratic hash method and the chain address method to solve the hash conflict problem. The specific steps are as follows:

1. Firstly, the hash value is calculated by using the hash function *Hash1*:

$$Value1 = Hash1(Key) \qquad (2)$$

Chain traversal is performed at *Value1* in the hash table to check if there is a hash conflict between different (*SA,DA*) address pairs. If there is no hash conflict, rule *f* is inserted at *Value1* of the hash table, where the node storing rule *f* becomes the header of the list at *Value1*.

2. If there is a hash conflict caused by different (*SA,DA*) address pairs, then the hash value is calculated again using the hash function *Hash2*:

$$Value2 = Hash2(Key) \qquad (3)$$

Chain traversal is performed at *Value2* position of the hash table to check if there are hash conflicts caused by different (*SA,DA*) address pairs. If there is no hash conflict, rule *f* is inserted at *Value2* of the hash table, where the node storing rule *f* becomes the header of the list at *Value2*.

3. If there is still a hash conflict, insert rule *f* at the end of the list at *Value2*.

In the above process, rules with the same hash value are grouped into the same subset *S*, and each subset becomes a bucket. The elements in each bucket are connected by a single linked list, and the head nodes of each linked list are stored in the hash table. The structure of the hash table that stores a subset of rules is shown below. The hash table is essentially an array of linked lists, with *Value* as an array index and a sequential list as an array element.

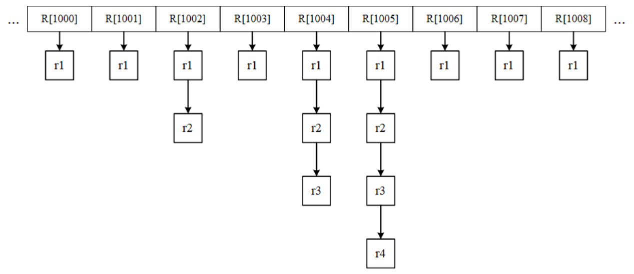

Figure 1. Structure of hash table

D. Construction of AVL Tree

For each rule *f* in the rule library, after inserting (*SA,DA*) into the hash table, insert *FL* into the AVL tree at the corresponding position in the hash table. The main steps are as follows:

1. Set the root node as the comparison node.

2. Set the comparison node as the parent node. Compare node-FL with input-FL. If node-FL > input-FL, the left child of the comparison node is set as the comparison node. If node-FL < input-FL, the right child of the comparison node is set as the comparison node. Otherwise the rule isrepeated, and the insertion ends.

3. Continue with step 2 until the comparison node is empty.

4. If node-FL > input-FL of the parent node, insert input-FL into the left child node of the parent node; If the parent node's node-FL < input-FL, input-FL is inserted into the right child of the parent node.

5. Rotate the AVL tree according to the balance factor of the parent node.

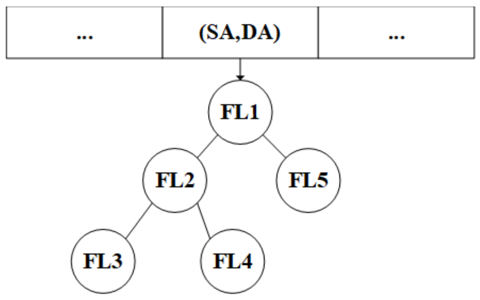

Figure 2. Structure of AVL tree on a Hashtable node

E. Matching Process

For a packet, the packet classification process can be divided into two steps. The first step is to find the corresponding storage node in the hash table with the flow direction based on (*SA,DA*) as the first-level input. The second step, with *FL* as the secondary input, performs a binary search in the AVL tree structure of the found storage node.

According to *Fig. 3*, when a packet arrives, the three-tuple fields in its header (SA_p, DA_p, FL_p) are first extracted. For (SA_p, DA_p), after prefix extraction is processed with the combination of three prefix lengths of /32,/48,/64, hash calculation is performed to obtain the index of the rule. In the worst case, nine times of prefix extracts may be required. According to the quadratic hash method and chain address method adopted in this paper, when the value obtained from the first hash calculation traverses the list of nodes without finding the corresponding matching rule, the second hash calculation will be carried out and the list of nodes will be continued to match.The single prefix processing will carry out at most two hash calculations. When the corresponding (*SA,DA*) rule is found in the linked list, it comes to the FL lookup phase. Compare the *node-FL* of the current root node with the FL_p, starting with the AVL root node to which the linked list node in hash table points. If *node-FL*>FL_p, enter the left child node; if *node-FL*<FL_p, enter the right child node. If *node-FL* = FL_p, the match succeeds.

Algorithm 1 PCHA

```
Input: SA,DA,FL
Output: action
for x,y in prefixlist do
    SA=SA & x;
    DA=DA & y;
    Value=Hash1(SA,DA)
    if $Hashtable[Value] \neq NULL$ then
        list=Hashtable[Value]
    else
        Value=Hash2(SA,DA)
        if $Hashtable[Value] \neq NULL$ then
            list=Hashtable[Value]
        else
            rule doesn't exist
            return default action
        end if
    end if
end for
member=list.header
while $member \neq NULL$ do
    if member.SA==SA&&member.DA==DA then
        break
    end if
    member=member.next
end while
if member==NULL then
    rule doesn't exist
    return default action
end if
node=m.root
while $node \neq NULL$ do
    if node.FL>FL then
        node=node.left
    else if node.FL<FL then
        node=node.right
    else
        rule match
        return node.action
    end if
end while
rule doesn't exist
return default action
```

Figure 3. ThePCHA algorithm

The PCHA algorithm proposed in this paper does not focus on the processing actions in the rule set. For the output processing actions, they can represent routing policies, ACL policies, QoS service policies, etc.

F. Complexity Analysis

The complexity of the algorithm will be analyzed from two perspectives: time complexity and space complexity.

1. Time complexity

The overall time complexity of the algorithm can be divided into the hash lookup complexity for (*SA,DA*) and the AVL tree lookup complexity for *FL*.

The time complexity of the hash algorithm is *O(1)*. In PCHA, due to the resolution of hash conflicts and the processing of different prefix lengths, the search time complexity of (*SA,DA*) in the worst case is O(*(x+y)*9*),where *x* and *y* are the maximum length of the list of hash table nodes in the case of quadratic hash.

Because of its inherent balancing property, AVL tree has $O(log_2N)$ time complexity of insert, search and delete,where *N* is the number of nodes in an AVL tree.

Therefore, the overall time complexity of the algorithm is $O((x+y)*9+log_2N)$. Since the probability of hashing conflicts in the case of quadratic hashing is small, the time complexity of the hash part is close to a constant, and the average time complexity is $O(log_2N)$.

2. Space complexity

The overall space complexity of the algorithm can also be divided into the space complexity of hash table and the space complexity of AVL tree.

The space complexity of the hash table is approximately *O(m)*, where *m* is the number of rule entries in the hash table.

The space complexity of AVL tree is *O(n)*, where *n* is the number of nodes per AVL tree.

The overall space complexity of the algorithm is *O(m*n)*. In the experimental test based onC language, the establishment of a hash table of 1024*1024 occupies a fixed space of 8MB, and the size of each item in the hash table is 96bytes, including the size of a single node of AVL tree of 24bytes. Assuming the algorithm memory consumptionis about 53776Kbytes in the case of 500,000 rule sets without hash conflicts. For a gateway devicewithmemoryof 64G, the algorithm's memory consumption is about 0.08%, and memory requests are acceptable.Asthe test code includes auxiliary data structures other than the basic storage structure, the test results are a little bit higher than the theoretical results.

IV. EXPERIMENTS AND RESULTS

A. Experimental Environment

To verify the performance of the proposed algorithm, Hicutsalgorithm[5] and PCHT algorithm[10] were selected for experimental comparison with PCHA under the same conditions. Throughput tests were conducted on the three algorithms using f-stack, a high-performance open source network framework based on Data Plane Development Kit(DPDK). The operating system used in the test environment isRed hat Linux Enterprise Server7.1, the CPU is Intel Xeon E5-2690 v4 processor, and the memory is 64G. F-stack basic performance tests have been conducted on gateway devices in advance, with a net throughput of 4.28Gbps for 78-bytes packets and 20Gbps for 1500-bytes packets.

The throughput of gateway was tested by Spirent flow generator. The basic process of the test is to randomly generate different numbers of rules, and then compare the performance of preprocessing time, throughput and memory consumption of the rule set when the number of rules changes constantly. The size of the sending packet is 1500 bytes. The experimental data were averaged over 10 tests. Before the matching algorithm test, a small rule set of 1000~5000 rules and a large rule set of 100000~500000 rules were generated. Each rule was composed of source address (with mask), destination address (with mask) and flow label, where IP address and flow label were randomly generated, and the mask was randomly assigned in the combination of 32/48/64. The rules are loaded into memory when the f-stack is started.

The performance of the algorithm will be analyzed from the following aspects:

1. Preprocessing time of rule set

The time to preprocess the rule set, which reflects the time consumed by the algorithm to establish the storage structure, is related to the performance of rule dynamic insertion.

2. Memory consumption

The memory space consumed by the algorithm runtime includes the storage space occupied by the rule set and the data structure storage space established by the algorithm for lookup.

3. Time complexity/throughput

The search time of the rule is directly related to the time complexity of the algorithm, which reflects the processing speed of the algorithm. Since the overall processing time of small packets is less than that of large packets, the processing abilityof small packets can directly reflect the time performance of the algorithm. The higher the throughput when sending small packets, the better the performance of rule matching is.

4. Resolution of hashing conflict

When hashing is adopted for matching, the lower the probability of hashing conflict, the higher the efficiency of hashing. If the probability of hash conflict is high, hash conflict should be solved by chain address method, which inevitably leads to the increase of time complexity. Therefore, it is one of the important factors to solve hash conflict in a reasonable way.

B. *Time Comparison of Rule Set Establishment*

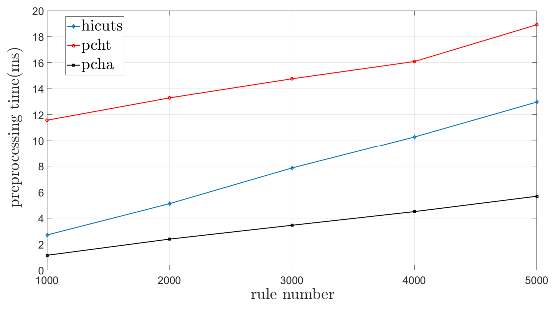

Figure 4. Comparison of preprocessing time with small rule sets

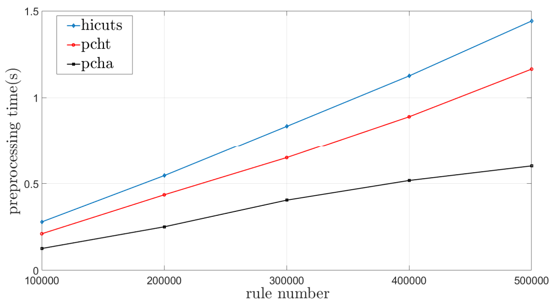

Figure 5. Comparison of preprocessing time with large rule sets

As can be seen from *Fig. 4*and *Fig. 5*, the performance of PCHA algorithm on preprocessing time of rule set improves with the increase of rule set. When the rule number N=5000, the preprocessing time of PCHA algorithm was 56% less than that of Hicuts algorithm and 70% less than that of PCHT algorithm. When the rule number N= 500,000, the preprocessing time of PCHA algorithm was 58% less thanHicuts algorithm and 48% less than PCHT algorithm. PCHA algorithm has good time performance when preprocessing rule sets of different sizes.

C. *Comparison of memory consumption*

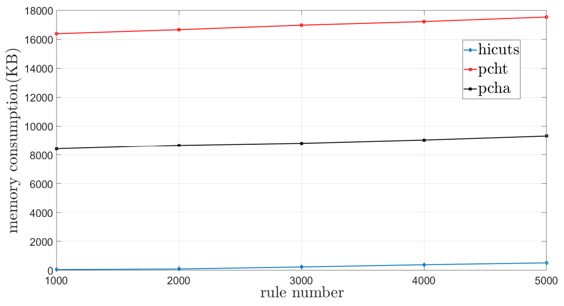

Figure 6. Comparison of memory consumption with small rule sets

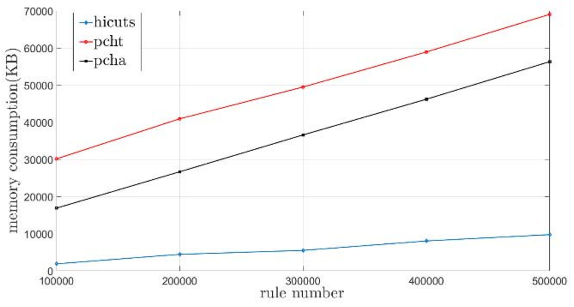

Figure 7. Comparison of memory consumption with large rule sets

As can be seen from *Fig. 6* and *Fig. 7*, both PCHA algorithm and PCHT algorithm occupy more memory than Hicuts algorithm due to the use of hash tables. In small rule sets, PCHA algorithms consume about 50% of the memory of PCHT. When the rule set reaches N= 500,000, the PCHA algorithm occupies 18% less space than the PCHT algorithm. In general, although PCHA algorithm increases memory consumption due to hashing based features, it has certain storage space advantages over PCHT in all levels of rule sets.

D. *Throughput comparison*

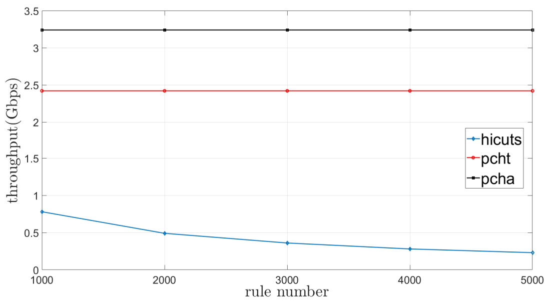

Figure 8. Comparison of throughput(78-bytes packet) with small rule sets

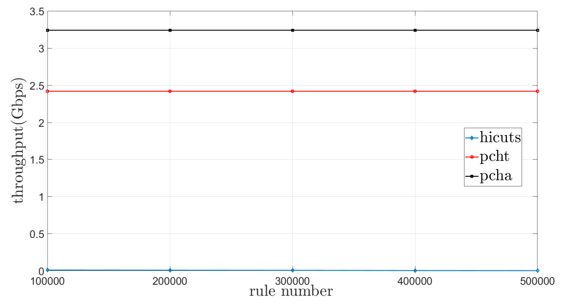

Figure 9. Comparison of throughput(78-bytes packet) with large rule sets

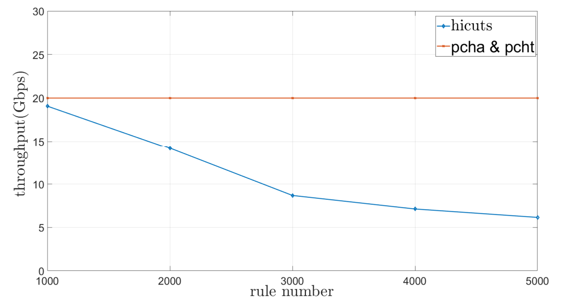

Figure 10. Comparison of throughput(1500-bytes packet) with small rule sets

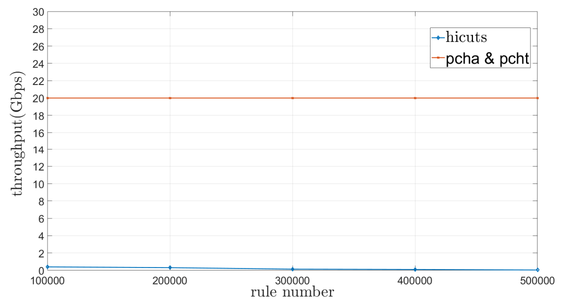

Figure 11. Comparison of throughput(1500-bytes packet) with large rule sets

As can be seen from *Fig. 8* and *Fig. 9*, Hicuts algorithm shows a significant decline in throughput as the number of rules increases. Accordingto*Fig. 8*,When N=5000, the throughput of a 78-byte packet flow is 0.23Gbps, a decrease of 29% compared with the throughput of N=1000.The throughput of N>100000 drops below 10Mbps. Above all indicate that Hicuts algorithm is no longer applicable to large-scale rule sets. With the increase of rule set size, the PCHA and PCHT algorithm's 78-bytes packet flow throughput stabilized at 3.24Gbps and 2.42Gbps, respectively. The overall performance of the PCHA algorithm is stable, maintaining the performance of 75% of the net gateway raw throughput of 78-bytes packet, and achieving the throughput of 20Gbps of 1500-bytes packet according to *Fig. 10* and *Fig. 11*.

E. *Situation of Hash Conflict*

TABLE I. HASH COLLISION STATISTICS

Rule number	Numbers of hash collision	Ratio of hash collision	Longest length of chain node
1000~5000	0	0%	1
100000	275	0.28%	3
200000	2042	1.02%	4
300000	6597	2.20%	5
400000	14584	3.65%	5
500000	26671	5.33%	6

In this paper, the quadratic hash method and chain address method are used in the hashing calculation of PCHA algorithm. To verify whether hash conflicts are resolved, the statistics of hash conflicts are carried out under different scales of rule sets. As shown in TABLE Ⅰ, there are no hash collisions under the small-scale rule set. However, as the rule set increases, the free space in the hash table decreases, and the proportion of hash conflicts increases. When the rule number reaches 500,000, the hash collision ratio is 5.33%, and in the case of the chain address method, each linked list extended due to the conflict has no more than 6 nodes. It meansthat in the case of hash matching, two hash calculations and matching of 12 linked list nodes are required in the worst case, and the overall time performance is still close to *O(1)*.

V. CONCLUSION

To solve the problem of IPv6 packet classification, this paper proposes a high performance IPv6 packet classification algorithm suitable for IPv6 network environment. The PCHA algorithm is based on the hash table for quick matching lookup, while under 500,000 rule sets the memory consumption of this algorithm is 55MB. It is acceptable for devices with large memory space such as securitygateway. At the same time, the algorithm has low time complexity and is still at constant level in the worst case. It can maintain 75% of the net throughput for 78-bytespackets andsupports the throughput of 20Gbps for the packet size of 1500-bytes. In general, this algorithm meets the requirements of fast classification of IPv6 packets and can well adapt to rule set size changes.

ACKNOWLEDGMENT

Wewould like to record our appreciationto Dr. Dongbin Wang from School of Cyberspace Security, Beijing University of Posts and Telecommunications forsupportandassistance.

REFERENCES

[1] J. Rajahalme, A. Conta, B. Carpenter, S. Deering, "IPv6 Flow Label Specification", RFC3697.

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

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

[4] 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.

[5] LAKSHMAN T V,.STILIADIS D. High Speed Policy Packet Forwarding Using Efficient Multidimensional Range Matching[J].ACM Computer Communication Review,1998,28(4):203-214.

[6] W. Li, X. Li, H. Li and G. Xie, "CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 2645-2653.

[7] Gahyun Park, Minseok Kwon, “An enhanced bloom filter for longest prefix matching,” 2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS), pp.1-6, 2013.

[8] SINGH S,BABOESCU F,VARGHESE G,et al.Packet Classification Using Multi-dimentional Cuttings[C]//Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications,2003.Karlsruhe:ACMSIGCOMM,2003:213-224.

[9] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM ’19). Association for Computing Machinery, New York, NY, USA, 256–269. DOI:https://doi.org/10.1145/3341302.3342221.

[10] EATHERTON W.Hardware based Internet protocol prefix lookups [D].St.Louis: Washington University,1999.

[11] W. Jiang and V. K. Prasanna, "Sequence-preserving parallel IP lookup using multiple SRAM-based pipelines", J. Parallel Distrib. Comput., vol. 69, no. 9, pp. 778-789, 2009.

[12] 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), 2005.

[13] GUPTA P, MCKEOWN N. Packet Classification on Multiple Fields [J]. ACM SIGCOMM Computer Communication Review, 1999,29(14):147-160.

[14] WALDVOGEL M,VARGHESE G,TURNER J, et al. Scalables high speed IP routing lookups[J]. ACM SIGCOMM Computer Communication Review, 1997,27(4): 25-26.

[15] EATHERTON W, VARGHESE G, DITTIA Z. Tree bitmap: Hardware/software IP lookups with incremental updates[J]. ACM SIGCOMM Computer Communication Reviwe,2004,34(2): 97-122.

[16] LI Y K,PAO D. Address lookup algorithms for IPv6[J]. IEEE Proceedings of Communication,2006,153(6): 909-918.

[17] HUSTON G. Analyzing the Internet’s BGP routing table[J]. The Internet Protocol Journal,2001,4(1):2-15.

[18] Pang Lihui, Jiang feng. Reaserch on High Perfomence Rule Matching Algorithm in IPV6 Networks[J]. Computer Science, 2017,44(3): 158-162.

[19] Eric C.K. Poh, Hong Tat Ewe, “IPv6 Packet Classification based on Flow Label, Source and Destination Addresses[J].Third International Conference on Information Technology and Applications, 2005: 1504-1508.

[20] Zhen Xu, Jun Sun, Jun Zhang. A Novel Hash-based Packet Classification Algorithm[C], 5th International Conference on Information Communications & Signal Processing,2005.

[21] Xiaoju Zhou, Xia hong Huang, Qiong Sun, Wei Yang, Yan Ma. A fast and scalable IPv6 packet classification[C]. IEEE International Conference on Network Infrastructure and Digital Content, 2009:275-279.

[22] Uday Trivedi, Mohan Lal Jangir. EQC16: An optimized packet classification algorithm for large rule-sets[J],.International Conference on Advances in Computing, Communications and Informatics, 2014: 112-119.