研究論文

A Hybrid Approach to Packet Classification

Xiaohui ZHAO
Graduate University for Advanced Studies
Yusheng JI
National Institute of Informatics
Yongcheng LEI
Lenovo R&D Institute

ABSTRACT

Packet classification is one of the key technologies to support differentiated services to classified flows. Combining the top-down lookup tree structure and intelligent constructing strategy of present algorithms, this paper proposes a hybrid approach to packet classification. The filter set, which is used to classify packets, are stored in the leaf chains of a special data structure, in which an index table acts as index to locate several lookup trees. Also, an evaluation formula shows the major criteria on how to build lookup trees and distribute filters to obtain satisfactory efficiency. The performance of the proposed algorithm is discussed by analytical computation and simulation. Theoretically, it has a logarithmic execution time cost with a polynomial space (storage) cost. Further, a simulation of packet classifier built on IBM Power Network Processor is performed to test its performance and do a comparison between multi-tree applied cases and the unapplied. The results show its superiority in complex filter handling and matching modes.

[Keywords]

IP flow, packet classification, filter set, lookup tree

1 Introduction

There are a number of network services that require packet classification, such as access-control in firewalls, IP routing, policy based routing, provisioning of differentiated qualities of service, and traffic billing[1-3]. In each case, it is necessary to determine which flow an arriving packet belongs to, so as to determine, for example, whether to forward or filter it, where to forward it to, what class of service it should receive, or how much should be charged for transporting it. This categorization function is performed by a packet classifier (also called a flow classifier), which maintains a set of filters, where each flow obeys at least one filter. To classify which flow a packet belongs to is based on the contents of the packet header(s). For example, a flow could be defined by particular values of source and destination IP addresses, and particular transport port numbers. Or a flow could be simply defined by a destination prefix and a range of port values. Table 1 lists some typical application areas of packet classification. Also this table gives the related requirements of number of fields for matching classification types, and filter exam-

Table 1: Packet Classification Examples.

Application	Number of Fields / Classification type	Filter Example
Switching, MPLS	Single / Exact Match	Send packets with Dest_MAC==68:10:01:ab:12:7a directly to end hosts
Forwarding	Single / Longest Prefix Match	Send all packets with Dest_IP==192.168.0.* to the ISP's router
Flow Identification, IntServ	Multiple / Exact Match	Give packet with Src_IP, Dest_IP, Src_Port, Dest_Port ==(192.168.4.5, 200.10.2.3, 21, 1030) highest priority
Filtering, DiffServ	Multiple / Prefix or Range Match	Drop all packets with Src_IP==192.168.* && Src_Port>1023 && Dest_Port <5000
Load Balancing	Multiple / Scan with Exact or Prefix Match	Re-direct packets having filenames ending with ".ra" in DATA field to audio server.
Intrusion Detection	Multiple / Scan and Match Reg. Expressions	Create alarm when packets having "get *.vbs" in DATA field.

ples.

In this area, a lot of work, such as [4-7], has been done from classification algorithms to hardware design. Typically, Stanford High-performance Networking Group[8] and Princeton Extensible Router Group[9] are two active teams in this field.

2 Related Work

The idea for packet classification was initiated in [10] and was later expanded in [11,12]. The architecture and algorithms presented in these papers were targeted mainly for a single end-point and their main goal was to isolate packets that are destined to specific protocols or specific connections. A variation was presented in [13] where the first hardware implementation of packet classification was reported.

The general packet classification problem can be viewed as a point location problem in multidimensional space. This is a classical problem in Computational Geometry, and is defined as follows: Giving a point in a d-dimensional space and a set of n-dimensional regions ($n \le d$, d represents the number of fields to be matched), the problem is to find the region that the point belongs to. When considering the general case of $d>3$ dimensions, as is the problem of packet classification, the best algorithms considering time or space have either an $O(\log^{d-1} n)$ time-complexity with $O(n)$ space, or an $O(\log n)$ time-complexity with $O(n^d)$ space. Let us assume that we want the router to be able to process 1000 filters of five dimensions with 1 μs (to sustain one million packets per second throughput). Unfortunately, since we have to spend 10000 memory accesses per packet or 10^{12} memory units, either of which seems impractical at present.

As same as the point location problem, packet classification algorithms also use two dominant resources, memory and time. All existing packet classification algorithms trade memory for time, ranging from schemes like Recursive Flow Classification (RFC)[14,15] (which is fast but takes excessive storage), to linear search (which is slow but takes minimal storage). The current algorithms with the best time-space tradeoffs appear to be EGT-PC[16] and HiCuts[17]. However, while the tradeoffs have been constantly improving, the time taken for a reasonable amount of memory is still too poor for practical deployment.

Because of problems with existing algorithmic schemes, most venders use ternary Content Addressable Memory (CAM), which uses brute-force parallel hardware to simultaneously check for all filters[18]. The main advantages of TCAMs over algorithmic solutions are speed and versatility (TCAM works for all classifiers, not only typical ones). However, CAM fundamentally has to contend with reduced density (using comparing logic per bit) and increased power (using parallel comparison). Two less fundamental problems are the need for filters with range specifications to be translated into several CAM entries, and the need for gluing logic. Actually, these problems have already made venders consider about algorithmic alternatives. These venders include Cypress, Fast-Chip, EZchip, and Integrated Silicon.

This paper aims to propose a generalized classification algorithm with efficient time-space tradeoffs and flexible supports to filter matching modes. And the rest of this paper is organized as follow: Chapter 3 defines the formal description of classification problem; Chapter 4 gives our basic idea of algorithm, as well as design goals; Chapter 5 and 6 respectively set out the data structure and detailed steps of the proposed algorithm; Chapter 7 discusses its performance and does comparisons with other present algorithms; Chapter 8 describes the simulation on IBM NP4GS3.

3 Packet Classification Problem

Giving a set of filters defining packet attributes or content, packet classification is the process of identifying the filter or filters to which a packet conforms or matches[19]. In this section, we formally define the packet classification problem. First, we develop some useful definitions.

Definition 1. A destination address D is a string of L bits, each bit can be 0,1 or * ('*' is a wildcard character).

Definition 2. A prefix P is a string of 0 to L bits, length(P) denotes its length in bit.

Definition 3. Prefix P matches address D, when the first length(P) bits of D equals to P.

Definition 4. The information relevant for classifying a packet is contained in K distinct header fields in the packet. These header fields are denoted as $H[1]$, $H[2]$, ..., $H[K]$.

Definition 5. A classifier (also known as filter table or filter database), FT, consists of N filters $F1$, $F2$,..., FN. Each filter Fj is an array of K values, where $Fj[i]$ is a specification in the i-th header field. The i-th header field

is sometimes referred to as the *i*-th dimension. The value *Fj*[*i*] specifies what the *i*-th header field of a packet must contain in order for the packet to match filter *Fj*.

These specifications often have (but need not be restricted to) the following forms: exact match, for example source address must equal to 192.168.0.16 or prefix match, like destination address must match prefix 192.168.*.

Definition 6. Each filter *Fj* has an associated directive, which specifies the action to perform for a packet that matches this filter. This directive may indicate whether to block the packet, send it to a particular interface, or perform some other actions.

Definition 7. Since a packet may match more then one filter, we associate a cost *Wi* for each filter *Fi* in this paper to resolve ambiguous matches. *Wi* is imported to help constructing the arbitrary mechanism so as to improve classification efficiency. Anyway, the cost can be set to 1 in the simplest case.

Finally, the packet classification problem is to find the lowest cost filter matching a given packet *P*, and it can be defined as follows,

Find the filter *Fm* in *FT* such that,

-*Fm* is a filter match for *P;*

-There is no other *Fn* in *FT* such that *Fn* is a filter matches for *P* and *Wn*<*Wm*.

Filter	Source Address	Destination Address	Source Port	Destination Port	Protocol Type	Action
F0	000*	111*	10	*	UDP	*Act0*
F1	000*	111*	01	10	UDP	*Act0*
F2	000*	10*	*	10	TCP	*Act1*
F3	000*	10*	*	01	TCP	*Act2*
F4	000*	111*	10	11	TCP	*Act1*
F5	0*	111*	10	01	UDP	*Act0*
F6	0*	1*	10	10	UDP	*Act0*
F7	0*	01*	*	*	TCP	*Act2*
F8	*	0*	*	*	TCP	*Act2*
F9	*	0*	*	01	UDP	*Act0*
F10	*	*	*	*	UDP	*Act3*
F11	*	*	*	*	TCP	*Act4*

Figure 1: A Simple Example with 12 Filters on Five Fields.

4 Basic Idea and Design Goals

We summary the present classification strategies as two characteristics, listed as below:

- Binary Search on Prefix Length: Filters can be mapped to tuples, according to different length of prefixes. Each tuple is maintained as a hash table that can be searched in one memory access. Inside a tuple, even a simple linear search of the tuple space can provide significant speedup over naive linear search over the filters. The most typical representative is Tuple Space Search (TSS) algorithm[20], which fully utilizes such pre-handling to diminish searching range.
- Pulling Filters onto the Decision Tree: Recursive cutting can be embodied using a decision tree in which each node represents a cut and leaves represent filters. HiCut algorithm[17] has applied this structure to store filters, the intermediate nodes of the tree represent the intelligent cutting over multi-dimensional filter set. In general, some linear searching at leaves is useful to reduce storage.

Our basic idea is just to combine the advantages of the both, i.e. borrowing the ideas of tuple grouping and lookup tree. And in the view of data structure, we joint the lookup trees to one hash table, which comes from tuple grouping. Further, a hybrid approach using such data structure is proposed.

Due to the high throughput and speed requirements of present network control as well as various requirements for different application listed in table 1, we outline the major criteria that an efficient classification algorithm must meet:

1. The algorithm must be fast enough for use in routers with gigabits links.
2. The algorithm must be able to process every packet arriving to the physical interfaces at wire-speed. Since the algorithm cannot use buffering to absorb the variation in execution times caused by packet size.
3. Classification filters must be based on several fields of the packet header, including among others source and destination IP addresses, source and destination port numbers and protocol types. The filters must be able to handle prefix match and not just exact values.
4. It is possible that some packets may match more than one filter. The algorithm must allow arbitrary priorities to be imposed on these filters, so that only one of these filters will finally be applicable to the packet.

5. Even though memory prices have continued to fall, the memory requirements of the algorithm should not be prohibitively expensive.

5 Data Structure Design

The basic strategy to multidimensional search, as used for packet filtering, is to use decomposable search so that the intersection step does not take more time than the required bound. However, as it was pointed out before, even a $\log^4 n$ solution for five-dimensional packet filtering is not practical for our application where n can be in the thousands. Therefore, our aim is to lower the dimensions by unfolding one or more prefix fields into a hash table, called index table. And decompose the filters into several trees connecting to the index table, according to the dependency among filters. While the leaf nodes store the actual filters within a so-called leaf chain.

Figure 2 shows the diagram of this classification data structure.

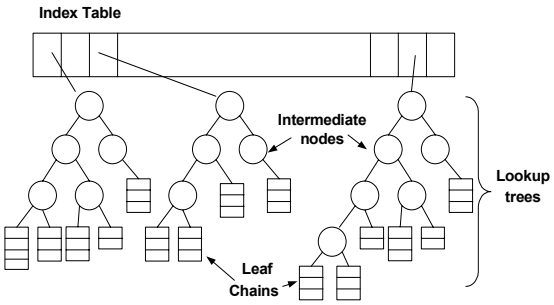

Figure 2: A modular classification data structure

In view of searching space, this model divides the searching space into several parts as follows:

Index table - All filters are grouped on proper bits of bit string. The entries of these groups, i.e. the entries of lookup trees, are stored in index table.

Lookup tree - The grouped filters construct a $2m$-ary lookup tree, searching m bits of the rule, and dividing into $2m$ groups. These m bits are chosen from the unvisited of current filter bit string, following two basic principles: decrease filter replication and balance the $2m$ sub trees. The building of lookup tree is a continuous course of filter grouping, and this course ends at leaf nodes, i.e. leaf chains.

Leaf chain - When the number of remaining filters is under a threshold, the decomposition stops. This node is called the leaf node. As there are not many filters in leaf nodes, we use another searching method, different from the method in lookup tree.

Intermediate nodes - The nodes positioning between root and leaf node are called intermediate nodes. Without function of filter storing, these nodes are only used for pattern searching.

After mapping the filter table in figure 1 into B, a two-dimensional array, and we define:

$B[i,.]$ - Elements in i-th row;

$B[.,j]$ - Elements in j-th column;

B^{-x} - x-th column deleted in B;

$B^{-x..y}$ - Columns x to y deleted in array B;

Wi - The cost of i-th row of B, i.e. the cost of Fi.

As to each column, e.g. j-th column, we give the following two formulas:

$$Nx_j(B) = \sum_{1 \le i \le n, B[i,j]=x} W_i \qquad (1)$$

Here, $X \in \{0,1,*\}$, and $Nx_j(B)$ is the sum of costs of all rows in B, whose j-th columns are X.

$$D_j(B) = \left| N0_j(B) - N1_j(B) \right| \qquad (2)$$

Here, $Dj(B)$ is the absolute value of the sum of costs of rows, whose j-th columns are 0, minus those with costs of 1 in j-th column.

As to the filter table FT, we relate the ordered pairs, i.e. $(N^*_j(FT), D_j(FT))$, to j-th column of FT. This relationship is shown in Figure 3.

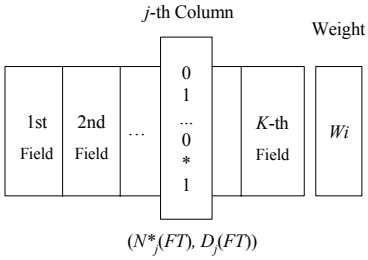

Figure 3: Concatenated view of packet filter table

6 Algorithm Description

The process of packet classification follows four steps, i.e. data structure initialization, new nodes insertion, selected nodes deletion, and proper filter searching. The former three steps are to create and maintain the whole structure, while the last does the actual classification.

6.1 Initialization Work

Suppose a k-dimensional filter table, FT, which can be viewed as a table-styled filter list. As to each dimension j ($1 \le j \le k$), field h_j is used to build the index table. This step has two important missions, i.e. to build the index table, and to build the lookup tree.

To build the index table, it will divide the original filter rule set into subsets according to h_j;

To build the lookup tree, it will try to enhance the searching performance following two principles:

I. Try to avoid filter replication, in order to save memory space;

II. Try to balance subtrees, in order to lower the mean searching time and the worst-case searching time.

6.2 Lookup Tree Building

This step focuses on the structure determination of lookup tree, following the two principles mentioned in previous section.

As to Principle I, i.e. for space saving, the replication often happens in the case that filters intercross. To keep the categoricalness and avoid searching back, we have to import some redundant filters, which may cost additional space. The trade-off between time and space complexities also needs to be considered.

As to Principle II, the memory access is a main bottleneck of tree's searching time. Hence, we can improve the efficiency by shortening the depth of tree, which may decrease the number of memory accesses. Actually this is also the best way to balance tree. Moreover, shortening the tree is also the key point of bit selection.

Analyzing the ordered pair $(N^*_j(FT), D_j(FT))$, where $N^*_j(FT)$ denotes the number of filters needed to replicate in the case that the j-th column of FT is selected to create index table. $D_j(FT)$ represents the balance extent of the lookup tree. Obviously, $N^*_j(FT)$ and $D_j(FT)$ represent space and time complexities respectively. The smaller they are, the better that tree is.

Then, we give the following evaluation formula to quantitate the preference.

$$preference[j] \quad \frac{D_j(FT) \quad D_{\min}(FT)}{D_{\max}(FT) \quad D_{\min}(FT)} \quad \frac{N^*_j(FT) \quad N^*_{\min}(FT)}{N^*_{\max}(FT) \quad N^*_{\min}(FT)} \quad (3)$$

Where $D_{min}(FT)$ is the minimum in $D_j(FT)$; $D_{max}(FT)$ is the maximum in $D_j(FT)$; $N^*_{min}(FT)$ is the minimum in $N^*(FT)$; and $N^*_{max}(FT)$ is the maximum in $N^*(FT)$.

In fact, we use a greedy algorithm in selection of j, i.e. every time select the best column, which has the current minimal preference value, to build the subtrees. This evaluation formula can be adapted to practical scenario by modifying the cost values. Therefore, this proposed strategy for tree building is flexible and general, since Wi can be given various meaning according to practical scenarios.

6.3 Leaf Chain Handling

A leaf chain is one of the basic elements in this algorithm. It has the following attributes:

- The filters positioning at the same leaf chain, should be relatively short.
- The filters positioning at the same leaf chain, should be as similar as possible.
- One filter may belong to multiple leaf chains, e.g. in the case that filters intercross each other in particular bits.

Due to the above reasons, to do the searching inside a leaf chain, it is not hard to find an appropriate method from present algorithms, e.g. linear searching, binary searching or CAM based searching.

6.4 Lookup Process

All fields of the packet header are considered as a bit string, called header string. The lookup process is explained below:

Step 1. Use specific bits of the header string as index, to query the index table and get the entry address of the corresponding lookup tree.

Step 2. Following the tree, trace until getting into one leaf node, as the value of 'm' (refer to $2m$-ary lookup tree) falls down every time.

Step 3. Inside the searched leaf node, find the filter with minimal cost.

7 Performance Analysis

This proposed modular classification model is not easy to analyze on time and space complexity. Its implementation is always affected by practical applied scenario, also its final performance parameters change with different filter set. For this reason, we try to analyze it by importing some additional parameters.

Let V denote the maximal capacity of leaf chain, and the root contain M filters. The time complexity can be ap-

proximated to be O(log(*M*/*V*)+*V*), in the simplest case that the filters are uniformly distributed in leaf chains of lookup trees, therefore, $M = N/2^{\sum h_k}$, here $2^{\sum h_k}$ denotes the number of bits selected to create the index table. And its space complexity is roughly $O(N(1+2/V)-2^{\sum h_k})$. Anyway, its complexity rockets quickly when the wildcard characters increase. In the worst case, that can be infinite. Also, *V* does opposite impacts in space and time aspects, i.e. it requires less space but costs longer time when *V* increases, and vice versa. Therefore, a trade-off should be considered. When this model is applied in practice, it needs some adjustments on parameters according to concrete demands and characteristics, in order to meet performance requirements.

Table 2 lists the time and space complexities of present classification algorithms.

Table 2: Performance Comparison of Classification Schemes

Algorithm	Worst time complexity	Worst space complexity
Linear search	N	N
Hierarchical trie tree	W^d	NdW
Pruning trie tree	DW	N^d
Grid-of-tries	W^{d-1}	NdW
Corss product	DW	N^d
RFC	D	N^d
Tuple Space	N	N
TCAM	1	N
Non-collision hash trie-tree	W^{d-1}	-
Bitmap-intersection	$DW+N$/(bits of memory unit)	DN^2
Our algorithm	$log(N/(2^{\sum h_k}V))+V$	$N(1+2/V)-2^{\sum h_k}$

N - Number of filters
W - Length of the bit string
D, d - Number of dimensions

Algorithms listed above the bottom row, are all designed under specific conditions, and applied in different scenarios. GOT is good at 2-dimensional prefix matching; RFC is strong in multi-dimensional application, and so on. Interested readers are referred to [16] for details of the implemented algorithms. These present algorithms lack of abilities of handling complex or flexible filters within economical resources. For example, some algorithms such as GOT etc., are only good at searching on two fields, i.e. source IP and destination IP; some can only deal with exact matching. And some, such as TCAM, are good at searching speed, but fail in implementation cost. Compared to these, our approach is good at filter adaptation and relatively small time and space complexities, i.e. log*N* and *N* level respectively.

8 Simulation

8.1 Simulation Environment

In order to approximate the actual performance of this algorithm, we developed a test bed built on IBM PowerNP4GS3. NP4GS3 is a programmable network processor optimized for packet processing at speeds up to OC48 (4Gbps). As superior as a switching and routing system on a single chip, NP4GS3 supports cost-efficient designs for high-level packet forwarding and filtering. NP4GS3 consists of two parts, i.e. central processor (CP) and network processor (NP), and NP4GS3 uses software-managed tree (SMT) as the classifier.

Figure 4 shows the process from tree building to searching execution. First, define the filter set for the simulation. Next, apply ChoiceBit algorithm to build lookup tree, i.e. the CP converts the filter set into a matrix, chooses the column with current minimal preference value to fill in the index table, builds SMT tree, and repeats this process until the filter set is empty. Then download a copy of the SMT tree with the same structure of nodes and leaves, to NP.

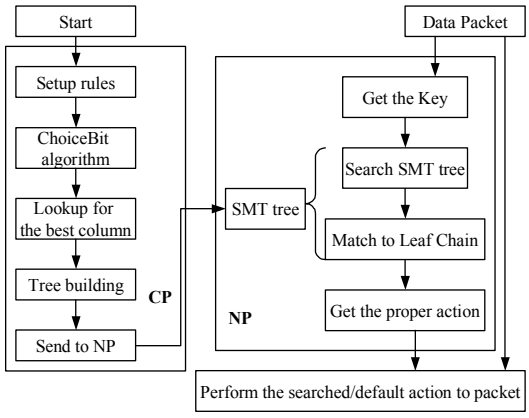

Figure 4: Flowchart for SMT Building and Search Mechanism

When a packet arrives, the header is extracted, and the classifier traces the SMT tree to find a matchable filter. The corresponding action will be performed if a result returns successfully, or a default action will be taken instead.

In this test, the packets are generated with special programs. The IP addresses of both source and destination follow a uniform distribution. And the protocol type follows the distribution of about TCP~35%, UDP~35%, ICMP~20%, others~10%, according to the statistics of real network traffic. Actually, the result data are obtained from NPSim, a soft simulator of NP4GS3. The filters are also generated with computers, in the five-tuple format as shown in figure 1. And the evaluation formula is denoted as follows,

$$preference[i] = (a0 - \frac{N}{2})^2 + (a1 - \frac{N}{2})^2 + (a* + ab)^2$$

Inside the leaf chain, we use linear searching to trace filters.

8.2 Simulation Result

Figure 5 shows the relationship between number of filter, and some attributes of lookup tree, namely the longest leaf chain, the number of leaf chains and intermediate nodes, in the case that there exists only one tree. This case means that index table is not valid at all.

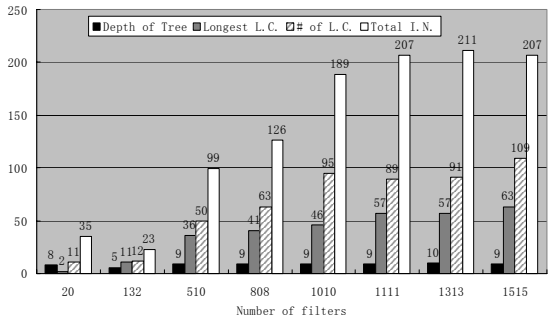

Figure 5: Relationship between the Number of Filters and Parameters of Lookup Tree

I.N. - Intermediate nodes

L.C. - Leaf chain

From above figure, we can see that the lookup tree is well balanced, i.e. with satisfactory depth and short leaf chains, when there are not so many filters. While in the case of more filters, the leaf chains grow very long and occupy much memory space, due to the intersection and dependency among these filters. When filters are over one thousand, the tree stops growing, as its depth and number of intermediate nodes and leaf chains change little. However, at the same time, the filters stored in leaf chains keep increasing, which can be witnessed from the number of filters in the longest leaf chain. This trend denotes that the lookup efficiency is decaying to be a linear searching.

To prevent this situation, a direct and simple solution is to put those filters into more trees. Theoretically, the filter set can be dynamically grouped so as to balance the distribution of filters within different trees. And the index table works as an index to locate the proper tree. In this scenario, our algorithm shows its superiority.

In the following simulation, the field of protocol type is chosen to create the index table, i.e. to classify the filters into four groups, namely TCP, UDP, ICMP and the others. Each group distributes its filters into one specific lookup tree. Actually, this step is just the process of "ChoiceBit" and "Tree building".

The test data, before and after grouping, are shown in figure 6. The result of the ungrouped in the case that the filters are over 1600, is not available.

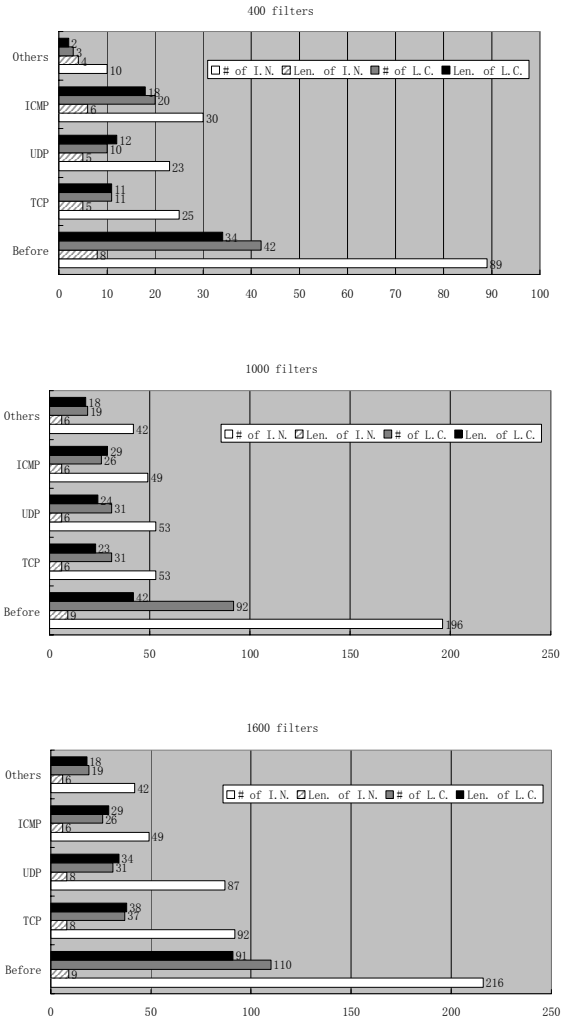

Figure 6 (1-4): Simulation Results

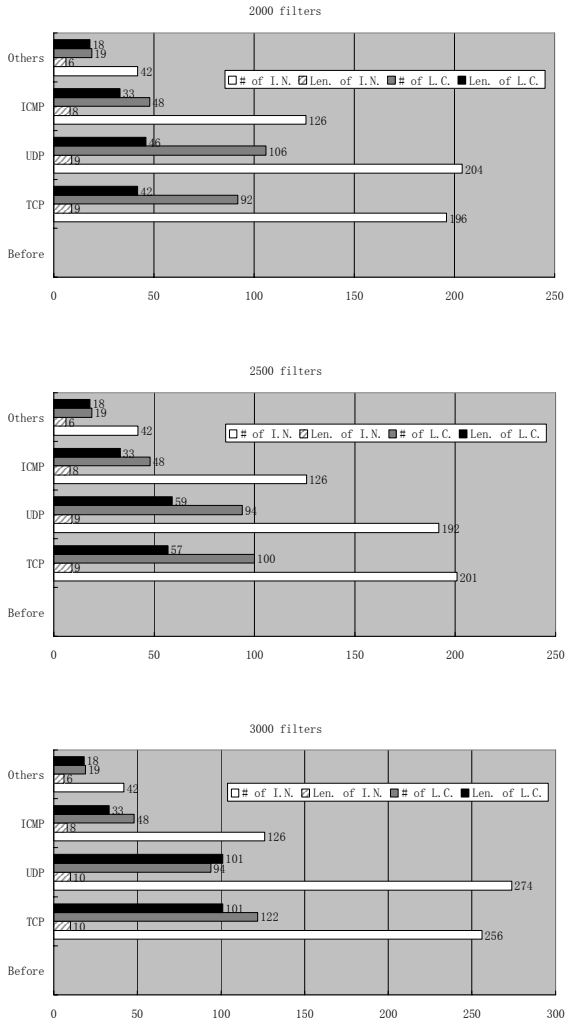

Figure 6 (5-6): Simulation Results

From figure 6, we can see that the longest leaf chain becomes shorter, because each tree has fewer filters to store than before. And the shorter depth will result in a shorter searching time. Anyway, another problem emerges as the number of filters keeps increasing. The groups of TCP and UDP congregate much more filters, due to the asymmetrical distribution of filters in different protocols. This trend will cause the decrease of efficiency.

9 Conclusion

Today, as internet has become more open and sharable, the requirement for differentiated and sophisticated service stands out increasingly. Thus, to find fast and adaptive packet classification algorithms, as a key factor to network management, is a hot topic.

This paper proposed an idea of combining the static top-down tree structure and tuple grouping strategy. Based on this hybrid approach, a classification algorithm is discussed, which groups the filters as several top-down lookup trees according to the inner dependency. Following the strategy of lowering dimensions, this algorithm unfolds selected header fields into index table to shorten the depth of lookup tree, and the final search is done inside the leaf chain, lying at leaf nodes. As to its implementation speed, running on the powerful NP chip, this algorithm with logarithmic time cost, is fast enough to deal with present packet flow without any loss. Also, by importing wildcards into filter formats, this algorithm is able to handle more flexible filters besides exact matching. In addition, the evaluation system based on filter cost, i.e. *Wi*, enables the arbitrary mechanism of filter selection.

Our future work is to refine and configure this algorithm into several schemes according to typical application scenarios. Besides, more simulations are also needed to test and validate its feasibility and practicability.

References

[1] Vijay, P. Kumar; T.V., Lakshman; D., Stiliadis, “Beyond Best Effort: Router Architectures for the Differenti-ated Services of Tomorrow's Internet”, *IEEE Commu-nication Magazine*, 1998, p.152-164.

[2] Yusheng, Ji; Shoichiro, Asano, “Virtual Rate-Based Queueing: A Generalized Queueing Discipline for Switches in High-Speed Networks”, *IEICE Transactions on Communications*, Vol.E77-B, No.12, p.1537-1545, 1994.

[3] T. V., Lakshman; Dimitrios, Stiliadis, “High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching”, *Proceedings of ACM SIGCOMM*, 1998, p.203-214.

[4] P., Warkhede et al., “Fast Packet Classification for Two-Dimensional Conflict-Free Filters”, *Proceedings of IEEE INFOCOM*, 2001, p.214-228.

[5] V., Srinivasan, “A Packet Classification and Filter Management System”, *Proceedings of IEEE INFOCOM*, 2001, p.1464-1473.

[6] T.Y.C., Woo, “A Modular Approach to Packet Classifi-cation: Algorithms and Results”, *Proceedings of IEEE INFOCOM*, 2000, p.174-185.

[7] H.Y., Tzeng, “Longest Prefix Search Using Compressed Trees”, *Proceedings of IEEE GLOBECOM*, 1998, p.8-12.

[8] Stanford High-performance Networking Group, http://klamath.stanford.edu/.

[9] Princeton Extensible Router, http://www.cs.princeton.edu/nsg/router.html.

[10] J.C., Mogul; R.F., Rashid; M.J., Accetta, "The Packet Filter: An Efficient Mechanism for User Level Network Code", Technical Report 87.2, Digital WRL, 1987.

[11] S., McCanne; V., Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", *Proceeding of USENIX Technical Conference*, 1994, p.259-269.

[12] M., Yuhara; B.N., Bershad; C., Maeda; J., Eliot; B., Moss, "Efficient Packet Demultiplexing for Multiple Endpoints and Large Messages", *Proceeding of USENIX Technical Conference*, 1994.

[13] M.L., Bailey; B., Gopal; M., Pagels; L.L., Peterson; P., Sarka, "PATHFINDER: A Pattern-based Packet Classifier", *Symposium on Operating Systems Design and Implementation*, 1994.

[14] S., Singh; F., Baboescu; G., Varghese; J., Wang, "Packet Classification Using Multidimensional Cutting", *Proceedings of ACM SIGCOMM*, 2003, p.213-224.

[15] P., Gupta; N., McKeown, "Packet Classification on Multiple Fields", *Proceedings of ACM SIGCOMM*, 1999, p.147-160.

[16] F., Baboescu; S., Singh; G., Varghese, "Packet Classifi-cation for Core Routers: Is there an alternative to CAMs?", *Proceedings of IEEE INFOCOM*, 2003.

[17] P., Gupta; N., McKeown, "Packet Classification using Hierarchical Intelligent Cuttings", *IEEE Micro*, Vol. 20, No.1, p.34-41, 2000.

[18] SiberCore Technologies Inc. Ultra TCAM Product Briefs. http://www.sibercore.com/.

[19] S., Iyer et al., "ClassiPl: An architecture for Fast and Flexible Packet Classification", *IEEE Network*, p.141-152, 2001.

[20] V., Srinivasan; S., Suri; G., Varghese, "Packet Classification using Tuple Space Search", *Proceedings of ACM SIGCOMM*, 1999, p.135-146.