A kind of packet classification algorithm based on compress, partition and index *

Tian Liqin[1] Chen Fuming[2] Lin Chuang[3]
[1]Computer Science and Technology of Department, University of Science and Technology Beijing 100083
Email: tianliqin@tsinghua.org.cn
[2]China University of Geosciences (Beijing) Beijing 100083
[3]Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China

Abstract:
In this paper a new fast scalable multi-dimension packet classification algorithm is given. This algorithm converts rules, by compress operation, partition operation and index operation, into two kinds of storage data structure: compress-partition-rule-table and index-list-tables. Compress operation is like RFC algorithm [Tian Liqin, Lin Chuang. Study and application of packet classification. Journal of computer research and development, 2003,6: 765-6775], but in order to increase the speed of the packet classification, the technique is used only in first step. Partition operation is like regional partition algorithm [Yan Tian-xin, Wang Yong-gong, Shi Jiang-tao, Dai Xue-long. Optimized implementation of regional partition algorithm for packet classification, Journal of China institute of communications, 2004,6:80-88], but it partitions a compressed-rule into several 16-bit fields and needn't partition it into only 1 or 0 matched rules. Index operation, which creates index-list-tables and has 64k rows for each partition field, can greatly speed up the classification. Moreover, These methods can make the algorithm support ten thousands multi-dimension rules, and have low expected storage complexity. It not only can be implemented by the software but also can be implement by the hardware. Compared with existing classification algorithms, the algorithm is better than those existing classification algorithms when ten thousands multi-dimension rules is concluded in the rule base, so it can be used in the real router when it deals with large packet classification rules.

Keywords: *Packet Classification algorithm, design and Implementation, Compress, Partition, Index*

1 Introduction

Packet classification is widely used in router, firewall, intrusion detection system and so on. Packet classification must be forwarded at wire rate, this means routers must forward minimum packet (little than 64 bytes) immediately and this is very important in various network application. The hypostasis of packet classification is its fast speed, but we also need consider other aspects of the packet classification such as the number of the rules and the storage needed. It is difficult to develop a fast and multi-dimensions packet classification algorithm, which can deal with a large number of the rules. Packet classification has already been a new bottleneck of router. It isn't suitable to only simply use these existing common fast search algorithms. We must find out new skill to deal with great rules and storage space needed in the packet classification. We should research relationship between the classification speed and characteristic of classification rules to

* This work is supported by the National Natural Science Foundation of China (No. 90412012,60429202, 60432030 and 90104002), NSFC and RGC (No. 60218003), the National Grand Fundamental Research 973 Program of China (No. 2003CB314804), and Intel IXA University Research Plan.

find efficient packet classification methods.

2 Compress Partition and Index Algorithm

2.1 The main idea of algorithm

The best characteristic in this algorithm is we convert the original rule set into two storage data structure: one is compress partition rules table and the other is index lists tables, which can decrease the requirement of storage space, improve the speed of classification, and also can support ten thousands of rules. The detail step of the algorithm is as follows:

1) **Delete redundant rules** and **sort rules** ascendingly by full matched rules

Research shows that 8% rules are redundant in classifier, if packet classification algorithm can identify and delete these redundant rules, it can improve the capability of packet classification. The detailed algorithm sees [1]. Then sort rules ascendingly by full matched rules, in other words, the aim of the sort is finding the most matched rule firstly when the rules is found in order, the detailed algorithm also sees [1].

2) Create compress-partition-rule-table

This algorithm don't store original rule set, but store compress-partition-rule-table which is created from the original rule set, the aim is decreasing storage space, speeding up packet classification.

It must deal with **compress operation** and **partition operation** when create compress partition rules table from original rule set. The idea of **compress operation** is like RFC algorithm's first step [1], the aim of **compress operation** is to compress width of classification region, but we don't use heuristic algorithm. We only use stationary compress algorithm, which can simple the classification operation, by analysizing characteristic of fields and distributing regulation of the rules. It can decrease storage space and speed up classification. The idea of **partition operation** like regional partition algorithm [11], but there are some different between them, one is our algorithm need partition the compressed rules into several fields with 16 bits width, the other is our algorithm don't need partition the compressed rules into only 1 or 0 matched rules but into several field matched rules as long as it can be read out from memory at a time. From the example in the following section we can see that the suitable rule's number is less than 7 not 1. The detailed algorithm also sees the example in the following section.

3) Create **index-list-tables:** We create P **index-list-tables** with P partition fields of compress-partition-rule-table. Then we can only lookup index-list-table with the value of partition field to find partition field-matched rules, this can speed up packet classification. Partition field has 16 bits, so index-list-table has 64k(65536) rows (2^{16}=64k), so index-list-table also can be named as 64k table. The reason of partition field has 16 bits is if we partition it into 8 bits (2^8=256), then the lookup efficiency is too low, if we partition it into 24 bits (2^{24}=8M), then the requirement of storage space is too large, moreover, when partition it into other bits that don't suit the hardware characteristic of computer to read and write. **Index-list-tables**' structure is: the first column is the matched rules number of partition field, others are pointer, which point to the position of the rules in the compress-partition-rule-table, and meanwhile they are stored by the order of rules. This can greatly speed up the lookup the partition field matched rules. Detail process can be seen from the figure 1 and author also gives the whole mapping graph at the same time in figure 2.

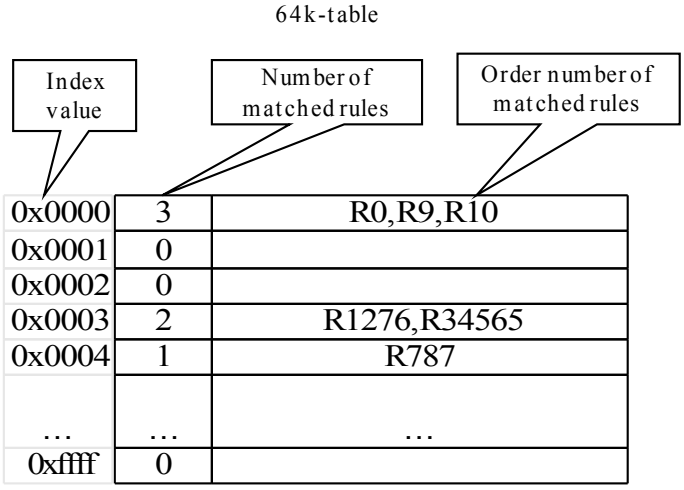


Figure 1. Structure sketch map of index-list-table for a partition field

*In this figure, R0 is rule0, R9 is rule9, R10 is rule 10, these are all the order-number of partition field matched rules in compress-partition-rule-table.

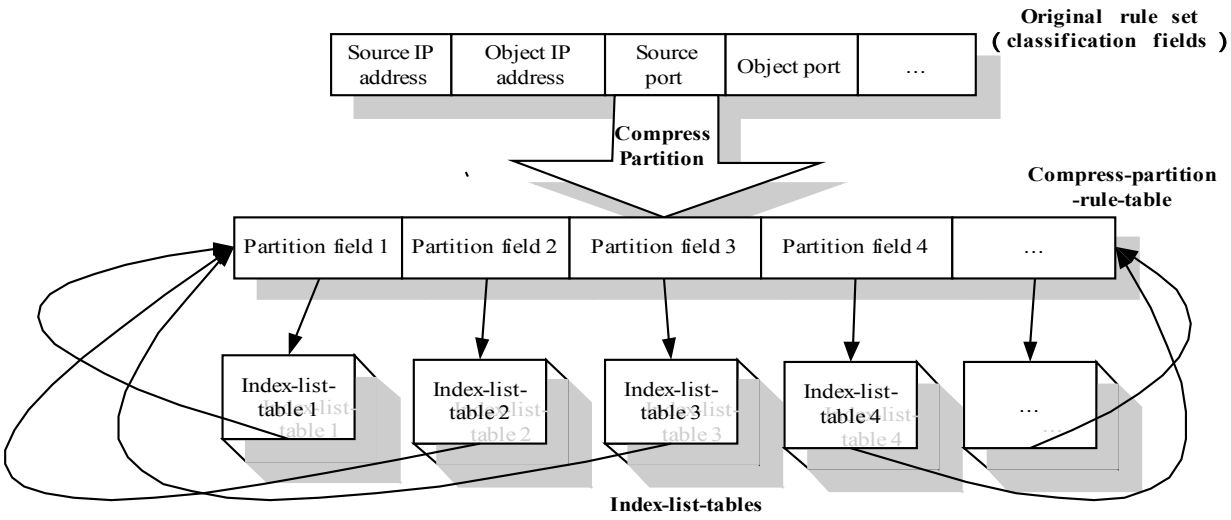


Figure 2. Whole mapping graph of this algorithm

4) **Lookup process**: lookup process is very simple. First, we take out all classification fields from packet when accept a packet, and **compress and partition** the classification fields into P 1**6bit-partition-fields**, then **lookup** P **index-list-tables with the value** of P partition fields and read out P lists. Get the value of **first column** (it contains matched rules number of partition field) of each list, if **0**, it means no matched rules, if not 0, then we compare values of first column of P lists, select the list which has smallest value to lookup compress-partition-rule-table with the values of other columns of the list, then we can get the matched rule of the accepted packet, because these values of other columns is sorted by the order number of rules in compress-partition-rule-tables. The detailed algorithm sees the example in following section.

2.2 Example of the algorithm

There are two kinds of assembled classification fields usually: (1)Source, object IP address (each 32 bits), it is usually for two dimension classification (totally 64 bits). (2)Source, object IP address(each 32 bits), source, object port(each 16 bits), protocol type number(8 bits), it is for five dimension classification (totally 104 bits). The following example algorithm is designed for(2),

the(1) and others packet classification can also be designed in similar way. The first step of the following example algorithm refers to section 2.1, it also depicted in detail in other papers, such as [1][2][10], so we don't depict it in this example.

2.2.1 Create compress-partition-rule-table from the original rule set by compress and partition

There are two steps in the creating compress-partition-rule-table. Compress operations' aim is to decrease the whole field width of classification and speed up packet classification, and partition operations' aim is to decrease the storage space when we create compress-partition-rule-table from the original rule set.

1. Compress operations

compress operations is like the first step of RFC algorithm, but we don't use heuristic algorithm. We use stationary compress algorithm by analysizing characteristic of fields and distributing regulation of the rules. It can decrease storage space and speed up classification. From [1] we know the protocol type only need 3 bits not 8 bits. Source port and object port each need 5 bits not 16 bits. So we can compress bits of the original these fields, so we can decrease the requirement of storage space, speed up classification.

Let the value of protocol type is x_8, the value of source port is y_{16}, the value of object port is z_{16}, and compressed values are x_4, y_6, z_6 respectively, then the value of m which is compressed from above fields and have 16 bits width is:

$$m=x_4+2^4y_6+2^{10}z_6$$

Then we compress 3 fields into a field with 16 bits, this means 40 bits is cut down to 16 bits, the compression rate is 60%. Mapping graph see figure 3.

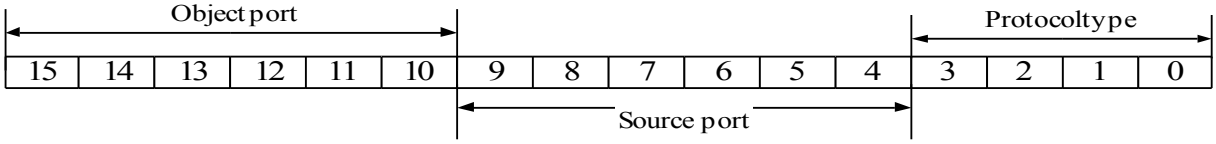


Figure 3. Object port, source port and protocol type mapping into a 16bit-field

Source and object IP address fields can be compressed like this too. For IPv6, source and object IP address fields are all 128 bits, it's too wide. If we don't compress, the efficiency of our algorithm is too low and storage space is too large, so we must compress. In this algorithm it is acceptable, so we don't need to compress the source and object IP address fields.

2. Partition operations

Partition operation is like regional partition algorithm [11], but there are some different between them. One is that our algorithm needs partition the compressed rules into 5 fields, which have 16 bits, and we call it *partition field*. The other is that our algorithm don't need partition the compressed rules into only 1 or 0 matched rules, but into 5 partition fields and matched rules of each partition field can be read out from memory at one time. In this example, it need less than 7 not 1, at the same time we need the number of matched rules of front partition fields is less than latter partition fields. When we create index-list-table from compress-partition-rule-table, if partition field use 8 bits, then we can create 2^8=256 index, although the index space is too small, the lookup efficiency is too low, if partition field use 24 bits, then we can create 2^{24}=8M index, the index space is too huge, so the requirement of storage space is too large, if use other bits that don't suit the hardware characteristic of computer. The compress operation has compressed 104 bits of a rule into 80 bits of a rule, then we partition 80 bits of a rule into 5 partition fields, each partition

field has a 64k rows table (1k=1024=2^{16}), we call it index-list-table. When partition 80 bits into 5 partition fields, the operations, like regional partition algorithm, first analysis the distribution of the values of 5 partition fields, compared all kinds of partition methods, find well-proportioned distribution method, especially the front partition fields. In other words the aim of partition operations is to let the values of each partition field is well-proportioned distribution, especially the front partition fields. Then we can read out matched rules and its number of each partition field, which is in list of index-list-table, from memory in one time, when we lookup index-list-table.

Figure 4 show an example which partition compressed rule set into compress-partition-rule-table in this algorithm. Table 1 shows the pseudo code used to create compress-partition-rule-table from figure 4.

Rule	SourceIP address	ObjectIP address	Combine field
Rule0	160.25.249.18 0xa019f912	161.15.248.158 0xa10f879e	0x0218
Rule1	164.66.39.25 0xa427ee19	168.25.105.24 0xa8196918	0x0008
Rule2	162.24.153.159 0xa218999f	160.62.23.130 0xa03e1782	0x1472
Rule3	162.19.53.84.57 0xa2135439	167.224.93.7 0xa7e05d07	0x0f12
…	…	…	…

Compressedruleset

Partition

Rule	Partitionfield1	Partitionfield2	Partitionfield3	Partitionfield4	Partitionfield5
Rule0	0x01f9	0x0f87	0x0218	0xa012	0xa19e
Rule1	0x27ee	0x1969	0x0008	0xa419	0xa818
Rule2	0x1899	0xe178	0x1472	0xa29f	0xa082
Rule3	0x1354	0xe05d	0x0f12	0xa239	0xa707
…	…	…	…	…	…

Compress-partition-rule-table

Figure 4. Partition compressed rule set into compress-partition-rule-table (P=5 partition fields)

Table 1 Pseudo code used to create compress-partition-rule-table from figure 4.

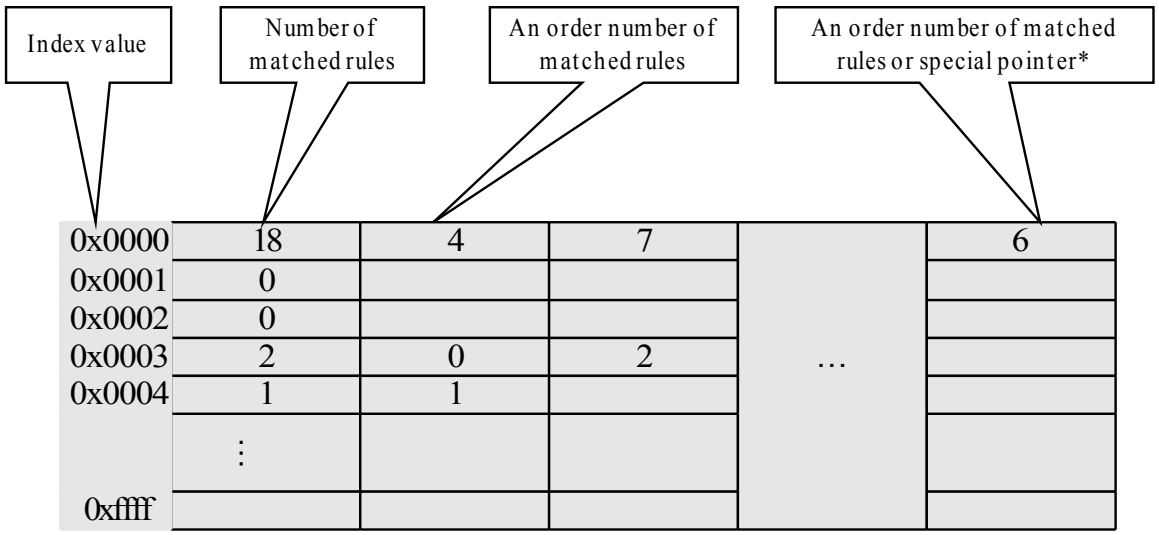
```
void create_ rule_table()
{ long int old_rule[N];   // old_rule[N] store original rule set
                                  // N is number of rules, only stored temporarily.
    old_rule⇐rules;      // initialize.
     long int new_rule[N];  // new_rule[N] store compress-partition-rule-table
                                  // N is number of rules, store it into SRAM for the best.
    new_rule⇐0;      // initialize.
    long int new_field ;  // used to store 5 partition fields of each rule temporarily.
   for (i=0; i<N; i++)  // i is rule i.
          {
             take out the i rule and compress it with compress operation;
             partition it into 5 16bits with partition operation;
      store 5 16bits into new_field;
      store new_field into new_rule[i] .
     }
}
```

2.2.2 Create index-list-table from compress-partition-rule-table

1) The data structure of index-list-table

As state at above, 5 partition fields create 5 index-list-tables. That is to say, there are 5 64k-tables. Structure sketch map of 64k-table is shown in figure 1.

The index value of index-list-table is the value of corresponding partition field. The first column of list of each index-list-table is the number of matched rules of partition field. The other columns are pointers (sequence number of rules) of compress-partition-rule-table, which point to these rules and to be saved in the order of the rules number. Furthermore, considered in special conditions, the matched rules of partition field isn't read out from memory at a time, the last column of list of index-list-table is alternative, one is pointer like other columns, another is special pointer to list of chain table, which is a special table and will be depicted more in 2.3.2. The alternative column is decided by the number of matched rules of partition field, if the number of matched rules of partition field many more than one time memory read, then the column is the special pointer to list of chain table. The detailed data structure of index-list-table (64k-table) is showed in figure 5.



*The last column is alternative, that is pointer to compress-partition-rule-table or special pointer to list of index-list-chain-table

Figure 5. Detail data structure of index-list-table (64k-table)

In figure 5, the last column is special pointer where the index value of the row of the column is 0x0002. In this example we assume the number of matched rules, which can be read out from memory in one time, is 7 (8 columns-1 number of matched rules, 8*16bits=128bits), because of 9>7, the last column is special pointer to list of index list chain table. Then we must create storage space for it, this storage space also include 64k rows, which can be read out from memory in 1 time. The structure of this storage space is like index-list-table, but the first column isn't the number of matched rule, it's order number of matched rule.

2) Index operation

Index operation is important operation in this algorithm. This section will depict the process of index operation in detail. A filter rule is compressed and partitioned into compress rule table, each rule is compressed and partitioned into 5 partition fields. Then 5 partition fields are inserted into corresponding list of 5 index-list-tables by the value of partition field, we called it index operation.

Now we narrate how to index, Let the value of a partition field is x, the partition operation of the y rule of compress-partition-rule-table is adding 1 to the value m of the first column of x row in

index-list-table, then store y to m+2 column. Figure 6 show us how to index and index result for a partition field.

Considered rule 4 of figure 6, the value of this partition field is 0x0218, use this value to lookup row in index-list-table, add 1 to the first column of the row, the value become 2, then store the order number of rule, which is 4, into 2+1 =3 column of the row. We can insert all partition fields into corresponding index-list-tables in this way. The whole mapping graph sees figure 2, there P=5,that is to say, there are 5 partition fields.

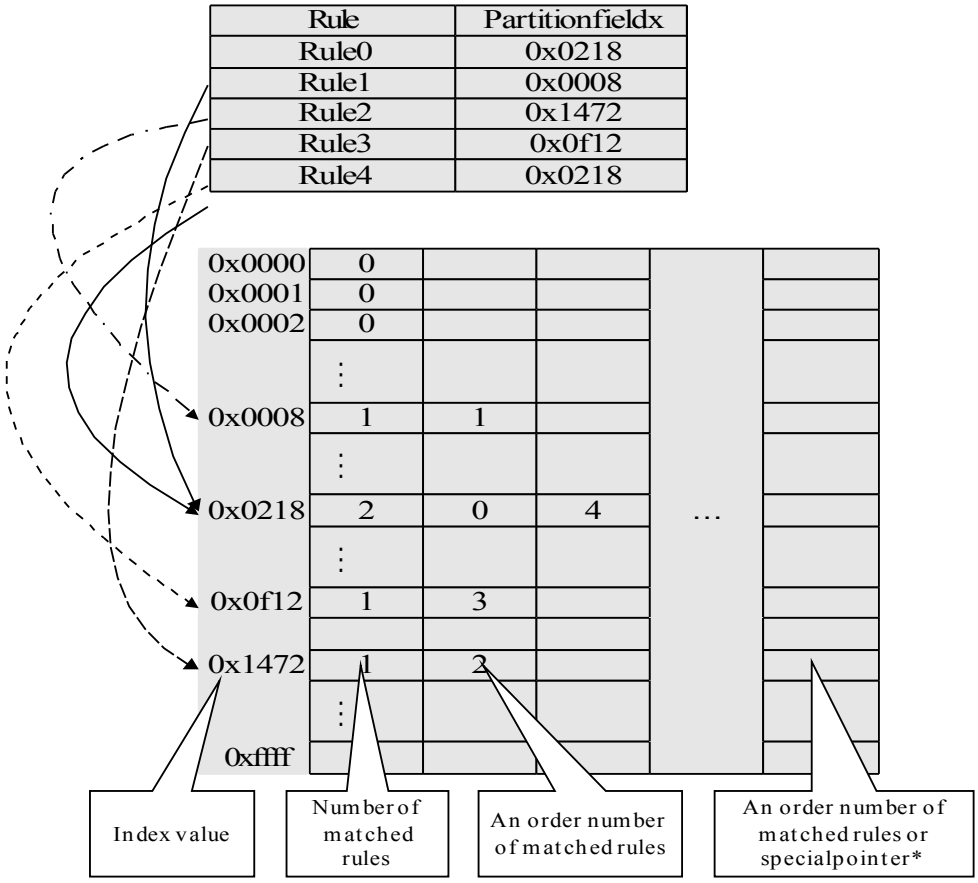


Figure 6 Structure of index-list-table and process of index for a partition field

2.2.3 Process of lookup

Process of lookup is we take out all classification fields in packet when we accept a packet, **compress and partition** the classification fields into 5 1**6 bits partition fields**, then **lookup** 5 **index-list-tables with the value** of 5 partition fields and read out 5 list in order. Look for the value of **first column** (order number of partition field matched rules) of the list, if **0** it means no matched rules, if not 0, then we compare with K+5-I (K will discuss in 2.3.1, 5-i is remainder index-list-tables which aren't read out yet), if less than **K+5-i**, we lookup compress-partition-rule-table with the values of other columns of this list to find **full matched rule**, for these values of other columns is the order number of rules in compress-partition-rule-tables. Under the **other** circumstance, we **intersect** tow list, then intersect to another list, etc. until the number of matched rules is less than **K**, then we also lookup compress-partition-rule-table with the values of columns of this intersect list to find **full matched rule**. Lookup flow chart is shown in figure 7.

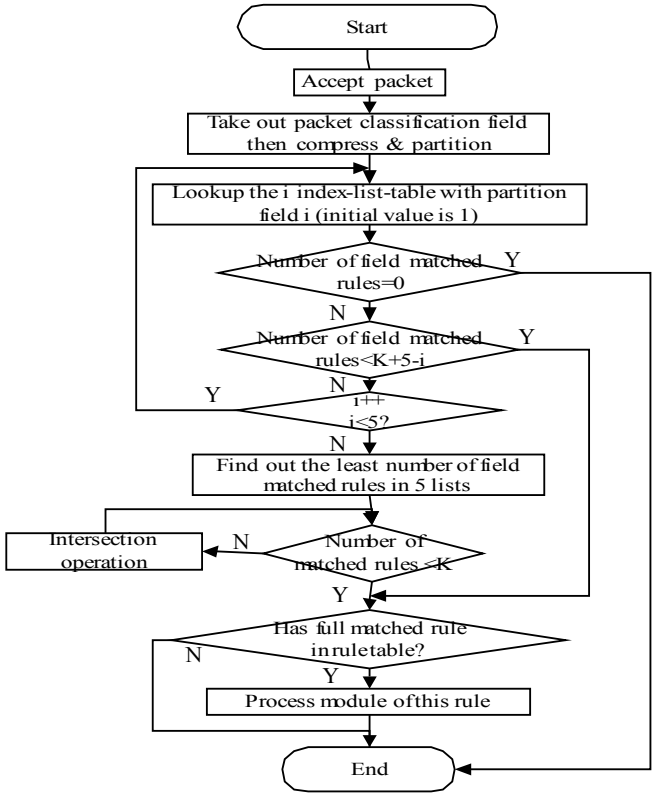


Figure 7. Flow chart of lookup operation

In this example we design each column of index-list-table as 16 bits, 2^{16}=65536, so we can complete classification less than 5+k times memory read under less than 65,536 rules. If more than 65,536 rules, we can design each column of index-list-table as 24 bits. We can conclude it from [1], so this algorithm needn't any modification usually.

2.3 Tow detail problem in implement of the algorithm

2.3.1 Determine the value of K in the algorithm

The value of K is relative to specific hardware, we can get different K in different hardware. There are several operation time in this algorithm, read out compressed rule in compress-partition-rule-table from memory, we assume it as M_{rule}; read out a list in index-list-table from memory, we assume it as M_{index}; compared with compressed rule in compress-partition-rule-table, we assume it as C; intersection operation, we assume it as I; lookup a column in a list, we assume it as L. After lookup several index-list-table, there are k field matched rules or partially matched rules, continue to intersect or not, we can conclude from this equation:

$$5M_{index}+k\,(L + M_{rule} + C) < 5M_{index}+I+L+M_{rule}+C$$

$$\text{that is: } k\,(L + M_{rule} + C) < I + L + M_{rule} + C$$

The left means the time of this k rules matched packet or not. The right means the time of continuous intersection. Then

$$k < \frac{I + L + M_{rule} + C}{L + M_{rule} + C}$$

$$k < 1 + \frac{I}{L + M_{rule} + C}$$

so K is:

$$K = 1 + \left[\frac{I}{L + M_{rule} + C} \right]$$

2.3.2 Index-list-chain-table

Index-list-chain-table is complementary table of index-list-table. This table is created for the number of partition field matched rules more than the columns of list of index-list-table. The

structure of this table like index-list-table, but the first column isn't the number of matched rules, it's also order number of matched rule. Because the list in index-list-table is used a little (this we can see partition operation and [1]), we don't consider it when we research. But we must create it when we implement this algorithm. We call it chain-table sometimes in this paper.

2.4 Complexity analysis of time and space

2.4.1 Space complexity of the algorithm

Assume the number of rules is N, if each rule is precise value, then the requirement of storage space is the least. Because each rule is compressed into a row in compress-partition-rule-table, the requirement of storage space is N. The requirement of storage space of P index-list-tables is PN, and a chain table is N, totally is (P+2)N. Because P is the number of partition fields, then P+2 $=(R*W/16)+2 =C_1D$, D is dimension of classification, R is compressed rate of compress operation, W is width of classification, C_1 is constant coefficient. So we can see the space complexity is O(DN). In this example, under less than 65536 rules, compress-partition-rule-table need: 2^{16}rules * 80bits / 8bits per byte = 640K<1M, 5 index-list-table and 1 index list chain table need: 6 * (2^{16} rows * 128 bits per list / 8 bits per byte) = 6 * 2^{20} bytes = 6M, totally need less than 7M.

2.4.2 Time complexity of the algorithm

Under the worst condition (mainly analyses memory read time, other include intersection operation), P lists in P index-list-table must be read out, each list 1 time memory read, totally P times memory read, and K times compress-partition-rule-table read. So memory read times in this algorithm is P+K under worst conditions. P is the number of partition fields, so P+K $=(R*W/16)+K=C_2D$, D is dimension of classification, R is compressed rate of compress operation, W is width of classification, C_2 is constant coefficient. Under the most condition, it's less than P+K times, so we can see the time complexity is O(D).

2.4.3 Compared with other algorithm

Table 2. Compared with other algorithm

Name of algorithm	time complexity	space complexity	Introduction
RFC	D	N^d	misfit large rule set, need large storage space and parallel computing capability.
Hierarchical cuttings	D	N^d	Storage space too large
Tuple-space search	N	N	Under the worst conditions, time complexity is O(N), too slow.
PCBNP	$\leq$ D	N^2	Multi-dimension, fast, not demand large storage and parallel computing capability, misfit large rule
This algorithm	$\leq$ D	DN	Multi-dimension, fast, not demand large storage and parallel computing capability, support ten thousand about rules.

From table 2 we can see, compared with other algorithm, this algorithm can support ten thousand rules about, has fast classification speed, need less storage space, so it's a better algorithm for packet classification under ten thousand rules about.

3 Conclusion

The development of Internet application necessitates next–generation router's ability to support those functions such as Network Intrusion Detection Systems, QoS, Loading Balance etc. Although implementation of these functions varies mostly, they all need packet classification.

But study has shown that it is difficult to develop a fast and multi-dimensions packet classification algorithm that can deal with a large number of the rules, packet classification has already been a new bottleneck of router. With the development of the network and appearance of IPv6, the situation will become worse. The conflict leads to packet classification be one of important research issues in network technology fields. So it attracts many researchers' attention in recent years.

In this paper a new fast scalable multi-dimension packet classification algorithm named CPI is given. It can support about ten thousands multi-dimension rules, and have fast speed of packet classification, low expected storage complexity. Compared with existing classification algorithms, this algorithm is better than those existing classification algorithms when ten thousands of multi-dimension rules are included, so it can be used in real application.

4 References

1. Tian Liqin, Lin Chuang. Study and application of packet classification. Journal of computer research and development, 2003,6: 765-6775
2. Tian Liqin, Lin Chuang, Xiao Renyi, Li Yajuan. Design and implementationof fast packet classification based onIXP1200, Journal of computer research and development, 2003,11: 1616-1625
3. A.N.M. Ehtesham Rafiq, M. Watheq El-Kharashi, Fayez Gebali. A fast string search algorithm for deep packet classification. Computer Communications 27 (2004) 1524–1538
4. F.Baboescu, G.Varghese. Fast and scalable confict detection for packet classifers, Computer Networks 42(2003) 717-735
5. Ying-Dar Lin, Huan-Yun Wei, Kuo-Jui Wu. Ordered lookup with bypass matching for scalable per-flow classification in layer 4 routers, Computer Communications 24 (2001) 667-676
6. F. Baboescu, G.Varghese. Fast and scalable confict detection for packet classifers, Computer Networks 42(2003) 717-735
7. Pankaj Gupta and Nick McKeown. Packet classification using hierarchical intelligent cuttings. In Proc.ACM Sigcomm'98,Sept.1998
8. Packet classification using tuple space search, Srinivasan, S.Suri, and G.Varghese, in Proceedings of SIGCOMM'99,1999
9. T.V. Lakshman and D. Stiliadis. High speed policy-based packet forwarding using efficient multi-dimensional range matching. in Proceedings of ACM SIGCOMM'98,1998
10. Zhu Qiuxiang, TaoJun. Algorithms for packet classification, Mini- micro systems, 2004,10:1802-1810
11. Yan Tian-xin, Wang Yong-gong, Shi Jiang-tao, Dai Xue-long. Optimized implementation of regional partition algorithm for packet classification, Journal of China institute of communications, 2004,6:80-88
12. Shan Zheng, Zhao Rongcai, Zhang Zheng. Research of algorithms for packet classification, Computer engineering and application, 2005,7:149-152

figures: 7
tables:2

Article received: 2005-09-22