ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification

Jeffrey Fong*, Xiang Wang*, Yaxuan Qi†, Jun Li† and Weirong Jiang‡
*Research Institute of Information Technology, Tsinghua University, Beijing, China
†Tsinghua National Lab for Information Science and Technology, Beijing, China
‡Ericsson Inc., San Jose, CA, USA

***Abstract*—Packet classification is a fundamental enabling function for various applications in switches, routers and firewalls. Due to their performance and scalability limitations, current packet classification solutions are insufficient in addressing the challenges from the growing network bandwidth and the increasing number of new applications. This paper presents a scalable parallel architecture, named *ParaSplit*, for high-performance packet classification. We propose a rule set partitioning algorithm based on range-point conversion to reduce the overall memory requirement. We further optimize the partitioning by applying the Simulated Annealing technique. We implement the architecture on a Field Programmable Gate Array (FPGA) to achieve high throughput by exploiting the abundant parallelism in the hardware. Evaluation using real-life data sets including OpenFlow-like 11-tuple rules shows that ParaSplit achieves significant reduction in memory requirement, compared with the-state-of-the-art algorithms such as HyperSplit [6] and EffiCuts [8]. Because of the memory efficiency of ParaSplit, our FPGA design can support in the on-chip memory multiple engines, each of which contains up to 10K complex rules. As a result, the architecture with multiple ParaSplit engines in parallel can achieve up to Terabit per second throughput for large and complex rule sets on a single FPGA device.**

***Keywords*-packet classification; terabit; OpenFlow; FPGA**

I. Introduction

The rapid growth of the Internet has led to a great demand for the Internet to provide higher bandwidth, lower latency and better security. In order to achieve these goals, network devices need to identify the data or packets being transmitted at high speed. The process of categorizing packets based on predefined rules is known as *packet classification*, a fundamental technique used by switches, routers, firewalls and network intrusion detection systems (NIDS). A packet is classified based on multiple fields extracted from its packet header. These fields are matched against predefined rules or filters to determine the type of actions (e.g. drop or forward) to be taken on the packet.

Although it has been extensively studied, traditional packet classification is used only for access control and firewall with relatively small static rule sets [1]. Most of existing methods do not meet the requirements for emerging applications on the Internet. For example, current data center networks require fine-grained flow control in order to route traffic efficiently. This brings a whole new set of requirements and challenges. First, the network devices within large-scale data centers already have Terabit per second (Tbps) switching and routing capacity [2]. It is a challenge for packet classification to meet such extremely high bandwidth. Second, as the numbers of new applications and protocols arise, the rule sets are becoming increasingly large and complex. Large-scale enterprise networks need fine-grained flow control over thousands of physical and virtual machines, while new network protocols such as OpenFlow [3] employ complex rule specifications containing more than 10 header fields. Thus the second challenge is to support a large number of complex rules.

The predominant industrial solutions today are based on multi-core or Ternary Content Addressable Memory (TCAM) platforms. Multi-core based solutions are cheap but they are limited in performance due to memory I/O bottlenecks [4]. Even high-end multi-core based platforms are limited to a classification rate of 10 Gbps which is significantly lower than the 100 Gbps links emerging in the Internet backbones and data centers. TCAM-based solutions are widely used because of their ability to process packets at high speed. However, TCAMs are expensive, and suffer from scalability and high power consumption [5]. TCAMs only supports prefix matching which means a rule represented in range must be converted into prefixes. Such range-to-prefix conversion may exponentially increase the number of rules.

On the other hand, Field Programmable Gate Array (FPGA) has recently become an attractive option for implementing packet classification engines [4], [5]. FPGA combines the flexibility of software with the performance of hardware. Although some existing designs can achieve high performance [4], [5], they are still limited to small and simple rule sets due to the large consumption of on-chip resources such as memory. Memory consumption is largely dependent on the complexity and the size (i.e. the number of rules) of the rule set [6]. Take the publicly available rule sets [7] as an example. It has been shown that with the same number of rules, the IP Chain (IPC) and Firewall (FW) rules consume 10–1000 times more memory than Access Control List (ACL) [6]. Because of the high memory consumption, current solutions need to store rules in external DRAM and thus cannot meet the throughput requirements.

This paper presents a scalable architecture, named *ParaSplit*, which is capable of classifying packets at Terabit throughput by allowing multiple engines implemented on

a single FPGA device. The scalability is achieved by using an optimized rule set partitioning algorithm that increases parallelism while reducing memory requirement. The main contributions include:

- We propose a rule set partitioning algorithm to reduce rule set complexity, which leads to memory reduction. By incorporating the range-point conversion, the algorithm converts rules to points and utilizes clustering algorithms for effective partitioning. Simulated annealing is then applied to find an optimal partitioning. Compared with the algorithms without rule set partitioning, our algorithm achieves on average 100-fold reduction in memory. Compared with EffiCuts [8] which is the state-of-the-art algorithm with rule set partitioning, our algorithm achieves 20%–500% reduction in memory.
- We design a parallelized decision-tree-based architecture to achieve high throughput. We employ the HyperSplit algorithm to build the decision tree for each rule subset. All decision tree are searched in parallel. The results from each decision tree is aggregated to obtain the final result. We pipeline the tree traversal to achieve the high throughput of one packet per clock cycle.
- We implement the ParaSplit architecture on a Xilinx Virtex 5 FPGA. It takes advantage of the dual-port Block RAM (BRAM) available on modern FPGAs to double the throughput. The design can classify 64-byte packets at a throughput up to 120 Gbps, while supporting 10K FW rules. To the best of our knowledge, this is the first FPGA design that supports 10K FW rules while sustaining over 100 Gbps throughput. With the new-generation FPGAs, over 10 ParaSplit engines can be placed on a single device to provide an aggregated throughput over 1 Tbps for 10K complex rules.
- We evaluate our algorithm using OpenFlow-like 11-tuple rule sets from real-life enterprise users. Preliminary results show that ParaSplit consumes up to three orders of magnitude lower memory than HyperSplit.

The rest of the paper is organized as follows. Section II introduces the background. Section III presents the rule set partitioning algorithms. Section IV describes the parallel search architecture and the FPGA design. Section V provides performance evaluation results. Section VI concludes the paper and discusses the future work.

II. Background

A. Problem Statement

Packet classification is to find the best matching rule from a predefined rule set for a packet. Each rule R contains F fields. Each field has a range that a packet could match. From a geometric point of view, each R represents a hyper rectangle in F-dimensional space. If a packet p matches a particular rule R, the point represented by p falls into the hyper-rectangle specified by R. Hence packet classification can be treated as a point location problem in computational geometry. For N rectangles in F-dimensional space, it has been shown that the best bounds for locating a point is either $\Theta(\log^{F-1} N)$ time with $\Theta(N \log^{F-1} N)$ space, or $\Theta(F \log N)$ time with $\Theta(N^F)$ space [6], [9]. Therefore, the mathematic complexity of packet classification is extremely high as the number of rules or dimensions increase. Fortunately, packet classification rules in real-life applications have structural redundancies [6], [8]. Although different types of rules have various statistical characteristics [10], the complexity of real-life rules is far less than the theoretical bounds. Therefore, packet classification algorithms can exploit such redundancies and achieve practical search speed with modest memory usage.

B. Packet Classification Algorithms

A lot of algorithms have been proposed for packet classification [1]. They can generally be categorized into two major schemes: decomposition-based and decision tree -based [5].

Decomposition-based algorithm such as RFC [15] and HSM [16] are considered to provide good performance at the expense of high memory consumption [1], [8]. These algorithms perform independent searches on each field and finally combine the search results from all fields. But they are memory-inefficient for large rule sets.

Decision tree based algorithms [6], [10], [11] are considered as the most popular algorithms for packet classification. 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. Such a recursive process builds a decision tree. An incoming packet would traverse the tree until it reaches the leaf node that stores the matching rule. There are a few variations of the decision tree, differing on the method of cutting the space. HiCuts [10] makes multiple evenly-spaced cuttings on a single dimension at each internal node. HyperCuts [11] extends HiCuts by allowing multiple dimensions to be cut simultaneously to reduce the height of the decision tree. To avoid rule replication by equal-sized cuts, HyperSplit [6] uses non-equal sized cuts for more efficient memory usage. However, even with the optimized binary space cutting (like HyperSplit), the memory usage of decision tree based algorithms still grows exponentially as the number of rules increases. As shown in Figure 1(a), the HiCuts algorithm cuts the search space into two equal-sized subspaces. The rules R2 and R3 are replicated in the respective subspaces. By aligning the cuttings at the edges of the rules, the HyperSplit algorithm is able to reduce rule replication (Figure 1(b)). However, HyperSplit still cannot eliminate all rule replication, especially for complex rule sets. For instance, using HyperSplit on the 10K FW rule set $fw1\_10K$ from [7], some rules are replicated by up to 7,000 times.

Observing the above problem in decision tree based methods, the EffiCuts algorithm [8] is proposed to reduce rule

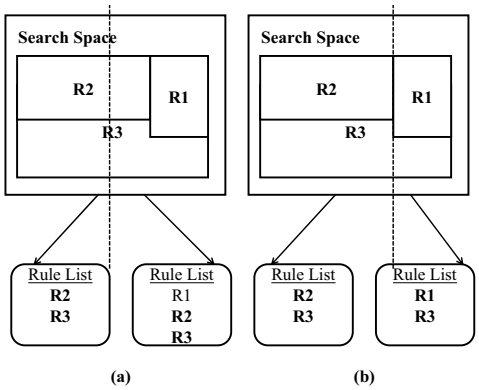

Figure 1. (a) Unaligned cut point (causing replication of R2); (b) Aligned cut point to the boundary of R1 and R2 (reducing replication)

replication through categorizing rules into separate trees. The idea is to group overlapping rules separately and build independent decision trees. The results show a significant reduction in rule replication. As a result, the memory usage is reduced by a factor of 57 compared with HyperCuts. However, the grouping algorithm used by EffiCuts is not scalable because it results in $\Theta(2^F)$ groups. For a typical 5-tuple IPv4 packet header (i.e. $F = 5$), EffiCuts would generate up to 32 subsets of rules. Although EffiCuts proposed selective tree merging to reduce the number of trees, the merging is computation-intensive and difficult to adopt for higher dimensional rules. For example, applying EffiCuts on OpenFlow 12-tuple rules will generate over 4000 groups.

C. FPGA Solutions

The majority of the FPGA-based packet classification solutions are hardware implementation of traditional packet classification algorithms. Because decomposition-based algorithms search each dimension independently, they suit the parallel nature of the hardware [12], [13]. However, these designs suffer from large memory consumption and cannot support large and complex rule sets such as 10K FW or IPC rule sets. Recent work [4], [5] is focused on decision tree based algorithms. By pipelining the tree traversal, these designs are capable of classifying multiple packets at the same time. [5] implements an optimized HyperCuts on a Xilinx Virtex 5 FPGA and achieves over 80 Gbps throughput. It extends the HyperCuts tree into a two-dimensional tree to reduce memory usage. Through deep pipelining, it is able to achieve high throughput at the cost of high latency. [4] proposes three optimization methods for the HyperSplit algorithm and implements it on a Xilinx Virtex 6 FPGA. The optimization techniques reduces pipeline depth to achieve lower latency. The implementation sustains over 100 Gbps for the ACL rule set.

Although decision tree based solutions provide efficient mapping to hardware through pipelining, they still suffer large memory consumption due to rule replication. While they may classify 10K ACL rules while sustaining 100 Gbps, they have difficulty in implementing complex rule sets such as FW or IPC. Even the smallest FW 1K rule set would consume 4 times more memory than the largest ACL 10K rule set.

III. Rule Set Partitioning

A. Motivation

As the number of overlapping rules or the number of dimensions increases, the complexity of the rule set increases exponentially. On the other hand, the amount of memory resources available on FPGA is limited. Since the rules are stored in the on-chip BRAM of the FPGA, a more memory-efficient algorithm means that the FPGA design can support larger, more complex rule sets. Hence, we propose an optimized rule set partitioning algorithm to reduce the complexity of the rule set and thus the memory requirement.

B. Rule Set Complexity

With N overlapping rules with F fields, it may require up to $(2N+1)^F$ non-overlapping hyper rectangles in the worst case. Hence the space complexity of the rule set is $\Theta(N^F)$ [14]. As the number of rules or the number of dimensions increases, the complexity increases exponentially. The goal of rule set partitioning is to reduce this complexity through intelligent grouping of rules. Assuming that the rule set R is divided evenly into K groups (i.e. S_1, S_2, S_3, ..., S_K) such that each group has $\frac{N}{K}$ rules. The overall complexity of the rule set becomes $\Theta(K \cdot (\frac{N}{K})^F) = \Theta(\frac{N^F}{K^{F-1}})$. Hence by partitioning the rules into K groups, the space complexity can be reduced by a factor of K^{F-1}.

Since overlapping rules cause rule replication during the cutting of the search space in decision tree based algorithms, a good rule set partitioning can reduce the rule replication by removing overlapping rules. However, finding an optimal solution for grouping is difficult. Grouping the rules can be seen as a combinatory mathematic problem. Given N differentiable rules and M non-differentiable groups, the number of different grouping choices is equal to:

$S(N,1) + S(N,2) + \cdots + S(N,M), N \geq M$

$S(N,1) + S(N,2) + \cdots + S(N,N), N \leq M$

where $S(N,x)$ is the Sterling number. For example, there are more than 9.0×10^{74} different ways to partition 100 rules ($N = 100$) into 6 groups ($M = 6$). This number increases exponentially with the number of rules (N) and the number of different groups (M). Furthermore, the performance/cost curve is not smooth and thus it is difficult to apply tradition optimization techniques for finding the global optimal partitioning. We study two algorithms to partition the rule set so that the complexity of the rule set is minimized. The first algorithm is to approximate the optimal partitioning through clustering. The second algorithm is to employ simulated annealing to approach the optimal partitioning. We propose to combine these two algorithms to obtain the optimal partitioning at low computation cost.

C. Clustering with Range-Point Conversion

It is difficult to cluster the F-dimensional hyper-rectangle rules directly. Since the point location problem is a dual problem with range search, a hyper rectangle in F-dimensional space can be converted to a point in $2F$-dimensional space [9]. The range-point (RP) conversion has been proven to be a dual-problem in computation geometry. This is accomplished by treating the starting point and the ending point of each field as separate dimensions. For a rule with its range represented by $a < x < b$, the mapping is done such that $(+a)$ is mapped to dimension x_s and $(-b)$ is mapped dimension x_e. Hence, for an incoming packet, x, to satisfy a rule, $a < x < b$, it must satisfy:

$(a < x)$ & $(x < b) \equiv (a < x)$ & $(-b < -x)$.

The condition of $(a < x)$ and $(-b < -x)$ represents a region formed by the line $x_s = a$ and $x_e = -b$. So all rules, represented as points, that satisfy the incoming packet, must be enclosed by the region. Table I shows a sample 1-dimensional rule set with 2 rules and 2 incoming packets to be classified. Rule 1 represents the range [2, 3] on Field-x, while Rule 2 represents [1, 6] on that same field. So for packet A whose value is 3 on Field-x, it matches both Rule 1 and Rule 2. Another packet, B, whose Field-x value is 5, matches only Rule 2. Figures 2 and 3 depict the geometric views of the rules and packets before and after range-point (RP) conversion, respectively. As shown in Figure 3, both Rule 1 and Rule 2 are enclosed by the region formed by packet A. This corresponds to that packet A matches both Rule 1 and Rule 2.

Table I
SAMPLE RULES AND PACKETS

Field-x	Start (s)	End (e)
Rule 1	2	3
Rule 2	1	6

	Field-x	Matched rules
Packet A	3	Rules 1, 2
Packet B	5	Rule 2

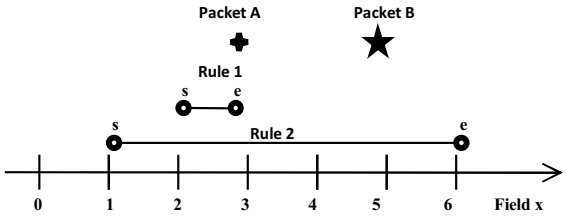

Figure 2. Traditional View

Once the rules are represented as points in $2F$-dimensional space, we employ the K-means clustering algorithm to group the rules. A number of grouping heuristic can be used to partition the rule set, such as:

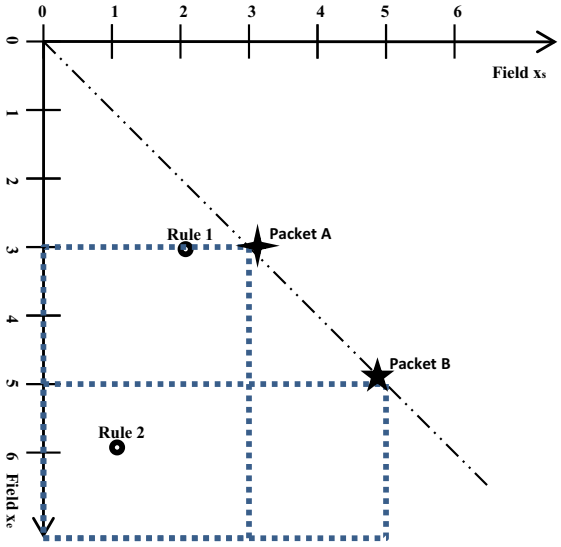

Figure 3. Range-Point Conversion View

1) *Minimum distance*: For rule R_i, select set S_j such that the summation of average distance between rules within all sets is minimized, $\min_j(\sum_{l=1}^{K} avg\_dist(R_i \cup S_j))$. This is to group similar rules together, i.e. the subset contains rules with similar values in each field.
2) *Maximum distance*: For rule R_i, select set S_j such that the summation of the average distance is maximized (i.e. minimizing the inverse distance), $\max_j(\sum_{l=1}^{K} avg\_dist(R_i \cup S_j))$. The idea is to remove overlapping rules by grouping together rules that are far from each other.
3) *Distance from origin*: For rule R_i, select set S_j such that the distance from the origin in $2F$ space is similar. This aims to balance between similar and dissimilar groups.

After the rule set is partitioned into multiple groups, each group is built into a decision tree. We choose the HyperSplit algorithm [6] as the decision tree algorithm for its superior memory efficiency (consuming at least an order of magnitude less memory) compared with other existing work. Based on our experiments, the first heuristic (Minimum distance) is chosen for its excellent reduction in memory usage for the HyperSplit algorithm.

D. Optimal Partitioning with Simulated Annealing

The clustering with range-point conversion cannot guarantee an optimal partitioning. Also note that not all overlapping rules cause replication if the rules are not broken into smaller segments. In fact, some rules are better grouped together even though they have overlapping. It is difficult to identify good overlapping rules from bad ones. We apply the Simulated Annealing (SA) technique to find the optimal rule set partitioning.

First, we consider the simulated annealing algorithm without the clustering. After defining the rule set and the number of groups, the simulated annealing algorithm works by first setting the initial temperature of the system and making the initial grouping of the rules randomly. Then the algorithm calculates the cost of the system and then randomly selecting 2 sets, S_i and S_j. There are three possible actions that could operate on the two groups:

1) Move R_i from S_i to S_j
2) Swap R_i from S_i with R_j from S_j
3) Move R_j from S_j to S_i

Actions 1 and 3 are to allow uneven number of rules within a group. An optimal grouping may have unevenly distributed rules in the groups. If only swapping rules is allowed, the number of rules in each group is restricted to the same initial number which is determined by the initial grouping algorithm. After performing the action, the new cost of the system is then calculated. The algorithm accepts this new system state with a probability of $P_{accept} = e^{-\Delta D/T}$, where ΔD is the change in cost (= new cost - initial cost) and T is the temperature of the system. The algorithm then repeats the process of selecting groups, and swapping rules until the system reaches a satisfactory cost or a predetermined number of iterations. To reduce the memory usage, the cost function is chosen to be the number of leaf nodes produces by the tree which the algorithm tries to minimize.

Because of the randomness in simulated annealing, as well as the large search space, it takes a long time for simulated annealing to reach an acceptable solution. We propose to combine the clustering and the simulated annealing algorithms: We use the clustering with range-point (RP) conversion to obtain the initial groups and then apply the simulated annealing (SA) to approach the optimal partitioning. This helps finding the optimal partitioning with faster convergence. We conduct experiments using $fw1\_1K$ rule set from [7]. Figure 4 shows that, by using the combined scheme (RP+SA), 5,000 to 10,000 iterations are sufficient to find an optimal partitioning where the cost reaches an asymptote. On the other hand, the algorithm with only simulated annealing (SA) requires 15,000 to 20,000 iterations before reaching the initial cost of the combined scheme. Note that the result of RP+SA with 0 iteration is equal to that of RP without SA. Thus we can see that SA can improve the cost of clustering with RP by approximately 15%.

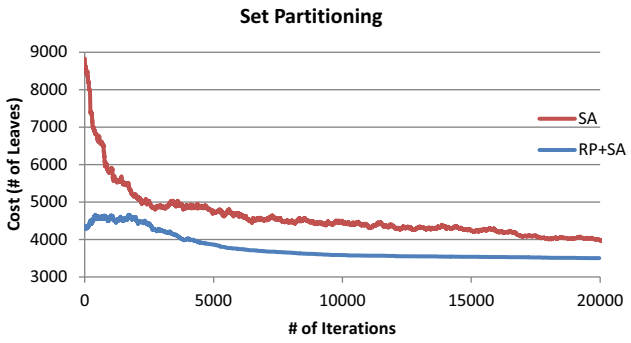

Figure 4. Simulated Annealing (SA) only vs. Range-Point (RP) + SA

IV. Architecture and Implementation

A. Overview

With the near-optimal rule set partitioning, each group (i.e. subset) is built into a separate decision tree using the HyperSplit algorithm. Each decision tree is mapped onto a dedicated pipeline. As an incoming packet arrives, the packet header is fed to all decision tree pipelines. Because a packet may match a rule in the different decision trees, a hardware module is required to select the highest priority rule from the possibly multiple matching rules.

B. Tree Mapping

We use the HyperSplit algorithm to build the decision tree for each rule subset. The tree traversal can be pipelined in the hardware to achieve high throughput [4], [5]. A decision tree is mapped into hardware by collecting all the nodes in each level of the tree and mapping them into a stage. Each stage contains the decision tree processing logic and a separate RAM storing the tree node information. The processing logics are the same for all stages with the only difference being the size of the BRAM allocated. Different stages can process different packets at the same time so that the overall throughput of the design is one packet per clock cycle. Because of the simplicity of the processing logic, the pipeline can run at high clock frequency. Each stage relies on the information provided by the previous stage to determine the traversal of the incoming packet.

C. FPGA Implementation

The dual-port BRAM on the FPGA can complete two reads per clock cycle. It can be used to implement two pipelines (i.e. dual-pipeline) without requiring any additional BRAM resources. The only cost is slightly higher logic usage. So on every clock cycle, two packets can be fed to the engine. Hence, the throughput is doubled.

Furthermore, due to low resource usage of the design, multiple engines can be implemented onto a single FPGA to achieve even higher performance. The throughput achieved by such a design is $2 * P * F * 64 * 8$ Mbps, where P denotes the number of engines and F the frequency of the engine (MHz). The overall architecture of the multi-engine, dual-pipeline design is shown in Figure 5.

D. Incremental Update

Incremental update is extremely important for highly dynamical applications and environments such as switching or routing. Incremental update such as rule insertion can be accomplished by inserting the rule into the group so that the memory usage is minimized. This may involve

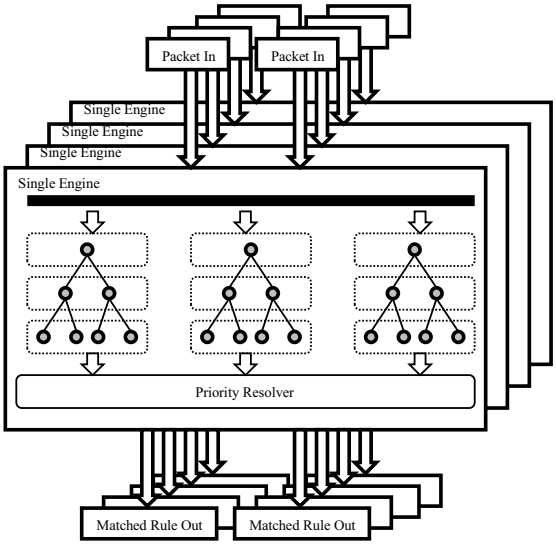

Figure 5. Multiple engines each with dual-pipeline

rebuilding multiple decision trees for each rule insertion. Rule removal can be completed by removing the rule from the associated group and rebuilding the decision tree for that group. However, a large number of rule insertion and removal could potentially move the rule set partitioning out of the optimal. Hence after a certain number of insertion and/or deletion, the algorithm should perform a complete update of the grouping.

On-the-fly update to the hardware can be done through inserting write-bubbles into the respective pipeline [5]. This allows the decision tree to be altered on-the-fly without halting the classification engine. However, for large scale changes to the data structure, a more efficient method called the back buffer engine. This method builds an extra pipeline and populates the extra pipeline with the new data set during dynamic update. Once the pipeline is ready, swap the pipelines. This method sacrifices extra FPGA logic and RAM for more efficient on-the-fly updates.

V. Performance Evaluation

A. Test Bed and Data Set

We conduct experiments using publicly available 5-tuple packet classification rule sets [7] to evaluate the effectiveness of our algorithms and hardware architecture. Only the IP Chain (IPC) and Firewall (FW) rules have been used for testing because they are the most representative of complex rule sets. The number of rules contained in these rule sets ranges from 100 to 10,000.

The FPGA device used for our implementation is the Xilinx Virtex-5 (model: XC5VSX240T) containing 4,200 Kb of Distributed RAM and 18,576 Kb of BRAM. All the experimental results are obtained through the post place and route simulation.

We also evaluate our algorithm for OpenFlow-like 11-tuple rule sets. The rule sets are generated based on 216 real-life 11-tuple rules from enterprise customers. The 11 tuples include (1) source IPv6, (2) destination IPv6, (3)source MAC, (4) destination MAC, (5) 32bit input port, (6) 32bit time stamp, and the traditional 5 tuples. The number of rules ranges from 400 to 2,000.

B. Algorithm Evaluation

We consider the following two performance metrics:

- Average memory required per rule: This is, on average, the amount of memory required per rule. This is computed by dividing the total memory requirement by the number of rules.
- Worst-case tree height: This is the height of the deepest decision tree. As the pipeline depth (i.e. the number of stages) is determined by the tree height, a shorter tree reduces latency and the amount of hardware logic.

For evaluating the rule set partitioning algorithm, different numbers of subsets have been tested. To determine the best number of subsets, 1 to 26 subsets have been tested. Results show that using 8 groups provides good balance between the number of trees and the memory usage and tree height. For this reason, we choose 8 groups for the majority of performance evaluation.

The ParaSplit algorithm (which builds multiple HyperSplit trees after rule set partitioning) is first compared with the original HyperSplit. The results are shown in Figures 6 and 7. FW and IPC both benefit greatly from rule set partitioning with memory reduction of more than 2 orders of magnitude for the $fw1\_10K$ rule set. ParaSplit always outperforms the original HyperSplit on all rule sets. On average, rule set partitioning reduces the memory consumption by an average of 150 fold. Rules that previously require several hundreds of megabytes to a gigabyte memory can now fit into the few megabytes of BRAM of a single FPGA.

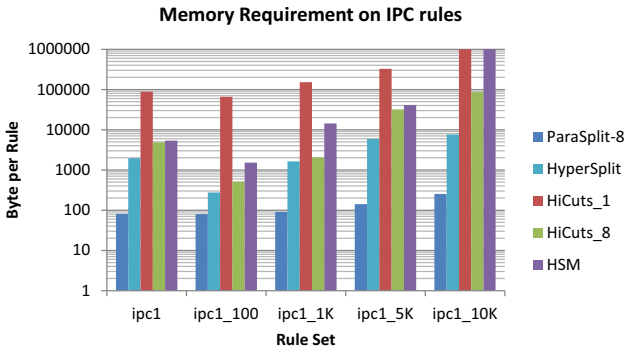

Figure 6. ParaSplit vs. well-known algorithms on IPC rules

Figures 6 and 7 also compare ParaSplit with other well-known packet classification algorithms such as HiCuts and HSM. We evaluate both HiCuts with linear search

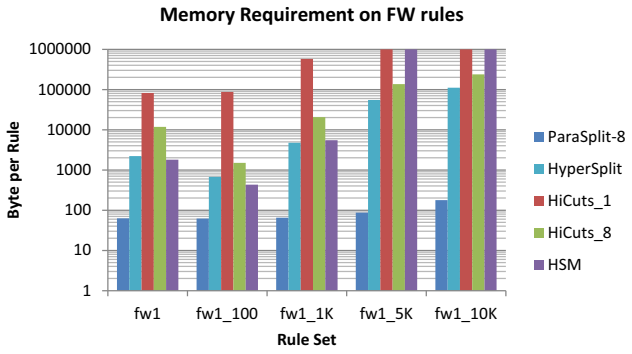

Figure 7. ParaSplit vs. well-known algorithms on FW rules

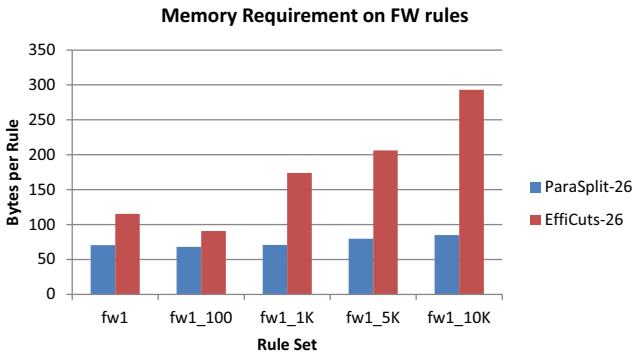

Figure 9. ParaSplit-26 vs. EffiCuts-26 on FW rules

(HiCuts_8) and HiCuts without linear search (HiCuts_1). Most decision tree based algorithms allow the tree leafs to contain more than 1 rule. A threshold is defined to stop decision tree algorithm from cutting the search space once the number of rules within the search space is less than this value, named $binth$. The leaf node stores a pointer to a list of rules which is searched linearly to determine which rule is matched. $binth = 8$ for HiCuts_8. For large and complex rule sets such as $ipc1\_10K$ and $fw1\_10K$, HSM and HiCuts_1 both fail to generate data structure due to exhaustion of memory (over 4GB). However, with ParaSplit, the data structure consumes 2.2MB and 1.6MB, respectively, for $ipc1\_10K$ and $fw1\_10K$. Hence it can easily fit within the 2.5MB BRAM available on the FPGA. Note that ParaSplit supports the decision tree with linear search, which is not implemented in our design because linear search will significantly increase the processing latency for high speed packet classification. Actually there is no need to use the linear search, as the ParaSplit algorithm already generates a data structure small enough to fit within the FPGA.

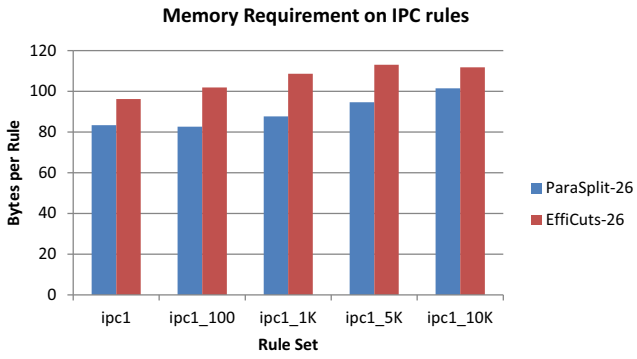

Figure 8. ParaSplit-26 vs. EffiCuts-26 on IPC rules

We then compare ParaSplit with EffiCuts [8] which is the state-of-the-art algorithm with rule set partitioning. The number of groups is 26 for both ParaSplit and EffiCuts. Because EffiCuts is not publicly available, we implement the EffiCuts scheme without tree merging. HyperSplit is used to build the decision trees for all the groups to isolate the grouping performance. Results are shown in Figures 8 and 9. ParaSplit requires 20% to 500% less memory than the EffiCuts scheme. This is because ParaSplit achieves more optimal rule set partitioning.

Table II compares the worst-case tree height achieved by different decision tree algorithms on various rule sets. Compared with HyperSplit, ParaSplit reduces the tree height dramatically. ParaSplit using only 8 groups achieves the similar or even better results than EffiCuts with 26 groups.

Table II
WORST-CASE TREE HEIGHT

Rule set	ParaSplit-8	HyperSplit	EffiCuts-26
$ipc1\_100$	12	18	16
$ipc1\_1K$	17	26	19
$ipc1\_5K$	21	32	23
$ipc1\_10K$	25	35	24
$fw1\_100$	12	20	13
$fw1\_1K$	16	28	20
$fw1\_5K$	20	34	26
$fw1\_10K$	25	37	27

C. FPGA Results

The performance and the resource utilization for the different rule sets are shown in Table III. Because the algorithm is optimized to suit the FPGA architecture, it is able to maintain high throughput and low memory consumption even with large and complex rule sets. The bottleneck of this architecture is the memory rather than the logic utilization. Our design can support up to 10K FW rule sets while sustaining over 100 Gbps throughput for a single ParaSplit engine. To the best of our knowledge, this is the first FPGA design that supports 10K FW rule sets while sustaining 100 Gbps throughput. Current Xilinx Virtex-7 FPGAs have up to 85 Mb of BRAM. Hence more than ten ParaSplit engines can fit within a single FPGA chip to achieve over terabit throughput (Table IV). Even for the large and complex $fw1\_5K$ rule set, ParaSplit can provide 1.18 Tbps throughput.

Table III
PERFORMANCE AND RESOURCE USAGE WITH A SINGLE ENGINE

Rule set	Max freq. (MHz)	Max thrupt (Gbps)	Tree height	# slice	# BRAM
$fw1\_100$	120.86	123	12	7270	48
$fw1\_1K$	118.02	120.8	16	10274	151
$fw1\_5K$	105.52	108.0	20	13834	253
$fw1\_10K$	100.23	102.6	25	12095	399

Table IV
PERFORMANCE WITH MULTIPLE ENGINES

Rule set	BRAM usage per engine	# Engines	Aggregated Throughput (Tbps)
$fw1\_100$	1.6%	60	7.38
$fw1\_1K$	5.2%	19	2.29
$fw1\_5K$	8.7%	11	1.18
$fw1\_10K$	13.9%	7	0.72

D. Scalability with OpenFlow-like Rules

We conduct experiments using 11-tuple rule sets of different sizes. Table V compares the performance between ParaSplit (with 8 groups) and HyperSplit. ParaSplit achieves up to 3 orders of magnitude lower memory consumption than HyperSplit. The worst-case tree height of ParaSplit is also at least 30% smaller than that of HyperSplit.

Table V
PERFORMANCE WITH 11-TUPLE RULE SETS

# of rules	ParaSplit-8		HyperSplit	
	Bytes/rule	Tree height	Bytes/rule	Tree height
400	1.25	4	8.75	16
800	2	11	196.25	23
1200	3.625	15	559.5	23
1600	5.625	16	10141.5	29
2000	17.625	20	14401.5	30

VI. CONCLUSION AND FUTURE WORK

This paper presents ParaSplit, a scalable packet classification architecture that achieves high throughput while supporting complex rule sets. We develop a rule set partitioning algorithm based on range-point conversion to reduce memory consumption. We further improve the partitioning algorithm by employing the Simulated Annealing technique. We design the high-throughput architecture upon the pipelining of decision trees. Multiple pipelines are built for different rule subsets to increase parallelism. The result is a FPGA-based packet classification engine capable of classifying a large number of complex rules, which previously is not possible. This design is extensible to future applications where larger, higher dimensional rule sets are used.

Future work includes FPGA implementation for OpenFlow-like rule sets and using heterogeneous classification engines. In the current design, each group builds a decision tree. However, it may be better to group rules according to the best method of classification. One group may classify better using decomposition method while other groups may classify best using decision tree based methods. FPGA provides the unique flexibility to classify different sets of rule differently according to the best classification method for each set. FPGA can support heterogeneous classification and can be used to improve both memory efficiency and classification throughput.

REFERENCES

[1] P. Gupta and N. McKeown, *Algorithms for packet classification*, IEEE Networks, vol. 15(2), Mar/Apr 2001, pp. 24-32.

[2] Juniper Networks Inc. www.juniper.net

[3] OpenFlow. www.openflow.org

[4] Y. Qi, J. Fong, W. Jiang, X. Bo, J. Li, and V. Prasanna, *Multi-dimensional Packet Classification on FPGA: 100Gbps and Beyond*, Proc. Intl. Conf. on Field-Programmable Technology (FPT '10), Dec. 2010.

[5] W. Jiang and V. Prasanna, *Large-scale wire-speed packet classification on FPGAs*, Proc. ACM/SIGDA International Symp. on Field programmable gate arrays (FPGA '09), Feb. 2009.

[6] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, *Packet classification algorithms: from theory to practice*, Proc. IEEE Conf. on Computer Communications (INFOCOM '09), Apr. 2009, pp. 648-656.

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

[8] B. Vamanan, G. Voskuilen, T. N. Vijaykumar, *EffiCuts: optimizing packet classification for memory and throughput*, Proc. SIGCOMM '10, Oct. 2010, pp. 207-218.

[9] M. Overmars and A. Stappen, *Range searching and point location among fat objects*, Journal of Algorithms, vol 21(3), Nov. 1996, pp. 240-253.

[10] P. Gupta and N. Mckeown, *Packet Classification using Hierarchical Intelligent Cuttings*, Proc. Hot Interconnects, 1999, pp. 34-41.

[11] S. Singh, F. Baboescu, G. Varghese and J. Wang, *Packet classification using multidimensional cutting*, Proc. SIGCOMM '03, Aug. 2003, pp. 213-224.

[12] H. Song and J. W. Lockwood, *Efficient packet classification for network intrusion detection using FPGA*, Proc. FPGA '05, 2005, pp. 238-245.

[13] Gajanan S. Jedhe, Arun Ramamoorthy, and Kuruvilla Varghese, *A Scalable High Throughput Firewall in FPGA*, Proc. FCCM '08, 2008, pp. 43-52.

[14] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, *Fast and Scalable Layer Four Switching*, Proc. SIGCOMM '98, Sept. 1998, pp. 191-202.

[15] P. Gupta and N. McKeown. *Packet classification on multiple fields*, Proc. SIGCOMM '99, August 1999, pp. 147-160.

[16] Bo Xu, Dongyi Jiang, and Jun Li; *HSM: a fast packet classification algorithm*, Proc. AINA '05, March 2005, pp. 987-992.