PROBABILISTIC SUFFIX TREE BASED PACKET CLASSIFICATION ON FPGA

S.B.Dhanya*

Mr V.Gopi**

Abstract—

The general packet classification problem has received a great deal of attention over the last decade. The ability to classify packets into flows based on their packet headers is important for QoS, security, virtual private networks (VPN) and packet filtering applications. Multi-field packet classification has evolved from traditional fixed 5-tuple matching to flexible matching with arbitrary combination of numerous packet header fields. In this project, we introduce a method for producing more efficient probabilistic suffix tree (PST) classifiers. Our empirical model is general enough to model many disparate problems. We considered the next-generation packet classification problems where more than 5-tuple packet header fields would be classified. When matching multiple fields simultaneously, it is difficult to achieve both high classification rate and modest storage in the worst case. Our classifier can be considered among the most algorithms which have high throughput and efficiency.

Keywords— Probabilistic suffix tree (PST), field-programmable gate array (FPGA), packet classification, pipeline, SRAM.

* PG Student, M.E (EST), PSN College of Engineering and Technology,Tirunelveli.

** M.E, (Ph.D) Professor, PSN College of Engineering and Technology, Tirunelveli

I. INTRODUCTION

The development of the Internet demands next-generation routers to support a variety of network applications, such as firewall processing, Quality of Service (QoS) differentiation, virtual private networks, policy routing, traffic billing, and other value added services. In order to provide these services, the router needs to classify the packets into different categories based on a set of predefined rules, which specify the value ranges of the multiple fields in the packet header. Such a function is called multi-field packet classification. In traditional network applications, packet classification problems usually consider the fixed 5-tuple fields: 32-bit source/destination IP addresses, 16-bit source/destination port numbers, and 8-bit transport layer protocol. In OpenFlow, up to 12-tuple header fields are considered [3].

Due to the rapid growth of the rule set size, as well as the link rate, multi-field packet classification has become one of the fundamental challenges to design high speed routers. For example, the current link rate has been pushed beyond 40 Gbps, which requires processing a packet every 8 ns in the worst case. Such throughput is impossible using existing software-based solutions. On the other hand, mapping decision-tree-based packet classification algorithms onto SRAM-based pipeline architecture appear to be a promising alternative. Thus in conventional method, the decision tree is used for packet classification.

Here, we develop a decision forest, to partition a given set of 12-tuple rules into multiple subsets so that each subset uses a small number of header fields to build a decision tree of certain depth. To tackle the problem of rule duplication when building the decision tree, we use two optimization techniques, called rule overlap reduction and precise range cutting. Then the tree is mapped onto the pipeline architecture, by a node-to-stage mapping scheme which allows imposing the bounds on the memory size as well as the number of nodes in each stage. As a result, the memory utilization of the architecture is maximized. The memory allocation scheme also enables using external SRAMs to handle even larger rule sets. We use the dual-port high-speed Block RAMs provided in state-of-the-art FPGAs to achieve a high throughput of two packets per clock cycle (PPC).

Next-generation packet classification on more header fields poses an even bigger challenge. Most of the existing work in high-throughput packet classification is based on ternary content addressable memory (TCAM) [5]–[7] or a variety of hashing schemes such as Bloom Filters [8]–[10]. However, TCAMs are not scalable with respect to clock rate, power consumption, or circuit area, compared to

SRAMs [11]. Most of TCAM-based solutions also suffer from range expansion when converting ranges into prefixes [6], [7]. Hashing-based solutions such as Bloom Filters have become popular due to their time performance and high memory efficiency [12]. However, hashing cannot provide deterministic performance due to potential collision and is inefficient in handling wildcard or prefix matching [13]. A secondary module is always needed to resolve false positives inherent in Bloom Filters, which may be slow and can limit the overall performance [14].

In this paper, we propose a Probabilistic Suffix Tree (PST)for packet classification. The Probability Suffix Tree is a data structure that, allows many problems on strings (sequence of characters) to be solved quickly. For the problems on the strings to be solved, it does not require the exact match between the header values of the packet to be matched and the reference packet. In PST, the suffix values of the reference packet and the packet to be classified are compared. If they matches, the packet gets into next stage and if not the packet is rejected. By using PST, it's very easy to classify complex strings. Advantages in using the Probability Suffix Tree are its high detection accuracy and high throughput.

The rest of this paper contained as follows: In Section 2, we review the related work. Section 3 presents our approach for generation of probabilistic suffix tree (PST) for efficient packet classification. Section 4 presents the pipeline architecture. Section 5 presents the implementation details. Finally, Section 6 concludes the paper.

II. RELATED WORK

A. FPGA designs for 5-tuple packet classification

Although traditional 5-tuple packet classification is considered a saturated area of research, little work has been done on FPGAs. Most of existing FPGA implementations of packet classification engines are based on decomposition-based packet classification algorithms, such as BV [15] and DCFL [11].

Lakshman *et al.* [15] propose the Parallel Bit Vector (BV) algorithm, which is a decomposition-based algorithm targeting hardware implementation. It performs the parallel lookups on each individual field first. The lookup on each field returns a bit vector with each bit

representing a rule. A bit is set if the corresponding rule is matched on this field; a bit is reset if the corresponding rule is not matched on this field. The result of the bitwise AND operation on these bit vectors indicates the set of rules that matches a given packet. The BV algorithm can provide a high throughput at the cost of low memory efficiency. By combining TCAMs and the BV algorithm, Song *et al.* [7] present an architecture called BV-TCAM for multi-match packet classification. A TCAM performs prefix or exact match, while a multi-bit trie implemented in Tree Bitmap [23] is used for source or destination port lookup.

Taylor *et al.* [11] introduce Distributed Crossproducting of Field Labels (DCFL), which is also a decomposition-based algorithm leveraging several observations of the structure of real filter sets. They decompose the multi-field searching problem and use independent search engines, which can operate in parallel to find the matching conditions for each filter field. Instead of using bit vectors, DCFL uses a network of efficient aggregation nodes, by employing Bloom Filters and encoding intermediate search results. As a result, the algorithm avoids the exponential increase in the time or space incurred when performing this operation in a single step. The authors predict that an optimized implementation of DCFL can provide over 100 million packets per second (MPPS) and store over 200K rules in the current generation of FPGA or application-specific integrated circuit (ASIC) without the need of external memories. However, their prediction is based on the maximum clock frequency of FPGA devices and a logic intensive approach using Bloom Filters. This approach may not be optimal for FPGA implementation due to long logic paths and large routing delays. Furthermore, the estimated number of rules is based only on the assumption of statistics similar to those of the currently available rule sets.

B. Hardware Accelerators for open flow

While OpenFlow switch technology is evolving, little attention has been paid on improving the performance of 12-tuple packet classification. Luo *et al.* [15] propose using network processors to accelerate the OpenFlow switching. Similar to the software implementation of the OpenFlow switching, hashing is adopted for simple rules while linear search is performed on the complex rules. When the number of complex rules becomes large, using linear search leads to low throughput. Moreover, hashing-based solutions may suffer from hash collision and cannot produce

deterministic throughput. Naous *et al.* [12] implement the OpenFlow switch on NetFPGA which is a Xilinx Virtex-2 Pro 50 FPGA board tailored for network applications. They use hashing for simple rules and a small TCAM implemented on FPGA for complex rules. Due to the high cost to implement TCAM on FPGA, their design can support no more than few tens of complex rules. Though it is possible to use external TCAMs for large rule tables, high power consumption of TCAMs remains a big challenge. To the best of our knowledge, none of existing schemes for OpenFlow-like packet classification can sustain throughput above 10 Gbps in the worst case where packets are of minimum size i.e., 40 bytes.

C. Decision tree

Decision-tree-based algorithms (such as HyperCuts) usually scale well and are suitable for rule sets where the rules have little overlap with each other. But they suffer from rule duplication which can result in memory explosion in the worst case. Hence an intuitive idea is to partition a table of complex rules into different subsets. The rules within the same subset specify nearly the same set of header fields. For each rule subset, we build the decision tree based on the specified fields used by the rules within this subset.

After rule set partitioning, the rule duplication due to wildcard fields will be reduced. However, the Hi- Cuts/HyperCuts algorithm may still suffer from rule duplication due to its own inefficiency, such as rule replication. The following two optimization techniques, called rule overlap reduction and precise range cutting are used to overcome inefficiency due to rule replication. In Rule overlap reduction, we store the rules which will be replicated into child nodes, in a list attached to each internal node. These rule lists are called internal rule lists. In Precise range cutting, we seek the cutting points which result in the minimum number of rule duplication, instead of deciding the number of cuts for this field.

Then, using the partitioned rule set, a decision tree is built. For each partitioned rule set a single decision tree is formed and thus we get a decision forest. The decisions trees thus formed are mapped into a pipeline architecture each. Each level of decision tree forms, one stage in the pipeline structure.

III. PROBABILISTIC SUFFIX TREE GENERATION

Our goal is to generate the probabilistic suffix tree (PST) for efficient packet classification. Initially, the incoming packet header field is converted into hexadecimal value by a logic and then using the probability of occurrences of the field values, a suffix tree is formed. This is known to as Probability suffix tree. This tree is matched with the probability suffix tree formed for the database entries. Initially, the header fields are classified into number of rules based on different values of the header fields as shown in Table 1 below.

The Table 1 is constructed only based on 5-tuples of the header field. Now, the tables are constructed based on 12-tuples for efficient classification. In Table 1, SA/DA are 32-bit fields, SP/DP are 16-bit fields, ingress port field is variable field, VLAN ID, VLAN priority, Type of service are some of the fields used in 12-tuple packet classification. These rules may be divided into several rule sets based on the common value ranges of the header fields.

TABLE 1

EXAMPLE 5-TUPLE RULE SET

Rule	SA	DA	SP	DP	Protocol	Priority	Action
R1	*	*	2 – 9	6 – 11	*	1	act0
R2	1*	0*	3 – 8	1 – 4	10	2	act0
R3	0*	0110*	9 – 12	10 – 13	11	3	act1
R4	0*	11*	11 – 14	4 – 8	*	4	act2
R5	011*	11*	1 – 4	9 – 15	10	5	act2
R6	011*	11*	1 – 4	4 – 15	10	5	act1
R7	110*	00*	0 – 15	5 – 6	11	6	act3
R8	110*	0110*	0 – 15	5 – 6	*	6	act0
R9	111*	0110*	0 – 15	7 – 9	11	7	act2
R10	111*	00*	0 – 15	4 – 9	*	7	act1

The binary ranges of the header fields are converted into corresponding hexadecimal values. Then the hexadecimal sequence is used for the construction of probabilistic suffix tree.

The hexadecimal sequence is initially taken for PST construction. For example, let the sequence be ‘abcabc’. The corresponding PST is as shown in Fig 1. The sequence is taken and probability of occurrences of the various pairs of hexa decimal values is considered for the suffix of the sequence.

Based on the various combinations of pairs a tree is constructed and its probability value is specified in it.

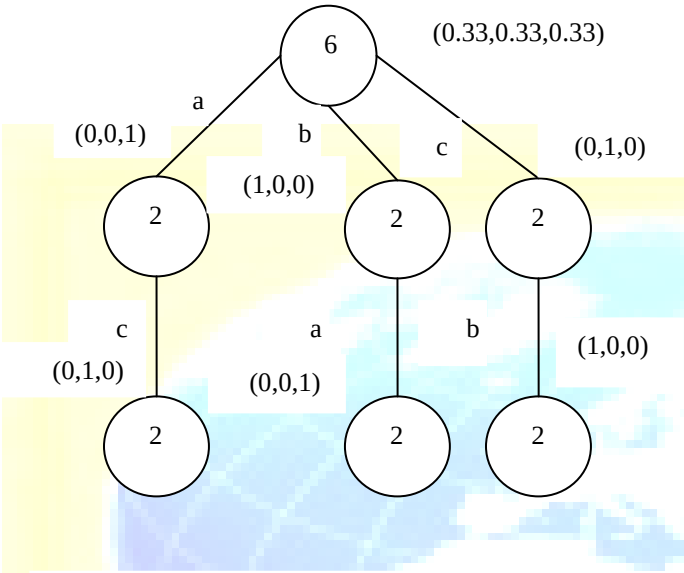

Fig 1: PST for the sequence 'abcabc'

IV. PIPELINE ARCHITECTURE

To achieve line-rate throughput, we map the PST a parallel multi-pipeline architecture with linear pipelines, as shown in Fig. 2. Each pipeline is used for traversing a decision tree as well as matching the rule lists attached to the leaf nodes of that tree. The pipeline stages for tree traversal are called the tree stages while those for rule list matching are called the rule stages. Each tree stage includes a memory block storing the tree nodes and the cutting logic which generates the memory access address based on the input packet header values. At the end of tree traversal, the index of the corresponding leaf node is retrieved to access the rule stages. Since a leaf node contains a list of listSize rules, we need listSize rule stages for matching these rules. All the leaf nodes of a tree have their rule lists mapped onto these listSize rule stages. Each rule stage includes a memory block storing the full content of rules and the matching logic which performs parallel matching on all header fields.

The PST thus built is then mapped into the pipeline architecture. Here the PST in rule stages are compared with the PST constructed out of the current packet header field in the tree stages. The pipeline architecture is shown in Fig 2. If more than one tree in the rule stage matches with the current packets PST tree, the tree with highest probability value is considered for the exact match for the current packet header.

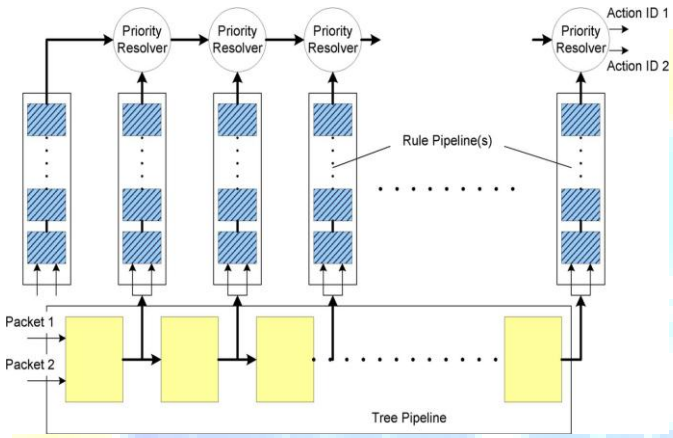

Fig 2: Block diagram of 2-D linear pipeline architecture

Each incoming packet goes through all the pipelines in parallel. A different subset of header fields of the packet may be used to traverse the trees in different pipelines. Each pipeline outputs the rule ID or its corresponding action. The priority resolver picks the result with the highest priority among the P outputs from the pipelines.

V. IMPLEMENTATION RESULTS

To implement the PST for 1K 12-tuple rules in FPGA, we examined the performance results of each tree. our mapping scheme outperformed the static mapping scheme with respect to both memory and node distribution. We mapped the above probabilistic suffix tree onto the pipeline architecture. Since Block RAMs were not used efficiently for blocks of less than 1K entries, we merged the rule lists of the first two pipelines and used distributed memory for the remaining rule lists.

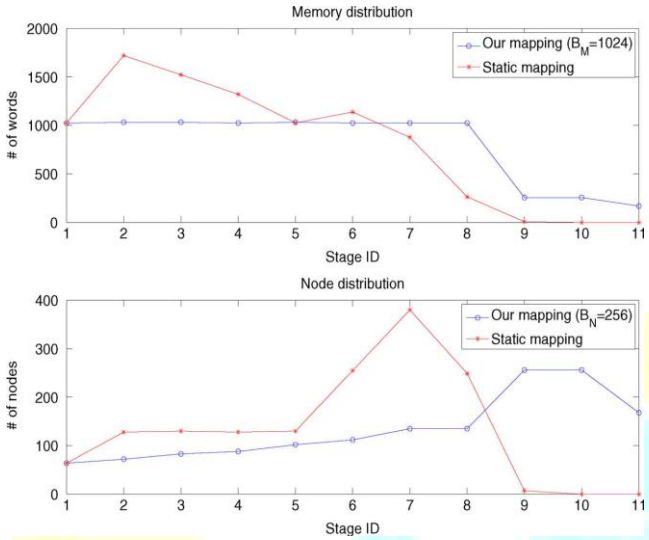

Fig 3: Distribution over tree pipeline stages

VI. CONCLUSION

This paper presented a novel probability suffix-tree-based linear multi-pipeline architecture on FPGAs for wire-speed multi-field packet classification. We considered the next-generation packet classification problems where more than 5-tuple packet header fields would be classified. Several optimization techniques were proposed to reduce the memory requirement of the state-of-the-art decision-tree-based packet classification algorithm, so that 10K 5-tuple rules or 1K 12-tuple rules could fit in the on-chip memory of a single FPGA. Our architecture provided the reconfiguration due to the linear memory-based architecture. Extensive simulation and FPGA implementation results demonstrate the effectiveness of our solution. The FPGA design supports 10K 5-tuple rules or 1K OpenFlow-like complex rules and sustains over 40 Gbps throughput for minimum size (40 bytes) packets. Our future work includes porting our design into real systems and evaluating its performance under real-life scenarios such as dynamic rule updates.

REFERENCES

[1] M. Casado, T. Koponen, D.Moon, and S. Shenker, "Rethinking packet forwarding hardware," in *Proc. HotNets—VII*, 2008, pp. 1–6.

[2] N. McKeown, T. Anderson,H.Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[3] OpenFlow Foundation, "OpenFlow Switch Specification, Version 1.0.0," 2009. [Online]. Available: http://www.openflowswitch.org/
documents/openflow-spec-v1.0.0.pdf.

[4] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.

[5] F. Yu, R. H. Katz, and T. V. Lakshman, "Efficient multimatch packet classification and lookup with TCAM," *IEEE Micro*, vol. 25, no. 1, pp. 50–59, Jan. 2005.

[6] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. SIGCOMM*, 2005, pp. 193–204.

[7] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," in *Proc. FPGA*, 2005, pp. 238–245.

[8] S. Dharmapurikar, H. Song, J. S. Turner, and J. W. Lockwood, "Fast packet classification using bloom filters," in *Proc. ANCS*, 2006, pp. 61–70.

[9] I. Papaefstathiou and V. Papaefstathiou, "Memory-efficient 5D packet classification at 40 Gbps," in *Proc. INFOCOM*, 2007, pp. 1370–1378.

[10] A. Nikitakis and I. Papaefstathiou, "A memory-efficient FPGA-based classification engine," in *Proc. FCCM*, 2008, pp. 53–62.

[11] W. Jiang and V. K. Prasanna, "Sequence-preserving parallel IP lookup using multiple SRAM-based pipelines," *J. Parallel Distrib. Comput.*, vol. 69, no. 9, pp. 778–789, 2009.

[12] H. Yu and R. Mahapatra, "A power- and throughput-efficient packet classifier with n bloom filters," *IEEE Trans. Comput.*, vol. 60, no. 8, pp. 1182–1193, Aug. 2011.

[13] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proc. FPGA*, 2009, pp. 219–228.

[14] I. Sourdis, "Designs & algorithms for packet and content inspection" Ph.D. dissertation, Comput. Eng. Div., Delft Univ. Technol., Delft, The Netherlands, 2007. [Online]. Available: http://ce.et.tudelft.nl/publicationfiles/ 1464_564 sourdis phdthesis.pdf

[15] G. S. Jedhe, A. Ramamoorthy, and K. Varghese, "A scalable high throughput firewall in FPGA," in *Proc. FCCM*, 2008, pp. 43–52.