A Fast IP Classification Algorithm Applying to Multiple Fields

Zhongchao Yu, Jianping Wu, Ke Xu, Mingwei Xu
Department of Computer Science, Tsinghua University
Beijing 100084 P.R.China

Abstract-**With the network applications development, routers must support those functions such as firewalls, provision of QoS and traffic billing etc. All these functions need classification of IP packets, according to which it is determined how different packets are processed subsequently. In this article, a novel IP classification algorithm is proposed based on the Grid of Tries algorithm. The new algorithm not only eliminates original limitations in the case of multiple fields but also shows better performance in regard to both time and space. It has better overall performance than many other algorithms.**

I. Introduction

Future IP network must provide more service types and better quality of service [5], which include differentiated service [1], firewalls [2], policy-based routing [3], virtual private network and traffic billing [4] etc. All these functions need classification of IP packets.

In this paper, we first provide the mathematical model of IP classification problem. Then we present a novel IP classification algorithm applying to multiple fields based on the two-dimension IP classification. We also compare our new algorithm to others. The simulation result shows that our algorithm has best overall performance.

II. Mathematical Model Of Ip Classification

A. *Terminology Definitions [8]*

An address D is a bit string of W bits in length;

A prefix P is a bit string of length between 0 and W. We use *length (P)* to denote the number of bits in a prefix;

A header H has K fields, which are denoted by $H[1],H[2],...,H[K]$ respectively. Each field is a string of binary bits;

A filter F has also K fields. Each field $F[i]$ in a filter can specify any of the three kinds of matches: exact match, prefix match, or range match;

It is called an exact match iff a single value is specified for the ith filter field (i.e. $F[i]$) and the header field $H[i]$ is equal to $F[i]$;

It is called a prefix match iff a prefix is specified for the ith filter field and the first $length(F[i])$ binary bits of the header field $H[i]$ are the same as those of $F[i]$;

It is called a range match iff a range of values $F[i] =[val1,val2]$ is specified for the ith filter field and the header field $H[i]$ falls into that range, i.e. $H[i] \in [val1,val2]$;

A filter F is said to be a matching filter for a header H iff each field $H[i]$ of H matches the corresponding field $F[i]$ of F. The type of match is specified by $F[i]$ and could be an exact match, a prefix match or a range match;

A set of N filters is called a filter database, which is denoted by FS;

Each filter F has a *cost* property denoted by *cost (F)*. For $\forall F_1, F_2 \in FS$, if $cost(F_1)=cost(F_2)$ then $F_1=F_2$. We use the *cost* property to assure that there is at most one matching filter.

B. *The Best Matching Filter Problem and IP Classification*

We define the following problem as the best matching filter problem:

Given a filter database $FS \neq \varnothing$ and a header H, find the best matching filter f_{best} which meets the following conditions:

(1) $f_{best} \in FS$;

(2) f_{best} matches H;

(3) $\forall f \in FS, f \neq f_{best}$, if f matches H, then $cost(f_{best})<cost(f)$.

IP classification is an instance of the best matching filter problem. In theory, seven fields can be used for the filter: destination/source IP address, destination/source transport port, type of service, protocol type and flag of transport layer. The sum of bits of these fields is 120 (we assume that all the seven fields reside in IP packet header for the sake of convenience, although some fields are in TCP header actually.) Statistical results of some actual filter databases used by ISPs show that 17% of the filters specify only one field, 23% specify three fields, and 60% specify four fields [6].

III. Related Works

Packet classification based on Patterns [7] is used in the operating system when dispatching data packets of input queue to different process spaces. It is the first algorithm avoiding linear lookup. Its performance has direct ratio with the number of fields and is independent of the number of filters. But this algorithm has very strong limitations on filters, thus it is not suitable for IP routers.

Crossproducting algorithm [8] is based on caches. For bigger classifiers, the authors propose a caching technique (on-demand crossproducting) with a non-deterministic classification time.

Modular algorithm [9] is an IP classification algorithm based on statistics. It may optimize the lookup data structure according to the distribution of filter matching ratio and IP traffic. Short of effective statistic parameters, this algorithm cannot be practically used for IP routers now.

RFC (Recursive Flow Classification) algorithm [6] is a simple multi-stage classification algorithm, which maps the S bits header to the T bits *ClassID* ($T << S$) step by step. It is the fastest algorithm ever known, but it needs a lot of pre-computation (usually more than ten seconds) and it may suffer from space explosion.

A solution called Grid of Tries is proposed in [8]. In this scheme, the trie-tree data structure is extended to two dimensions. This is a good solution if the filters are restricted to only two fields, but it is difficult to extend it to apply to more fields.

A hardware-only algorithm could employ a ternary CAM (content-addressable memory). Ternary CAMs store words with three-valued digits: '0', '1' or '*' (wildcard). The rules are stored in the CAM array in the order of decreasing priority. Given a packet-header to classify, the CAM performs a comparison against all of its entries in parallel, and a priority encoder selects the first matching rule. While simple and flexible, CAMs are currently suitable only for small tables; they are too expensive, too small and consume too much power for large classifiers. Furthermore, some operators are not directly supported, and so the memory array may be used very inefficiently.

In this article, we proposed a novel lookup algorithm called non-collision hash trie-tree algorithm, which is based on Grid of Tries algorithm. Average time consumed and space requirement of this algorithm are less than those of Grid of Tries, and it gets rid of the limitation of filters in Grid of Tries. It is the most attractive candidate if implemented by means of software.

IV. Non-Collision Hash Trie-Tree Algorithm

A. *Basic Algorithm*

We mentioned that seven fields in IP packets might be the candidate fields of the filter. But actual filter databases usually use only five fields: destination/source IP addresses, destination/source ports and protocol type. The width of protocol type field is 8 bits. To program conveniently, we extend the protocol type field to 16 bits.

TABLE I An Example Database Of Filters

ClassID	Dest-IP	src-IP	Dest-port	src-port	Prot
0	10.1.*.*	10.2.*.*	*	*	*
1	10.3.*.*	10.4.*.*	80	*	17
2	10.5.*.*	10.6.*.*	80	*	17
3	10.5.*.*	10.6.*.*	[20,21]	*	6
4	10.7.*.*	10.7.*.*	*	gt 1023	6
5	*	*	*	*	*

The value of destination port, source port and protocol type ranges from 0 to 65535, but in actual filters they only use a very small part of the whole range. Currently, the value of protocol field is limited to TCP, UDP, ICMP, IGMP, (E)IGRP, GRE and IPINIP. In most Client-Server software architectures, ports can be roughly divided into two categories [10]. One is the reserved port which numbers in 1-1023 and the other is the ephemeral port which numbers larger than 1023. Ephemeral ports are usually used in client software and are assigned by the kernel. They are nothing but to identify an endpoint of a connection. It is almost impossible that a filter specify a specific port larger than 1023. Usually, filters specify a range such as gt1023 meaning that all ports larger than 1023 (but less than 65535). The most widely used reserved ports are 20,21 (FTP) and 80 (www), other ports are used less frequently.

Our analysis above shows that the number of combination of destination/source ports and protocol value in actual filters is very small. Based on this observation, we construct a two-stage lookup table that can be used to lookup without collision at all. We take table I for example to illustrate our idea.

Table I is a filter database that contains six filters. We assume *ClassID* is the same as the filter cost in this database. Take the destination port for example. We assign each port in 0-65535 a bitmap. This bitmap denotes the filters the port matches (the length of bitmap is equal to the number of filters). For example, the bitmap of ports 21 and 22 are both 100111, which means they match filter 0,3,4 and 5. According to these bitmaps, we classify all possible destination ports into different equivalent classes. The ports with the same bitmap belong to the same class. The bitmap of equivalence class A is denoted by $bmp(A)$. The set of all such destination port equivalent classes is denoted by D_Set. And the total number of destination port equivalent classes is denoted by D. For example, the D_Set of Table I is {{80}, [20,21], {0-65535 except 20,21,80}}. In the same manner, we construct the set of source port equivalent classes S_Set and protocol equivalent classes P_Set, whose element numbers are S and P respectively.

If $a \in D\_Set$, $b \in S\_Set$, $c \in P\_Set$, the 3-tuple (a,b,c) is

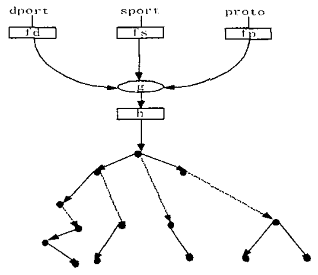

Fig.1. Non-collision hash trie-tree algorithm

called a cross-combination.

Now we further divide the set of all the cross-combinations into different equivalent classess called *DSP_Set*. We divide as follows:

Consider two cross-combination *(a,b,c)* and *(d,e,f)*. If *bmp(a)&bmp(b)&bmp(c) = bmp(d)&bmp(e)&bmp(f)* then *(a,b,c)* and *(d,e,f)* belong to the same class, otherwise they belong to different classes.

Each element of the *DSP_Set* (a cross-combination equivalent class, notice that it is a set itself) has a corresponding set of destination and source IP prefix pairs. These IP prefix pairs are those of the filters whose destination port, source port and protocol number 3-tuple belongs to the element. The cross-combinations that belong to the same element of *DSP_Set* share the same pointer to a destination and source IP prefix set. Non-collision hash trie-tree algorithm first finds out the pointer to the destination and source prefix pair set by looking up a non-collision hash table according to destination port, source port and protocol number. Then we perform a two-dimension trie-tree lookup in the destination and source IP prefix pairs to obtain the final *ClassID*.

B. Non-collusion hash lookup

When classifying a packet with its header *H(dport,sprot,proto)* (representing destination port, source port and protocol respectively, we do not consider IP address fields in this step). We first look up three tables using *dport*, *sport* and *proto* as the indices respectively. We use a function of these lookup result *g(fd(dport),fs(sport),fp(proto))* as the index to perform another lookup. The result is *h(g(fd(dport),fs(sport),fp(proto)))*, which is the pointer to a

ALGORITHM I CONSTRUCTION OF fd(fs,fp) TABLE

```
*table_fx table_fx_setup()
{
    /*allocate memory and initialization */
    p=new_table_fx();
    for(n=0;n<65536,n++){
        Get bmp(n);
        eq=search_in_eqivalence_class_x_set(bmp(n));
        if(eq == NULL) { /* new bmp */
          eq=new_eqivalence_class_x(bmp(n));
          add eq into x_set;
        }
        p->table[n].ID=eq->ID;
    }
    return p;
}
```

ALGORITHM II CONSTRUCTION OF h TABLE

```
*table_h table_h_setup()
{
    indx=0;
    /* allocate memory and initialization */
    p=new_table_h(D,S,P);
    for eqd in d_set,eqs in s_set,eqp in p_set {
        bmp=eqd->bmp&eqs->bmp&eqp->bmp;
        eq=search_eqvivalence_class_dsp(bmp);
        if(eq is null) {     /*new bmp*/
            eq=new_eqivalence_class_dsp(bmp);
            add eq into dsp_set;
        }
        p->table[indx++].ID=eq->ID;
    }
}
```

set of destination and source IP prefix pairs. We number the *D* equivalent classes' IDs of *D_Set* as *0,1,2,...,D-1* and define *fd(dport)* as the equivalent *classID* of *dport*. And it is the same with *fs* and *fp*.

Fig. 1 shows the lookup process stated above. The rectangle represents a lookup table and *g* is a hash function. We choose *g(d,s,p)=PSd+Ps+p*. According to our definitions of *fd, fs and fp*, we have $0 \le d \le D-1$, $0 \le s \le S-1$, $0 \le p \le P-1$. Now we prove *g* will not cause any collision.

Theorem Define $g(d,s,p)=PSd+Ps+p$ $(d,s,p,D,S,P \in Z)$, $0 \le d \le D-1$, $0 \le s \le S-1$, $0 \le p \le P-1$, if $g(d_1,s_1,p_1)=g(d_2,s_2,p_2)$, then $d_1=d_2$, $s_1=s_2$, $p_1=p_2$.

Proof From $PSd_1+Ps_1+p_1=PSd_2+Ps_2+p_2$, we get $p_2-p_1=PSd_1+Ps_1-PSd_2-Ps_2=P[S(d_1-d_2)+s_1-s_2]$. Taking the absolute values of both sides, we have $|p_2-p_1|=P|S(d_1-d_2)+s_1-s_2|$. Because $p_1, p_2 \in [0,P-1]$, $|p_2-p_1|<P$ and $|S(d_1-d_2)+s_1-s_2|$ is an integer, we conclude that $p_2-p_1=0$, i.e. $p_2=p_1$. For the same reason , we have $s_2=s_1$, $d_1=d_2$.□

Now we have proved that *g* is a non-collision function. Construction of these tables and the two-dimension trie-tree is completed by reading the filter database during the pre-computation stage. Thus, we can find the pointer to the set of destination-source IP prefix pairs with four memory accesses. The next step is to look up through the two-dimension trie-tree using destination and source IP addresses in the packet header. Algorithm I and II are the setup algorithms of these tables.

C. Lookup In Destination-Source IP Prefix Pairs

In this section, we introduce a simplified Grid of Tries lookup algorithm. Extending the trie-tree data structure from one dimension to two-dimension, we have the two-dimension trie-tree. We take the filter database in table II as an example to show this process (assume that the width of IP address in the table is 2).

We first build up a trie-tree (denoted by Dest-Trie tree) according to the destination IP prefixes. For each node in Dest-Trie tree, if there exists the corresponding destination IP prefix, it points to a source IP prefix trie-tree (denoted by Src-Trie) otherwise the pointer is null. A Dest-Trie node not only contains the corresponding source IP prefixes but also those of its ancestors in Dest-Trie. In that case, time complexity of lookup in the two-dimension trie tree is *O(W)*, but since each Dest-Trie node stores both the pair source IP prefixes of its own and those of its ancestors, the space complexity turns out to be $\Theta(N^2)$.

We can get rid of the redundant copies. Every Dest-Trie node only contains the corresponding source IP prefixes in the database. But in this case, in order to find out the final *ClassID* with the least cost, we need search not only the Src-Trie but also that of its ancestors. Thus the time complexity rises up to $O(W^2)$, although we need less space.

The answer is to introduce a switch pointer. In the process of pre-computation, we direct the null pointer of the Src-Trie node to a Src-Trie node of one of its Dest-Trie ancestors' so that we can proceed further when we go along the longest matching path. In addition, we must make sure that the longer a destination-source prefix pair is, the lower its cost. Take filters 2,3 and 4 of table 2 for example, filter 2 is shorter than filter 3 in destination-source pair length, and filter 3 is shorter than filter 4. But the fact is that filter 2's cost is lower than filter 3 and filter 3's cost is lower than filter 4's. So they do not accord with our principle of a longer pair with a lower cost. However, we observe that if we remove filter 3 and 4 from the table, our lookup result does not change. That is because a header matching filter 3 and filter 4 will surely match filter 2 and filter 2 has a lower cost. In other words, filter 3 and filter 4 are redundant. There are two ways to deal with the problem. The first one is to guarantee that there is no redundancy at all when building up

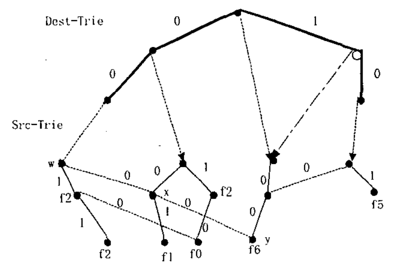

Fig.2. Improved data structure of 2D trie-tree

TABLE II An Example Of IP Pairs

ClassID	dest-IP	src-IP
0	0*	10*
1	0*	01*
2	0*	1*
3	00*	1*
4	00*	11*
5	10*	1*
6	*	00*

our filter database; the other is to change the filter 3 and 4's *ClassID* to 2 in the pre-computation stage. Both will guarantee the correctness of the algorithm.

The ultimate two-dimension trie-tree is shown as Fig. 2, where the number beside the letter "*f*" denotes the corresponding *ClassID* (it is also the sequence number of the filter and its cost). Given this figure, we look up the matching filter with the lowest cost for a coming header as follows: First perform a longest destination IP prefix matching process ending at some node in Dest-Trie. Then go along the 0 or 1 pointer (or if null, a switch pointer) of the corresponding Src-Trie to perform a longest source IP prefix matching according to the header's destination and source IPs. We go as further as we can, and the *ClassID* of the filter with the lowest cost along the path is the final result we want.

V. Lookup Performance

In the worst case, it takes four serial lookups to obtain the pointer to the two-dimension trie-tree, i.e. lookup in the table *fd,fs,fp* and *h*. Lookup through the two-dimension trie-tree needs to visit *2W* nodes in the worst case. So the total number of memory access is *2W+4* in the worst case. What's more, the time consumed is irrelevant to the number of filters. In contrast, even with a hash function without collision, Grid of Tries needs *4(1+2W)* memory accesses.

It is a little more complicated as for the space complexity of non-collision hash algorithm. The numbers of entries of the *fs,fd* and *fp* tables are all 65536 and the number of entries of table *h* is $D \times S \times P$. Theoretically speaking, the number of table *h*'s entries could be up to 65536×65536×65536. However, as analyzed above, *D*, *S* and *P* are rather small in normal cases, so we expect that the number of table *h*'s entries is quite small. As for the two-dimension trie-tree, since a filter needs *2W* trie nodes at the most and there are *N* filters together, space needed is about *2NW*. Thus we could estimate that the total space is *Table_Size+2NW* in the worst case, while Grid of Tries for multi-fields also needs about *Hash_Size+2NW*, where *Hash_Size* denotes the space for hash table. In order to gain more time efficiency, the hash table usually consumes a lot of memory. In our test, non-collision hash trie-tree algorithm also shows better performance in space.

It is difficult to analyze the average performance of both time and space. Even worse, little is done in sampling for both filters and IP flow in the real Internet. Because of that, we design a virtual environment to perform a testing. Our concern focuses on the relative performance between non-collision hash trie-tree and Grid-of-Trie, so the virtual environment will suffice. We make reasonable assumptions about IP flows and filters and generate IP packet flow and filters from a random generator. For the sake of comparability, we add in the same limitations needed by the Grid-of-Trie algorithm. We observe that even our assumptions favor the Grid-of-Trie, non-collision hash algorithm still shows better performance in both time and space. So we expect the difference will be more obvious in practice. Testing results are shown in Fig.3 and Fig. 4.

VI. Conclusion And Future Work

In this article, a novel IP classification algorithm is proposed based on the Grid of Tries algorithm. The new algorithm not only eliminates original limitations in the case of multiple fields but also shows better performance in regard to both time and space. It has better overall performance than many other algorithms.

The algorithm we presented can be improved further. In the process of lookup through two-dimension trie-tree, our algorithm will go one step according to one bit of the header. If it can lookup several bits at one time, the depth of trie-tree will reduce greatly and the performance will improve. But to do this we need more memory. Future work is to explore the distribution of IP prefixes [11], by which we hope that we can select the depth of trie-tree and decide which bits to look at when going down the trie-tree.

References

[1] W.Weiss. QoS with Differentiated Services. Technical Journal v3 n4, 1998:48~62

[2] S.Bellovin and W.Cheswick. Network Firewalls. IEEE Communications Magazine v32 n9 1994: 50~57

[3] Y.Jyh-haw, C.Randy and N.W.Richard. Interdomain. Access Control with Policy Routing. In Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems. Oct. 1997: 46~52

[4] Richard Edell, Nick McKeown, and Pravin Varaiya. Billing Users and Pricing for TCP. IEEE Journal on Selected Areas in Communications v13 n7 Sept. 1995:1162—1175

[5] Xu Ke, Xiong Yong-qiang, Wu Jian-ping. Analysis of Broadband IP Router Architecture. Journal of Software, v11 n2 2000:179~186

[6] P.Gupta and N.McKeown. Packet classification on multiple fields. ACM Computer Communication Review, v29 n 4 1999:146~160

[7] M.L.Bailey, B.Gopal, M.A.Pagels, L.L.Peterson. PATHFINDER: A Pattern-Based Packet Classifier, In Proceedings of 1st symposium on Operating System Design and Implementation. Usenix Association, Nov 1994: 95~104

[8] V.Srinivasn, G.Varghese, S.Suri, et al. Fast Scalable Level Four Switching. ACM Computer Communication Review, v28 n4 1998:191~205

[9] T.Y.C.Woo. A Modular Approach to Packet Classification: Algorithms and Results. In Proceedings of IEEE Infocom2000. San Francisco, CA: IEEE Computer Society Press, 2000: 1210~1217

[10] W.R.Stevens. UNIX Networking Programming vol 1(2nd Edition). Englewood Cliffs, NJ: Prentice Hall,Inc. 1998

[11] Merit Inc. IPMA Statistics. http://nic.merit.edu/ipma

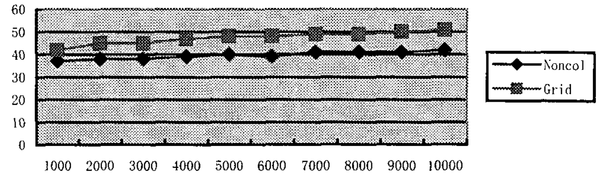

Fig.3. Comparison of time performance between non-colision trie-tree (denoted by Noncol) and Grid of Tries (denoted by Grid), where x-axis plots the number of filters and y-axis plots the total seconds consumed while processing 10^7 packets.

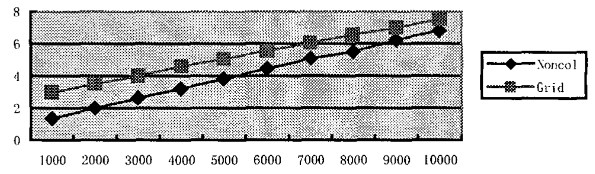

Fig.4. Comparison of space performance between non-colision trie-tree (denoted by Noncol) and Grid of Tries (denoted by Grid), where x-axis plots the number of filters and y-axis plots the maxium memory (MB) consumed.