

# Energy-Efficient Multi-Pipeline Architecture for Terabit Packet Classification

Weirong Jiang and Viktor K. Prasanna  
Ming Hsieh Department of Electrical Engineering  
University of Southern California  
Los Angeles, CA 90089, USA  
Email: {weirongj, prasanna}@usc.edu

**Abstract**—Energy efficiency has become a critical concern in designing high speed packet classification engines for next generation routers. Although TCAM-based solutions can provide high throughput, they are not scalable with respect to power consumption. On the other hand, mapping decision-tree-based packet classification algorithms onto SRAM-based pipeline architectures becomes a promising alternative to TCAMs. However, existing SRAM-based algorithmic solutions need a variable number of accesses to large memories to classify a packet, and thus suffer from high energy dissipation in the worst case. This paper proposes a partitioning-based multi-pipeline architecture for energy-efficient packet classification. We optimize the HyperCuts algorithm, which is considered among the most scalable packet classification algorithms, and build a decision tree with a bounded height. Then we study two different schemes to partition the decision tree into several disjoint subtrees and map them onto multiple SRAM-based pipelines. Only one pipeline is active for classifying each packet, which takes a bounded number of accesses to small memories. Thus the energy dissipation is reduced. Simulation experiments using both real-life and synthetic traces show that the proposed architecture with 8 pipelines can store up to 10K unique rules in 0.336 MB SRAM, sustains 1 Tbps throughput, and achieves 2.25-fold reduction in energy dissipation over the baseline pipeline architecture that is not partitioned.

## I. INTRODUCTION

Multi-field packet classification is a critical function that enables network routers to support a variety of applications such as firewall processing, Quality of Service (QoS) differentiation, virtual private networks, policy routing, traffic billing, and other value added services. An Internet Protocol (IP) packet is usually classified based on the five fields in the packet header: the 32-bit source/destination IP addresses (denoted **SA/DA**), 16-bit source/destination port numbers (denoted **SP/DP**), and 8-bit transport layer protocol (denoted **Prtl**). Individual entries for classifying a packet are called *rules* which specify the value ranges of each field in the packet header. As the network traffic continues to grow rapidly, packet classification has become a major performance bottleneck in routers. For example, the current link rate has been pushed towards 100 Gbps [1], which requires processing a packet every 3.2 ns in the worst case (where the packets are of minimum size i.e. 40 bytes).

To address the throughput challenge, recent research in this area seeks to combine algorithmic and architectural approaches, most of which are based on ternary content addressable memories (TCAMs) [2]–[4] or various hashing-based schemes such as Bloom Filters [5]–[7]. However, Bloom-Filter-based solutions suffer from false positives, and always need a secondary module for verification, which may be slow and can limit the overall performance [8]. TCAMs can perform fast parallel searches over all rules in one clock cycle. But as shown in [9]–[11], TCAMs are expensive and not scalable with respect to clock rate, power consumption, or circuit area, compared to static random access memories (SRAMs). It is estimated that the power consumption per bit of TCAMs is 150 times more than that for SRAMs [10].

On the other hand, mapping decision-tree-based packet classification algorithms (e.g. HyperCuts [12]) onto SRAM-based pipeline architectures becomes a promising option for high throughput packet classification engines [13], [14]. However, existing SRAM-based algorithmic solutions still suffer from high energy dissipation,<sup>1</sup> which comes mainly from two sources: First, the size of the memories to store the search structure is large. Second, the number of accesses to these large memories is variable [2] and can be large in the worst case. To address these problems, we propose a SRAM-based multi-pipeline architecture for energy-efficient terabit packet classification. Following contributions are made in this paper.

- We optimize the original HyperCuts algorithm by bounding the height of the decision tree.
- To reduce the memory size for classifying each packet, we propose two different schemes to partition the decision tree into multiple disjoint subtrees and map them onto multiple pipelines, while keeping the memory requirement among the pipelines to be balanced.
- A fine-grained node-to-stage mapping scheme is developed to achieve a balanced memory distribution across stages within each pipeline.
- Simulation experiments using real-life and synthetic rule sets show that our architecture can store 10K rules

This work is supported by the United States National Science Foundation under grant No. CCF-0702784.

<sup>1</sup>Although the terms *power* and *energy* are often used interchangeably, this paper considers *energy* as the product of average power consumption and latency [15].

in 0.336 MB memory, sustains 1 Tbps throughput for minimum size (40 bytes) packets, and achieves 2.25-fold reduction in energy dissipation over the baseline pipeline architecture that is not partitioned.

The remainder of the paper is organized as follows. Section II briefly reviews the decision-tree-based packet classification algorithms and the existing work on energy-efficient packet classification engines. Section III gives an overview of our architecture. Section IV presents the set of algorithms including the optimized HyperCuts, the tree partitioning schemes, and the node-to-stage mapping method. Section V evaluates the performance. Section VI concludes the paper.

## II. BACKGROUND AND RELATED WORK

### A. Decision-Tree-based Packet Classification Algorithms

Packet classification algorithms have been extensively studied in the past decade. Comprehensive surveys can be found in [10], [16]. Among these algorithms, decision-tree-based algorithms (e.g. HyperCuts [12]) usually scale well and are suitable for pipelining. Such algorithms take the geometric view of the packet classification problem. Each rule defines a hypercube in a  $D$ -dimensional space where  $D$  is the number of header fields in a rule. Each packet defines a point in this  $D$ -dimensional space. The decision tree construction algorithm employs several heuristics to cut the space recursively into smaller subspaces. Each subspace ends up with fewer rules, which to a point allows a low-cost linear search to find the best matching rule. After the decision tree is built, the algorithm to look up a packet is simple. Based on the value of the packet header, the algorithm follows the cutting sequence to locate the target subspace (i.e. a leaf node in the decision tree), and then performs a linear search on the rules in this subspace.

The representatives of decision-tree-based packet classification algorithms are HiCuts [17] and its enhanced version HyperCuts [12]. HiCuts builds a decision tree using local optimization decisions at each node to choose the next dimension to cut, as well as how many cuts to make in the chosen dimension. HyperCuts, on the other hand, allows cutting on multiple fields per step, resulting in a fatter and shorter decision tree. Figure 1 shows the example of the HiCuts and the HyperCuts decision trees for a set of 2-field rules which can be represented geometrically.

Decision-tree-based algorithms classify a packet by traversing the tree. Such search process can be pipelined well to achieve high throughput of one packet per clock cycle [12], [14]. However, as pointed out in [2], decision-tree-based algorithms require a variable number of memory accesses for classifying a packet. In other words, the height<sup>2</sup> of the decision tree, denoted  $H_T$ , can vary widely for different rule sets. The pipeline depth,<sup>3</sup> denoted  $H_P$ , is determined by the worst-case tree height which can be the total number of bits of the packet header being classified (denoted  $W$ ). For example,

<sup>2</sup>The *height* of a tree node is the maximum directed distance from the tree node to a leaf node. The height of a tree refers to the height of the root.

<sup>3</sup>The *depth* of a pipeline is the number of stages in the pipeline.

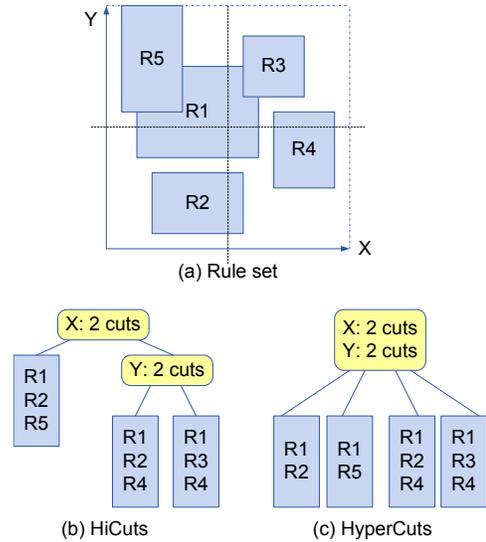


Fig. 1. Example of HiCuts and HyperCuts decision trees. ((a) X and Y axes correspond to the 2 fields of the rules. (b)(c) A rounded rectangle in yellow denotes an internal tree node, and a rectangle in gray a leaf node.)

$W = 32 + 32 + 16 + 16 + 8 = 104$  for 5-field IPv4 packet classification. Such a deep pipeline is neither energy-efficient nor practical for real hardware implementation.

### B. Energy-Efficient Packet Classification Engines

As routers achieve aggregate throughputs of trillions of bits per second, energy dissipation by packet classification engines becomes an increasingly critical concern for backbone router design [10], [18]. Most of existing work on designing energy-efficient packet classification engines focus on reducing the power consumption of TCAM-based solutions.

As discussed in Section I, TCAM is power-hungry. Various schemes have been proposed to partition a TCAM into multiple blocks. Only one or few blocks are searched for a packet so that the power consumption is reduced [2], [19], [20]. The common goal of these schemes is to ensure each block contains equal numbers of rules, so that the power consumption reduction is maximized. We believe that such a partitioning-based idea can be applied for SRAM-based pipeline engines as well. However, unlike TCAM-based solutions where each rule consumes the same amount of memory words [2], the memory requirement of SRAM-based engines is determined by the size of the decision tree. Hence a different and efficient method is needed to partition the decision tree so that each partition contains equal amount of memory.

Little work has been done on SRAM-based energy-efficient packet classification engines. Kennedy et al. [21] simplify the HiCuts and the HyperCuts algorithms by using a large number of cuts at each internal node and storing up to hundreds of rules in each leaf node. These schemes result in a decision tree with a small height. Thus the power consumption, which is proportional to the number of memory accesses needed for classifying a packet, is reduced. The authors later propose an adaptive clocking unit which dynamically changes the clock

frequency to minimize the power consumption while matching the traffic fluctuation [22]. However, the size of the memories accessed by each packet is large. Matching hundreds of rules in each leaf node also results in high latency. The power reduction achieved by the adaptive clocking scheme depends heavily on the characteristics of the traffic traces. The energy dissipation will still be quite high in the worst case.

### III. ARCHITECTURE OVERVIEW

We propose a SRAM-based multi-pipeline architecture for energy-efficient packet classification. As shown in Figure 2, the architecture consists of  $P$  SRAM-based pipelines each of which stores a portion of the rule set. The rule subset stored in a pipeline is disjoint with that in any other pipeline. Thus only one of the  $P$  pipelines is searched for each input packet. All the pipelines have the same depth, which guarantees that the packets are output in the same order as they are input. The memory distribution among the  $P$  pipelines as well as across the stages within each pipeline is balanced so that the reduction in power consumption is maximized.

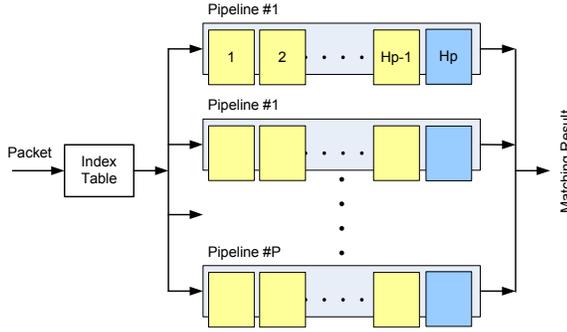


Fig. 2. Block diagram of the multi-pipeline architecture.

Any packet to be classified looks up the index table and retrieves the ID of the pipeline to which the packet is directed. Then the packet goes through the  $H_P$  stages in that pipeline. Within each pipeline, the first  $H_P - 1$  stages store the internal nodes of the decision tree while the last stage stores the rules contained in the leaf nodes. The packet traverses the tree until it reaches the leaf node. A small number of rules contained in that leaf node are matched against the packet in parallel, which produces the classification result to be finally output.

### IV. ALGORITHMS

#### A. Tree Building

As discussed in Section II-A, a decision tree should have a bounded height  $H_T$  so that the tree can be mapped onto a pipeline whose depth  $H_P$  must be  $H_P \geq H_T$ . A simple way to bound the tree height is setting a lower bound (denoted  $Stride_B$ ) for the number of cuts at each internal node. Note that each cut always cuts a value space by half, which is equal to extracting one bit from the input packet header. We define  $stride$  as the number of cutting bits at an internal node. Thus  $Stride_B$  also indicates the minimum stride that must be used at each internal node. Since the number of bits in the packet

header is  $W$  which is 104 for 5-field IPv4 packet classification, the tree height will be  $H_T \leq \frac{W}{Stride_B}$ .

Motivated by the above idea, we integrate  $Stride_B$  into the original HyperCuts construction algorithm. We note that both HiCuts and HyperCuts use a configurable parameter  $spf$  (called the space factor) to control the trade-off between the number of memory accesses and the total memory size. A larger  $spf$  allows more cuts at each internal node so that the tree height can be reduced, at the potential cost of consuming more memory. Unlike the original HyperCuts which has no explicit control on the tree height and applies the same value of  $spf$  to all internal nodes, we consider  $Stride_B$  and set adaptive values of  $spf$  for different internal nodes.

Moreover, though  $Stride_B$  is fixed after configuration, the actual lower bound of stride for different internal nodes can be different. For example, suppose  $Stride_B = 4$ . If an internal node takes a stride of 5 which is  $> 4$ , all the children of this node can have a  $Stride'_B = 4 - (5 - 4) = 3$ , instead of 4. We implement such adaptive  $Stride_B$  by adding a field,  $strideCredit$ , for each internal node. The value of  $strideCredit$  of a node is the actual stride used at the node subtracted by  $Stride_B$ .

Figure 3 shows the algorithm to build a HyperCuts tree with a bounded height.  $n$  denotes a node,  $bucketSize$  the maximum number of rules that can be contained in a leaf node, and  $NB$  the number of cutting bits i.e. the stride.  $\Delta$  is a small increment, which is 0.1 in our implementation.

**Input:** A rule set.

**Input:**  $Stride_B$ .

**Output:** A HyperCuts tree whose height is  $\leq \frac{W}{Stride_B}$ .

- 1: Initialize the root node and push it into  $nodeList$ .
- 2: **while**  $nodeList \neq null$  **do**
- 3:    $n \leftarrow Pop(nodeList)$
- 4:   **if**  $n.numRule < bucketSize$  **then**
- 5:     Push  $n$  into the list of leaf nodes.
- 6:     Continue.
- 7:   **end if**
- 8:    $spf \leftarrow 1.0$
- 9:   **while**  $NB < Stride_B - n.strideCredit$  **do**
- 10:      $NB \leftarrow OptNumBit(n, spf)$
- 11:      $spf \leftarrow spf + \Delta$
- 12:   **end while**
- 13:   **for**  $i \leftarrow 0$  to  $2^{NB} - 1$  **do**
- 14:      $n_i \leftarrow CreateNode(n, i)$
- 15:      $n_i.strideCredit \leftarrow n.strideCredit + NB - Stride_B$
- 16:     Push  $n_i$  into  $nodeList$ .
- 17:   **end for**
- 18: **end while**

Fig. 3. Algorithm: Building the decision tree with a bounded height

#### B. Tree Partitioning

We first define the *size* of a tree as the sum of the memory size of the nodes contained in the tree.

We propose two different schemes to partition the decision tree and map the subtrees onto  $P$  pipelines while balancing the memory requirement among the pipelines. One scheme uses a simple method to partition the tree and packs the subtrees into  $P$  pipelines. The other scheme takes one pipeline at a time, splits subtrees out of the tree and map them onto the pipeline.

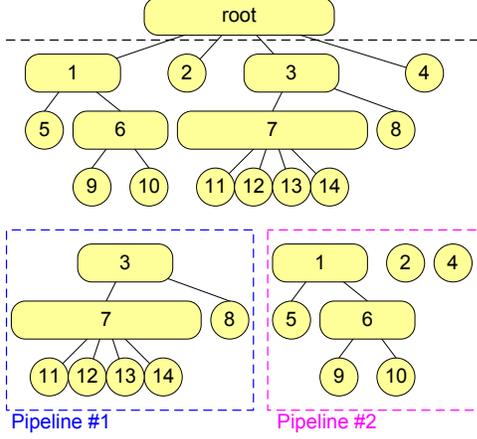


Fig. 4. Partition the tree and pack the subtrees into  $P = 2$  pipelines.

1) *Packing-based*: As shown in Figure 4, we simply partition the original decision tree by assigning the children of the root to be the roots of the subtrees. We need to map these subtrees of various size onto  $P$  pipelines to achieve balanced memory distribution among the pipelines. Such a problem is NP-complete, which can be shown by a reduction from the partitioning problem [23]. We propose a polynomial-time approximation algorithm shown in Figure 5. The complexity of this algorithm is  $O(KP)$ , where  $K$  denotes the number of subtrees and  $P$  the number of pipelines. According to [23], in the worst-case, the resulting largest pipeline may require 1.5 times the memory size as the optimal mapping.

**Input:**  $K$  subtrees:  $T_i, i = 1, 2, \dots, K$ .

**Input:**  $P$  empty pipelines.

**Output:**  $P$  pipelines, each of which contains a set of subtrees  $S_i, i = 1, 2, \dots, P$ .

- 1: Set  $S_i = \phi$  for all pipelines,  $i = 1, 2, \dots, P$ .
- 2: Sort  $\{T_i\}$  in the decreasing order of  $size(T_i), i = 1, 2, \dots, K$ .
- 3: Renumber the subtrees so that  $size(T_1) \geq size(T_2) \geq \dots \geq size(T_K)$ .
- 4: **for**  $i = 1$  to  $K$  **do**
- 5: Find  $S_m$  so that  $size(S_m) = \min_{j=1,2,\dots,P} size(S_j)$ .
- 6: Assign  $T_i$  to the  $m$  th pipeline:  $S_m \leftarrow S_m \cup T_i$ .
- 7: **end for**

Fig. 5. Algorithm: Packing-based subtree-to-pipeline mapping

The packing-based scheme cannot achieve perfectly balanced memory distribution among multiple pipelines. On the other hand, due to its simple partitioning, the index table in the architecture can be implemented as a single SRAM. Each

input packet uses the cutting bits at the root to directly access the index table and retrieves the ID of the destination pipeline.

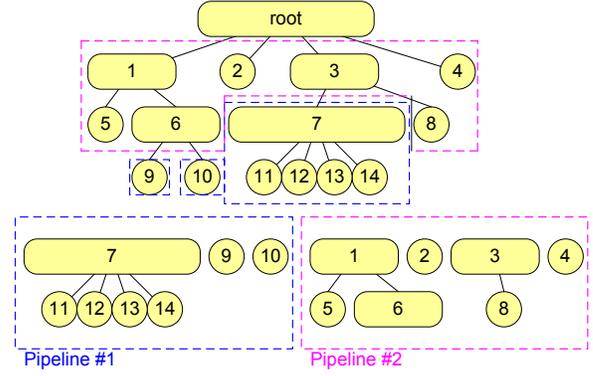


Fig. 6. Split subtrees out of the tree and map onto  $P = 2$  pipelines.

2) *Splitting-based*: To achieve perfectly balanced memory distribution among  $P$  pipelines, we consider one pipeline at a time, carefully split subtrees out of the decision tree, and map them onto the pipeline, as shown in Figure 6. The complete algorithm is shown in Figure 7, where  $r$  denotes a root node and  $T(r)$  the tree whose root is  $r$ . We first calculate the memory capacity of the pipeline being mapped onto. According to the gap between the capacity and the current memory size of the pipeline, we split out of the tree the subtree whose size is most fit into the gap. The split procedure is implemented as a recursive function shown in Figure 8. The complexities of the algorithms in Figures 7 and 8 are both  $O(N)$ , where  $N$  denotes the number of tree nodes.

**Input:** The root of the decision tree:  $r$ .

**Input:**  $P$  empty pipelines.

**Output:**  $P$  pipelines, each of which contains a set of subtrees  $S_i, i = 1, 2, \dots, P$ .

- 1: Set  $S_i = \phi$  for all pipelines,  $i = 1, 2, \dots, P$ .
- 2: **for**  $i = 1$  to  $P$  **do**
- 3:  $Capacity \leftarrow \frac{size(T(r))}{P+1-i}$
- 4: **while**  $size(S_i) < Capacity$  **do**
- 5:  $r_s \leftarrow Split(r, Capacity - size(S_i))$
- 6: Assign  $T(r_s)$  to the current pipeline:  $S_i \leftarrow S_i \cup T(r_s)$ .
- 7: **end while**
- 8: **end for**

Fig. 7. Algorithm: Splitting-based subtree-to-pipeline mapping

The splitting-based scheme can guarantee the perfectly balanced memory distribution among multiple pipelines. However, since the roots of the resulting subtrees have different sequence of cutting bits from the root of the original tree, a TCAM is needed as part of the index table to index these subtrees. To reduce the number of TCAM entries, we use a SRAM to index the subtrees whose roots are at the first level, i.e. the children of the root of the original decision tree.

**Input:** The root of the tree:  $r$ .

**Input:** Memory size requirement:  $m$ .

**Output:** The root of the split subtree:  $r_s$ .

- 1: **if**  $size(T(r)) \leq m$  **then**
- 2:   Return  $r$ .
- 3: **end if**
- 4: Among the children of  $r$ , find the child  $chd_M$  so that  $size(T(chd_M)) = \max_{chd \in r.children} size(T(chd))$ .
- 5: **if**  $(r_s \leftarrow Split(chd_M, m)) == chd_M$  **then**
- 6:    $r.chd_M \leftarrow Null$
- 7: **end if**
- 8: Return  $r_s$ .

Fig. 8.  $Split(r, m)$ : Splitting a subtree out of the tree

### C. Node-to-Stage Mapping

We now have a set of subtrees for each pipeline. Within each pipeline, the size of the memory in each stage must be determined before hardware implementation. The static mapping scheme is to simply map each level of the decision tree onto a separate stage. However, the memory distribution across the stages can vary widely. Allocating memory with the maximum size for each stage also results in large memory wastage [13]. Our task is to map the decision tree onto a linear pipeline to achieve balanced memory distribution over stages, while sustaining a throughput of one packet per clock cycle. The challenge comes from the various number of words needed for different tree nodes.

The problem is a variant of bin packing problems [23], and can be proved to be NP-complete. We use a fine-grained mapping scheme similar to [14]<sup>4</sup> to allow the nodes on the same level of the tree to be mapped onto different pipeline stages. This provides more flexibility to map the tree nodes, and helps achieve a balanced memory distribution across the stages in a pipeline. Such fine-grained node-to-stage mapping can be enabled in hardware using a simple method. Each node stored in the local memory of a pipeline stage has one extra field: the distance to the pipeline stage where the child node is stored. When a packet passes through the pipeline, the distance value is decremented by 1 every time it goes through a stage. When the distance value becomes 0, the child node's address is used to access the memory in that stage.

## V. PERFORMANCE EVALUATION

To evaluate the effectiveness of our schemes, we conducted simulation experiments using 5 real-life and synthetic rule sets from [24]. The synthetic rule sets were generated using ClassBench [25], with parameter files extracted from real-life rules. The size of the rule sets varies from hundreds to of thousands of rules, as shown in Table I.

TABLE I  
RULE SETS FOR 5-FIELD PACKET CLASSIFICATION

Rule sets	<i>acl1</i>	<i>acl_100</i>	<i>acl_1K</i>	<i>acl_5K</i>	<i>acl_10K</i>
# of rules	752	98	916	4415	9603

<sup>4</sup>The node-to-stage mapping scheme in [14] is for an optimized Hypercuts tree and tailored to maximize the FPGA memory utilization.

We set  $Stride_B = 5$  to build the optimized HyperCuts trees for these rule sets. The pipeline depth needed was thus bounded by  $\frac{W}{Stride_B} = \frac{104}{5} = 21$ . We applied the two proposed partitioning schemes, respectively, for mapping the trees onto  $P$  pipelines. Their mapping results are shown in Figures 9 and 10, respectively. In these figures, the X axis shows the IDs of pipelines and the Y axis shows the ratio of the memory size needed for each pipeline to the total memory requirement. By comparing Figures 9 and 10, it is clear that the splitting-based scheme achieved much more balanced memory distribution than the packing-based scheme.

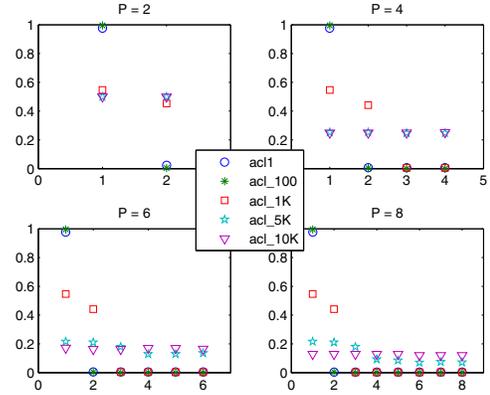


Fig. 9. Memory distribution among  $P$  pipelines after using the packing-based scheme. ( $P = 2, 4, 6, 8$ )

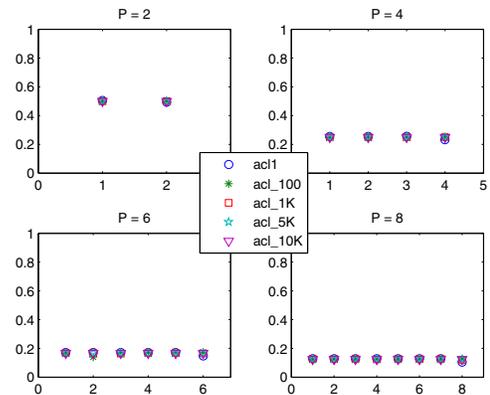


Fig. 10. Memory distribution among  $P$  pipelines after using the splitting-based scheme. ( $P = 2, 4, 6, 8$ )

After using the splitting-based scheme for mapping the trees onto a 8-pipeline architecture, we applied the fine-grained node-to-stage mapping method within each pipeline. The decision tree construction results showed that the tree height for all rule sets was no larger than 13. Hence we set the pipeline depth to be 14 which was larger than the height of all subtrees. Figure 11 shows the mapping results, where the X axis indicates the IDs of stages and the Y axis the number of words. Within each pipeline, except the first few stages, all the stages had almost equal amount of memory.

According to the mapping results for *acl\_10K*, each stage

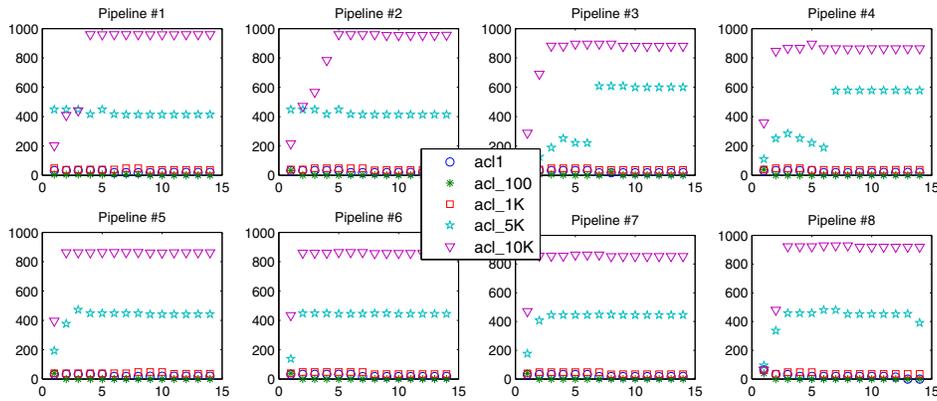


Fig. 11. Memory distribution among 14 stages in each pipeline. ( $P = 8$ )

needs 3 Kbytes SRAM. The total memory requirement is thus 0.336 MB. Using CACTI 5.3 [26], a 3 KB SRAM with 45 nm CMOS technology needs 0.31 ns to access and dissipates 0.0067 nJ energy. The architecture also needs a TCAM with 269 entries as part of the index table. The TCAM was so small that its effect on the overall performance can be omitted. Hence the architecture can achieve a clock rate of 3.26 GHz, resulting in a throughput of 1 Tbps for minimum size (40 bytes) packets. The energy dissipation for classifying a packet is  $0.0067 \times 14 = 0.0938$  nJ. In contrast, the baseline pipeline architecture that is not partitioned needs 24 KB SRAM in each stage, resulting in 800 Gbps throughput and 0.211 nJ energy dissipation for classifying a packet. In other words, our partitioning schemes resulted in 2.25-fold reduction in energy dissipation over the baseline pipeline architecture.

## VI. CONCLUSION

This paper proposed a SRAM-based multi-pipeline architecture for terabit packet classification. We first addressed the challenge in pipelining decision-tree-based algorithms by building a height-bounded HyperCuts tree. Two different schemes were proposed to partition the tree into multiple disjoint subtrees which were then mapped onto multiple pipelines, while balancing the memory requirement among these pipelines. A fine-grained node-to-stage mapping scheme was used to achieve balanced memory distribution across stages within each pipeline. Simulation experiments using real-life and synthetic rule sets show that our architecture with 8 pipelines can store 10K rules in 0.336 MB memory, sustains 1 Tbps throughput for minimum size (40 bytes) packets, and achieves 2.25-fold reduction in energy dissipation over the baseline pipeline that is not partitioned. We are implementing the architecture in FPGA and the preliminary design shows to be practical and consumes small amount of on-chip resources.

## REFERENCES

- [1] Verizon moving to 100 Gbps network in '09, "http://www.networkworld.com/news/2008/031008-verizon-100gbps-network.html."
- [2] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-based distributed parallel packet classification with range encoding," *IEEE Trans. Computers*, vol. 55, no. 8, pp. 947–961, 2006.
- [3] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. SIGCOMM*, 2005, pp. 193–204.
- [4] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *Proc. FPGA*, 2005, pp. 238–245.
- [5] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast packet classification using bloom filters," in *Proc. ANCS*, 2006.
- [6] I. Papaefstathiou and V. Papaefstathiou, "Memory-efficient 5D packet classification at 40 Gbps," in *Proc. INFOCOM*, 2007, pp. 1370–1378.
- [7] A. Nikitakis and I. Papaefstathiou, "A memory-efficient FPGA-based classification engine," in *Proc. FCCM*, 2008, pp. 53–62.
- [8] I. Sourdis, "Designs & algorithms for packet and content inspection," Ph.D. dissertation, Delft University of Technology, 2007. [Online]. Available: [http://ce.et.tudelft.nl/publicationfiles/1464\\_564\\_sourdis\\_phdthesis.pdf](http://ce.et.tudelft.nl/publicationfiles/1464_564_sourdis_phdthesis.pdf)
- [9] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" in *Proc. INFOCOM*, 2003.
- [10] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
- [11] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," in *Proc. INFOCOM*, 2008, pp. 1786–1794.
- [12] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. SIGCOMM*, 2003.
- [13] F. Baboescu, D. M. Tullsen, G. Rosu, and S. Singh, "A tree based router search engine architecture with single port memories," in *Proc. ISCA*, 2005, pp. 123–133.
- [14] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proc. FPGA*, 2009, pp. 219–228.
- [15] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy-efficient signal processing using FPGAs," in *Proc. FPGA*, 2003, pp. 225–234.
- [16] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.
- [17] —, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.
- [18] M. Gupta and S. Singh, "Greening of the Internet," in *Proc. SIGCOMM '03*, 2003, pp. 19–26.
- [19] H. Lu and M. Pan, "Partition filter set for power-efficient packet classification," in *Proc. Globecom*, 2006.
- [20] M. Nourani and M. Faezipour, "A single-cycle multi-match packet classification engine using TCAMs," in *Proc. High-Performance Interconnects*, 2006, pp. 73–80.
- [21] A. Kennedy, X. Wang, and B. Liu, "Energy efficient packet classification hardware accelerator," in *IPDPS*, 2008, pp. 1–8.
- [22] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *Proc. ANCS*, 2008, pp. 131–140.
- [23] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.
- [24] Packet Classification Filter Sets, "http://www.arl.wustl.edu/~hs1/pclasseval.html."
- [25] D. E. Taylor and J. S. Turner, "Classbench: a packet classification benchmark," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, 2007.
- [26] CACTI 5.3, "http://quid.hpl.hp.com:9081/cacti/."