2013 IEEE 21st Annual Symposium on High-Performance Interconnects

HybridCuts: A Scheme Combining Decomposition and Cutting for Packet Classification

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

***Abstract*—Packet classification is an enabling function for a variety of Internet applications such as access control, quality of service and differentiated services. Decision-tree and decomposition are the most well-known algorithmic approaches. Compared to architectural solutions, both approaches are memory and performance inefficient, falling short of the needs of high-speed networks. EffiCuts, the state-of-the-art decision-tree technique, significantly reduces memory overhead of classic cutting algorithms with separated trees and equi-dense cuts. However, it suffers from too many memory accesses and a large number of separated trees. Besides, EffiCuts needs comparator circuitry to support equi-dense cuts, which makes it less practical. Decomposition based schemes, such as BV, can leverage the parallelism offered by modern hardware for memory accesses, but they have poor storage scalability. In this paper, we propose HybridCuts, a combination of decomposition and decision-tree techniques that improves storage and performance simultaneously. The decomposition part of HybridCuts has the benefits of traditional decomposition-based techniques without the trouble of aggregating results from a large number of bit vectors or a set of big lookup tables. Meanwhile, thanks to the clever partitioning of the rule set, an efficient cutting algorithm following the decomposition can build short decision trees with significant reduction on rule replications. Using ClassBench, we show that HybridCuts achieves similar memory reduction compared to EffiCuts, but it outperforms EffiCuts significantly in terms of memory accesses for packet classification. In addition, HybridCuts is more practical for implementation than EffiCuts, which maintains complicated data structures and requires special hardware support for efficient cuts.**

Keywords-Packet Classification; Decomposition; Decision-Tree; Rule Replication; Performance

I. INTRODUCTION

Modern Internet routers provide services beyond basic packet forwarding, such as access control, quality of service, and differentiated services. All such functionalities require packet classification, which decides the action to be taken on a packet based on multiple fields in the packet header. A predefined classifier consisting of a set of rules is looked up for a match for these purposes. To match a rule, packet classification needs to compare multiple header values of the incoming packet against the field values of all the rules in the classifier to determine the type of actions (e.g., drop or permit) to be taken on the packet.

Numerous packet classification techniques have been proposed in the past fifteen years [1]. They can be categorized broadly into two major approaches: architectural and algorithmic. Ternary Content Addressable Memories (TCAMs) based techniques are the representative architectural approaches [2-6]. TCAMs are widely used because of their ability to process packets at line speed. But TCAMs are expensive, and suffer from scalability and high power consumption [7]. To address the high power consumption problem, recent TCAM related research efforts try to make tradeoffs between power consumption and lookup performance with multiple accesses to a set of TCAM subarrays, such as the TreeCAM [8]. Another issue with TCAMs is the rule duplication problem related to the port number fields when transforming a single range into an equivalent set of multiple prefixes for the convenience of TCAM operations [9]. As a result, efficient algorithmic solutions using ordinary memories such as DRAM/SRAM are still under active investigation.

Decision-tree and decomposition are the most well-known algorithmic approaches. Decision-tree based schemes, such as HiCuts [10] and HyperCuts [11], separate the search space into many equal-sized sub-spaces using local optimizations. But both schemes have the same rule replication problem, which might cause large memory overhead. Although EffiCuts [12], the state-of-the-art decision-tree technique, can avoid large memory overhead with Separable trees and Equi-dense cuts, it suffers from several problems: 1) too many memory accesses, which worsen the low-speed problem of algorithmic techniques; 2) a large and variable number of separated trees, which raises its barrier for practical implementation; 3) and the needs of specialized comparator circuitry to support equi-dense cuts, which further limits its adoption in practice. Decomposition based schemes, such as BV [13], can leverage the parallelism offered by modern hardware to improve performance, but they have poor storage scalability. Therefore, these algorithmic approaches are memory and performance inefficient, falling short of the needs of high-speed networks. Thus, it is worth investigating new algorithms with higher performance and better scalability.

In this paper, we propose HybridCuts, a combination of decomposition and decision-tree techniques that improves storage and performance simultaneously. The decomposition part of HybridCuts has the benefits of traditional decomposition-based techniques, but without the trouble of aggregating results from a large number of bit vectors or a set of big lookup tables. Meanwhile, thanks to the clever partitioning of the rule set, an efficient cutting algorithm following the decomposition is employed to build short decision trees with significant reduction on rule replications. The main contributions of this paper include:

978-0-7695-5103-6/13 $26.00 © 2013 IEEE
DOI 10.1109/HOTI.2013.12

IEEE computer society

- A rule set decomposition algorithm based on the observation that most rules have at least one small field. The proposed decomposition has the benefits of traditional decomposition-based techniques, but without the trouble of expensive aggregations.
- A novel one-dimensional cutting algorithm called FiCuts, which has a global view on the characteristics of the subset of rules, and therefore is simpler and works more intelligently than HiCuts.
- A two-stage cutting framework which combines one-dimensional cutting and multi-dimensional cutting techniques, and is adaptive at building short decision trees with significant reduction on rule replications.

We evaluate our algorithm using ClassBench [14] and show that HybridCuts is able to produce a very small number of short decision trees with low memory overhead. Even for rule sets up to 100K entries, the constructed data structure can be accommodated in on-chip memory. In addition, it is quite feasible that a carefully pipelined and parallelized hardware implementation of HybridCuts will achieve a line-speed performance.

The rest of the paper is organized as follows. Section II introduces the background. Section III briefly summarizes the related work. Section IV presents the technical details of HybridCuts. Section V provides experimental results. Finally, Section VI concludes the paper.

II. BACKGROUND

A. The Packet Classification Problem

The purpose of packet classification is to find a matching rule from a packet classifier for a packet. A packet classifier is a set of rules, with each rule R consisting of a tuple of F field values (exact value, prefix or range) and an action to be taken in case of a match. 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 number (i.e., Prot). The rules are often prioritized to resolve potential multiple match scenarios.

B. Complexity in Theory

There are several standard problems in the field of computational geometry that resemble packet classification. From a geometric point of view, each rule R represents a hyper-rectangle and an incoming packet p represents a point in F-dimensional space. If a packet p matches a particular rule R, then the point represented by p falls into the hyper-rectangle specified by R. Therefore, packet classification can be treated as a point location problem in computational geometry. According to [15], for N non-overlapping hyper-rectangles in F-dimensional space, 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. Therefore, the mathematic complexity of packet classification is extremely high as the number of rules or dimensions increase. Clearly, this is impractical: with just 1000 rules and 4 fields, a solution is either impracticably (N^F is about 1000G) or too slow ($\log^{F-1} N$ is about 1000 memory accesses).

C. Complexity in Practice

As the above example illustrates, it is infeasible to design a single algorithm that can perform well in all cases. Fortunately, packet classification rules in real-life applications have some inherent characteristics that can be exploited to reduce the complexity [11, 16-22]. The following is a distillation of previous observations relevant to our work:

- The protocol field is restricted to a small set of values, e.g., TCP, UDP, and the wildcard.
- Rules specify a limited number of distinct transport port ranges.
- The number of address prefixes matching a given address is typically five or less.
- The number of rules matching a given packet is typically five or less.
- Many different rules share the same field values.

III. RELATED WORK

A lot of algorithmic approaches have been proposed for packet classification in the past fifteen years. They can be categorized into two major groups: decomposition based [9, 13, 18, 19] and decision-tree based [9-12, 16, 19-22] algorithms. In decomposition based schemes, independent search on each header field is performed for the whole rule set, and then the results are integrated to get the matching rule. These approaches offer high throughput but require a large amount of storage in order to aggregate the results from individual search operations efficiently. In decision-tree based schemes, the geometric view of the packet classification problem is taken and a decision tree is built. They work by recursively cutting the search space into smaller subspaces. This is repeated until a predefined number of rules are contained by each subspace. When a packet arrives, the decision tree is traversed to find a matching rule at a leaf node. Next, we give a more detailed review on a few representative techniques.

A. Parallel Bit-Vectors (BV)

Parallel Bit-Vectors scheme [13] is one of the most representative decomposition based solutions. It works on the individual fields of rules independently for partially matching results, which are encoded as bit vectors. Each bit in a bit vector stands for a partial matching result to a single rule. Then a bit-wise AND operation on all bit vectors is performed to get the final result. The most significant "1" bit in the final bit vector denotes a matched rule with the highest priority. Parallel BV approach has $\Theta(\log N)$ search time and a rather unfavorable $\Theta(N^2)$ memory requirement. Aggregate Bit-Vector(ABV) [18] can be viewed as an improved version of BV, which seeks to improve the performance of BV based on the fact that real filter sets often result in sparse vectors containing a large portion of "0"s inside. ABV makes use of this phenomenon by compressing the bit vectors into several chunks containing "1" bits. Although ABV can improve performance to some extent, the unfavorable memory problem still exists.

B. *HiCuts and HyperCuts*

HiCuts [10] and HyperCuts [11] take a geometric view of the packet classification problem. Each rule is viewed as a hypercube in an F-dimensional space, where F is the number of fields in a rule. Each packet defines a point in this F-dimensional space. HiCuts cuts the search space into many equal-sized subspaces recursively until the rules covered by each subspace is less than the pre-defined bucket size called *binth*. This cutting process is carried out using a tree data structure, which is called a decision-tree. The root node of the tree covers the whole searching space which contains all rules. HiCuts selects one dimension to cut and decides how many subspaces should be cut using a space optimization function with a parameter called *spfac*. In case a rule spans multiple subspaces, rule replication happens, which is an undesirable case. Upon the construction of the decision tree, an incoming packet searches along the tree using its field values until a leaf node is hit, then a linear search is taken to get a best matched rule, if any, among the ones in the leaf node. HyperCuts is 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. Both HiCuts and HyperCuts have the same rule replication problem, leading to significant memory overhead, especially for large rule tables.

C. *EffiCuts*

EffiCuts [12], a state-of-the-art decision-tree technique, identifies two primary causes for large memory overhead: big rules suffer a lot of replications, and equi-sized cuts create many ineffective nodes. Based on these observations, EffiCuts proposes two techniques to reduce memory overhead. One is that EffiCuts uses a partition method to separate rule set into several subsets, depending on whether the value of each dimension is wildcard (or almost wildcard) and each subset creates its own decision-tree independently using HyperCuts. The other one is equi-dense cuts instead of equi-size cuts, which tries to combine similar nodes into one. Although EffiCuts reduces memory overhead dramatically, Efficuts suffers from too many memory accesses and a large number of separated trees (31 in the worst-case, and 12 on average). Efficuts alleviates this problem with selective tree merging. However, the merging is restricted to pairs of trees meeting special conditions. Otherwise, it will destroy separability and result in significant rule replications with arbitrary tree-merging. In our experiments, we observe that the selective tree merging results in 8 trees on average.

D. *ParaSplit*

ParaSplit [19] is another recent work that is decision-tree based and makes use of rule set partitioning to significantly reduce rule replications. It is different from EffiCuts in that its objective is for efficient hardware implementation using FPGAs. It employs a complex heuristic for rule set partitioning, which may require 5000 to 10000 iterations to reach an optimal partitioning. For hardware acceleration, it applies parallelism and pipelining in conjunction: the multiple trees are searched in parallel, and the search on individual trees is pipelined. Although HybridCuts proposed in this work is an algorithmic solution, it is perfectly suitable for similar hardware accelerations.

IV. HybridCuts

As the authors of EffiCuts observed, the overlapping of rules in a classifier varies vastly in size, causing extensive replications of large rules during the algorithms' fine cuts for separating the small rules. An effective solution to address this problem is to decompose the original set into several subsets for cutting-based algorithms. But EffiCuts applies this principle aggressively, leading to an uncontrolled large number of trees. In this paper, we propose a new decomposition algorithm to separate rules into a few subsets, and then we introduce a set of efficient cutting algorithms on the individual subsets.

A. *Decomposition-based Framework*

EffiCuts produces up to 31 decision trees with a partition method that takes the properties of rules on all five dimensions into consideration. Unlike EffiCuts, we decompose rules based on their characteristics shared in a single dimension. This decomposition produces only 6 subsets for a typical 5-tuple rule set. In fact, since we do not consider protocol dimension, just 5 subsets will be generated in our baseline algorithm, and this number is further reduced to 3 with an optimization.

The rationale behind this strategy of decomposition is simple: by grouping rules that are small in the same field, we get a dimension in the space where the extensive replications bothering traditional cutting-based algorithms by wide overlaps are significantly reduced. In addition, subsets decomposed this way enable a simple and space-efficient one-dimensional cutting algorithm.

Definitions. To proceed to more detailed discussion, we first introduce some definitions and observations used by HybridCuts. Given an N-dimensional rule $R = (F_1, F_2, F_3 \cdots F_N)$, Len_i represents the length of field F_i, and a threshold value vector $T = (T_1, T_2, T_3 \cdots T_N)$. We call F_i is a **small field** if $Len_i \leq T_i$, or a **big field** if $Len_i > T_i$. Then, we define the following concepts for R:

Big rule: $\forall i \in \{1, 2, 3 \cdots N\}$, F_i in R is a big field;
Small rule: $\exists i \in \{1, 2, 3 \cdots N\}$, F_i in R is a small field;

TABLE I. Percentage of Big rules

Big rules%	T'(16,16,8,8)	T'(12,12,6,6)	T'(8,8,4,4)
ACL_1K	0.55%	0.55%	3.93%
ACL_10K	0.22%	0.22%	3.11%
ACL_100K	0.00%	0.00%	0.00%
FW_1K	1.14%	1.52%	3.29%
FW_10K	0.79%	1.13%	3.87%
FW_100K	1.28%	1.70%	4.71%
IPC_1K	0.21%	0.96%	4.85%
IPC_10K	0.04%	0.94%	3.75%
IPC_100K	0.48%	1.86%	9.84%

For a typical 5-tuple rule, since the protocol field is restricted to a small set of values (tcp, udp, *, etc), we just consider the other four fields. Then the threshold value vector T for 5-tuple rules is simplified to a four-dimensional vector and a rule R(SA, DA, SP, DP, Prot) is called a small rule if there exists at least one small field in {SA, DA, SP, DP}. For the sake of convenience in writing, we use a logarithmic vector T' to represent the threshold value vector T. For example, if we set threshold value vector T = (2^{16}, 2^{16}, 2^8, 2^8), then index logarithmic T' = (16, 16, 8, 8).

Table I shows the percentage of big rules under three different thresholds for a number of rule sets from ClassBench. It is clear that the percentage of big rules is very low even under very demanding thresholds. This indicates that the vast majority of the rules have at least one small field satisfying a threshold T as narrow as Table I gives.

Field-wise decomposition. Based on above definitions and observations, we decompose a 5-tuple rule set into the following five subsets without duplicates among each other.

1) Big-subset: SA, DA, SP and DP are all big field;
2) SA-subset: SA is a small field for each rule;
3) DA-subset: DA is a small field for each rule;
4) SP-subset: SP is a small field for each rule;
5) DP-subset: DP is a small field for each rule.

Many rules may have multiple small fields, and there exists options on deciding where to go for a rule with multiple small fields. A simple metric for decision is the balance on the sizes of the subsets. For example, when processing a rule R with two small fields: SA and DA, suppose the sizes of the SA-subset and the DA-subset up to now are N_1 and N_2 respectively, then rule R will go to SA-subset if $N_1 \leq N_2$, or to DA-subset vice versa. There exist several other guidelines for decomposition, which will not be discussed here for the limitation of space.

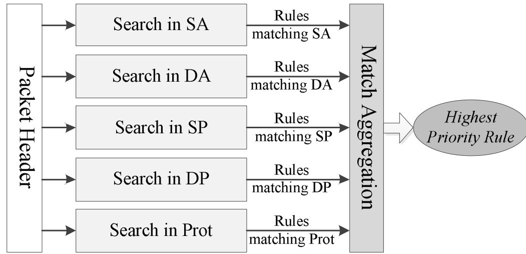

Figure 1. Traditional Decomposition

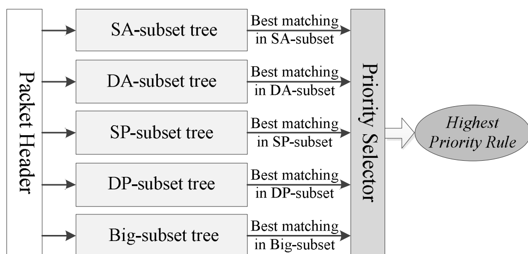

Figure 2. Improved Decomposition

Post-decomposition. The proposed decomposition is somewhat like the first step of a traditional decomposition based approach, where rules are projected into a specific field for a field-wise matching. However, the decomposition here is quite different from traditional schemes in two aspects. First, rules are partitioned into multiple subsets, instead of all being projected onto each field. Second, the partitioned subset will undergo a cutting process to find a single match, instead of finding multiple rules (often encoded in a long bit vector) matching the incoming packet on the projected field. Figure 1 and Figure 2 highlights their differences clearly.

From Figure 1 and Figure 2, it can be seen that the proposed decomposition scheme can leverage the benefits of traditional decomposition-based techniques without the time- and space- inefficient aggregation step. In more details, instead of searching in each dimension for the whole rule set, we first partition the rule set into several subsets using the algorithm described above. For each subset, we build a decision-tree with a hybrid of cutting algorithms efficient at different tree-construction stages. Then matching operations can be performed on these decision trees in parallel. Finally, instead of aggregating multiple bit vectors to get the final matching in traditional decomposition based techniques, we just need to perform a simple priority-based selection among the outcomes from the individual search processes.

B. FiCuts: A One-Dimensional Cutting Technique

After presenting the decomposition-based framework, we elaborate on the decision-tree construction process embedded in the middle part of Figure 2. First, we introduce a simple but effective one-dimensional cutting algorithm called Fixed intelligent Cuttings (FiCuts), which will be applied in the first stage of decision-tree construction. FiCuts derives from HiCuts, but with a better global view on the characteristics of the rule set. We use a small example of two-dimensional rule set shown in Figure 3 for subsequent discussion.

HiCuts is an 'Intelligent' cutting algorithm since it can choose the cutting dimension and decide the number of cuts with local heuristics at each tree node. However, constrained by the lack of a global view on the properties of a rule set, HiCuts does not always make intelligent cutting decisions. Take the rules shown in Figure 4 for example (a subset of 'vertical' rules in Figure 3), if we use the

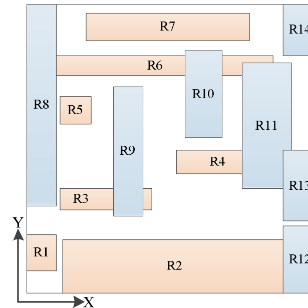

Figure 3. An example rule set

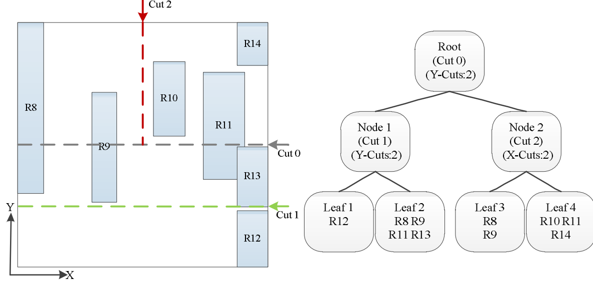

Figure 4. Non-intelligent cutting

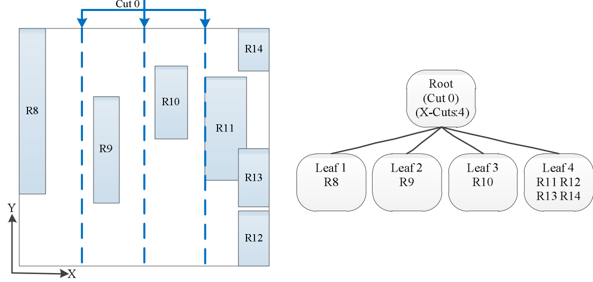

Figure 5. Intelligent cutting

heuristics described in HiCuts that cuts along the dimension with the largest number of distinct components (projected intervals) of rules in that dimension, we can see that the number of distinct components of rules in X, Y dimension is 5 and 7 respectively. As a result, the dimension been chosen at the root node is Y. Figure 4 shows the tree built if we set *binth* = 4 and *spfac* = 2.

From the tree built, we can see that three rules (R8, R9, and R11) are replicated and the depth of the tree is 2. But if HiCuts is intelligent enough, it should choose dimension X as Figure 5 shows, where a tree of depth one is built without any rule replication.

In Figure 5, the rules in the subset are all small in X-dimension. This facilitates cuts along the X-dimension with little rule replications. Since the rules have been grouped into several subsets, with each subset sharing the same small field, FiCuts is fed with this information, and exploits it for efficient cuttings. In essence, FiCuts has two features as follows:

1) Simplicity: FiCuts conducts cuttings on the subset along a fixed dimension instead of changing cutting dimensions dynamically based on local optimization considerations.
2) Adaptivity: FiCuts can decide when to stop this one-dimensional cutting and resort to other more effective cutting methods.

The first feature enables FiCuts to build short trees with significant reduction on rule replications, as contrasted in Figure 4 and Figure 5. FiCuts uses a heuristic to pick a suitable np (i.e., the number of cuts to make) in the fixed dimension. But with the shrinking of the search space, rule replications begin to rise with fine cuts. The second feature of FiCuts is to address this problem, which will be elaborated in the next part.

C. *Multi-dimensional Cutting*

As discussed in the previous part, rule replications become intense at some fine cuts, and the fixed-dimensional FiCuts is no longer the optimal algorithm. Fortunately, FiCuts is able to detect this problem and signals switching to a more appropriate multi-dimensional cutting.

FiCuts makes the decision as follows. If the number of cuts *np* at a node is less than a predefined MAXCUTS. Based on the space measure function described in HiCuts ($sm = \sum_i NumRules(child_i) + np$), it is not difficult to see that there are two cases resulting in *np* < MAXCUTS: either the subspace becomes small enough that causes serious rule replications, or there are very few number of rules for satisfying the space measure function above with MAXCUTS.

We show how HybridCuts works on the example in Figure 3. First, the decomposition step partitions the rule set into two subsets, as shown in Figure 6 and Figure 7 respectively. Then a two-stage cutting process is carried out.

At the one-dimensional cutting stage, FiCuts processes the corresponding subset until a leaf node is formed, or it discovers that it no longer works efficiently. Figure 6 shows the decision tree built for the Y-Subset by FiCuts if we set *binth*=2 and *spfac*=2. From this example, we can see that FiCuts just works fine: it builds the whole decision tree without any rule replication. However, Figure7 shows a different scenario, where pure FiCuts does not solve the problem. When FiCuts reaches Node1, it is no longer effective by continuing cutting along the X-axis. Therefore, it is necessary to resort to other effective techniques. Fortunately, this case only happens at tree nodes near the leaf level, and a multi-dimensional cutting algorithm will be applied to address it.

In the multi-dimensional cutting stage, we use HyperCuts because it works efficiently for small rule sets and the number of rules in nodes is much smaller than original subset after the processing of FiCuts. The result of the multi-dimensional cutting is illustrated with two red dotted lines in Figure 7.

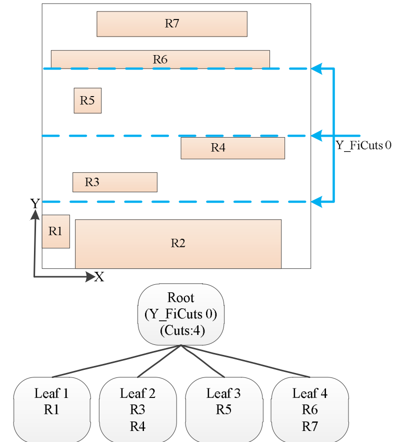

Figure 6.Y-Subset and hybrid structure

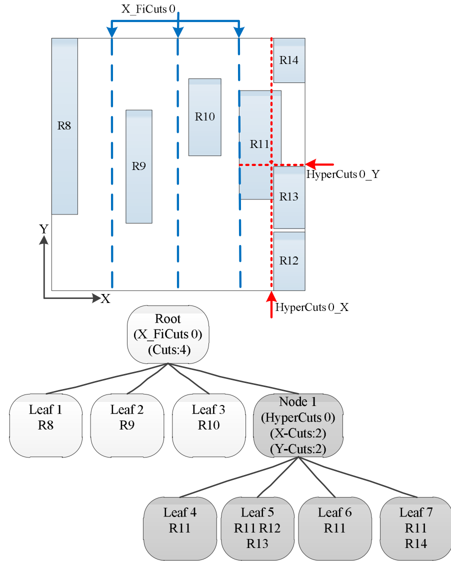

Figure 7. X-Subset and hybrid structure

D. *Optimization*

Based on previous observations that many classifiers specify a very limited number of port ranges, it can be derived that most rules have at least one small IP address field (SA or DA). Table II shows the percentage of big rules for rule set generated from ClassBench.

Based on this observation, we propose an optimization on decomposition to further reduce the number of decision trees. For a 5-tuple rule set, we decompose it into three subsets:

1) Big-subset: Both SA and DA are big field;
2) SA-subset: SA is a small field for each rule;
3) DA-subset: DA is a small field for each rule.

The processing procedure for each subset is the same as above, and in the experimentation, the result of this optimized HybridCuts will be used for comparison.

V. EXPERIMENTAL RESULTS

In this section, we present the performance results of HybridCuts with two other representative cutting techniques: EffiCuts and HyperCuts. The rule sets used in our experiments are publicly available from [23], 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., ACL_10K refers to the access control list rule set containing about 10,000 rules. Since 100K rule sets are not available in [23], we generate them using ClassBench [14], which creates synthetic classifiers with characteristics representative of real-word classifiers. We are also very grateful to the authors of EffiCuts for providing us the source code, which contains EffiCuts and an implementation of HiCuts and HyperCuts by them. This enables us to make a fair and justifiable comparison. In response, our implementation of HybridCuts has also been made public to the open-source community hosted at GitHub (https://github.com/lwj4333765/HybridCuts).

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.

TABLE II. PERCENTAGE OF BIG RULES

Big rules%	T'(16,16)	T'(12,12)	T'(8,8)
ACL_1K	0.55%	0.55%	3.93%
ACL_10K	0.22%	0.22%	3.11%
ACL_100K	0.00%	0.00%	0.00%
FW_1K	4.17%	4.30%	6.07%
FW_10K	1.54%	1.85%	4.59%
FW_100K	2.15%	2.54%	5.56%
IPC_1K	0.64%	1.49%	5.33%
IPC_10K	0.10%	1.38%	4.33%
IPC_100K	0.78%	2.80%	13.28%

A. *Memory Consumption*

Table III shows the memory consumption per rule for HyperCuts, EffiCuts, and our HybridCuts. Both EffiCuts and HybirdCuts achieve significant memory reduction compared to HyperCuts. This is especially obvious for FW classifiers (HyperCuts runs out of memory for FW-100K rule set, so its results are N/A in Table III and IV). The reason of this memory saving, as explained in [12], is that a significant fraction of FW rules (about 30%) have many wildcard fields, incurring rampant replication of rules in HyperCuts. Our HybridCuts consumes similar amount or less memory compared to EffiCuts. 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 Table III that the excellent memory efficiency of EffiCuts is kept in our work.

As for another partition-based technique ParaSplit that is implemented on FPGA, we are unable to repeat their experiments. But from the results reported in [19], we can see that our work consumes less memory than ParaSplit. For example, they report that IPC_10K consumes roughly 110 bytes per rule for EffiCuts, and 100 bytes for ParaSplit. In our experimentation, HybridCuts consumes only 46 bytes on IPC_10K. It is worth noting that our experiments reports 50 bytes for EffiCuts, significantly less than the result reported in [19], partly because their implementation of EffiCuts does not perform tree merging. More interestingly, for FW_10K, ParaSplit reports 80 bytes per rule, where our HybridCuts consumes only 25 bytes.

TABLE III. MEMORY CONSUMPTION (BYTES/RULE)

Binth=8 Spfac=4 T'=(16, 16)	**ACL**			**FW**			**IPC**		
	1K	**10K**	**100K**	**1K**	**10K**	**100K**	**1K**	**10K**	**100K**
HyperCuts	146	198	104	2.6K	2.2M	N/A	210	3.3K	0.2M
EffiCuts	39	48	52	46	36	81	39	50	36
HybridCuts	31	47	29	38	25	40	26	46	37

TABLE IV. NUMBER OF MEMORY ACCESSES

Binth=8 Spfac=4 T'=(16, 16)	ACL			FW			IPC		
	1K	10K	100K	1K	10K	100K	1K	10K	100K
HyperCuts	24	24	22	35	27	N/A	26	31	26
EffiCuts	41	41	36	95	76	117	92	110	80
HybridCuts	24	29	29	31	17	42	25	40	31

B. *Memory Accesses*

Table IV shows the performance in terms of the number of memory accesses for HyperCuts, EffiCuts, and our HybridCuts. It is clear that our work performs much better than EffiCuts. The improvement on FW rule sets is more striking, achieving an average of 3.4x speed-up. This improvement, combined with the result in Table III, justifies our claim that a careful combination of decomposition and cutting can avoid rule replications and produce efficient decision-trees at the same time.

To gain more insights, we look into the sizes of the individual subsets and their respective decision trees. The results are shown by Figure 8 and Figure 9 respectively. From Figure 8, it can be seen that very few rules are categorized as big rules, and the subsets corresponding to SA and DA fields are balanced in size. What's more important is that this balance extends to the corresponding decision-trees. As shown in Figure 9, the SA tree and DA tree are almost close in sizes in many cases. Although in a few cases, the big rules contribute an appreciable amount of memory, it does not affect the whole memory consumption seriously. Thus, enabled by the clever partition algorithm and effective FiCuts embedded in the first stage for subset trees, we can build fatter and shorter trees with seldom rule replications for each subset compared with traditional cutting algorithms.

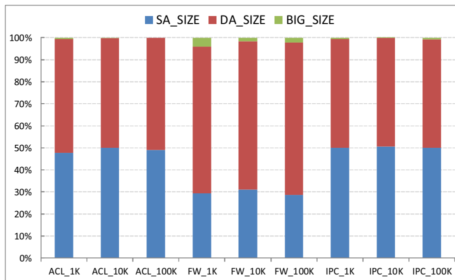

Figure 8. The sizes of subsets

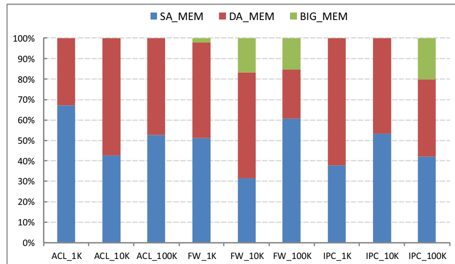

Figure 9. The sizes of trees

C. *Potential of Parallelization*

If parallel searching on multiple trees is desired for speedup in implementation, a natural concern is whether the size balance of the subsets and trees leads to similar balance in 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 10. It can be seen that in most cases, the worst-case height of a single tree (with rules in leaf nodes included) is half of or less than the overall number of memory accesses on all the trees. This means that the trees constructed are balanced in height among each other, amenable to a parallel implementation with a potential 2x speedup.

ParaSplit does not report the number of overall memory accesses for a packet, as it aims at hardware implementation, whose performance is decided by the worst-case height of the trees. Therefore, we can compare our results with ParaSplit in terms of worst-case height of trees. In ParaSplit, the results on IPC_10K and FW_10K are both 25. Our results are 18 and 10 respectively, as shown in Figure 10. The shorter trees mean that if similar hardware acceleration is employed, HybridCuts is likely to achieve higher performance than ParaSplit. Moreover, our technique requires less hardware resources, as only three decision-trees are constructed.

VI. CONCLUSION

This paper presents HybridCuts, a new packet classification technique that combines decomposition and decision-tree techniques to improve storage and performance simultaneously. We develop a rule set decomposition algorithm based on the observation that most rules have at least one small field. This decomposition avoids the trouble of result aggregation suffered by traditional decomposition-based approaches. Instead, it yields subsets of rules amenable to very efficient cutting algorithms. To better exploit the characteristics of the subsets, a two-stage cutting algorithm is carried out. Being aware of the global characteristics of the subset under processing, the first stage employs a one-dimensional cutting algorithm called FiCuts. It either ends at the leaf nodes, or invokes an alternate multi-dimensional cutting algorithm when realizing that it no longer works efficiently.

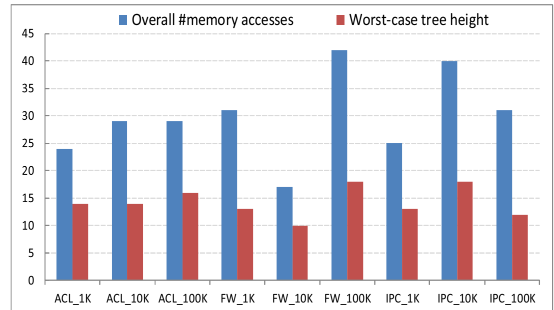

Figure 10. Potential of parallelization

Compared to the state-of-the-art EffiCuts algorithm, HybridCuts achieves significant speedup with the same level of reduction on memory consumption. Meanwhile, the very small and fixed number of decision-trees, which are balanced in size and height, enables a much easier implementation than EffiCuts in practice. Compared to ParaSplit, a recent work that also employs rule set partitioning and speeds up its algorithm with an FPGA implementation, our HybridCuts works better in terms of memory consumption and memory accesses, and is also amenable for FPGA acceleration.

Acknowledgment

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

References

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

[2] H. Liu, "*Efficient mapping of range classifier into Ternary-CAM*," in IEEE HOTI, 2002.

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

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

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

[6] Q. Dong, et al, "*Packet classifiers in ternary CAMs can be smaller*," in ACM SIGMETRICS, 2006.

[7] W. Jiang and VK. Prasanna, "*Large-scale wire-speed packet classification on FPGAs*," in FPGA, 2009.

[8] B. Vamanan and T. Vijaykumar, "*TreeCAM: Decoupling Updates and Lookups in Packet Classification*," in ACM CoNEXT, 2011.

[9] V. Srinivasan, et al, "*Fast and Scalable Layer Four Switching*," ACM SIGCOMM, 1998.

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

[11] S. Singh , F Baboescu, G. Varghese, J. Wang, "*Packet classification using multidimensional cutting*," in ACM SIGCOMM, 2003.

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

[13] TV. Lakshman and D. Stiliadis, "*High-speed policy-based packet forwarding using efficient multi-dimensional range matching*," in ACM SIGCOMM, 1998.

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

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

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

[17] P. Gupta and N. McKeown, "*Packet classification on multiple fields*," in ACM SIGCOMM, 1999.

[18] F. Baboescu and G. Varghese, "*Scalable packet classification*," in ACM SIGCOMM, 2001.

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

[20] A. Feldman and S. Muthukrishnan, "*Tradeoffs for packet classification*," in IEEE INFOCOM, 2000.

[21] Y. Qi, B. Xu, F. He, B. Yang, J. Yu and J. Li. "*Towards High-performance Flow-level Packet Processing on Multi-core Network Processors*," in ANCS, 2007.

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

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