Pseudo-TCAM: SRAM-based Architecture for Packet Classification in One Memory Access

Journal:	*IEEE Communications Letters*
Manuscript ID	Draft
Manuscript Type:	Original Article
Key Words:	packet classification, prefix inclusion code, ternary content addressable memory, FPGA implementation
EDICS:	CL1.7.1 High-speed Networks < CL1.7 Network Architecture and Design, CL1.9.7 Security < CL1.9 Network Operations and Management

Pseudo-TCAM: SRAM-based Architecture for Packet Classification in One Memory Access

Author One, Author Two, and Author Three

***Abstract*— Packet classification is used to support a number of network management functions, such as access control, firewall, quality of service provisioning, policy based switching, and traffic policing. Conventional approach implements packet classifier using ternary content addressable memory (TCAM). We shall present a simple SRAM-based hardware architecture that emulates the behavior of TCAM for packet classification. Header fields of a classification rule are encoded using the prefix inclusion coding method. Encoded rules are mapped to multiple SRAM-based match units using a bit-selection approach. Rules mapped to the same match unit have distinct values in the selected bits. Selected bits of the input key are used as the address to access a rule in the SRAM for comparison. The proposed method is implemented on a virtex-7 FPGA. The average memory cost is 23 bytes per rule, and the processing logic is simple. Throughput of the classifier can reach 400 million packets per second.**

***Index Terms*—Packet classification, prefix inclusion code, ternary content addressable memory, FPGA implementation.**

I. Introduction

Packet classification is used to support a number of network management functions, e.g. access control, firewall, quality of service provisioning, policy based switching, and traffic policing. Typically, a classification rule contains 5 fields, namely the source and destination IP addresses (SA and DA), source and destination port numbers (SP and DP), and protocol (PT). The SA and DA fields are represented by address prefixes, the SP and DP fields are represented by number ranges, and the PT field may be a specified value or wildcard. By convention, rules are prioritized according to the order in which they appear in the ruleset. In general, the last rule in the list is the default rule with all 5 fields equal to wildcard. An input packet may match multiple rules, and the action of the highest priority matching rule is applied. Readers may refer to [1, 2] for tutorials on packet classification algorithms.

Ternary content addressable memory (TCAM) is a popular device used in commercial routers for IP address lookup and packet classification. Simplicity is a major advantage of TCAM. The 5 header fields of an incoming packet are sent to the TCAM, and the TCAM outputs the address of the highest priority matching rule. The system will then use the address to retrieve the rule action from an associated SRAM. The SP and DP fields are converted to prefix format when a rule is stored in the TCAM. A range may be expanded to multiple prefixes, and this is called the *port range expansion* problem. A classifier with n rules occupies $e \times n$ TCAM entries. According to [2], the average expansion factor e is about 2.32.

In addition to the port range expansion problem, TCAM has a few drawbacks including high cost, high power consumption, and limited lookup rate. Commercial TCAM can be configured to have 36, 72, 144, 288, and 576 bits word width. An IPv4 rule requires 104 bits word width, whereas an IPv6 rule requires 296 bits word width. Hence, the effective average TCAM storage per IPv4 rule is about 42 bytes. The maximum lookup rate is 133 million packets per second (MPPS) for 72 and 144 bits word width. When the TCAM word width is increased to 288 and 576 bits, the lookup rate is reduced to 66.5 and 33.25 MPPS, respectively. Several encoding schemes [3-6] have been proposed to minimize port range expansion and the required TCAM word width. However, codeword lookup for the SA/DA field is equivalent to the IP address lookup problem. If the codeword lookup is performed by the TCAM, the overall throughput of the TCAM is reduced. On the other hand, if the codeword lookup is performed by a FPGA, the system requires a FPGA in addition to the TCAM.

A few studies on implementing TCAM on FPGA using block RAM (BRAM) and logic elements (LUTs) can be found in [7-8]. The Z-TCAM [7] method requires 36 BRAMs and 1781 LUTs to implement a 512×36 TCAM array, i.e. using 72 bits of SRAM storage to implement 1 bit of TCAM storage. The S-DIRECT method of [8] uses 36108 LUTs to implement a 1024×32 TCAM, and it is only applicable to IP address lookup with wildcard bits appearing on the right-hand side of the address prefix. One 6-input LUT is equivalent to a 64-bit memory array. The implementation cost is very high. Xilinx provides an intellectual properties core for realizing TCAM on FPGA [9]. It requires 5 BRAMs and 141674 LUTs to implement a 1024×128 TCAM array. The lookup latency is 12 clock cycles. One can see that the aforementioned methods [7-9] are not cost effective for packet classification.

There are substantive researches on algorithmic approaches for packet classification [10-22]. The published methods make use of some combinations of techniques, e.g. decision tree,

Manuscript received July 2018.
Authors One, Two and Three are with the same University.

hierarchical trie, cross-producting, field encoding, pipelined and parallel processing. The best achievable lookup rate is 388 MPPS [22], and the best memory performance is about 19.2 bytes per rule [21]. In this letter, we shall present a SRAM-based architecture for packet classification that will emulate the operation of TCAM. The hardware architecture is much simpler than the existing algorithmic approaches, and it offers very good performance. The average memory cost of the proposed method is 23 bytes per rule, and the lookup rate is 400 MPPS when implemented on virtex-7 FPGA.

II. Proposed Hardware Architecture

We shall use a simple example to illustrate our idea. Consider a set of 4 12-bit artificial rules shown in Fig. 1. The rules are shown in binary format, where the symbol *x* represents a wildcard bit. By selecting 2 appropriate bits from each rule such that the values of the selected bits are distinct, the 4 rules are mapped to distinct locations in a memory array as shown in Fig. 2. To determine if an input key matches any one of the 4 rules, the hardware uses 2 multiplexors (MUXs) to extract the selected bits from the input key and uses them as the address to access the SRAM. The retrieved rule is then compared to the input key to determine if there is a match.

Rule	Value
R_1	0001 x11x xx10
R_2	001x x100 xx1x
R_3	10xx x010 xx01
R_4	1100 1001 011x

Fig. 1. Sample set of 12-bit artificial rules.

Address	Data (stored rule)
00	001x x100 xx1x (R_2)
01	0001 x11x xx10 (R_1)
10	1100 1001 011x (R_4)
11	10xx x010 xx01 (R_3)

Fig. 2. Mapping of the 4 sample rules to a 4-entry SRAM memory array based on 2 selected (underlined) bits.

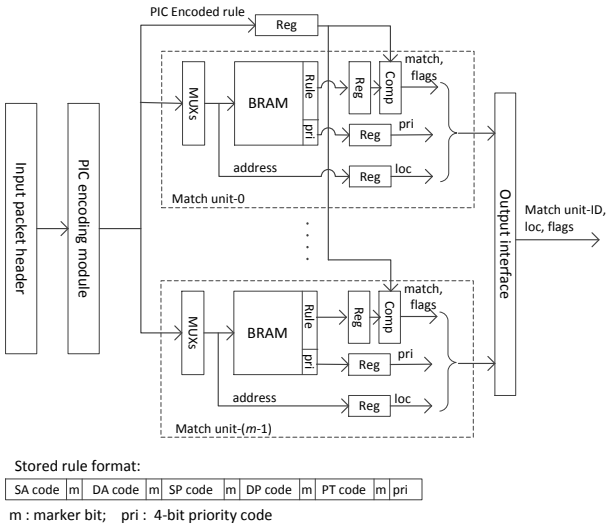

Fig. 3. Organization of the pseudo-TCAM classifier.

The organization of the pseudo-TCAM classifier is depicted in Fig. 3. The hardware contains multiple match units. Each match unit consists of a SRAM memory array, a set of MUXs that are used to extract the selected bits from the input key to form the memory address, and a comparator that compares the selected rule with the input key. Each MUX is controlled by a register that stores the index of the selected bit. Hence, the MUXs are programmable. A 4-bit priority code is associated with each rule stored in a match unit. If the input key matches multiple rules in different match units, the output interface selects the highest priority match result. The match result contains the match unit ID, the address of the matching rule and 2 control flags. The priority code and the control flags will be explained later. In our design, we would optimize the hardware cost, i.e. minimize the number of match units required for a given ruleset.

We shall encode the 5 header fields using the prefix inclusion code (PIC) method [4]. The PIC encoding module in Fig. 3 is similar to the phase-1 processing module of [21]. PIC encoding offers 2 major advantages. First, the prefix-form codewords preserve the inclusion property of the original data values, and an arbitrary range can be represented by one codeword. Second, the lengths of the codewords are shorter than the original data fields. Synthetic rulesets generated using the Classbench ruleset generator [23] are used in this study. Each ruleset contains about 10K rules. The number of distinct values in each data field, and the PIC code lengths for the 12 rulesets are shown in Fig. 4. The length of an encoded rule is between 32 and 45 bits.

Ruleset (No. of rules)	SA	DA	SP	DP	PT
acl1 (9856)	2212 (13 bits)	601 (12 bits)	1 (0 bit)	108 (9 bits)	3 (2 bits)
acl2 (10414)	247 (11 bits)	765 (12 bits)	1 (0 bit)	27 (6 bits)	4 (3 bits)
acl3 (9533)	635 (13 bits)	283 (10 bits)	3 (2 bits)	188 (9 bits)	4 (3 bits)
acl4 (9640)	609 (13 bits)	979 (13 bits)	3 (2 bits)	234 (9 bits)	6 (3 bits)
acl5 (10840)	656 (11 bits)	1761 (12 bits)	1 (0 bits)	40 (6 bits)	4 (3 bits)
fw1 (9731)	2056 (13 bits)	7360 (15 bits)	13 (4 bits)	43 (6 bits)	4 (3 bits)
fw2 (9659)	6578 (14 bits)	3512 (13 bits)	9 (4 bits)	1 (0 bit)	4 (3 bits)
fw3 (9978)	152 (9 bits)	8063 (14 bits)	9 (4 bits)	39 (6 bits)	3 (2 bits)
fw4 (9551)	4056 (13 bits)	7238 (15 bits)	28 (6 bits)	49 (7 bits)	8 (4 bits)
fw5 (9923)	2157 (14 bits)	6458 (15 bits)	11 (4 bits)	33 (6 bits)	3 (2 bits)
ipc1 (10170)	135 (9 bits)	543 (12 bits)	34 (6 bits)	54 (7 bits)	6 (3 bits)
ipc2 (10001)	4701 (14 bits)	8963 (14 bits)	3 (2 bits)	3 (2 bits)	3 (2 bits)

Fig. 4. Number of distinct prefixes/ranges/values, and the PIC code lengths (numbers shown in brackets) for 12 rulesets.

The reduced rule length helps to improve the memory efficiency of the hardware classifier, and the PIC code format facilitates the mapping of the rules to match units. In the PIC code format, an address prefix (or range) that does not have

any longer length sub-prefix (or more specific sub-range) will be assigned a full-length codeword. This property will enable a more efficient strategy to select suitable bits for mapping the rules to the hardware match units.

Conventional approach to represent a rule is to use 2 bit-vectors, namely the mask-vector and the data-vector. The data-vector stores the value, and the mask-vector defines which bits are wildcard bits. This approach uses $2L$ bits storage to represent an L-bit rule. Assume the PIC code lengths are fixed for a given ruleset. We can store a k-bit prefix using k+1 bits. A *marker* bit with value 1 is placed after the last data bit, and the remaining bits are padded with 0. Consider three 4-bit codes 1001, 01* and *. The physical 5-bit representations are 10011, 01100, and 00000, respectively. The marker bit is underlined. The comparator circuit determines the rightmost bit with value 1, and compares the input key with the bits on the left hand side of the marker bit.

The marker bit of the PT field has a different meaning. It is not uncommon to have a pair of rules in the ruleset with identical values in the SA, DA, SP and DP fields, and one rule in the pair has a specified PT value and the other rule has the PT field equal to wildcard. The pair of rules can be combined into 1 entry in the match unit. If the PT code is 0, then it means that it is a wildcard field. If the PT code is non-zero and the marker bit is set, then it means that we have another rule with identical SA, DA, SP and DP fields, but the PT field is a wildcard. The comparator circuit generates a control flag to indicate if the PT field matches the stored PT value or it matches the wildcard (if the marker bit is set). Using this approach, an L-bit rule can be represented using L+5 bits.

The priority code of each rule is determined as follows. A directed graph is constructed by inserting the rules to the graph from low to high priority. The root node corresponds to the default rule. When rule R_i is added to the graph and R_i overlaps with another rule R_j already in the graph, a directed edge from R_j to R_i is created. The priority code of the root is 0. The priority code of R_i is equal to the length of the longest path from the root to R_i in the graph. The largest priority code value is up to 10, hence a 4-bit priority code is sufficient.

A 36-Kbit BRAM in the FPGA has 2 independent 36-bit I/O ports. In order to support 2 parallel reads to the BRAM, the BRAM is configured as 1024×36 bits memory array. An encoded rule requires 41 to 54 bits of storage. We may connect 2 BRAMs in parallel to form 72-bit words. However, the utilization of the memory is low. To improve the memory efficiency, we shall connect 3 BRAMs in parallel to form 108-bit words, such that each memory word (bucket) can store 2 encoded rules. Two comparator circuits are used to compare the 2 rules against the input key. By default, the rule on the left has higher priority. A second control flag in the output of the hardware match unit is used to indicate if the match result corresponds to the left or the right entry in the bucket.

III. Mapping Rules to Match Units

In this section we shall explain the rule mapping strategy. A small number of rules in each ruleset have short address prefixes, and the SP and DP fields are equal to wildcard. These rules can be handled by a special BRAM. The BRAM is configured as 1024×36 bits, and each 36-bit word is further divided into four 9-bit entries. Rules with the SP and DP fields equal to wildcard, and the lengths of the SA and DA codes are no more than 6 bits are mapped to the special BRAM using the leaf pushing approach. The hardware simply takes the most significant 6 bits from the SA and DA codes of the input key to access 1 of the 4K entries in the special BRAM. Each entry stores the PT code, the marker bit, and the priority code. The remaining rules are mapped to regular match units.

The initial mapping phase makes use of the numbers of distinct values in each data field to guide the bit selection process. For example, acl1 has 2212 distinct SA prefixes. Selecting the last 10 bits of the SA code will allow us to fill a match unit with good utilization. If more than 2 rules are mapped to the same bucket, the 2 rules with the largest number of wildcard bits are chosen. The remaining rules will have more defined bits such that it easier to find suitable selected bits in the next mapping rounds. For acl1, the DA field has 601 distinct prefixes. Hence, choosing the last 10 bits of the DA code may fill up 50% of the buckets in the match unit. The mapping algorithm will try out some combinations of 2 or 3 fields, e.g. taking 5 bits from SA code and 5 bits from DA code, or 6 bits from SA code and 4 bits from DP code, and so on. After trying out a few combinations and the utilization drops below some threshold, e.g. 20%, the mapping algorithm will switch over to use a more dynamic bit-selection strategy.

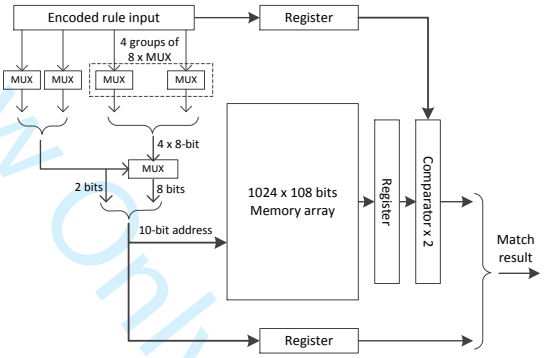

Fig. 5. Match unit with 2-level bit-selection.

After the initial mapping phase that uses field-based bit selection, we shall apply a 2-level bit selection approach to find a mapping of the remaining rules to match units. A match unit is divided into 4 partitions to allow more flexibility in the bit-selection process. Each partition has 256 buckets. The refined structure of the match unit is shown in Fig. 5. The algorithm will first select 2 bits such that the list of rules is divided into 4 groups with more or less balanced sizes. Three arrays rc_0, rc_1, and BF are used in the bit-selection algorithm. Let $rc_0[i]$ be the number of rules with the ith bit equal to 0, and $rc_1[i]$ be the number of rules with the ith bit equal to 1. The balance factor of the ith bit $BF[i] = \min(rc_0[i], rc_1[i]) / n$, where n is the number of rules in the list. Index of the first selected bit is b_0 where $BF[b_0]$ is the largest value in the array BF. Two sub-lists are obtained based on the value of the selected bit. The algorithm then computes BF_0 and BF_1 for the 2 sub-lists, respectively. Note that $BF_0[b_0]$ and $BF_1[b_0]$ are equal to 0. With 2 sub-lists, $BF[i] = BF_0[i]+BF_1[i]$. Index of

the second selected bit is b_1 where $BF[b_1]$ is the largest value in BF.

Four groups of rules are obtained based on the 2 selected bits. The same bit-selection process is applied to each group to select 8 bits. The rules in a group are mapped to the corresponding partition based on the 8 selected bits. Each bucket can store 2 rules. The mapping process goes through 3 rounds. In the first round, we shall only consider rules without wildcard bit in the 8 selected bit indexes. If 2 or more rules are mapped to the same bucket, the rule with the largest number of wildcard bits is chosen. In the second round, we shall consider rules that may contain wildcard bits in some of the selected bit indexes. The rule may be expanded to occupy multiple buckets. In the third round, we shall try to fill the buckets with the remaining rules. The above mapping process is repeated until all rules are mapped to match units.

IV. Performance Evaluation and Discussion

Performance of pseudo-TCAM is evaluated using 12 synthetic rulesets generated by the Classbench ruleset generator [23]. The numbers of regular match units (excluding the special BRAM unit for rules with short SA/DA codes) required for the 12 rulesets are shown in Fig. 6. For proof of concept, pseudo-TCAM is implemented on a virtex-7 FPGA. The processing logic consumes 5179 LUTs, and the FPGA can operate at 200 MHz. Two lookups can be performed in parallel, and the system throughput can reach 400 MPPS.

Ruleset	acl1	acl2	acl3	acl4	acl5	fw1
Match units	10	12	15	12	13	12
Ruleset	fw2	fw3	fw4	fw5	ipc1	ipc2
Match units	8	11	13	12	14	7

Fig. 6. Regular match units required for the 12 rulesets.

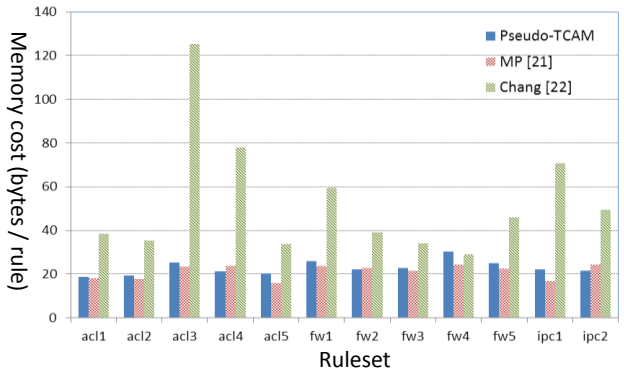

Fig. 7. Comparison of memory cost with 2 other methods.

A comparison of the memory cost with 2 state-of-the-art algorithmic methods [21, 22] is shown in Fig. 7. The average memory cost of pseudo-TCAM is 19% higher than MP [21], but it is 57% lower than [22]. The cost of the processing logic of pseudo-TCAM is only 36% of MP. Hence, the overall hardware cost of pseudo-TCAM is lower than MP. The speed of pseudo-TCAM is 17% faster than MP due to the simpler processing logic. Memory cost of pseudo-TCAM can be reduced further if it is implemented in ASIC with more flexible block memory sizes. Performance of pseudo-TCAM is better than conventional TCAM and FPGA-based TCAM implementation methods [7-9]. Simplicity is a major advantage of pseudo-TCAM over algorithmic methods.

References

[1] P. Gupta and N. McKeown, "Algorithms for packet classification", IEEE Network, Vol. 15, No. 2, pp. 24-32, 2001.

[2] D. E. Taylor, "Survey and taxonomy of packet classification techniques", ACM Computing Surveys, Vol.37, pp. 238-275, 2005.

[3] J. van Lunteren and T. Engbersen, "Fast and Scalable Packet Classification", IEEE JSAC, Vol. 21, No. 4, pp. 560-571, 2003.

[4] D. Pao Y. K. Li, P. Zhou, "Efficient packet classification using TCAMs", Computer Networks, Vol. 50, pp. 3523-3535, 2006.

[5] D. Pao, P. Zhou, B. Liu, X. Zhang, "Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory", IET Computers & Digital Techniques, Vol. 1, pp. 572-580, 2007.

[6] C. R. Meiners, A. X. Liu and E. Torng, "Topological transformation approaches to TCAM-based packet classification", IEEE/ACM Trans. Networking, Vol. 19, pp. 237-250, 2011.

[7] Z. Ullah, M. K. Jaiswal, R. C. C. Cheung, "Z-TCAM: An SRAM-based architecture for TCAM", IEEE Trans. VLSI Systems, Vol. 23, No. 2, pp. 402-406, 2015.

[8] N. Ayyildiz, E. G. Schmidt, H. C. Curan, "S-DIRECT: Scalable and dynamically reconfigurable TCAM architecture for high-speed IP lookup", The Computer Journal, Vol. 58, No. 6, pp. 1443-1455, 2015.

[9] Xilinx, "TCAM search IP for SDNet", www.xilinx.com/support/documentation/ip_documentation/tcam/pg190-tcam.pdf

[10] P. Gupta and N. McKeown, "Classification using hierarchical intelligent cuttings", IEEE Micro, Vol. 20, No. 1, 34-41, 2000.

[11] S. Singh, F. Baboescu, G. Varghese, J. Wang, "Packet classification using multidimensional cutting", ACM SIGCOMM, pp. 213-224, 2003.

[12] F. Baboescu and G. Varghese, "Scalable packet classification", IEEE/ACM Trans. Networking, Vol. 13, pp. 2-14, 2005.

[13] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed corssproducting of field labels", IEEE INFOCOM, pp. 269-280, 2005.

[14] D. Pao, C. Liu, "Parallel tree search: an algorithmic approach for multi-field packet classification", Computer Communications, Vol. 30, pp. 302-314, 2007.

[15] P. C. Wang, "Scalable packet classification with controlled cross-producting", Computer Networks, Vol. 53, 821-834, 2009.

[16] B. Vamanan, G. Voskuilen, T. Vijaykumar, "Efficuts: optimizing packet classification for memory and throughput", ACM SIGCOMM, pp. 207-218, 2010.

[17] A. G. Alagu Priya, H. Lim, "Hierarchical packet classification using a Bloom filter and rule-priority tries", Computer Communications, Vol. 33, pp. 1215-1226, 2010.

[18] H. Lim, S. Lee and E. E. Swartzlander Jr., "A new hierarchical packet classification algorithm", Computer Networks, Vol. 56, pp. 3010-3022, 2012.

[19] W. Jiang, V. K. Prasanna, "Scalable packet classification on FPGA", IEEE Trans. VLSI Systems, Vol. 20, pp. 1668-1680, 2012.

[20] A. Kennedy, X. Wang, "Ultra-high throughput low-power packet classification", IEEE Trans. VLSI Systems, Vol. 22, pp. 286-299, 2014.

[21] D. Pao and Z. Lu, "A multi-pipeline architecture for high-speed packet classification", Computer Communications, Vol. 54, pp. 84-96, 2014.

[22] Y. K. Chang, H. C. Chen, "Fast packet classification using recursive endpoint-cutting and bucket compression on FPGA", The Computer Journal, in press, 2018.

[23] D. E. Taylor, J. S. Turner, "Classbench: a packet classification benchmark", IEEE/ACM Trans. Networking, Vol. 15, pp. 499-511, 2007.