Gear up the Classifier: Scalable Packet Classification Optimization Framework via Rule Set Pre-Processing

Kai Zheng[1,2], Zhiyong Liang[1], Yi Ge[1]
[1] IBM China Research Lab; [2]Department of Computer Science, Tsinghua University, P.R.China;
{zhengkai, liangzhy, geyi}@cn.ibm.com

***Abstract*—As one of the critical data path functions for many emerging networking applications, packet classification is gaining more and more concerns nowadays. It is commonly believed that conventional software-based classification algorithms are much more time-consuming than hardware-based solutions, i.e., the costly and power consuming TCAM-based mechanism, and incompetent for future high-end applications. In this paper, we propose an efficient optimization framework which can be applied to "gear up" most exiting software-based packet classification algorithms. Under this framework, the large rule set is pre-partitioned into several small subsets, according to some heuristics and dedicated methods. Then the conventional classification process can be significantly simplified and results in a distinct performance improvement by converging the classification power on only a small portion of the rule set. According to the results of our experiment, in which the framework is applied to one of the best algorithms EGT-PC [2], the memory accesses can even be reduced by up to 70%. This provides a much lower cost and more power-efficient alternative to TCAM-based solutions. Another advantage is that the framework requires no change to the hardware environment and little system cost overhead, making it especially suitable for the modern network processor based network solutions.**

***Key words*—Packet Classification, System Design, Framework**

I. INTRODUCTION

The rapid growth of the Internet and fast emergence of the new network applications have brought great challenges and complex issues in deploying high-speed and Qos guaranteed network. Packet classification is one of the basic critical data path functions for many networking applications, such as ACL, Firewall, and Qos Control, etc. It is important, complicated, yet not well solved.

Conventional software based solutions using binary trie or decision tree are said to be either time or storage consuming [1], and can not be easily scaled to support high-speed packet forwarding. On the contrary, some hardware based solutions, especially the ones using TCAM, are thought to be much more promising because of their deterministic and fast processing speed. The design of such schemes/mechanism is straightforward and easy to implement. All these phenomena seem to imply that there may not be alternatives to TCAM-based hardware solutions for high-end applications, despite many of their drawbacks, such as high cost to density ratio and very high power consumption.

However, a currently emerging and popularity-gaining concept *Network Processor* (NP) is going to change our viewpoints step by step. The concept of NP is firstly developed for the reason that the emergence of new network applications are much faster than the speed we develop hardware devices. So there should be a kind of re-programmable and high speed solution to meet the needs of today's network rapidly progressing in both functions and performance. NPs are specialized and programmable engines that are optimized to perform communication functions.

Equipped with abundant parallel processing resources, they can deliver hardware-level performance for software programmable systems. This powerful combination of performance and flexibility offers a revolutionary approach to the design of communication systems. It allows system designers to focus on higher-level services and ensures longer product lifecycles, rather than the conventional hardware solutions which simply meet the "speeds and feeds" of the moment. Since the widely adoption of NP, software based algorithms have again gained their popularity and become a hotspot in the literature.

According to our study, we find that by utilizing the abundant parallel resources of NP and partitioning the original rule set based on certain principles and heuristics, conventional software-based solutions can be optimized to be suitable for high-end applications, achieving much better performance without distinct cost. In this paper, instead of proposing new algorithms, we develop a novel concept, *Optimization Framework*; we are not seeking a way to discard old algorithms/solutions, but to inherit the strengths from them and cooperate with them. Firstly, we analyze and classify most kinds of rule set cutting methods. Then, by studying the characteristics of most existing algorithms and various real-world rule databases, we develop an efficient optimization framework based on rule set pre-cutting methods.

II. DEFINITIONS AND TERMS.

A. Definition: Rule, Key, Key Space, and (Hyper) Layer

A rule table, or a policy filtering table, includes a set of rules. A *Rule* is composed of a match condition and the corresponding action. Here we consider the matching condition as a typical five-tuple including five packet header fields, which, in sequence, are DIP(1-32), SIP(1-32),

Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06)
0-7695-2588-1/06 $20.00 © 2006 IEEE

DPORT(1-16), SPORT(1-16), and PROT(1-8)[i].

A *Search Key* is a 104-bit[ii] string composed of the five IP packet header fields.

Accordingly, in a geometrical view, the *Key Space* is actually a hyper-space with multi-dimensions; a specific search key represents one point in the hyper-space, while each rule represents a hyper-cube in the key space.

In order to introduce the following concepts, we develop a related term (*Hyper*) *Layer*. Note that rules may overlap with each other on some dimensions. (Hyper) Layer is defined as a set of rules that does not overlap with each other on specific dimensions/field, as depicted in Fig.1. Then the key space can be partitioned into several (hyper) layers.

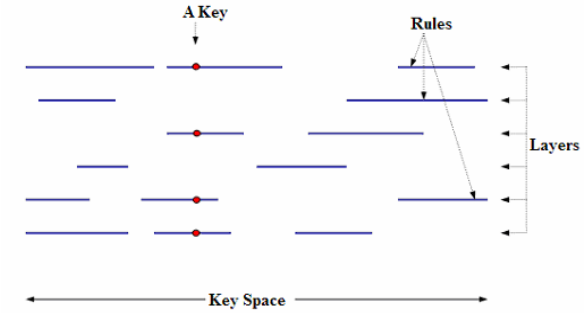

Fig.1 The concepts and relationships of Key Space, Key, Rules, and Layers in a 1-D case. 15 rules are included in the example. The figure shows one of the layer-partition schemes, which includes 6 layers. And there are 4 rules matching the given key, which are indicated by the solid points.

B. Definition: Rules Set Cutting and Key Space Partitioning

Let R be a specific rule set. If a set of its subsets $\{Q_{j\,,\,j=1,\ldots,N}\}$, satisfies $\forall Q_j \subset R, j=1,\ldots,N$, and $\bigcup_{n=1,\ldots,N} Q_n = R$, then we call $\{Q_{j\,,\,j=1,\ldots,N}\}$ an *N-Cutting* of rule set R.

Let $\overline{S}$ be the key space, $k, k \in \overline{S}$ be one of the Keys within the key space. Given a function of the keys: $F: \overline{S} -> \{1,2,\ldots,M\}$, which map the keys into M subsets, $\{K_i, i=1,..,M\}$, where $K_i = \{k \mid k \in \overline{S}, F(k)=i\}$, and $\bigcup_{i=1,\ldots,M} K_i = \overline{S}$. Then we call $\{K_i, i=1,..,M\}$ an *M-Partition* of the key space, and F the corresponding *M*-Partition function. (Note that $\forall i, j, i \neq j, K_i \cap K_j = \phi$)

C. Definition: Completeness

1. Completeness of a rule subset to a Key and Key partition

We say a subset Q_j of rule set R is complete to a key k, if Q_j contains all of the rules in R that matching k, which is denoted as $k \overset{R}{\in} Q_j$; Further, for a given key partition K_i, if Q_j satisfies, $\forall k, k \in K_i, k \overset{R}{\in} Q_j$, we say Q_j is complete to K_i within rule set R, which is denoted as $K_i \overset{R}{\subset} Q_j$.

The completeness of a subset Q_j indicates that to traverse Q_j and find the matching rules of any given key $k \in K_i$ is a sufficient condition of finding the matching rules of k in the whole rule set R.

2. Completeness of Rule cutting

Given an *M*-Partition function F of the key space $\overline{S}$, $\overline{S}$ is partitioned into $\{K_i, i=1,..,M\}$. For a rule set R, if an *N*-Cutting of R, $\{Q_{j\,,\,j=1,\ldots,N}\}$, satisfies:

$\forall K_i, i=1,\ldots,M, \exists Q_j, j=1,\ldots,N, K_i \overset{R}{\subset} Q_j$, then we call $\{Q_{j\,,\,j=1,\ldots,N}\}$ is an complete (completeness-guaranteed) *N*-Cutting of R under key partitioning function F.

Completeness of a rule Cutting $\{Q_{j\,,\,j=1,\ldots,N}\}$ ensures that for any given key, to traverse only one specific rule group is sufficient and correct.

III. Packet Classification Optimization Framework Based on Rule Set Cutting

A. The Overall Idea of the Optimization Framework

Actually, in a general word, the packet classifier is a kind of filter. It filtrates the rules from the original rule set and finds out the ones matching the input key. Most existing well-known multi-fields classification algorithms, such as EGT-PC [2] (an extension of [8]), HiCuts [5], HyperCuts [6], Modular [7], etc., are very similar in their basic ideas. They usually include 2 steps: First, an efficient filtering algorithm (e.g., decision tree based algorithms) is adopted in certain fields to filtrate the rules and the target rules are then focalized in a reduced subset, called the *Interim Set*. Secondly, since the Interim Set is relatively much smaller, still adopting the complex filtering algorithm may, on the contrary, be less efficient. So usually a simple algorithm, e.g., linear search, is employed to match with the key within the Interim Set and find out the final matching result.

The idea of the proposed optimization framework is as shown in Fig.2. Since policy rules are usually "stable" and with much lower modification frequency, which indicates that it would neither cost much nor do harm to the overall performance if we employ some pre-processing of the rule set to optimize the search performance. Based on this observation, we try to cut the rules into several groups based on certain principles in advance. And then, according to the key, we may focalize the filtering algorithm within only one of the subsets via a simple hashing. Hence the classification workload will become much lighter and the size of the Interim Set will be further reduced as well, resulting in a distinctive improvement of the performance.

However, we also note that to divide the original rule set into several sub-sets/groups for classification optimization is NOT a trivial work. Since we should ensure not only the completeness of the sub-set to any input key, but also that no rules be duplicated and that the process to find the corresponding subset of the key be simple.

[i] DIP, SIP, DPORT, SPORT, and PROT represent Destination IP prefix, Source IP prefix, Destination Port range, Source Port range, and PROTocol number, respectively.

[ii] 32+32+16+16+8=104.

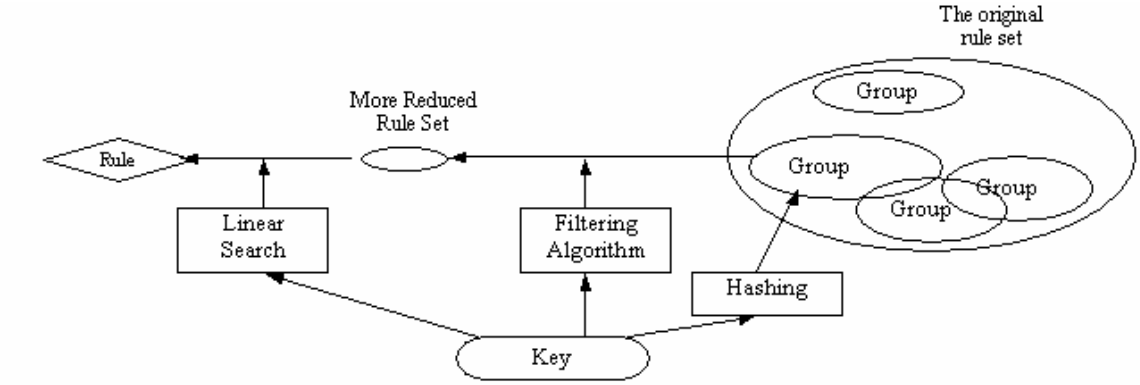

Fig. 2 The optimization idea based on rule cutting

B. Rule Set Cutting methods.

Before presenting the details of the optimization framework, first let's begin with the introduction of two kinds of rule set cutting methods. Fig. 3 depicts the examples of Complete and Incomplete rule cutting methods.

1. Algorithms of V-Cuts

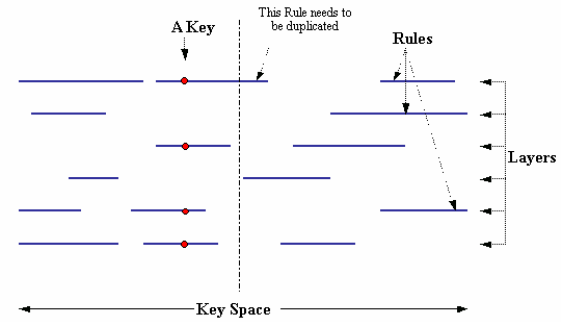

Fig.3 V-Cuts: A complete cutting

Fig.3 shows a rule cutting scheme based on key ranges/intervals (the rules on the left side of the cut belong to one group while the ones on the right side belong to the other. The ones across the cut would be duplicated and belong to both groups). It is straightforward that this kind of cutting ensures completeness, since all rules that matching an arbitrary key are allocated in the same subset. Visually, we name such kind of rule cutting methods as **Vertical Cuts**, or **V-Cuts**. This kind of cutting method actually includes two sub-procedures: first, to cut the original rule set into groups; second, to provide a mapping function for the input key, so as to map it to its complete rule group.

V-Cuts based on range intervals

The V-Cuts method that appears in Fig.3 belongs to this kind. The rules are grouped based on range/prefix intervals on one specific field (within its matching condition). This method is easy to implement and can be applied to all kinds of packet fields (i.e., prefix, range, or exact matching fields).

V-Cuts based on ID

The concept of ID is firstly introduced in one of our previous work [9]. As far as a rule is concerned, its *Rule-ID* is defined as a *P-bit* ternary ('1','0', or '*') bit-string, each bit of which is extracted from certain bit position within the rule's matching condition. Accordingly, the term *Key-ID* is defined as the P binary bits ('1'or '0') extracted from the corresponding bit positions within a given key. For example, suppose P=2, and the 2 bit positions are SIP(16) and PROT(8). Then the Rule-ID of rule <1.1.*.*, 2.*.*.*, *, *, 6> is"*0"and the Key-ID of search key <1.1.1.1, 2.2.2.2, 1028, 34556, 11> is"01".

The rules can then be divided into several groups according to their Rule-IDs. For instance, the rules with Rule-ID (suppose P=2) '*0' belong to both Key-ID Group '10' and Key-ID Group '00'. This method is also complete, since the rules matching one arbitrary key must match the same Key-ID, and therefore they must belong to the same Key-ID group. This method is only suitable for the packet fields which is in the form of prefix (i.e., DIP/SIP) or exact matching (i.e., PROT), since a port range may not be represented by a ternary bit string.

2. Algorithms of H-Cuts

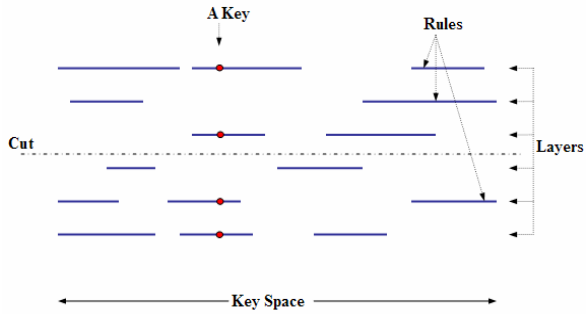

Fig.4 H-Cuts: A complete cutting

Fig.4 shows a rule cutting scheme based on layer-partition (the rules above the cut belong to one group and the ones below the cut form the other). And it is straightforward that this kind of cutting is not complete. Because the rules

matching a given key are separated into different groups; to traverse either of the groups may be insufficient for a given key. We denote such kind of rule cutting in a visual way as **Horizontal Cuts**, or **H-Cuts**. Compared with V-Cuts, H-Cuts are not complete, implying that multiple or even all rule groups generated by the H-Cuts may need to be traversed to ensure completeness.

H-Cuts based on (hyper-) layers

The H-Cuts method mentioned in Fig.4 belongs to this kind. Firstly, the hyper-layer hierarchy is formed, and then all the rules are categorized into layers.

H-Cuts based on prefix level

Firstly, we need to introduce the concept of *Prefix Level*, which is originally introduced in one of our previous work [10]. As the example shown in Fig.5, for a given prefix node *n*, there are multi paths from node *n* to its descendant leaf nodes. Among these paths, let P_{max} be the path containing the most prefix nodes. The number of prefix nodes (excluding node *n* itself) in P_{max} is called the *Prefix Level* of node *n*. For instance, node *g* in Fig.5 is with Level 0, and node *d* is with Level 2, etc. And then we can further cut the ten prefixes into 3 groups according to their prefix Level, which is called *Level Set*, i.e., LevelSet(0)={b, c, g, f, h, i, j}, LevelSet(1)={a, e}, LevelSet(2)={d}. This kind of H-Cuts is efficient, however it is only suitable for the packet fields in prefix form.

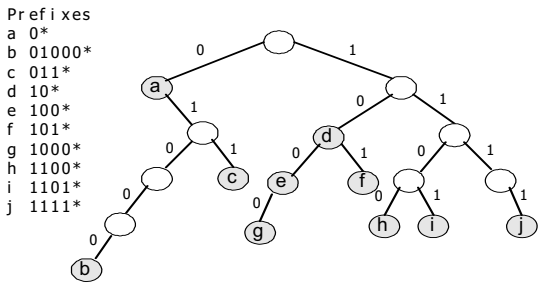

Fig.5 A prefix trie example

C. The Packet Classification Optimization Framework

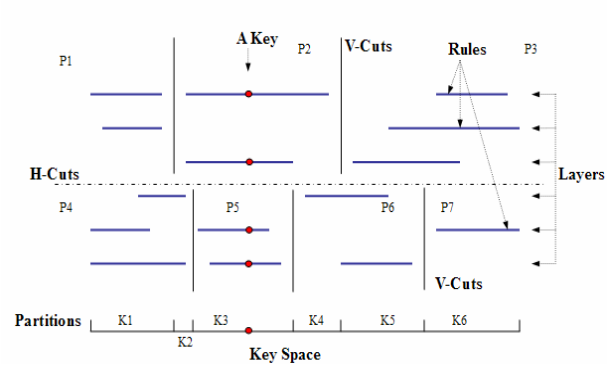

Fig 6 An example combining both V-Cuts and H-Cuts to group the rules

According to the introduction of the cutting methods in the previous sub-section, simple H-Cuts can not ensure completeness while simple V-Cuts may incur huge storage redundancy. The only way to achieve the goal is to combine the two methods, adopting their strength points accordingly.

As the example shown in Fig.6, after forming the (hyper) layer hierarchy, the cutting approach starts from an H-Cuts, it divides the rules into 2 groups according to their layers. This H-Cut eliminates/reduces the possibility of duplications caused by the V-Cuts. Without the H-Cut (i.e. each V-Cuts should be performed in all of the 6 layers), the V-Cuts would always cut through some rules and incur rule duplications in this example, no matter at which point/interval we launch them.

Then 2 V-Cuts are performed within the upper sub-set while three within the lower one, without any rule duplication. Then the original rule set is divided into seven sub-sets $\{P_1, P_2, ..., P_7\}$.

On the other hand, the five V-Cuts partition the key space into six partitions $\{K_1, K_2, ..., K_6\}$ (as shown in Fig. 6). By a simple combination, we can get a cutting scheme $\{ Q_1 = P_1 \bigcup P_4 , \; Q_2 = P_1 \bigcup P_5 , \; Q_3 = P_2 \bigcup P_5 , \; Q_4 = P_2 \bigcup P_6 , \; Q_5 = P_3 \bigcup P_6 , \; Q_6 = P_3 \bigcup P_7 \}$, in which each rule group Q_i is complete to the corresponding key partition K_i, $i = 1,2,...,6$. No rule duplication is required at all.

The classification for any given key now can be done within only a group of subsets of the rule database. For instance, in the example shown in Fig.6, to classify the given key we only need to search rule group Q_3, i.e., subset P_2 and P_5, which may be searched independently in serial or in parallel; namely, 11 rules out of 15 are already excluded for classification. Note that the cuttings are done in pre-computation, and the only processing overhead during the classification is one very simple hashing to map the input key to the corresponding rule group.

The pseudo-code of the optimization framework is as the following:

i. Select the appropriate H-Cuts method and packet fields to perform rule set cutting, according to the target classification algorithm.
ii. Form the (hyper) layer hierarchy on the selected packet field(s), according to the selected H-Cuts methods.
iii. Perform H-Cuts and divide the (hyper) layers into several groups[iii].
iv. Perform V-Cuts at the intervals where would not incur rule duplication, respectively within each layer group. And then result in a series of rule sub-sets.
v. Group the partitioned subsets and get the complete rule groups for each key partition. Then form the hash table.
vi. Apply the target classification algorithm to each rule subset and construct the corresponding data structures, respectively.

[iii] The selection of the number of layer-groups depends on the number of classifiers provided. Although partitioning more layer-groups (i.e., performing more H-Cuts) may result in fewer overlaps in each layer-group so that more V-Cuts may be performed to reduce the size of each subset; however, more layer-groups may also lead to more subsets be inspected in serial if the classifier is less than the layer-groups.

IV. PERFORMANCE EVALUATION

A. Optimization performance evaluation

According to the analysis in the previous sections, we notice that the performance of the proposed optimization framework is dependent on the target algorithm as well as the characteristics of the rule sets, which may vary largely case by case. Therefore it would be hard for us to provide very accurate theoretical analysis on the optimization performance. In the follows, we estimate the average-case performance of the optimization framework in term of the time and storage complexity.

Given that we have h parallel classifiers and h-1 H-Cuts are performed within the target rule set, which contains N rules. The H-Cuts result in h H-Cuts groups. Further, v_1, v_2,…,v_h V-Cuts are performed within the h H-Cuts groups respectively. Assume that the number of each resulting subsets is similar with each other, so each subset contains approximately $N / \sum_{i=1,\ldots,h} v_i$ rules.

Suppose that the time and storage complexity of the target algorithm are $O[T(N_r)]$ and $O[S(N_r)]$ respectively, where N_r is the number of rules. Tab.1 presents the comparison of the performance between different schemes.

Scheme	Time	Storage
Native Algorithm	$O[T(N_r)]$	$O[S(N_r)]$
Duplicating Rule Set	$O[T(N_r)/h]$	$O[h \times S(N_r)]$
+Optimization Framework	$O[T(N_r / \sum_{i=1,\ldots,h} v_i)]$	$O[S(N_r)]$

Tab.1 Performance comparison between three schemes

We see that for the native algorithm, though we have h classifiers, since the completeness of each classification must be guaranteed, all rules should still be inspected in sequence, and no speedup is gained.

For the case that the whole rule set are duplicated to h memory chips, the rules can, therefore, be accessed by the h classifiers in parallel, resulting in a speedup of approximately h; however, remember that this is at a cost of distinct storage overhead.

In the case of employing the optimization frame work, no rule is duplicated, and to guarantee the completeness of classification, only h subsets out of $\sum_{i=1,\ldots,h} v_i$ should be queried, which can be accessed in parallel by the h classifiers. Therefore the classification latency is reduced to approximately that for single subset. Note that all these gains are contributed by the job done in pre-computation.

B. Experiment results

We use the packet classification algorithm evaluation tool-set "Class-Bench" (developed by Taylor etc. from Washington University [3]) and the source code of the EGT-PC algorithm (presented by the authors) to evaluate the performance of proposed framework. We focus our energy on the optimization caused by the framework.

Tab.2 shows the experiment results. For the sake of checking the universality of the framework, we select all three kinds of rule sets, i.e., Access Control List (*acl*), Firewall (*fw*), and IP Chain (*ipc*). In these cases we use the SP and PRTO fields to perform rule set pre-cutting. Only one H-Cut is performed in all cases (supposing that only two classifiers are provided). Since memory access is commonly much slower than the processor and always the performance bottleneck of software-based packet classification, we assume that the overall performance is mainly determined by memory accesses.

According to the experimental results, as shown in Tab.2, we find that with the optimization framework, the number of memory accesses is dramatically reduced. For instance, for the *fw* and *acl* sets, the number of memory accesses is reduced by more than 70% when C=2. Even without multiple parallel classifiers (i.e., when C=1), the performance can also be distinctively improved.

All of the pre-cutting process and EGT-PC trie construction are done within a few seconds (<10s) using a 1.8GHz Intel Pentium IV-m laptop. This shows that the pre-computation work load is trivial, which does not cause substantial impact on the classifier(s) (e.g., co-processors of the NP).

C. The updating issue

First of all, for rule deletions, employing the framework will not cause the disability of incremental update so long as the target algorithm supports incremental update.

For rule insertion, it may possibly introduce new ranges which cross the intervals where some V-Cuts are performed, and therefore incurs rules duplication. However, this case happens only when the new ranges cross V-Cuts in all of the layer groups. In this case, the V-Cuts which cross the new rule ranges will be cancelled and the associated subsets will be combined. And then the corresponding data structure of the target algorithm should also be re-constructed.

Fortunately, since the rules are commonly modified infrequently, the update will not be a very critical problem even if incremental method is infeasible in some extreme cases.

V. CONCLUSIONS

In this paper, we proposed an optimization framework for packet classification algorithms, which can be employed in most NP environments and cooperate with most existing algorithms. By developing two kinds of rule cutting methods (i.e. V-Cuts and H-Cuts), pre-partitioning the original rule set into a series of sub-sets according to some optimization principles/heuristics, and focalizing each classification within only a small portion of the rule set, the framework can dramatically improve the performance of most algorithms without rule duplication. The optimization framework does not impact the hardware environment of the classifier and can be easily implemented.

Rule Set	Rule set features					Scheme	*M.H*	*Avg. N.A*	*Avg. R.M*	*Avg. M.A.C**	*M.A.C Reduced*
	R	***D.R.***	***L***	***S.S.***	***T***						
acl	6982	227	13	55	209460	EGT-PC	0	24.8	24.9	74.5	--
						+Framework *C*=1	4	15.7	5.5	30.7	58.8%
						+Framework *C*=2**	2	10.6	4.6	21.7	70.9%
fw	7881	43	6	39	236430	EGT-PC	0	21.4	96.7	214.8	--
						+Framework *C*=1	4	21.9	23.1	72.2	66.4%
						+Framework *C*=2	2	17.1	21.5	62.1	71.1%
ipc	6753	53	7	37	202590	EGT-PC	0	15.3	9.6	34.6	--
						+Framework *C*=1	4	17.7	4.7	31.1	10.1%
						+Framework *C*=2	2	12.1	4.6	23.3	32.7%

Tab.2 Experiment results and comparison between adopting the proposed framework and not adopting

R : number of Rules; *D.R.*: number of Distinct Range in the source port field; *L* : number of Layers; *S.S.*: number of Sub-Sets divided. *T* : number of packet Trace used. *M.H* : number of Memory accesses needed for range Hashing; *N.A* : number of trie Node Accessed; *R.M.* : number of Rules Matched with at the leaf nodes; *M.A.C* : number of Memory Accessed Cycles; *C* : number of Classifiers.

*: Supposing that each N.A takes one memory access cycle while each R.M takes two, according to typical data structures.

**: Note that for each classification, 2 EGT-PC tries should be accessed. In the case of C=2 (two classifiers work in parallel), the corresponding index are accumulated according to the classifier performing more memory accesses. While in the case of C=1, the indexes are accumulated by summing up the numbers of both the two classifiers.

The future work of our team will focus on the detailed implementation of IPv6 scheme and the framework extension to support other pattern matching problems, such as IDS/IPS or other security associated issues.

VI. Related Works

Due to its high complexity of the search and widely adoption as the critical data path function in IP packet forwarding, packet classification is often a performance bottleneck in network infrastructure and it has received much attention in the research community. Besides the Decision Tree based algorithms [2] [5-8] we mentioned in the previous sections, several other kinds of algorithms or hardware schemes are proposed, recently.

One category is the ones via Decomposition [11-12]. The idea is to decompose the multiple field search into instances of single field searches, and perform independent searches on each packet field, then combine the results. The latest work on this category is the DCFL (Distributed Crossproducting of Field Labels) [11], which presents a combination of new and existing packet classification techniques that leverages observations of the structure of real filter sets and takes advantage of the capabilities of modern hardware technology. High performance is achieved at the cost of high implementation complexity. Another category is the hardware based ones [4] [9], in which TCAM is adopted. High performance is gained at the cost of high price and low power efficiency, supposing that the range matching problem of TCAM can be perfectly solved. However, the major problem of such algorithms/schemes is not their performance, but their flexibility for the NP environments, that they can hardly be transplanted smoothly from one system to the other, and is with much higher re-developing cost and longer time-to-market. These are the reasons why we based our work on the Decision Tree based software algorithms, as far as the NP environment is concerned.

VII. References

[1] P. Gupta and N. McKeown, "Algorithms for Packet Classification", IEEE Network, March/April, 2001

[2] F. Baboescu, S. Singh, G. Varghese, "Packet Classification for Core Routers: Is there an alternative to CAMs?", Proc. of IEEE INFOCOM, San Francisco USA, March 2003.

[3] ClassBench, a classification algorithm evaluation tool set, available at: http://www.arl.wustl.edu/~det3/ClassBench/

[4] K. Lakshminarayanan, A. Rangarajan, S. Venkatachary,, " Algorithms for Advanced Packet Classification with Ternary CAMs ", Proc. of ACM SIGCOMM '05, Philadephia USA, Aug 2005

[5] P. Gupta, and N. McKeown, "Packet Classification using Hierarchical Intelligent Cuttings", IEEE Micro Magazine, Vol. 20, No. 1, pp 34-41, January- February 2000.

[6] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting", Proc. of ACM SIGCOMM 2003.

[7] T.Y.C.Woo, " A Modular Approach to Packet Classification: Algorithm and Results," Proc. of IEEE INFOCOM, March 2000.

[8] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching", Proc. of ACM SIGCOMM '98.

[9] K. Zheng, H. Che, Z. Wang, and B. Liu, " TCAM-based Distributed Parallel Packet Classification Algorithm with Range-Matching Solution", Proc. of IEEE INFOCOM, Miami, USA, March 2005.

[10] Z. Liang, K. Xu, J. Wu, "A Scalable Parallel Lookup Framework Avoiding Longest Prefix", Lecture Notes in Computer Science, vol. 3090,2004, pp.616-625.

[11] D. E. Taylor and J. S. Turner, "Scalable Packet Classification using Distributed Crossproducting of Field Labels", Proc. of IEEE INFOCOM'05, Miami, USA, March 2005

[12] F. Baboescu and G. Varghese,"Scalable packet classification", IEEE/ACM Transactions on Networking, v 13, p2-14, Feb, 2005.