Scalable Packet Classification Using Hybrid and Dynamic Cuttings

Wenjun Li (liwenjun@sz.pku.edu.cn), Xianfeng Li (lixianfeng@pkusz.edu.cn)
Engineering Lab on Intelligent Perception for Internet of Things (ELIP)
School of Electronic and Computer Engineering, Peking University, Shenzhen, China

***Abstract*—Network packet classification is an important functionality provided by modern Internet routers for enabling many network applications such as quality of service, security and differentiated services. Decision-tree based schemes are the most well-known algorithmic solutions for packet classification. But most of them, such as HiCuts and HyperCuts, suffer from memory explosion problem due to uncontrolled rule replications. EffiCuts, a state-of-the-art decision-tree technique, was proposed to reduce memory consumption at the cost of excessive memory accesses. Therefore, these decision-tree based approaches are either memory or performance inefficient, falling short of the needs of high-speed networks. In this paper, we propose HD-Cuts, a scalable decision-tree based packet classification using hybrid and dynamic cuttings to improve storage and performance simultaneously. Using ClassBench, we show that HD-Cuts achieves similarly excellent memory reduction as EffiCuts, but the performance of HD-Cuts is significantly better than EffiCuts in terms of memory accesses. In addition, HD-Cuts is more practical for implementation than EffiCuts.**

Keywords—Packet Classification, Algorithm, Router

I. Introduction

Modern Internet routers provide services beyond basic packet forwarding, such as quality of service (QoS), security and differentiated services. Packet classification is the core functionality for supporting these Internet services. For each incoming packet, the packet classifier compares multiple packet header fields against the corresponding field values of all the rules in a predefined rule table, and an action associated with the highest-priority rule will be taken on the packet.

The importance of packet classification has long been recognized, and extensive research efforts have been made in the past fifteen years [1]. They can be categorized broadly into two major approaches: architectural approach and algorithmic approach. Ternary Content Addressable Memory (TCAM) based techniques are the representative architectural approaches. TCAMs are the de-facto industry standard for packet classification in high-end routers because of their line-speed classification capability [2-4]. But TCAMs are expensive, and suffer from poor scalability, excessive high power consumption and serious range expansion problems [5]. As a result, efficient algorithmic approaches using commodity memory chips are still under active investigation. The quality of an algorithmic approach is evaluated primarily with two metrics: the memory space consumption of the classification data structure, and the number of memory accesses (which determines the classification speed) for the classification of an individual network packet. Decision-tree base schemes, such as HiCuts [6] and HyperCuts [7], are prevailing algorithmic solutions. But most of them suffer from memory explosion problems due to uncontrolled rule replications during the process of decision-tree construction. Another recent decision-tree based scheme, EffiCuts [8], was proposed to address the memory explosion problem through two techniques: Separable trees and Equi-dense cuts. However, EffiCuts achieves excellent memory reduction at the cost of excessive memory accesses and difficulty of implementation. In summary, these existing decision-tree based algorithmic approaches are either memory or performance inefficient, falling short of the needs of high-speed networks.

In this paper, we propose HD-Cuts, a scalable decision-tree based scheme using hybrid and dynamic cuttings, which is capable of improving storage and performance simultaneously. Unlike classic decision-tree based algorithms such as HiCuts and HyperCuts, HD-Cuts first partitions the classifier (rule set) into subsets with insights on their characteristics, then builds decision-trees for each subset by exploiting the characteristics of each individual subset. Note that EffiCuts also employs rule set partitioning, but its large number of decision-trees (up to 32 trees for 5-tuple rule sets) is the source of poor search performance, as each individual decision-tree needs a complete search process, resulting in more overall memory accesses compared to single-tree schemes. Unlike EffiCuts, HD-Cuts just partitions the rule set into at most 4 subsets based on two IP address fields. This reduction of subsets and the resulting decision-trees contributes directly to the reduction of memory accesses for packet classification. With these subsets, HD-Cuts employs a set of hybrid cutting algorithms which combine traditional one-dimensional cutting (HiCuts) and multi-dimensional cutting (HyperCuts) to build decision-trees. HD-Cuts is able to transfer between different cutting strategies adaptively to best exploit the characteristics of the subsets along the tree construction. Using ClassBench [9], we show that HD-Cuts achieves similarly excellent memory reduction as EffiCuts, but the performance of HD-Cuts is significantly better than EffiCuts in terms of memory accesses. In addition, HD-Cuts is more practical for implementation than EffiCuts, which maintains complicated data structures and requires special hardware support for efficient cuts.

The rest of the paper is organized as follows. Section II introduces the background information and briefly summarizes the related work. Section III gives a detailed description of the proposed HD-Cuts. Section IV provides the experimental results. Finally, Section V concludes the paper.

978-1-4799-2827-9/13/$31.00 ©2013 IEEE

II. BACKGROUND

A. The Packet Classification Problem

Packet classification is performed using a packet classifier, which is a collection of rules with priorities. Given a predefined packet classifier, the packet classification problem is to find out the highest-priority matching rule from the packet classifier for each incoming packet. Each rule is associated with an action. After classification, the corresponding action will be performed on each packet. In today's classifiers, a typical rule is a 5-tuple: the source and destination IP addresses (i.e., SA, DA), the source and destination ports (i.e., SP, DP), and the protocol (i.e., Prot). Table I shows a simple example classifier with fourteen 5-tuple rules and the priority of each rule is identified by rule order in classifier. Packet classification can be treated as a point location problem in computational geometry and it has been proved that the best bounds for locating a point are either $\Theta(\log N)$ time with $\Theta(N^F)$ space, or $\Theta(\log^{F-1} N)$ time with $\Theta(N)$ space for N non-overlapping hyper-rectangles in F-dimensional space [5, 10]. Clearly, this is impractical. Fortunately, packet classification rules in real-life applications have structural redundancies and some inherent characteristics that can be exploited to reduce the complexity [1, 4-8, 11-16].

B. Related Work

Numerous algorithmic approaches have been proposed for packet classification in the past fifteen years. [1, 5-8, 11-16]. We mainly discuss some representative decision-tree based approaches, because they are related to our proposed scheme. In decision-tree based schemes, the geometric view of the packet classification problem is taken and a decision tree is built. The root node of the tree covers the whole search space which contains all rules. They work by recursively cutting the search space into smaller subspaces. Each child node covers one sub-space, and the parent node records the selected dimensions and the number of cuts as *cut information*. This is repeated until a predefined number of rules (i.e., *binth)* are contained by each subspace. During packet classification, the search starts at the root node of the decision-tree. At each tree node, the relevant header field of the incoming packet is compared against the *cut information* stored at the tree node to decide which child node to go in the next step. By traversing the decision tree, a set of candidate rules stored at a leaf node will be examined using linear search to find the best matched rule, if any.

TABLE I. A SIMPLE EXAMPLE CLASSIFIER WITH 14 RULES ON 5 FIELDS

Rule	SA	DA	SP	DP	Prot	Action
R1	*	000*	80	*	TCP	*Act1*
R2	0010	*	*	>1024	*	*Act2*
R3	0*	0110	>1024	128	UDP	*Act3*
R4	1010	1*	80	[0:15]	UDP	*Act4*
R5	*	100*	[3:15]	128	TCP	*Act5*
R6	0100	*	*	>1024	TCP	*Act6*
R7	*	100*	32769	65530	UDP	*Act7*
R8	0010	*	*	*	UDP	*Act8*
R9	*	100*	80	32769	TCP	*Act9*
R10	*	100*	65530	[0:15]	TCP	*Act10*
R11	0010	111*	80	65530	UDP	*Act11*
R12	1101	0100	>1024	128	TCP	*Act12*
R13	*	1100	65530	>1024	UDP	*Act13*
R14	*	*	>1024	>1024	UDP	*Act14*

Classic decision-tree based algorithms, such as HiCuts [6] and HyperCuts [7], cut the search space hierarchically into many sub-spaces. HiCuts builds a decision tree using local optimization decisions at each node to choose one dimension to cut, as well as the number of cuts to make in the chosen dimension. HyperCuts can be viewed as an improved version of HiCuts, which is more flexible in that it allows cutting on multiple fields per step, resulting in a fatter and shorter decision tree. But both HiCuts and HyperCuts have the same rule replication problem for rules spanning multiple sub-spaces. This replication leads to extraordinary memory overhead, especially for large rule tables. EffiCuts [8] addresses this problem by separating rules into multiple trees to reduce replications incurred by large rules. However, EffiCuts suffers from several problems such as too many memory accesses and the needs of specialized comparator circuitry to support equi-dense cuts.

III. HD-CUTS

In this section, we make some observations and propose a new partition algorithm to separate rules into at most four subsets, and then we introduce a set of efficient cutting algorithms on the individual subsets.

A. Observations

Real-life rules have some inherent characteristics and Table II shows more details of several rule sets which are publicly available from [17] (100K rule sets are generated using ClassBench [9] by ourselves). From Table II, we can see that rules specify a much smaller number of distinct port ranges and protocol values compared with IP address field values especially for larger rule sets, which is consistent with previous observations mentioned in [4-8, 11-16]. Thus, we know that many rules in a rule set share common port/protocol field values with each other, which means that the cost of distinguishing rules by their port/protocol fields is much higher than using their IP address fields. Therefore, IP address fields are the primary fields to identify best matching rule from rule table for packets, and the port and protocol fields will only be used as auxiliary when needed.

TABLE II. NUMBER OF UNIQUE FIELD VALUES

Type	Size	Number of unique field values				
		SA	**DA**	**SP**	**DP**	**Prot**
ACL	**1K**	103	297	1	99	4
	10K	4784	733	1	108	4
	100K	42404	78051	1	96	4
FW	**1K**	124	175	13	42	5
	10K	3638	6951	13	43	5
	100K	21964	66027	13	43	5
IPC	**1K**	336	436	28	44	6
	10K	1515	2726	34	54	7
	100K	86323	90448	35	55	7

B. *Rule Set Partitioning Algorithm*

Our algorithm consists of two steps: *rule set partitioning* followed by *decision-tree construction* of the partitioned subsets. An efficient partitioning step plays a vital role in reducing rule replications in our framework. According to the observation introduced earlier, we only use the source and destination IP addresses as candidates for partitioning to contain the number of partitioned subsets. In particular, we partition the rules into the following four subsets to separate large rules from small rules without duplicates among each other (assuming wildcard as large field):

1) Subset_SA: rules with wildcard in SA;
2) Subset_DA: rules with wildcard in DA;
3) Subset_Small: rules with no wildcard in SA and DA;
4) Subset_Large: rules with wildcard in both SA and DA.

Therefore, rule set shown in Table I can be assigned into the following subsets: Subset_SA = {R1, R5, R7, R9, R10, R13}, Subset_DA = {R2, R6, R8}, Subset_Small = {R3, R4, R11, R12} and Subset_Large = {R14}. So far, we have equated large rules with wildcard. As described in EffiCuts, a true wildcard may be too demanding to classify a rule as large and considerable chances for reducing rule replications will be lost. In light of this, we relax the definition of largeness in a dimension to include each rule that covers a predefined portion, called *thresh_large*, of the dimension. Thus, if we set *thresh_large* to be 0.5 for rules shown in Table I, then these rules can now be categorized in the following subsets: Subset_SA = {R1, R3, R5, R7, R9, R10, R13}, Subset_DA = {R2, R4, R6, R8}, Subset_Small = {R11, R12} and Subset_Large = {R14}. For real-life 5-tuple rules with 32-bit IP addresses, a small *thresh_large* of 0.05 is able to capture 5% of the fields as large ones.

C. *Hybrid and Dynamic Cuttings*

After the processing of rule set partitioning, at most 4 subsets are generated (instead of 32 subsets in EffiCuts). This significant reduction of subsets contributes directly to the reduction of memory accesses for packet classification. However, a new challenge is that these subsets do not exhibit the similar characteristics as those in EffiCuts for subsequent HyperCuts processing, the core single-tree construction algorithm invoked by EffiCuts. To produce short trees for each of the four subsets, we introduce a set of hybrid and dynamic cutting algorithms to exploit the different characteristics of our subsets as follows:

1) Subset_SA: We first conduct HiCuts by just considering two address fields, then extend to all five fields with HyperCuts once the next cut field is not DA or when the rules in current node are smaller than a pre-defined threshold;
2) Subset_DA: We first conduct HiCuts by just considering two address fields, then extend to all five fields with HyperCuts once the next cut field is not SA or when the rules in current node are smaller than a pre-defined threshold;
3) Subset_Small: We first conduct HyperCuts by just considering two address fields, then extend to all five fields with HyperCuts once rule replications become serious or when the rules in current node are smaller than a pre-defined threshold;
4) Subset_Large: We conduct HyperCuts considering all five fields simultaneously as traditional HyperCuts does.

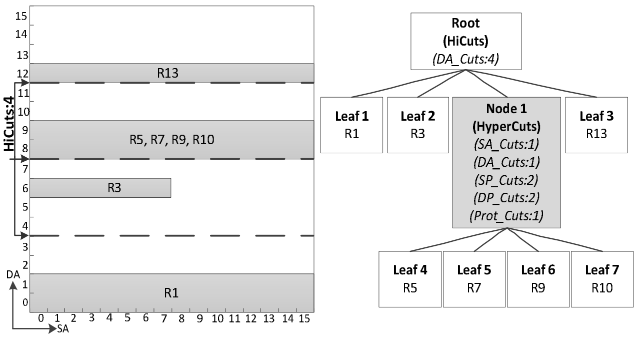

Figure 1. Decision tree for Subset_SA

To explain the rationale behind above cutting algorithms, we use Subset_SA as example. Apparently, rules in Subset_SA are small in their DA field; otherwise they will be in Subset_Large. This means cutting along DA is likely to separate the rules better than along SA, so we let HiCuts to continue as long as its decision meets this expectation. Once HiCuts chooses SA as the cutting dimension, it means cutting along DA will no longer produce efficient partitions. On the other hand, since the rules in Subset_SA are all large in the SA field, it is easy to cause replications by cutting along SA due to the overlaps of rules on this field. Therefore, HiCuts considering only DA and SA will not work efficiently any more, and it is time to switch to HyperCuts for the rest of decision-tree construction.

Taken Subset_SA (i.e., {R1, R3, R5, R7, R9, R10, R13}) generated from rules shown in Table I as example. Figure 1 shows the decision tree built for Subset_SA using hybrid and dynamic cuttings. We can see that HD-Cuts first conducts HiCuts just among two IP address fields, and DA is selected to cut the space into four subspaces. This decision intuitively matches the rule distribution pattern shown by the left part of Figure 1, where these rules are separated horizontally along the DA dimension. When reaching node1 which contains a few number of similar addresses rules, HD-Cuts finds that it will no longer work efficiently by continuing cutting using two IP address fields. At this point, it automatically switches to HyperCuts using all five fields in the following steps.

IV. EXPERIMENTAL RESULTS

In this section, we present the performance results of our HD-Cuts with three other representative cutting techniques: HiCuts, HyperCuts and EffiCuts. The rule sets used in our experiments are publicly available from [17], which provides three types of rule sets: Access Control List (ACL), Firewall (FW) and IP Chain (IPC). Each rule set is named by its type and size (e.g. FW_10K refers to the firewall rule set containing about 10,000 rules). The primary metrics for evaluating the performance of a packet classification are memory consumption and the number of memory accesses, and we report them as follows.

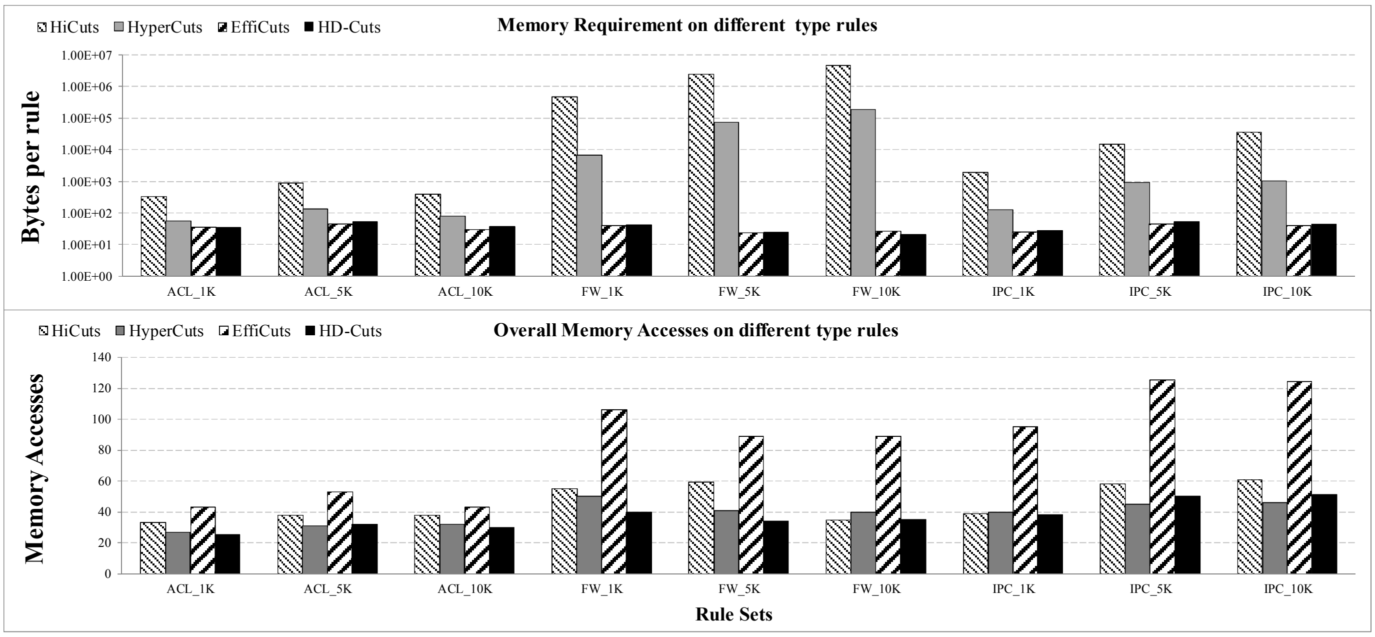

Figure 2. Total Memory (top) and Accesses (bottom) for HiCuts, HyperCuts, EffiCuts and HD-Cuts (*binth=8, thresh_large=0.05*)

A. Memory Consumption

Figure 2 (top) shows the memory consumption per rule for HiCuts, HyperCuts, EffiCuts, and our HD-Cuts. Both EffiCuts and HD-Cuts achieve significant memory reduction compared to HiCuts and HyperCuts. This is especially obvious for FW classifiers. The reason of this memory saving is that a significant fraction of FW rules have many wildcard fields, incurring rampant replication of rules in HiCuts and HyperCuts [8]. By partitioning rule set into subsets, EffiCuts and HD-Cuts can avoid rule replications dramatically. Note that the primary objective of our work is to achieve less memory accesses without sacrificing memory consumption and ease of implementation. We can see from Figure 2 (top) that the excellent memory efficiency of EffiCuts is kept in our proposed HD-Cuts.

B. Memory Accesses

Figure 2 (bottom) shows the performance in terms of the number of overall memory accesses for HiCuts, HyperCuts, EffiCuts, and our HD-Cuts. It is clear that our work performs much better than EffiCuts. The improvement on complex rule sets (FW and IPC) is more striking - an average of 2.3x speed-up is achieved. We can also see from Figure 2 (bottom) that HD-Cuts consumes similar or even less memory accesses compared with HyperCuts. But HD-Cuts has orders-of-magnitude reduction in memory than HyperCuts. Table III reports this memory reduciton for rule sets of 10K scale (HyperCuts runs out of memory for some 100K rules).

C. Scalablity of HD-Cuts

Table IV shows the scalability of HD-Cuts for large rule sets compared with EffiCuts. Since 100K rule sets are not available in [17], we generate them by ourselves using ClassBench which creates synthetic classifiers with characteristics representative of real-word classifiers. From Table IV, we can see that HD-Cuts keep smart memory efficiency and high performance for larger rule sets even up to 100K, which justifies our claim that HD-Cuts is a scalable packet classification algorithm.

D. Worst-case Tree Height

If parallel searching on multiple trees is desired for speedup in implementation, a natural concern is the worst-case tree height, as the overall performance of a parallel implementation is constrained by the worst case. Therefore, we look at the worst-case tree height among the multiple trees. This result is presented in Figure 3. It can be seen that the worst-case height of a single tree ranges from a quarter to half of the overall number of memory accesses on all the trees. This means HD-Cuts can achieve about 2x to 4x speedup than HyperCuts with a parallel implementation. The improvement, combined with the result in Figure 2, justifies our claim that HD-Cuts can improve storage and performance simultaneously.

TABLE III. HD-CUTS' MEMORY REDUCTION OVER HYPERCUTS

	ACL_10K	FW_10K	IPC_10K	Mean
HyperCuts (MB)	0.77	1650	9.1	
HD-Cuts (MB)	0.33	0.18	0.38	
Reduction	2.3	9166	24	3064

TABLE IV. SCALABLITY OF HD-CUTS

	Bytes per rule		Memory Accesses	
	EffiCuts	HD-Cuts	EffiCuts	HD-Cuts
ACL_100K	32	25	33	22
FW_100K	70	51	136	63
IPC_100K	27	22	99	43

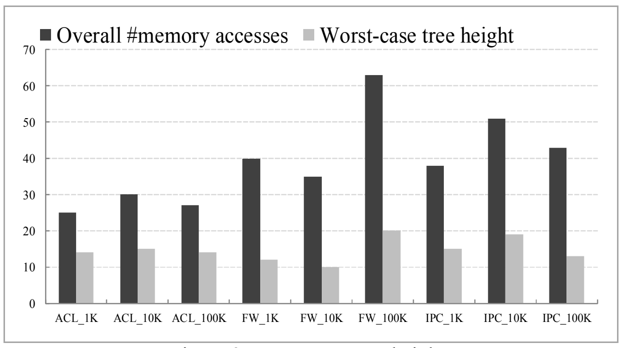

Figure 3. Worst-case tree height

V. CONCLUSION

This paper presents HD-Cuts, a scalable decision-tree based algorithmic scheme for packet classification using hybrid and dynamic cuttings, which can improve storage and performance simultaneously. HD-Cuts employs a far more efficient rule set partitioning method than EffiCuts, and a set of hybrid and dynamic cutting algorithms are proposed to adaptively exploit the different characteristics of these subsets. Compared to the state-of-the-art EffiCuts algorithm, HD-Cuts achieves significant speedup with the same level of reduction on memory consumption. Meanwhile, HD-Cuts is more practical for implementation than EffiCuts, which maintains complicated data structures and requires special hardware support for efficient cuts.

ACKNOWLEDGMENT

This work is supported by the grant of Shenzhen municipal government for basic research on Internet technologies (Outstanding Young Scholar, No. JC201005270274A) and Shenzhen municipal government for basic research on information technologies (No. JCYJ20130331144751105).

REFERENCES

[1] D.E. Taylor. “Survey *and taxonomy of packet classification techniques*,” ACM Computing Surveys (CSUR), 2005. 37(3): p. 238-275.

[2] E. Spitznagel, D. Taylor, and J. Turner. “*Packet classification using extended TCAMs*,” in IEEE ICNP, 2003.

[3] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. “*Algorithms for advanced packet classification with ternary CAMs*,” in ACM SIGCOMM, 2005.

[4] Y. Ma. and S. Banerjee. “*A smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification*,” in ACM SIGCOMM, 2012.

[5] V. Srinivasan, et al. “*Fast and Scalable Layer Four Switching*,” in ACM SIGCOMM, 1998.

[6] P. Gupta and N. McKeown. “*Packet classification using hierarchical intelligent cuttings*,” in IEEE HOTI, 1999.

[7] S. Singh, et al. “*Packet classification using multidimensional cutting*,” in ACM SIGCOMM, 2003.

[8] B. Vamanan, G. Voskuilen, and T. Vijaykumar. “*EffiCuts: optimizing packet classification for memory and throughput*,” in ACM SIGCOMM, 2010.

[9] D.E. Taylor. and J.S. Turner. “*Classbench: A packet classification benchmark*,” in IEEE INFOCOM, 2005.

[10] M. Overmars and A. Stappen. “*Range searching and point location among fat objects*,” Journal of Algorithms, 1996. 21(3): p. 629-656.

[11] F. Baboescu, S. Singh, and G. Varghese. “*Packet classification for core routers: Is there an alternative to CAMs?*” in IEEE INFOCOM, 2003.

[12] P. Gupta and N. McKeown. “*Packet classification on multiple fields*,” in ACM SIGCOMM, 1999.

[13] F. Baboescu and G. Varghese. “*Scalable packet classification*,” in ACM SIGCOMM, 2001.

[14] J. Fong , X. Wang, Y. Qi, J. Li, W. Jiang. “*ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification*,” in IEEE HOTI, 2012.

[15] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li. “*Packet Classification Algorithms: From Theory to Practice*,” in IEEE INFOCOM, 2009.

[16] A. Feldman and S. Muthukrishnan. “*Tradeoffs for packet classification*,” in IEEE INFOCOM, 2000.

[17] http://www.arl.wustl.edu/~hs1/PClassEval.html.