International Journal of Wireless Communications and Mobile Computing
2015; 3(3): 27-32
Published online May 8, 2015 (http://www.sciencepublishinggroup.com/j/wcmc)
doi: 10.11648/j.wcmc.20150303.11
ISSN: 2330-1007 (Print); ISSN: 2330-1015 (Online)

FPGA Based Packet Classification Using Multi-Pipeline Architecture

R. Sathesh Raaj, J. Kumarnath

Department of Electronics and Communication Engineering, PSNA College of Engineering and Technology, Dindigul, India

Email address:
Sathesh2311psna@gmail.com (R. S. Raaj), jkumarnath@gmail.com (J. Kumarnath)

To cite this article:
R. Sathesh Raaj, J. Kumarnath. FPGA Based Packet Classification Using Multi-Pipeline Architecture. *International Journal of Wireless Communications and Mobile Computing*. Vol. 3, No. 3, 2015, pp. 27-32. doi: 10.11648/j.wcmc.20150303.11

Abstract: This paper proposes a decision-tree-based linear multi-pipeline architecture on FPGA's for packet sorting. We reflect on the next-generation packet classification problems where more than 5-tuple packet header fields has been classified. From traditional fixed 5-tuple matching, Multi-field packet classification has been evolved for flexible matching with arbitrary combination of numerous packet header fields. The recently proposed Open Flow switching requires classifying each packet using up to 12-tuple packet header fields. It become a great task to develop scalable solutions for next-generation packet classification that support larger rule sets, additional packet header fields and higher throughput. This paper proposes a 2-D multi-pipeline decision-tree-based architecture for next-generation packet classification which exploits the abundant parallelism and other desirable features such as current field-programmable gate arrays (FPGAs),. We propose several optimization techniques for the state-of-the-art decision-tree-based algorithm by examine the various traditional 5-tuple packet classification methods. By using set of 12-tuple rules, the framework has been developed to partition the rule set into multiple subsets each of which is built into an optimized decision tree. To maximize the memory utilization. a tree-to-pipeline mapping scheme is carefully designed while underneath high throughput. Our proposed architecture can store up to 1K synthetic 12-tuple rules or 10K real-life 5-tuple rules in on-chip memory of a single up to date FPGA, and maintain 80 or 40 Gbps throughput for least packets of size (40 bytes) respectively. To utilize the memory properly and to sustaining high throughput, a mapping scheme based on tree-to-pipeline is designed carefully. This paper deal with the profuse parallelism and other preferred features provided by present field-programmable gate arrays and propose a 2-D multi-pipeline decision tree based architecture for next-generation packet sorting. The Verilog Hardware description languages (VHDL) are used to design the proposed architecture and synthesized using Xilinx Software.

Keywords: Field Programmable Gate Array (FPGA), Multi-Pipeline Architecture, Multi-Field Packet Classification, Open Flow Switching, 2-D Multi-Pipeline Decision-Tree-Based Architecture, 12-Tuple Rules, 5-Tuple Rules, Verilog Hardware Description Languages (VHDL)

1. Introduction

The development of the next-generation internet demand routers to support a several value added services such as firewall processing ,network functionalities, quality of service (QoS) differentiation, traffic billing, virtual private networks, policy routing, and other value added services. To proffer these services, the router needs to organize the packets into diverse categories based on a set of rules which are predefined, which specify the value ranges of the several fields in the packet header. Such kind of function is called multi-field packet classification. In previous network applications, problems based on packet classification frequently consider the fixed 5-tuple fields: 32 -bit source/destination IP addresses, 8-bit trans-port layer protocol and 16-bit source/destination port numbers. In recent times network virtualization emerges as an needed features for next-generation enterprise, cloud computing networks and data center. This entails the underlying data plane be flexible and offer clean interface for control plane. Such efforts can be seen in Open Flow switch which handle explicitly the network laws by a rule set with rich definition as the hardware- software interface [2]. In Open Flow, up to 12-tuple header fields are considered such as Open-Flow-like

packet classification and the *next-generation packet classification* problems. To design a high speed router, we need rule set size and multi -field packet classification. It has become one of the fundamental challenges. For example, the present link rate has been pushed above the OC-768 rate that is 40 Gbps, which requires meting out a packet every 8 ns in the most awful case (where the packets are of minimum size that is 40 bytes). Such high throughput is not possible using present software-based solutions [4]. Forth coming generation packet classification on more header fields poses an even higher challenge. Most of the present work in done by variety of hashing schemes such as Blooms filters and ternary content addressable memory .However, TCAMs are not scalable with respect to clock rate when compared to SRAMs, power consumption, or circuit area Most of TCAM-based solutions also suffer from range expansion when converting ranges into prefixes .Hashing-based solutions like Bloom Filters have become popular due to their $O(1)$ time performance and high memory proficiency. It is to be noted that, hashing cannot provide deterministic performance due to potential collision and is incompetent in handling wildcard or prefix matching [13].In Blooms filter, the secondary module is always needed to resolve false positives inherent, which may be slow and can limit the overall performance [14]. As an alternative, our work focuses on optimizing and mapping state-of-the- art packet classification algorithms onto SRAM-based parallel architectures such as field-programmable gate array (FPGA). FPGA technology has become an attractive option for implementing real-time network processing engines [7], [10], [15] due to its ability to reconfigure and to offer abundant parallelism. State- of-the-art SRAM-based FPGA devices such as Xilinx Virtex -6 [16] and Altera Stratix-IV [17] provide high clock rate, low power dissipation and large amounts of on-chip dual- port memory with configurable word width. We make use of these desirable features in current FPGAs for scheming high-performance next-generation packet classification engines.

Table 1. *Shows the Header field supported in current open flow.*

Header field	Notation	# of bits
Ingress port		Variable
Source Ethernet addresses	Eth src	48
Destination Ethernet address	Eth dst	48
Ethernet type	Eth type	16
VLAN ID		12
VLAN Priority		3
Source IP address	SA	32
Destination IP address	DA	32
IP Protocol	Prtl	8
IP Type of Service	ToS	6
Source port	SP	16
Destination port	DP	16

Ingrained node-to-stage mapping scheme is used for mapping the tree structure onto the pipeline architecture, which allows imposing the bounds on the memory size as well as the number of nodes in each stage. As a result, the memory consumption of the architecture is increased. Using external SRAM the memory allocation scheme is enabled to handle even larger rule sets. We make use of the dual-port high-speed Block RAMs provided in modern FPGAs to achieve a high throughput of two packets per clock cycle (PPC). Service interruption become possible due to memory-based linear architecture. All the routing path are located to avoid large routing delay hence high clock frequency is obtained,

2. Related Work

Traditional 5-tuple packet classification is considered a drenched research area, few work has been done on FPGAs. Decomposition based packet classifications algorithms such as DCFL{22} and BV{19}are commonly used in most of the readily available FPGA. Lakshman *et al.* [19] 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. Taylor *et al.* [22] introduce Distributed Cross producting of Field Labels (DCFL), which is also a decomposition-based algorithm leveraging several observations of the structure of real filter sets. They putrefy the multi-field searching problem and use independent search engines, which can function in parallel to find the alike conditions for each filter field. Jedhe et al. [15] realize the DCFL architecture in their entire firewall implementation on a Xilinx Virtex 2 Pro FPGA, using a memory intensive approach, as opposed to the logic intensive one, so that on-the-fly update is possible. Two recent works [24], [25] discuss several issues in implementing decision-tree-based packet classification algorithms on FPGA, with different motivations. Luo *et al.* [24] propose a method called *explicit range search* to allow more cuts per node than the Hyper Cuts algorithm. Based on the cost of increased memory utilization, the tree height is radically reduced.At each internal node in order to find which child node to traverse it is needed to determine the a varying number of memory accesses, which may be infeasible for pipelining. Since the authors do not implement their FPGA design, the actual performance results are undecided.

3. Proposed Architecture Design

To achieve line-rate throughput, we map the decision forest including trees onto a parallel multi-pipeline architecture with P linear pipelines, as shown in Fig. 6, where P=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 tree stages are the pipeline stages for tree traversal which is known as tree stages while those used for rule list matching are called the rule stages. Each tree stage contains 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 consequent leaf node is regain to access the rule stages. Since a leaf node contains a list of list Size rules, we need list Size rule stages for matching these rules. All the leaf nodes of a tree have their rule lists mapped onto these list Size 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 .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 outputs from the pipelines.

Table 2. *Formation of Rules.*

Header Filed	Bits Allocated	Rule Type
Source Address	32	Rule1
Destination Address	32	Rule2
Source Port Number	16	Rule3
Destination Port Number	16	Rule4
Transport layer protocol	3	Rule5
Ethernet Type	16	Rule6
VLAN ID	12	Rule7
VLAN Priority	3	Rule8

3.1. Pipeline

Like the HyperCuts with the *push common rule upwards* heuristic enabled, our algorithm may reduce the memory consumption at the cost of increased search time, if the process to match the rules in the *internal rule list* of each tree node is placed in the same critical path of decision tree traversal. Any packet traversing the decision tree must perform: 1) matching the rules in the internal rule list of the existing node and 2) branching to the child nodes, in series. The number of memory accesses all along the grave path can be very large in the worst cases. Though the throughput can be improved by using a deep pipeline, the large delay transitory the packet classification engine need the router to use a large buffer to store the payload of all packets being classified. Moreover, since the search in the rule list and the traversal in the decision tree have different structures, a heterogeneous pipeline is needed, which complicates the hardware design.

3.2. Tree-to-Pipeline Mapping

Before the FPGA implementation, the size of the memory in the pipeline stages should be known. However, when simply mapping each level of the decision tree onto a separate stage, the memory distribution across stages can vary extensively. Allocating memory with the maximum size for each stage results in large memory wastage. This propose a Ring pipeline architecture which employs TCAMs to achieve balanced memory distribution at the cost of halving the throughput to one packet per two clock cycles, i.e., 0.5 PPC, due to its non-linear structure. Our task is to map the decision tree onto a pipeline (i.e., Tree Pipeline in our architecture) to achieve balanced memory distribution over stages, while sustaining a throughput of one packet per clock cycle (which can be further improved to 2 PPC by employing dual-port RAMs). The memory distribution across stages should be balanced not only for the Tree Pipeline, but also for all the Rule Pipelines. Note that the number of words in each stage of a Rule Pipelines depends on the number of tree nodes rather than the number of words in the corresponding stage of Tree Pipeline, as shown in Fig. 8. The challenge comes from the various number of words needed for tree nodes. As a result, the tree-to-pipeline mapping scheme requires not only balanced memory distribution, but also balanced node distribution across stages. Moreover, to maximize the memory utilization in each stage, the sum of the number of words of all nodes in a stage should approach some power of 2. Otherwise, for example, we need to allocate 2048 words for a stage consuming only 1025 words. The above problem is a variant of bin packing problems, and can be proved to be NP-complete. We use a heuristic similar to our previous study of trie-based IP lookup, which allows the nodes on the same level of the tree to be mapped onto different stages. This provides more flexibility to map the tree nodes, and helps achieve a balanced memory and node distribution across the stages in a pipeline, as shown in Fig. 3. Only one constraint must be followed.

Constraint 1: If node is an ancestor of node in the tree, then must be mapped to a stage preceding the stage to which is mapped.

We impose two bounds, namely and for the memory and node distribution, respectively. The values of the bounds are some power of 2. The criteria to set the bounds is to minimize the number of pipeline stages while achieving balanced distribution over stages. The complete tree-to-pipeline mapping algorithm, where denotes a tree node, the number of stages, the set of remaining nodes to be mapped onto stages, the number of words of the the stage, and the number of nodes mapped onto the Nth stage. We manage two lists, Ready List and Next Ready List. The former stores the nodes that are available for filling the current stage, while the latter stores the nodes for filling the next stage. We start with mapping the nodes that are children of the root onto Stage 1.When filling a stage, the nodes in Ready List are popped out and mapped onto the stage, in the decreasing order of their heights.2 After a node is assigned to a stage, its children are pushed into Next Ready List. When a stage is full or Ready List becomes empty, we move on to the next stage. At that time, Next Ready List is merged into Ready List. By these means, Constraint 1 is met. The complexity of this mapping algorithm is , where denotes the total number of tree nodes. Our tree-to-pipeline mapping algorithm allows two nodes on the same tree level to be mapped to different stages. We implement this feature by 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 is passed through the pipeline, the distance value is decremented by 1 when 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. External SRAMs are usually needed to handle very large rule sets, while the number of external SRAMs is constrained by the number of I/O pins in our architecture. By assigning large values of and for one or two specific stages, our mapping algorithm can be extended to allocate a large number of tree nodes onto few external SRAMs which consume controllable number of I/O pins.

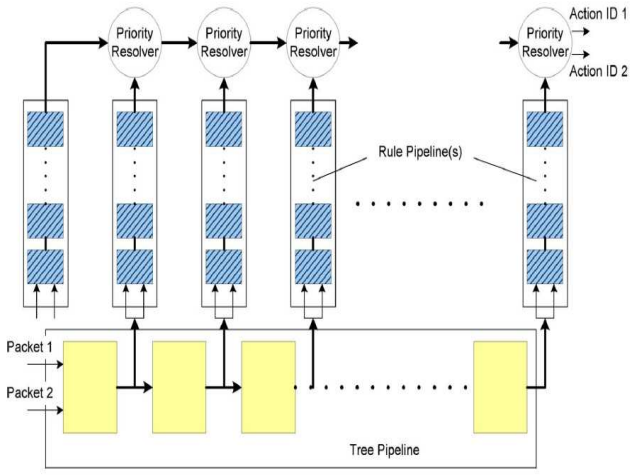

Figure 1. *2-D Multi-Pipeline Architecture.*

3.3. Pipeline for Rule Lists

When a packet accesses the memory in a Tree Pipeline stage, it will obtain the pointer to the rule list associated with the current tree node being accessed. The packet uses this pointer to access all stages of the Rule Pipeline attached to the current Tree Pipeline stage. Each rule is stored as one word in a Rule Pipeline stage, benefiting from the largeword width provided by FPGA. Within a stage of the Rule Pipeline, the packet uses the pointer to retrieve one rule and compare its header fields to find a match. When a match is found in the current Rule Pipeline stage, the packet will carry the corresponding action information with the rule priority along the Rule Pipeline until it finds another match where the matching rule has higher priority than the one the packet is carrying.

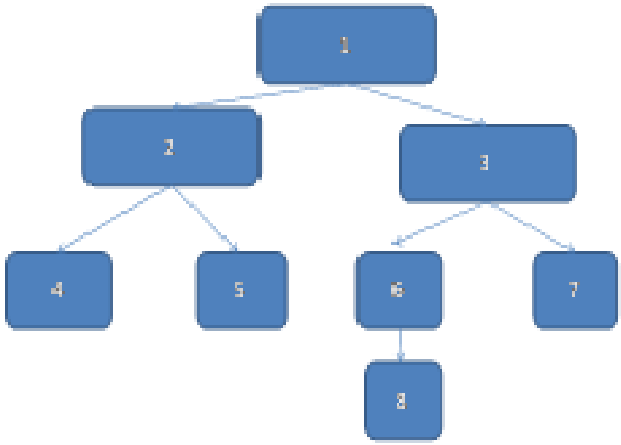

Figure 2. *Decision Tree.*

4. Simulation Results

Table 3. *Shows the Device Utilization Summary.*

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	682	9,312	7%	
Number of 4 input LUTs	1,987	9,312	21%	
Logic Distribution				
Number of occupied Slices	1,034	4,656	22%	
Number of Slices containing only related logic	1,034	1,034	100%	
Number of Slices containing unrelated logics	0	1,034	0%	
Total Number of 4 input LUTs	2,008	9,312	21%	
Number used as logic	1,987			
Number used as a route-thru	21			
Number of bonded IOBs	18	232	7%	
IOB Flip Flops	8			
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	22,689			
Additional JTAG gate count for IOBs	864			

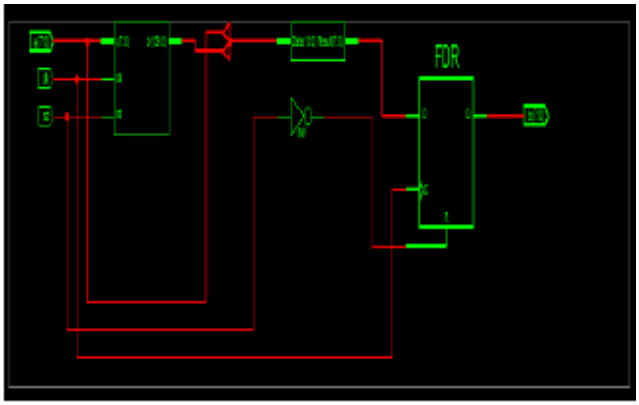

Figure 3. *RTL View of Proposed Architecture.*

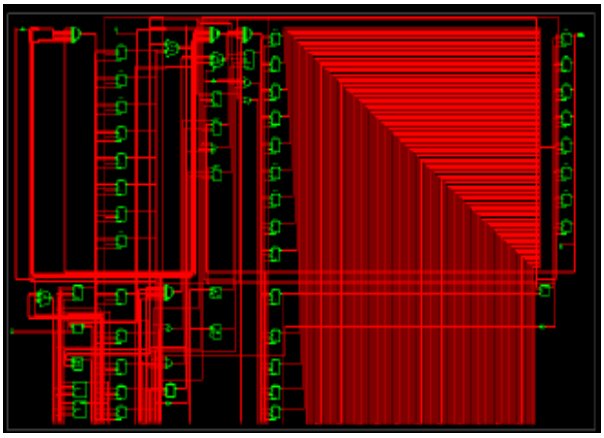

Figure 4. *Technology Schematic View.*

References

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

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

[3] Open Flow Foundation, "Open Flow 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. Mc Keown, "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.

[16] Xilinx, Inc., San Jose, CA, "Xilinx Virtex-6 FPGA family,"2009.[Online].Available:www.xilinx.com/products/virtex6/

[17] Altera Corp., San Jose, CA, "Altera Stratix IV FPGA," 2009.[Online].Available:http://www.altera.com/products/devices/stratix-fpgas/ stratix-iv/

[18] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[19] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *Proc. SIGCOMM*, 1998, pp. 203–214.

[20] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. SIGCOMM*, 2003, pp. 213–224.

[21] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.

[22] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *Proc. INFOCOM*, 2005, pp. 269–280.

[23] W. Eatherton, G. Varghese, and Z. Dittia, "Tree bitmap: Hardware/ software IP lookups with incremental updates," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 97–122, 2004.

[24] Y. Luo, K. Xiang, and S. Li, "Acceleration of decision tree searching for IP traffic classification," in *Proc. ANCS*, 2008, pp. 40–49.

[25] A. Kennedy, X.Wang, Z. Liu, and B. Liu, "Low power architecture for high speed packet classification," in *Proc. ANCS*, 2008, pp. 131–140.

[26] Y. Luo, P. Cascon, E. Murray, and J. Ortega, "Accelerating Open Flow switching with network processors," in *Proc. ANCS*, 2009, pp. 70–71.

[27] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. Mc Keown, "Implementing an Open Flow switch on the Net FPGA platform," in *Proc. ANCS*, 2008, pp