Toward Predictable Performance in Decision Tree based Packet Classification Algorithms

Peng He*†, Hongtao Guan*, Laurent Mathy§, Kavé Salamatian‡ Gaogang Xie*
*Institute of Computing Technology, Chinese Academy of Sciences, China
†Graduate University of Chinese Academy of Sciences, China
‡LISTIC-PolyTech, Université de Savoie, France
§University of Liège, Belgium

***Abstract*—Packet classification has been studied extensively in the past decade. While many efficient algorithms have been proposed, the lack of deterministic performance has hindered the adoption and deployment of these algorithms: the expensive and power-hungry TCAM is still the *de facto* standard solution for packet classification. In this work, in contrast to proposing yet another new packet classification algorithm, we present the first steps to understand this *unpredictability* in performance for the existing algorithms. We focus on decision-tree based algorithms in this paper. In order to achieve the predictability, we firstly revisit the classical and many state-of-art packet classification algorithms. Through a detailed analysis, we conclude that two features of ruleset usually dominate the performance results: 1) the *uniformity* of the range distribution in different dimensions of the rules; 2) the existence and the number of *"orthogonal structure"* and wildcard rules in the ruleset. We conduct experiments to show the correctness of these observations, and discribe some potential applications for those results. Our work provides some insight to make the packet classification algorithms a credible alternative to the TCAM-only solutions.**

***Index Terms*—Predictability, Packet Classification, Decision Tree Algorithms.**

I. INTRODUCTION

Packet classification is a key functionality for both routers and network middleboxes to provide advanced network services. Although it has been studied extensively in the past decade, the growing size of classifier encourages a renewed interest in the study of packet classification[3][9]. On one hand, because of the need for a finer grained processing in traditional network applications, such as VPN, load balancer, firewall, traffic engineering etc, the size of classifier is becoming larger and larger. According to a recent study[3], the number of packet classification rules in a typical ISP network has grown to 15K. On the other hand, the current trend of moving network services, such as the firewalling, into the cloud[2], requires consolidating multiple rulesets from different customers, potentially increasing the size of classifiers. This growing size and complexity of rulesets causes issues for packet classification systems.

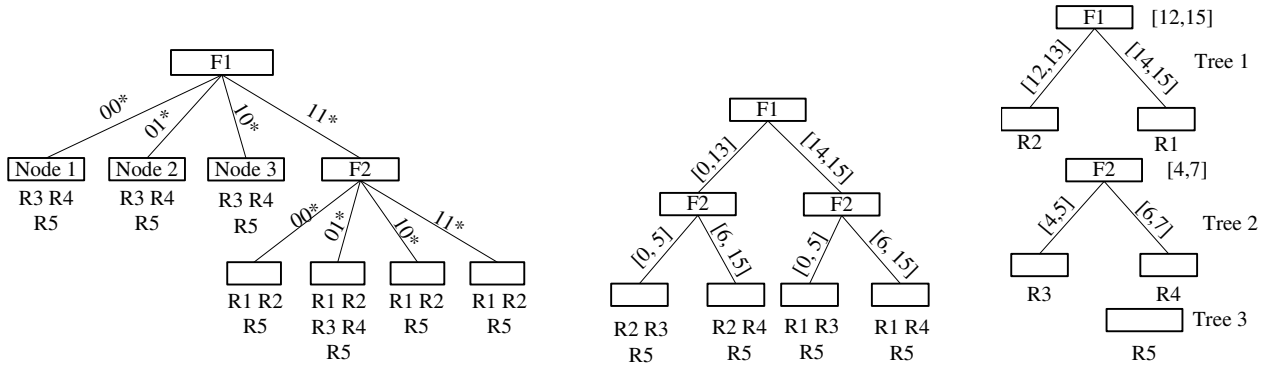
Previous packet classification solutions can be categorized into two types: RAM-based and TCAM-based. RAM-based solutions, also known as algorithmic solutions, through building compact and efficient data structures for classification rules in fast and cheap memory, can usually achieve a better price/performance ratio and lower power-consumption than TCAM-based solutions. However, because TCAMs guarantee deterministic performance, these expensive and power-hungry devices are still the *de facto* standard solution for packet classification. As the size of the classifier continues to grow, network device vendors have to spend more to enlarge the volume of TCAM, which increases both the cost and power-consumption of their products. In this paper, we argue that the lack of an effective method to predict the performance of different rulesets is the key problem which prevents the deployment of RAM-based solutions. By performance, we mean the number of the worst case memory accesses (which ultimately governs feasible data rates) and memory size required by one algorithm. We begin with a simple example to illustrate this *unpredictability* issue.

TABLE I
PERFORMANCE COMPARISON ON DIFFERENT RULESETS

algorithm	ruleset(num)	memory size	tree depth or mem. accesses
HyperSplit[5]	fw1_10K(9396)	770MB	33
	fw2_10K(9616)	23MB	24
	fw2_1K(965)	221KB	15
	fw3_1K(792)	221KB	19
HyperCuts[6]	fw2_1K(965)	150KB	6
	fw3_1K(792)	6MB	22
EffiCuts[9]	fw1_10K(9396)	610KB	73

Table I gives the performance results of different algorithms on different rulesets. Results are clearly vastly different. For two firewall rulesets, fw1_10K and fw2_10K, containing nearly equal number of rules, the memory size of fw1_10K is around 30 times larger than fw2_10K. For ruleset fw2_1K, HyperCuts and HyperSplit achieve nearly equal memory consumption, however, HyperSplit needs around 2x more memory accesses than HyperCuts. For another ruleset fw3_1K, the performance of HyperCuts is worse than HyperSplit again. Although the memory size of EffiCuts on ruleset fw1_10K is small, it requires around 2x more memory accesses. This unpredictability in performance is an issue for both network device vendors and users alike.

A naive method to overcome this problem is to simply compare the performance of the various algorithms on a given ruleset. However, network systems often do not have enough resources to carry out such comparison, not to mention that



(a) HiCuts (b) HyperSplit (c) EffiCuts

Fig. 1. The built decision trees by different algorithms

some algorithms have very long preprocessing times (according to [5], the HiCuts[1] algorithm needs more than 24 hours for some large rulesets). Meanwhile, such method cannot provide answers to the following related questions either: 1) Why the performance on different rulesets is so different? 2) Which features of rulesets dominate the result of performance? How can we measure these features? 3) For a specific ruleset, how can we make an algorithm recommendation by using the measurement results of these features ? 4) Can we propose a hybrid RAM-based solution by splitting rulesets into subsets, and choosing the right algorithm for the subsets? Or can we build a hybrid TCAM and SRAM packet classification engine? Which subset of rules need to be put on TCAM, and which subset of rules should be put on SRAM using algorithms?

In order to make headways towards increased performance predictability and answers to these questions, we firstly revisit the classical and most efficient state-of-art Decision-Tree (DT) based algorithms, and investigate what the key features of rulesets are and how they affect performance. We have found that: 1) the uniformity of the range distribution in the rulesets can affect the number of memory accesses dramatically; 2) the existence and the numbers of "orthogonal structures" and wildcard rules can affect the memory size significantly. We use ClassBench[8] to generated the synthetic rules, and validate our results through experiments. We also describe some potential applications based on the measurement results in the end of this paper.

The rest of this paper is organized as follows: Section 2 provides some background on the DT (decision-tree) based algorithms. The key observations of the ruleset features are presented in Section 3, and the measurement results are shown in Section 4. We provide some potential applications and discuss the future work in Section 5.

II. BACKGROUND

In this section, we introduce the basic idea of different DT algorithms with one example ruleset. We make a detailed comparison between these algorithms before we introduce the key observations in the next section.

TABLE II
AN EXAMPLE CLASSIFICATION RULESET

Field 1	Field 2	Action
111*	*	DROP
110*	*	PERMIT
*	010*	DROP
*	011*	PERMIT
*	*	PERMIT

A packet classification ruleset can be viewed as a combination of different *ranges* on different fields. To note, in this paper, we use the term *field* and *dimension* interchangeably. Table II shows one example of one two dimensional classification ruleset. Each field has 4 bits, which means the the value in each field can be from $[0, 15]$. On Field 1, we have three unique *ranges*, which are $[14, 15], [12, 13], [0, 15]$ and Field 2 also has three: $[4, 5], [6, 7], [0, 15]$.

Figure 1 shows the built decision trees of different algorithms on this example ruleset.

A. HiCuts and HyperCuts

We begin with two classic DT algorithms: HiCuts and HyperCuts. Since they are related, we only show the HiCuts decision tree in Figure 1(a). The HiCuts algorithm builds the decision tree by choosing a single dimension of rules to separate rules. As shown in Figure 1(a), in the HiCuts decision tree, Field 1 is cut into four equal-sized sub-space — $[0, 3], [4, 7], [8, 11], [12, 15]$. We call this kind of cutting scheme the *equal-sized cutting*. Since the ranges in the Field 1 are aggregated with prefix 11*, 4 equal-sized cuts cannot separate R1 and R2. More cuts are needed on the fourth children of the root node, which brings more memory accesses. However, since the ranges in Field 2 are also aggregated with prefix 01*, four equal-sized cuts still cannot separate R1 and R2 rules. As shown in Figure 1(a), in the process of cutting fields, because of the skewed distribution of ranges, we have created a lot of identical nodes. For example, the first three child nodes of the root node, (from left to right), all contains rules R3, R4 and R5. While HiCuts algorithm can

use the *Node Reuse* optimization technique to merge these three nodes, but the extra pointer array required by the node merging would also increase the memory size. According to [9], about $30\% \sim 50\%$ of total memory usage is caused by the storage of the child pointer arrays.

HyperCuts extends HiCuts by allowing multiple fields to be cut in one node. However, since HyperCuts still adopts an *equal-sized cutting* strategy, it still suffers from inefficiencies when the distribution of ranges in the chosen dimension is skewed. Because realistic packet classification rules are used to filter a small amount of packets in the traffic, they are usually very specific, or in other words, the distribution of the ranges is usually skewed. This skewness makes HiCuts and HyperCuts inefficient on realistic ruleset.

B. HyperSplit

In order to overcome the skewness problem of HiCuts and HyperCuts, HyperSplit[5] adopts a different method to separate rules. It picks a single point in the chosen field to split the space into two unequal-sized sub-spaces which contain nearly equal number of rules. As shown in Figure 1(b), the HyperSplit algorithm splits the Field 1 into two unequal-sized intervals: $[0, 13]$ and $[14, 15]$. Thus R1 and R2 are separated at the cost of only one memory access. Comparing to HiCuts, which needs at least 3 times as many memory accesses, HyperSplit is obviously more efficient. This binary cut on one single dimension, can also bring other benefits. Because each time, HyperSplit only performs a binary cut, wildcard rules like R5 will yield fewer duplicates, and thus a smaller memory size can be achieved. Note however that the average number of memory accesses for each search is usually larger than in HyperCuts due to this binary cut on a single dimension.

C. EffiCuts

Instead of building a single decision tree with all the rules, EffiCuts builds multiple trees for one ruleset to avoid rule duplication. EffiCuts categorizes the ranges into two types, *small range* and *large range*. One range can be formed as [*min*, *max*]. A *large range* is actually a range whose interval length $max - min$ is comparable with the maximal value on the dimension. For example, the ruleset shown in Table II has one *large range*: $[0, 15]$ and two *small ranges*: $[14, 15]$ and $[12, 13]$. We represent a *small range* as R_s and a *large range* as R_*. After this classification of ranges, each rules can be represented as a large-small ranges combination pattern. For example, the toy ruleset in Table II has three patterns: $\{R_s, R_*\}$, $\{R_*, R_s\}$ and $\{R_*, R_*\}$.

Before constructing the decision tree, EffiCuts first classifies the rules into different groups, each group having one or only a few combination patterns. As shown in Figure 1(c), building decision trees on these groups will not lead to rule duplication, thus reducing the memory footprint. Besides, by using the *Range Compaction* optimization technique from HyperCuts, the boundary of Field 1 is shrunk to [12, 15]. The same effect has also impacted on the Field 2 in the second tree. This will reduce the memory accesses in each tree. In theory, 5-tuple packet classification rules can be divided into $2^5 = 32$ groups. In realistic rulesets, it usually ends up with 10 trees. After using the *Tree Merge* technique of EffiCuts, there are still $4 \sim 5$ trees. Although EffiCuts has almost eliminated rule replication, the memory accesses brought by traversing multiple trees can reduce throughput significantly. According to [9], EffiCuts needs $2 \sim 3$x times more memory accesses compared to the single tree HyperCuts algorithm.

III. The Key Observations

In this section, we first propose the key observations about the features of the ruleset. Then, we present our measurement results of these features and the actual performance results on the rulesets.

A. Skewness of ranges distribution in different dimensions

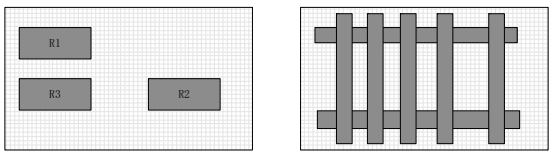
Although we have discussed the inefficiency of the equal-sized cutting when the rulesets are skewed, we further discuss the issue here.

Why more memory accesses? One reason is that *large range* rules enlarge the boundary of nodes, which undoes the effectiveness of the *Range Compaction* optimization techniques in HyperCuts. The other reason is related to how HyperCuts calculates the number of cuts: HyperCuts uses a cost function when calculating the number of cuts. In each node, HyperCuts maximizes the cut number K so that after K cuts, the number of rules in the child node N_{rule}, the number of child nodes N_{child}, and the number of rules in the parent node N_{parent} satisfy:

$$\sum N_{rule} + \sum N_{child} < spfac \times N_{parent} \qquad (1)$$

Since at each round cuttings along a dimension are cut into two pieces of equal sizes, N_{child} is doubled as the number of cutings increases. If the distribution of ranges is not uniform, more cuts do not separate the rules, but only increase the duplication of rules. Since the N_{child} increase quickly, soon the Formula 1 will not be satisfied. However, more cuts on child nodes are still needed to separate the rules.

Why more memory? The skewness of ruleset not only increases the memory accesses of HiCuts and HyperCuts, it also increases the memory consumption. As shown in Figure 1, it is easy to discover that there are many rule duplications inside the decision tree. This is because more cuts on one field usually lead to more duplication of rules which have wildcard range on that field. Decision-tree based algorithms use pointers to represent one rule, thus, a lot of rule duplication usually leads to a large amount of memory for the storage of these pointers. In the implementation of many DT algorithms, the memory consumption for pointers usually accounts for around $50\% \sim 90\%$ of the total memory size. Thus this skewness has a significant affect on performance. In other words, the degree of *uniformity* of the distribution of ranges is a key feature to capture for predictable performance.



(a) sparse rules (b) orthogonal structure

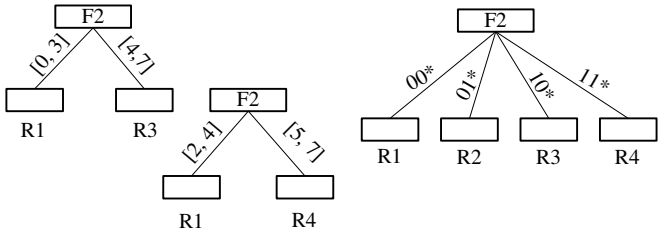
Fig. 2. Geometric view of packet classification rules

TABLE III
THE "ORTHOGONAL STRUCTURE" RULES

Field 1	Field 2
S_1	*
...	*
S_n	*
*	D_1
*	...
*	D_n

TABLE IV
UNIFORM DIMENSION HIDES THE "ORTHOGONAL STRUCTURE"

Field 1	Field 2	Field 3
00*	*	01
01*	01	*
10*	*	10
11*	10	*



(a) EffiCuts (b) HyperCuts

Fig. 3. Decision Trees of EffiCuts and HyperCuts on Table IV

B. "Orthogonal structure" and wildcard rules

Packet classification rules can be viewed as a set of hypercubes in a multi-dimension space. Fig 2(a) shows the geometric view of a 2-dimension rule-set. We call this rule-set sparse because the number of disjoint sub-regions is linear in the number of rules. This kind of ruleset is suitable for the decision-tree based algorithms. Figure 2(b) shows another type ruleset. In this kind of ruleset, the number of disjoint sub-regions are much larger than the number of rules. We say that such ruleset has an "orthogonal structure". Table III shows the form of this ruleset. In the worst case, A K-dimension ruleset with N rules will generate $O(N^K)$ disjoint sub-regions, which renders the efficient separation of the ruleset by decision-tree based algorithm difficult. From the Figure 2(b), we can deduce that it's impossible to perform smart cutting to separate these rules without incurring any rule duplication. Unfortunately, this structure is common in the many firewall rulesets. Firewall rulesets usually contain a lot of distinct *small range* on both source and destination IP fields. They also contain a lot of *large range* rules on these fields. When preprocessing these rulesets, HyperCuts or HiCuts will usually choose either the source IP or the destination IP field to cut, While this separates the rules which have *small range* on the corresponding field, this also causes the duplication of other rules which have wildcard on this field. Even worse, the rules which have *large range* on both the source and destination fields will get duplicated whenever there is any cuts on either field. We call these rules *wildcard rules* for short.

When EffiCuts groups rules by the combination of the *small range* and *large range*, it has also removed this kind of structure in the ruleset. This is the key reason why EffiCuts can reduce the memory footprint by several orders of magnitude.

C. The connection between two features

We have shown two important features that determine the performance of the decision-tree based algorithms. Here, we show how these two features interact with each other. If on one dimension, the distribution of the prefixes or the ranges is not skewed (or is very uniform), the HyperCuts algorithm or other decision-tree based algorithm can use only this dimension to separate all the rules, while ignoring any "orthogonal structure" that might exist on other dimensions (see Table IV).

On this ruleset, cutting on Field 1 is enough to separate the rules into four small groups, so the "orthogonal structure" on Field 2 and Field 3 does not have any impact on the algorithm performance. However, previous work ignores this uniformity feature, resulting in more memory accesses. For example, since there are two patterns inside this ruleset: $\{R_s, R_*, R_s\}$ and $\{R_s, R_s, R_*\}$, EffiCuts will simply divides all the rules into two groups, thus generating two trees. Although the depth of each tree is only one, it takes two memory accesses to traverse two trees. However if we build one decision tree, only one memory access is needed. Figure 3 shows the decision trees of both EffiCuts and HyperCuts.

IV. The measurement results

We use ClassBench[8] to generate synthetic classifiers for various classification applications such as accesses control list (ACL), firewall (FW) and IP chain (IPC). For our experiment, we have generated 1K and 10K rule scenarios.

For comparison, we implement two algorithms in this paper: HyperCuts and HyperSplit. For the HyperCuts algorithm, we adopt the implementation in [7]. It disables the *Range Compaction* and *Node merging* optimization techniques, and only allows at most 64 cuttings in one node. It uses 64-bits EPB (external pointer bitmap) to store the child node index, thus eliminating the pointer array of the original implementation. We use the HyperSplit code from [4]. For a fair comparison, we set the parameter $binth$ for both algorithms to 16. We set the *spfac* of HyperCuts to 4. Note that the HyperSplit code does not account for the memory used for the rule pointer in the leaf. We add these part of memory in our comparison.

TABLE V
THE DISTRIBUTION OF THE CUTS NUMBER

ruleset(num)	algo.	sIP	dIP	sPort	dPort	Protocol
ACL1K(953)	HyperSplit	42%	46%	0%	7%	4%
	HyperCuts	49%	46%	0%	4%	0
ACL1K(989)	HyperSplit	25%	34%	0%	23%	18%
	HyperCuts	34%	45%	0%	21%	0%
ACL10K(9792)	HyperSplit	46%	15%	0%	2%	37%
	HyperCuts	22%	74%	0%	1%	2%
ACL10K(9453)	HyperSplit	30%	37%	1%	21%	11%
	HyperCuts	12%	65%	0%	18%	6%
FW1K(815)	HyperSplit	40%	22%	6%	29%	3%
	HyperCuts	74%	5%	5%	5%	12%
FW1K(964)	HyperSplit	38%	42%	0%	0%	20%
	HyperCuts	64%	35%	1%	0%	0%
IPC1K(963)	HyperSplit	43%	31%	0%	4%	22%
	HyperCuts	49%	29%	1%	21%	0%
IPC1K(644)	HyperSplit	49%	31%	0%	0%	19%
	HyperCuts	2%	98%	0%	0%	0%
IPC10K(9515)	HyperSplit	33%	29%	4%	28%	6%
	HyperCuts	17%	55%	8%	10%	11%
IPC10K(10000)	HyperSplit	51%	47%	0%	0%	2%
	HyperCuts	29%	71%	0%	0%	0%

TABLE VI
THE DISTRIBUTION OF THE CUTS NUMBER

ruleset(num)	ss	sw	ws	ww	HyperCuts mem.	HyperSplit mem.
ACL1K(953)	927	19	5	2	36K	7K
ACL1K(989)	738	128	118	5	86K	19K
ACL10K(9792)	9570	176	37	9	2.0M	133K
ACL10K(9449)	7304	757	1217	171	134M	2.5G
FW1K(815)	104	208	448	55	12M	125K
FW1K(964)	187	631	133	13	151K	221K
FW10K(9395)	974	2423	5691	307	-	770M
FW10K(9615)	1801	6501	1279	25	14M	24M
IPC1K(963)	777	74	106	6	243K	27K
IPC1K(644)	302	84	258	0	124K	33K
IPC10K(9515)	7617	809	1050	39	66.8M	11M
IPC10K(10000)	3671	1077	5252	0	9.8M	5.5M

A. The cuts distribution

Before we start to measure these features of rulesets, we should first ask whether we need to measure all the dimensions of the rulesets? Here we present the cuts distribution of two algorithms. Since all the performance metrics are related to the cuts on the specific dimensions, studying the cuts distribution can help us to find the principal components that determine the performance.

From Table V, we can conclude that, although the cutting schemes of HyperSplit and HyperCuts are very different, their cuts distribution are similar. In both algorithms, around $50\% \sim 90\%$ cuts are performed on the IP fields. Especially for the FW and IPC rules, the number of cuts on IP fields is the largest component in the total cuts number. This is because in many packet classification rulesets, IP fields usually have most unique ranges, which leads the algorithm to cut on the IP fields.

Also, in our experiment, we have found that, almost 90% of memory are used for the storage of rule pointers in the leaf node. We do not show the results here due to the space limit.

B. The "orthogonal structure" and wildcard rules in IP fields

Since we have found that the largest part of cuts are performed on the IP fields, we now analyze the "orthogonal structure" in the IP fields of the rulesets. According to the *large range* and *small range* combinations on IP fields, we first split all the rulesets into four groups. We use the methods listed in [9] to determine whether a range is large or small. We named these four groups of rules as $\{R_*, R_s\}$, $\{R_s, R_*\}$, $\{R_s, R_s\}$ and $\{R_*, R_*\}$, and use four values ws, sw, ss, ww, respectively, to represent the number of rules in these four groups.

As shown in Table VI, the memory consumption of the HyperCuts and HyperSplit, is related to the number of "orthogonal structure" and wildcard rules in IP fields. For ACL rules, from 1K to 10K, we can see that the number of sw, ws and ww is fairly small compared to the total number of rules. Thus the memory consumption of ACL rules is usually small compared to FW rules. The ACL10K(9449) ruleset is a particular case. The memory size of ACL10K(9449) is far larger than the memory size of other rulesets. This is because the ww of this ruleset is relative larger. As presented above, during the building process of the decision tree, most cuts are performed on the IP fields and will leads to more duplications of $\{R_*, R_*\}$ rules. A large ww means that when algorithms perform cuts on IP fields, a large amount of rules will get duplicated, resulting in a large memory consumption.

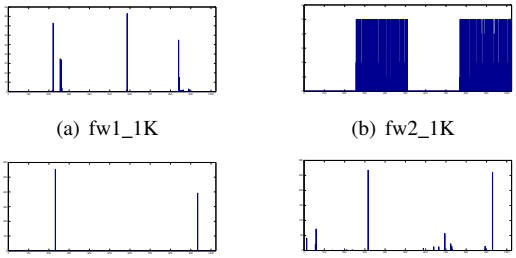
All the FW rules seem to have large sw, ws and ww. For example, the FW10K(9395) ruleset has a large ww and also has a large ws, so its memory consumption is beyond 500MB. A counterexample is the FW10K(9615) ruleset: its ww is only 25, which makes its memory consumption around 30x smaller than the memory consumption of FW10K(9615).

C. The uniformity of range distribution in IP fields

We use a simple method to test the uniformity of range distribution in IP fields. We transform the range representation on IP fields into the prefix representation. So by measuring the prefix distribution, we can measure the range distribution on IP fields.

For each ruleset, we form a counter array with $2^{10} = 1024$ entries to measure the distribution of the first 10-bits of source IP prefixes. For each prefix, if the prefix length p is equal or larger than 10, we extract the first 10-bits of this prefix, use these 10 bits to index one counter, and add one to that counter. If the prefix length p is smaller than 10, we expand the prefix into a 10-bit prefix, and add one into all the counters whose indexes share the first p bits. Therefore, by checking the distribution of the value inside the counter array, we can have a sketch of the uniformity of the prefix distribution. In our experiments, we only measure the rules which have *small ranges* on the source IP field, as rules with large ranges do not separate and simply get duplicated. Figure 4 shows the counter value distribution of 1K rulesets.

In Figure 4, the X axis shows the counter index, and the Y axis shows the corresponding counter value for the specific



(a) fw1_1K (b) fw2_1K

(c) acl1_1K (d) ipc1_1K

Fig. 4. The built decision trees by different algorithms

TABLE VII
THE TREE DEPTH OF THE FOUR RULESETS

Ruleset(num)	HyperCuts	HyperSplit
fw1_1K(815)	27	18
fw2_1K(964)	6	15
acl1_1K(953)	15	11
ipc1_1K(963)	18	13

index. As shown in Figure 4, the distribution of the ranges on the source IP fields of fw1_1K, acl1_1K and ipc1_1K is highly skewed. Most of the counter values are 0, and a lot of non-zero values are aggregated in a narrow index range. In contrast to these rulesets, fw2_1K has a wide non-zero range, so compared to the other three rulesets, the fw2_1K ruleset is more uniform.

We present the *Tree Depth* of the decision tree built on four rulesets in Table VII. The fw2_1K ruleset is the only ruleset which has fewer memory accesses when using the HyperCuts algorithm. According to Table VI, the memory size of HyperCuts on fw2_1K is also smaller than the memory size of HyperSplit. These results show the correctness of our observation that when the range distribution of the ruleset is skewed, it is better to use the HyperSplit algorithm for this ruleset. Otherwise, it is better to use HyperCuts algorithm as it results in fewer memory accesses and thus higher throughput.

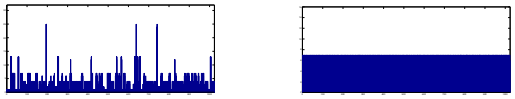
V. IMPLICATION DISCUSSION AND FUTURE WORK

In Figure 5, we have found that the range distribution of both source IP and destination IP fields of fw1_10K ruleset is uniform. We thus split the rulesets into three groups $\{R_s, R_*\}$, $\{\{R_*, R_s\}, \{R_s, R_s\}\}$ and $\{R_*, R_*\}$, and build HyperCuts tree on these three sub-ruleset. We find that this ruleset split method can achieve 2x less memory accesses than EffiCuts while still maintaining memory efficiency.

Further, by using these measurement results, one can:

1) Recommend algorithms for a specific ruleset according to the uniformity of the range distribution of the ruleset. For example, if the range distribution in one ruleset is skewed, then algorithms using the unequal-sized cutting scheme are preferred, like HyperSplit. If the range distribution is uniform, algorithms using the equal-sized cutting scheme are preferred, like HyperCuts.

2) Split ruleset based on the observation on the "orthogonal structure" on IP fields. For example, if the number of "orthogonal structure" rules is large, we should split the ruleset to reduce the memory consumption. Further, by measuring the uniformity of the range distribution, we can choose the right algorithm for the split sub-rulesets.



(a) sip_fw10K (b) dip_fw1K

Fig. 5. The prefix distribution of source and destination IP fields in fw1_10K

3) If we can capture the uniformity inside one ruleset, we can split one ruleset into several parts, leading to a hybrid TCAM and SRAM packet classification engine.

As future work, we will be performing the same experiments on the rulesets from production networks. The measurement method described in this paper will also be refined. Future work is needed to design finer grained metric to characterize ruleset features, and an automatic method will be designed for algorithm recommendation and rule splitting.

ACKNOWLEDGMENT

This work was supported in part by National Basic Research Program of China with Grant 2012CB315801, by National Natural Science Foundation of China (NSFC) with Grants 61133015 and 61202411, by National High-tech R&D Program of China with Grant 2013AA013501, by Strategic Priority Research Program of CAS with Grant XDA06010303, by the Instrument Developing Project of CAS with Grant YZ201229 and the ANR-NSFC pFLower project.

REFERENCES

[1] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Hot Interconnects VII*, pages 34–41, 1999.

[2] A. Khakpour and A. Liu. First step toward cloud-based firewalling. In *Proceedings of the 31st IEEE International Symposium on Reliable Distributed Systems (SRDS), Irvine, California*. IEEE, 2012.

[3] Y. Ma and S. Banerjee. A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 335–346. ACM, 2012.

[4] Y. Qi. Hypersplit source code. Online: http://security.riit.tsinghua.edu.cn/share/index.html, 2009.

[5] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li. Packet classification algorithms: From theory to practice. In *INFOCOM 2009, IEEE*, pages 648–656. IEEE, 2009.

[6] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 213–224. ACM, 2003.

[7] H. Song. Evaluation of packet classification algorithms. http://www.arl.wustl.edu/ hs1/PClassEval.html.

[8] D. Taylor and J. Turner. Classbench: A packet classification benchmark. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 2068–2079. IEEE, 2005.

[9] B. Vamanan, G. Voskuilen, and T. Vijaykumar. Efficuts: optimizing packet classification for memory and throughput. In *ACM SIGCOMM*, volume 40, pages 207–218. ACM, 2010.