A High-Speed and Memory Efficient Pipeline Architecture for Packet Classification

Yeim-Kuan Chang, Yi-Shang Lin, and Cheng-Chien Su
Department of Computer Science and Information Engineering
National Cheng Kung University, Tainan, 701, Taiwan
{ykchang, p7696104, p7894104}@mail.ncku.edu.tw

Abstract—**Multi-field Packet classification is the main function in high-performance routers. The current router design goal of achieving a throughput higher than 40 Gbps and supporting large rule sets simultaneously is difficult to be fulfilled by software approaches. In this paper, a set pruning trie based pipelined architecture called Set Pruning Multi-Bit Trie (SPMT) is proposed for multi-field packet classification. However, the problem of rule duplications in SPMT that may cause a memory blowup must be solved in order to implement SPMT with large rule sets in FPGA devices consisting of limited on-chip memory. We will propose two rule grouping schemes to reduce rule duplications in SPMT. The first scheme called Partition by Wildcards (PW) divides the rules into subgroups based on the positions of their wildcard fields. The second scheme called Partition by Length (PL) rules partitions the rules into subgroups according to their prefix lengths. Based on our performance experiments on Xilinx Virtex-5 FPGA device, the proposed pipeline architecture can achieve a throughput of over 100 Gbps with dual port memory. Also, the rule sets of up to 10k rules can be fit into the on-chip memory of Xilinx Virtex-5 FPGA device.**

I. INTRODUCTION

As the Internet becomes widely used, the next generation routers need to support a variety of value-added network services such as QoS differentiation, traffic billing, network security, and others. In order to support these services, packet classification that classifies packets traversing the Internet into flows is applied. It is well known that multi-field packet classification is a difficult problem [11].

Algorithms for packet classification can be divided into two groups: software-based and hardware-based. Software-based algorithms have an advantage of the flexibility but are slow. For example, pipeline and parallel techniques are hard to apply to software-based algorithms and can hurt throughputs. Today, researchers have turned to the hardware-based solutions that use field programmable gate array (FPGA), application specific integrated circuit (ASIC), and ternary content addressable memory (TCAM). ASIC and FPGA based approaches usually store the rule set in static random access memory (SRAM) or TCAM that stores not only 0 and 1 but also “don’t care” value on TCAM cells. At present, the achievable link rate is OC-768, i.e., 40Gbps, which is equivalent to the processing speed of one 40-byte packet every 8 ns.

In order to achieve such high throughput, we develop a data structure called set pruning multi-bit trie (SPMT) which is very suitable for the pipeline architecture. In order to reduce the memory blowout problem caused by the rule duplications in set pruning tries, two rule partitioning schemes are proposed to divide rules into subgroups based on wildcards and prefix lengths. Then, SPMT is built for each subgroup and all the SPMTs are searched in parallel to obtain the best matched rule.

Our experimental results conducted on Xilinx Virtex-5 XC5VFX200T FPGA [13] device show that the proposed pipeline architecture can achieve a throughput of over 100 Gbps with dual port memory. Also, the block RAMs of size 16,416 Kbits available in the Xilinx Virtex-5 XC5VFX200T FPGA are capable of storing the rule sets of up to 10k rules.

The rest of the paper is organized as follows. In section 2, we introduce the background about packet classification problem and related work. The section 3 illustrates the proposed design. Section 4 describes implementation of our proposed design. In section 5, we present our experiment results. The last section is the conclusion.

II. BACKGROUND AND RELATED WORK

Packet classification in the routers classifies packets into flows by searching the rule set which consists of a finite set of rules (or called filters) for obtaining the action, i.e. flow ID. Each filter consists of a set of fields and the associated action. The types of match conditions in the fields are typically prefixes for 32-bit source and destination IP fields (SA/DA), ranges for the 16-bit source and destination ports (SP/DP), and an 8-bit protocol number for transport layer protocol (PROT).

Formally, a filter F with d fields is defined as F = (f_1, f_2, …, f_d). A packet P is said to match a particular filter F if the i^{th} header field of packet P is contained in f_i for all i. The packet may match multiple filters and the classifier only applies the action with highest priority among all the matching filters [11]. An example rule set is shown in Table I while we assume that the packet header fields consist of 8-bit source/destination IP address prefix, 5-bit source/destination port range, and 8-bit protocol number.

Not so many packet classification proposals have been implemented on FPGAs. Existing FPGA implementations of packet classification algorithms include Improved HyperCuts [3], Power Saved HyperCuts [6], Dual Stage Bloom Filter Classification Engine (2sBFCE) [7], Bloom Based Packet Classification (B2PC) [8], and Nest Level Tuple Merging and Cross-product (NTLMC) [2]. Also, it is worth mentioning that several pipeline architectures have been proposed for IP lookup based on trie data structures, such as [4][5].

An improved data structure called *set pruning trie* [10] avoids the backtrack operations and thus increases the search speed. However, set pruning trie incurs a memory blowout problem caused by the rule duplications in the tries at the

978-0-7695-4056-6/10 $26.00 © 2010 IEEE
DOI 10.1109/FCCM.2010.40

TABLE I. A SAMPLE RULE SET

Filter						*Action*
#	SA	DA	SP	DP	PROT	FlowID
R1	0*	10*	[16:23]	[0:31]	UDP	0
R2	0*	00*	[8:15]	[28:31]	UDP	1
R3	0*	1*	[16:31]	[28:31]	UDP	2
R4	001*	11*	[28:31]	[28:31]	UDP	3
R5	001*	0010*	[4:5]	[0:15]	UDP	4
R6	*	111*	[28:31]	[0:15]	UDP	5
R7	001*	0*	[0:15]	[16:23]	UDP	6

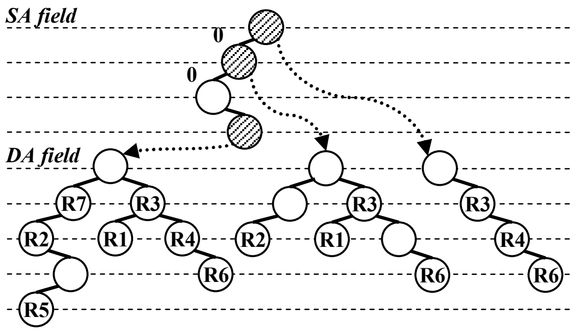

Figure 1. A 2-D 1-bit set pruning trie example.

lower levels. Fig. 1 shows an example of the 1-bit set pruning trie constructed from the rules in Table 1 by using the two prefix fields. Many rules are duplicated in the lower levels to avoid backtracks for finding the correct results. Notice that the duplicated rule R6 is duplicated in many places. Since trie-based data structure is very suitable for pipeline design, our proposed SPMT pipeline architecture is based on the set pruning trie.

III. PROPOSED SCHEMES

In this section, we propose to use the multi-bit version of set pruning trie as our basic data structure because it is easy to have a very fast pipeline implementation for the trie-based structure. Due to a lot of rule duplications in the lower level of the set pruning trie, we focus on reducing memory consumption by using the rule partitioning schemes to avoid duplicating rules as much as possible. Additionally, we target on the 5-field rule tables.

A. Set Pruning Multi-Bit Trie

It is well-known that the traditional multi-bit trie is very suitable for the IP address lookup pipelined search engine [5]. Because the 5-field set pruning trie uses too many stages, we develop a multi-bit version of the set pruning trie called set pruning multi-bit trie (SPMT) to minimize the number of pipeline stages. The SPMT building process is very similar to the set pruning trie. The only difference is that when building SPMT we have to avoid the redundant data structures caused by the address expansion onto the multi-bit nodes. In other words, the nodes should be shared and pointed to by as many pointers as possible. For the reason of limited space, Fig. 2 only shows the 2-bit SPMT constructed from the source and destination IP address fields of the rules in Table I. Each node contains four elements with indices of 00, 01, 10, 11 in the 2-bit SPMT. Each element contains a next-field link and a next-level link that are drawn by dotted and solid lines in Fig. 2, respectively. The root node has 2

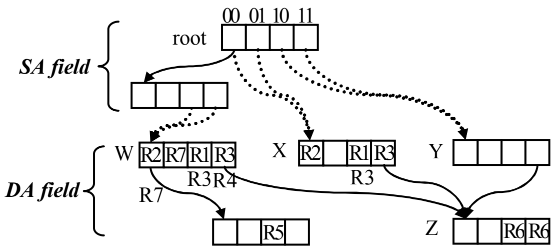

Figure 2. The 2-bit SPMT constructed from the rules in Table I.

prefixes "*" and "0*" in the SA field. The prefix "0*" is expanded to 00 and 01 and the prefix "*" is expanded to 00, 01, 10 and 11. Therefore, the next-field link of the two elements 00 and 01 of the root node point to the same next-field node X that consists of the sub-rules from R1, R2, R3, and R6. Similarly, the elements 10 and 11 of the root node point to node Y that consists of sub-rules from only R6. When we construct DA field, the nodes W, X, Y could share the same node Z pointed to by the same next-level link. Our experimental results show that the node sharing can reduce 40%~45% of the trie nodes.

The search operation is similar to that of the traditional set pruning trie [10]. The only difference is that we use multiple bits to traverse the SPMT nodes. Although some nodes can be shared, duplications inside the nodes are not removed. Also, the duplications caused by pushing the rules in upper levels to the lower levels incur a lot of memory usage. Hence, we proceed to seek improvements by analyzing the wildcard field values contained in rules as follows.

B. Partitioning by Wildcards (PW)

We can divide 5-field rule set into 32 possible cases (subgroups) such that the wildcard field values appear in some specified fields. However, we ignore the protocol field for dividing rules into subgroups because the protocol field has more values of TCP and UDP than wildcard and thus the impact on rule duplication from protocol field in set pruning trie is much smaller than the other four fields. As shown in the example of Fig. 1, wildcard fields will be duplicated in many places in the lower levels. Hence, we can divide the rules into at most 16 subgroups and an SPMT is built for each subgroup by ignoring the wildcard fields. Thus, a lot of rule duplications can be removed from the resulting SPMT.

As shown in Table II, we analyze three rule sets containing about 10K rules generated by *ClassBench* [12]. The IPC tables have more non-empty subgroups than ACL and Firewall tables which just have 5-6 subgroups. Although we divide the rule set into subgroups by wildcards, non-wildcard fields are still covered by short prefix of lengths 1 and 2. Based on our analysis on the length distribution of the source and destination IP fields, some of the rule tables contain up to 50 percentages of the rules whose lengths are 1 or 2. Therefore, we can further divide the subgroups partitioned by wildcards into many smaller subgroups and thus the reduction of rule duplications can be reduced further.

C. Partitioning by Lengths (PL)

TABLE II. SUBGROUPS PARTITIONED BY WILDCARD POSITIONS.

Rule name	*ACL1_10K*	*FW1_10K*	*IPC_10K*
# of filters	12947	13146	11831
# of groups	5	6	12

In some cases, using PW scheme to partition Firewall tables still needs a lot of memory because Firewall has many prefixes of length 0, 1 and 2 in destination IP address fields. The prefixes of lengths 0, 1, and 2 almost occupy over half of rules in Firewall tables. Hence, we focus on partitioning the rule set by prefix length. The main idea is that we like to select k boundary prefix lengths in the first field and divide the rule set into $k + 1$ subsets. If $k = 1$ and the selected length is t, the rules with the first field value of prefix length less than or equal to t are put into one subset and the other rules are put into another subset. This case is exactly the same as the simple *k_set* technique [1]. Thus, the original rules in which prefix field lengths shorter than and equal to t in the subset will not be duplicated in the other subset containing the prefixes of lengths longer than t.

To further explore the simple *k_set* technique, we propose the dynamic programming formula to select k boundary prefix lengths for the purpose of partitioning rules into (k + 1) subsets. By using the dynamic programming technique, we can construct a simple formula to compute what is the best set of k lengths to minimize the rule duplications. The dynamic programming formula is based on the *Duplicating Cost* (DC) which is defined to be number of times that a rule is duplicated in SPMT. We use the example in Fig. 1 to describe how to calculate the DC of a rule R6. Assume R6 = (*, 111*) is a 5-field rule with the first and second field values set to * and 111*, respectively. Rule R6 should be duplicated in the valid nodes (prefixes) covered by R6 in the binary trie of the first field. Fig. 1 shows that there are two valid nodes covered by R6 in the first field that are marked as gray circles. For each valid node covered by R6 in the first field binary trie, we have to calculate the number of nodes in the second field trie that are covered by the second field value of R6. We continue the same process in the following fields until the penultimate field. Finally, the DC of rule R6 can be obtained by summing up all the duplications in all the fields up to the penultimate field. Since the second field is already the penultimate field, no further computation is needed in the third field of Figure 1. Totally, rule R6 is duplicated 3 times and thus the duplicating cost of R6 is 3 (denoted as *DuplicatingCost*(R6) = 3).

We will design our dynamic programming formula. Consider the general problem of determining the minimum duplicating cost DC [L, k] for prefixes of lengths 0 to L by using k boundary lengths. The number k is constrained by the maximum boundary length L, i.e., k must not be greater than L. The prefix length of rule r is denoted by PLen(r). When we select length t as the first boundary length, all the prefixes of lengths greater than or equal to t have to be expanded to length L which is primed to the rule duplications. After selecting one boundary length, k – 1 more boundary lengths have to be selected afterwards. The remaining k – 1 boundary lengths are in the range [k – 1, L – 1]. Finally, we recursively find the minimum DC for the prefixes of lengths

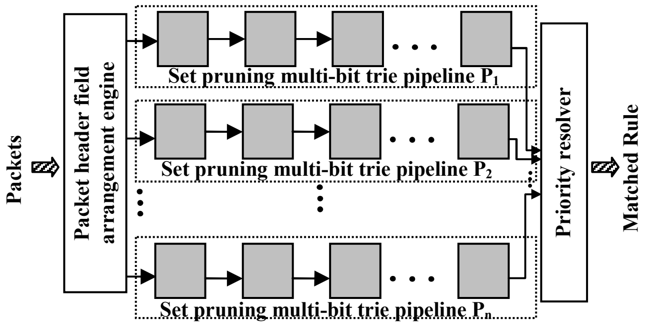
Packets
Packet header field arrangement engine
Set pruning multi-bit trie pipeline P1
Set pruning multi-bit trie pipeline P2
Set pruning multi-bit trie pipeline Pn
Priority resolver
Matched Rule

Figure 3. Block diagram of the parallel linear pipeline.

1 to t and the calculate DC for the rules of lengths greater than t. As a result, we can form the dynamic programming formula as follows.

$$DC[L,k] = \underset{t\in\{k-1,\ldots,L-1\}}{Min}\left\{ DC[t,k-1] + \sum_{r\in\{R\ |PLen(R)\le L\}} DuplicatingCost(r) \right\}$$

$$DC[L,0] = \sum_{r\in\{R\ |PLen(R)\le L\}} DuplicatingCost(r)$$

The initial problem is DC [L_{max_1}, k], where L_{max_1} is the maximum prefix length of the first field that is targeted for partition.

IV. IMPLEMENTATION

A pipelined and parallel design is used to improve the throughput of our proposed SPMT. The trie nodes in the same level are mapped to a single pipeline stage. Each packet traversing the SPMT level by level is equivalent to traversing the pipeline architecture stage by stage. Each pipeline stage must perform the following actions:

1. The memory of the SPMT is accessed to obtain the next level address and next field address.
2. The next field memory address of the longest prefix is recorded from the first field to the last field.
3. If the last field is reached, the matching rule information is obtained and then its priority is compared with matching rule obtained from previous stages.

FPGAs provide massive parallelism and high-speed dual-port Block RAMs distributed across the device. We exploit these features and propose a parallel architecture, as shown Fig. 3. The design is based on the following considerations:

1. A pipeline P_i is constructed for each subgroup (PRT_i) of the original rules after applying the proposed PW and PL partitioning schemes. The entire rule set is equal to $\sum PRT_i$ for $i = 1 \ldots n$.
2. Traversing the SPMT level to level and each level is mapped to a stage in pipeline. It is shown as gray color blocks in Fig 3.
3. Assume a field is X-bit wide and stride is Y-bit. The number of pipeline stages required is $\lceil X/Y \rceil$.
4. The incoming packets will be input in all pipelines in parallel. Then, each pipeline is executed independently. The packet header field arrangement engine will select a field order to construct the proposed SPMT for each subgroup.

Each pipeline will output a result which will be sent to priority resolver to select the rule with highest priority as the final result.

TABLE III. MEMORY USAGE

Rule sets			Methods (Mb): No Partition	Proposed PW	Proposed PW and PL combined
	Rule name	# of rule			
Real Rule	ACL1	752	1.71	0.62	0.55
	FW1	269	13.31	5.12	1.33
	IPC1	1,550	551.14	7.12	3.41
Synthetic Rule	ACL1_1K	916	5.04	2.40	1.60
	ACL1_5K	4,415	overflow	9.52	8.64
	ACL1_10K	9,603	overflow	31.20	15.36
	FW1_1K	791	41.20	16.48	2.48
	FW1_5K	4,653	overflow	485.60	11.28
	FW1_10K	9,311	overflow	2171.20	16.88
	IPC1_1K	938	510.56	4.08	2.08
	IPC1_5K	4,460	overflow	31.04	9.12
	IPC1_10K	9,037	overflow	94.08	16.08

TABLE IV. HARDWARE RESOURCE COMPARISON WITH IMPROVED HYPERCUTS

	Improved HyperCuts [7]	Proposed PW	Proposed PW and PL
# of rules	ACL1_10k	ACL1_5k	ACL1_10k
# of slices / utilization	10,307/33%	2,242/8%	6,854/24%
# of Block RAMs / utilization	407/89%	263/58%	429/94%
Frequency (MHz)	125.4	167.44	173.02
Throughput (Gbps)	80.23	107.16	110.73

TABLE V. COMPARING THROUGHPUT WITH OTHER METHODS

Approaches	# of rules	Throughput in Mpps	Throughput in Gbps
Proposed PW	4,451	334.88	107.16
Proposed PW and PL	9,603	346.04	110.73
Power Saved HyperCuts [6]	≈25,000	32.00	10.24
Improved HyperCuts [3]	9,603	250.8	80.23
BV-TCAM [9]	222	32	10.00
2sBFCE [7]	4,000	5.86	1.88
B2PC [8] in ASIC	3300	42.5	13.60
NTLMC [2]	12507	38	12.16

V. PERFORMANCE

We use Xilinx ISE 10.1 development tools and the target device is Xilinx Virtex-5 XC5VFX200T [13] with '–2' speed grade. The tables are obtained from a publicly available tool, *ClassBench* [12] and three real-life rule tables. We evaluated the memory usage of the proposed SPMT of 4-bit strides with the rule sets of different sizes. Based on our results, we can save a large number of memory usages for the three real-life tables. The results shown in Table III are obtained after the redundant nodes are removed. When no partitioning approach is used, the required memory is too large to be hold in Xilinx Virtex-5 XC5VFX200T when the rule set size is large than 5k. The memory saves for all the tables are tremendous after both PW and PL partitioning schemes are applied. For example, the FW1_10K needs only 16.88 Mb after both PW and PL partitioning schemes are applied compared with 2171.2 Mb when only PW scheme is applied.

The FPGA implementation results are shown in Table IV. The method called Improved HyperCuts proposed by Jiang *et al.* [3] is also compared. In order to have a fair comparison with Improved HyperCuts, we use the same FPGA device and dual-port memory. Our hardware cost is lower than Improved HyperCuts and clock rate also is higher than Improved HyperCuts. As we can see, both schemes can achieve the throughput of OC-768 but our proposed schemes are better. For the table of 10k rules, our slice utilization is lower (24% vs. 33%) but our Block RAM utilization is higher (94% vs. 89%) than Improved HyperCuts. Hence, Xilinx Virtex-5 XC5VFX200T is sufficient to support the proposed SPMT with 10K rules. Table V compares the throughputs of the proposed SPMT and other existing schemes. Our proposed schemes are implemented with Xilinx Virtex-5 XC5VFX200T and packet size is 40 bytes. We can see that the proposed SPMT outperforms all the schemes.

VI. CONCLUSION

In this paper, we proposed a pipeline packet classification architecture which aims at achieving a very high throughput for large rule table. The Set Pruning Multi-bit Trie based on the set pruning trie is first proposed. The drawback of memory blowout for large rule tables caused by rule duplication is solved by the proposed rule table partitioning schemes PW and PL. The PL partitioning scheme uses dynamic programming for choosing the best lengths to divide rule table into many independent subgroups. Based on our performance experiments on Xilinx Virtex-5 FPGA device, our proposed design can achieve the throughput over 100 Gbps with dual port memory while supporting up to l0k rules.

REFERENCES

[1] Y.-K. Chang, "Efficient Multidimensional Packet Classification with Fast Updates," *IEEE Transactions on Computers*, Vol. 58, No. 4, pp. 463-479, Apr. 2009.

[2] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast Packet Classification Using Bloom Filters," In *ACM/IEEE ANCS*, 2006.

[3] W. Jiang and V. K. Prasanna, "Large-Scale Wire-Speed Packet Classification on FPGAs," In *ACM/SIGDA FPGA*, 2009.

[4] W. Jiang and V. K. Prasanna, "Parallel IP Lookup using Multiple SRAM-based Pipelines," In *IEEE IPDPS*, 2008

[5] W. Jiang and V. K. Prasanna, "A Memory-Balanced Linear Pipeline Architecture for Trie-Based IP Lookup," In *IEEE HOTI*, 2007

[6] A. Kennedy, X. Wang, Z. Liu, and B. Liu, "Low Power Architecture for High Speed Packet Classification," In *ACM/IEEE ANCS*, 2008.

[7] A. Nikitakis and I. Papaefstathiou, "A Memory-Efficient FPGA-Based Classification Engine," In *IEEE FCCM*, 2008.

[8] I. Papaefstathiou and V. Papaefstathiou, "Memory-Efficient 5D Packet Classification at 40 Gbps," In *IEEE INFOCOM*, 2007.

[9] H. Song and J. W. Lockwood, "Efficient Packet Classification for Network Intrusion Detection Using FPGA," In *ACM FPGA*, 2005.

[10] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," In *ACM SIGCOMM*, 1998.

[11] D. E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys*, vol. 37, no.3, pp. 238-275, Sep. 2005.

[12] D. E. Taylor and J. S. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Transations on Networking*, vol. 15, no. 3, pp. 499-511, June 2007.

[13] Xilinx, "Virtex-5 Family Overview", Product Specification, DS100 (v5.0), Feb. 6, 2009, at http://www.xilinx.com