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

Weiwen Yu, Srinivas Sivakumar, and Derek Pao

***Abstract*—A SRAM-based hardware architecture that emulates the behavior of ternary content addressable memory for packet classification is presented. Header fields of the packet are encoded using the prefix inclusion coding method. Encoded rules are mapped to SRAM-based match units using a bit-selection approach. Selected bits of the input key are used as the address to access a rule in the SRAM for comparison. The average memory cost is 26.3 and 18.5 bytes per rule for rulesets with 10K and 100K rules, respectively. The proposed method is implemented on Xilinx UltraScale FPGA. Throughput of the classifier can reach 426 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. Typical 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. Rules are prioritized according to the order in which they appear in the ruleset. 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] for a tutorial on packet classification algorithms.

Ternary content addressable memory (TCAM) is commonly used in commercial routers for IP address lookup and packet classification. Simplicity is a major advantage of TCAM, but the major disadvantages are limited capacity, high cost and power consumption. The 5 header fields of the packet are sent to the TCAM, and the TCAM outputs the smallest address among the matching rules. 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 ruleset with n rules occupies $e \times n$ TCAM entries, and e ranges from 1 to 6.2 for some real-life rulesets [1]. A few encoding schemes [2]–[5] have been proposed to minimize port range expansion and the TCAM word width (much desirable for IPv6 with 128-bit addresses). 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 [6]–[7]. The Z-TCAM [6] method requires 36 BRAMs and 1781 LUTs to implement a 512×36 TCAM array, i.e., using 72 bits of SRAM to implement 1 bit of TCAM. The S-DIRECT [7] method uses 36108 LUTs to implement a 1024×32 TCAM, and it is only applicable to IP address lookup. Xilinx offers an intellectual property core for realizing TCAM on FPGA [8]. 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 [6]–[8] are not cost effective for packet classification.

There are substantive researches on algorithmic approaches for packet classification [9]–[20]. Published methods make use of some combinations of techniques, e.g., decision tree, hierarchical trie, cross-producting, bucket compression, field encoding, ruleset partitioning, pipelined and parallel processing. The best achievable lookup rate is 388 MPPS (Virtex-6) [20], and the best memory performance is about 19.2 bytes per rule [19]. In this letter, we shall present a SRAM-based architecture for packet classification that emulates the operation of TCAM. The hardware architecture is much simpler than existing algorithmic methods, and it offers good performance. The average memory cost of the proposed method is 26.3 bytes per rule for rulesets with 10K rules, and 18.5 bytes per rule for rulesets with 100K rules. The lookup rate of the proposed method is 426 MPPS when implemented using Xilinx UltraScale FPGA (device model xcvu080).

II. Proposed Pseudo-TCAM 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)

Manuscript received July 30, 2018; revised November 18, 2018; accepted January 31, 2019. Date of publication February 6, 2019; date of current version May 23, 2019. The associate editor coordinating the review of this paper and approving it for publication was J. Griffioen. *(Corresponding author: Derek Pao.)*

The authors are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong (e-mail: weiwenyu2-c@my.cityu.edu.hk; ssivakuma2-c@my.cityu.edu.hk; d.pao@cityu.edu.hk).

Digital Object Identifier 10.1109/LNET.2019.2897934

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	**0**01x x1**0**0 xx1x (R_2)
01	**0**001 x1**1**x xx10 (R_1)
10	**1**100 10**0**1 011x (R_4)
11	**1**0xx x0**1**0 xx01 (R_3)

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

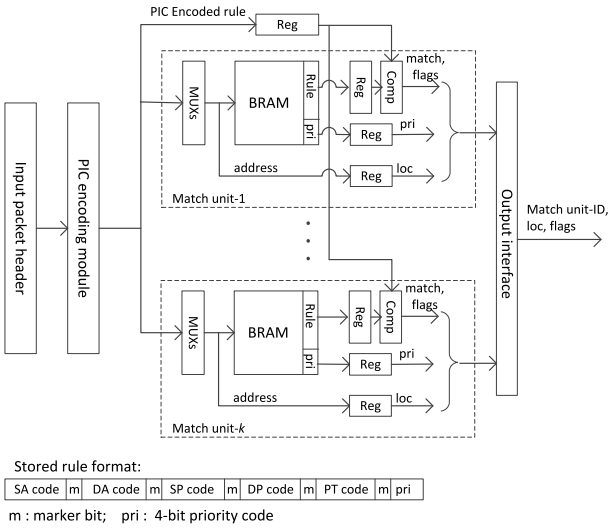

Fig. 3. Organization of the pseudo-TCAM classifier.

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.

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 matching rule is identified by the match unit ID, the address of the matching rule and 2 control flags. The hardware can be modified to output the action of the best matching rule instead of the address information. The priority code and the control flags will be explained later. In our design, we would optimize the number of match units required for a given ruleset.

Ruleset (10K rules)	SA	DA	SP	DP	PT
acl1	2212 (13 bits)	601 (12 bits)	1 (0 bit)	108 (9 bits)	3 (2 bits)
acl2	247 (11 bits)	765 (12 bits)	1 (0 bit)	27 (6 bits)	4 (3 bits)
fw1	2056 (13 bits)	7360 (15 bits)	13 (4 bits)	43 (6 bits)	4 (3 bits)
fw2	6578 (14 bits)	3512 (13 bits)	9 (4 bits)	1 (0 bit)	4 (3 bits)
ipc1	135 (9 bits)	543 (12 bits)	34 (6 bits)	54 (7 bits)	6 (3 bits)
ipc2	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 6 rulesets.

The 5 header fields are encoded using the prefix inclusion code (PIC) method [3]. The PIC encoding module in Fig. 3 is a 4-stage pipeline similar to the phase-1 processing module of [19]. The prefix-form PIC codewords preserve the inclusion property of the original data values, and an arbitrary range can be represented by one codeword. 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.

Real-life rulesets are not available in the public domain. Twelve synthetic rulesets generated using Classbench [21] are used in this letter. Each ruleset contains about 10K rules. The number of distinct values in each data field, and the PIC code lengths for 6 synthetic rulesets are shown in Fig. 4. The length of an encoded rule is between 32 and 45 bits.

Conventional approach to represent a rule is to use 2 bit-vectors, i.e., the data-vector and mask-vector. The data-vector stores the data bits, and the mask-vector defines which bits are wildcard bits. This approach uses 2*L* bits storage to represent an *L*-bit rule. Assume the maximum PIC code lengths are pre-determined 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.

To determine the priority codes, 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 $R_j \rightarrow R_i$ is created. The priority code of the root is 0, and priority code of $R_i > R_j$ if there is a directed edge $R_j \rightarrow R_j$.

The marker bit of the PT field has a different meaning. It is not uncommon to have a pair of rules (R_v, R_w) in the ruleset with identical values in the SA, DA, SP and DP fields, and R_v

has a non-wildcard PT value and R_w has the PT field equal to wildcard. The pair of rules (R_v, R_w) can be combined into 1 entry in the match unit. If the PT code of a match entry is 0, it means that the stored rule has the PT field equal to wildcard. If the PT code is non-zero and the marker bit is set, then it means that the match entry stores to a pair of rules (R_v, R_w). The comparator circuit uses a control flag to indicate if the input key matches R_v or R_w.

Minor refinements to the priority code assignment and the comparator circuit of the PT field are required to support the merging of (R_v, R_w) in 1 match entry. In the dependency graph, we may have a directed edge $R_w \rightarrow R_v$ or indirect edges $R_w \rightarrow R_o \rightarrow R_v$, where R_o is some other rule. Let the priority code of R_w be p_w. The priority code p_v of R_v is set to $p_w + 2$ in the above 2 cases. When (R_v, R_w) are combined into 1 entry, the priority code of R_v is stored in the match entry. If the input key matches R_v, priority code p_v is output. If the input key does not match R_v but matches R_w, the comparator circuit adjusts the output priority code to $p_v - 2$. If there are 2 or more rules in between R_w and R_v in the dependency graph and $p_v \neq p_w + 2$, then (R_v, R_w) cannot be combined into 1 match entry. However, this situation is not observed in the 12 rulesets. The largest priority value for the 12 rulesets based on the above priority assignment scheme is equal to 10. Hence, 4-bit priority code is used in the hardware design.

A 36-Kbit BRAM in the FPGA has 2 independent 36-bit I/O ports. An encoded rule requires 41 to 54 bits of storage. To improve the memory efficiency, 3 BRAMs (configured as 1024×36 bits) are connected in parallel to form 108-bit words such that each memory word (bucket) can store 2 rules. Two comparator circuits are used to compare the 2 rules against the input key. 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 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 some rulesets have short SA/DA prefixes, and the SP and DP fields are equal to wildcard. These rules are located near the bottom of the list with low priority (priority code ≤ 7). We handle this group of rules with a special match unit. Each word stores 32 3-bit priority codes. The special match unit has an address space of 2^{15}. We define the SA code length threshold L_{SA} = (15 – PT code length) / 2, and DA code length threshold L_{DA} = 15 – PT code length – L_{SA}. Rules with SA and DA code lengths less than or equal to the given threshold, SP and DP equal to wildcard, and priority code ≤ 7 are mapped to the special match unit using the leaf pushing approach. The hardware uses the most significant L_{SA}/L_{DA} bits of the SA/DA codes and the PT code to retrieve the priority code of the matching rule in the special group.

The remaining rules are mapped to regular match units. The initial mapping phase makes use of the numbers of distinct values in a 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 is 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 2-level bit-selection strategy.

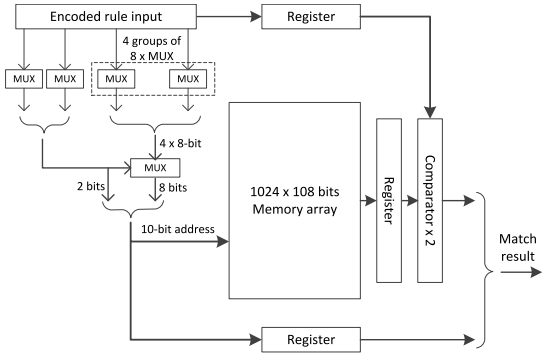

Fig. 5. Match unit with 2-level bit-selection.

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 *i*th bit equal to 0, and $rc_1[i]$ be the number of rules with the *i*th bit equal to 1. The balance factor of the *i*th bit *BF*[*i*] is equal to $\min(rc_0[i], rc_1[i])/n$, 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 bit b_0. The algorithm then computes BF_0 and BF_1 for the 2 sub-lists, respectively. With 2 sub-lists, *BF*[*i*] is equal to $BF_0[i] + BF_1[i]$. Index of the second selected bit is b_1 where $BF[b_1]$ is the largest value in *BF*.

Four disjoint 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.

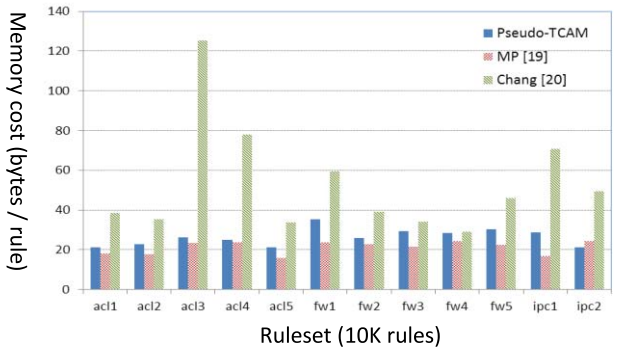

Fig. 6. Comparison of memory cost with 2 other methods.

IV. Performance Evaluation and Discussion

For proof of concept, pseudo-TCAM is implemented on an UltraScale FPGA. The processing logic consumes about 5200 LUTs, and the FPGA can operate at 213 MHz. Two lookups can be performed in parallel, and the system throughput can reach 426 MPPS. The method requires 7 to 18 regular match units for rulesets with 10K rules. A comparison of memory cost with 2 state-of-the-art algorithmic methods [19], [20] is shown in Fig. 6. The average memory cost of pseudo-TCAM is 26.3 bytes per rule. It is 37% higher than MP [19], but it is only 50% of Chang's method [20]. The cost of the processing logic of pseudo-TCAM is only 36% of MP. Hence, the overall hardware cost of pseudo-TCAM is comparable to MP. The speed of pseudo-TCAM is 6.5% faster than MP (also implemented on UltraScale) 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. We also evaluate the memory cost of pseudo-TCAM for rulesets with 100K rules. The length of an encoded rule is between 40 and 54 bits, and the word width of the match unit is increased to 144 bits (i.e., 4 BRAMs per match unit). When the size of the ruleset is increased, the numbers of distinct SA and DA prefixes increase accordingly. This allows the mapping algorithm to fill majority of the match units with good utilization (e.g., $> 90\%$). The method requires 51 to 75 regular match units. The average memory cost is reduced to 18.5 bytes per rule, which is better than Chang's method [20] that requires 63.5 bytes per rule for rulesets with 100K rules.

Supporting dynamic updates to the ruleset is an important issue. The hardware organization (e.g., number of pipelines and length of the pipelines) of many algorithmic methods [17], [18], [20] depends on the distribution of field values in the ruleset. It is difficult to decide on the hardware design parameters without a priori knowledge of the ruleset properties. Moreover, these methods require hardware reconfiguration if the ruleset properties change significantly over time. The pseudo-TCAM method has simple and uniform hardware organization for all rulesets. It can support dynamic updates, and the classifier needs not be suspended when the update process is in progress. The PIC encoding method with guard ranges [4] can be used to reduce the number of codewords affected by incremental updates. A significant proportion of match units are not 100% utilized, and a few match units have utilization below 10%. If the new rules cannot fit into the current match units, the system selects some match units with smaller number of rules and re-compute the mapping of the affected rules. Rules of the selected match units are relocated to spare units based on new sets of selected bits. When all rules in a match unit are vacated, the match unit serves as a spare unit for rules relocation. The design of the software to minimize the update cost will be a topic for future research.

References

[1] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surveys*, vol. 37, no. 3, pp. 238–275, 2005.
[2] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 560–571, May 2003.
[3] D. Pao, Y. K. Li, and P. Zhou, "Efficient packet classification using TCAMs," *Comput. Netw.*, vol. 50, no. 18, pp. 3523–3535, 2006.
[4] D. Pao, P. Zhou, B. Liu, and X. Zhang, "Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory," *IET Comput. Digit. Techn.*, vol. 1, no. 5, pp. 572–580, Sep. 2007.
[5] C. R. Meiners, A. X. Liu, and E. Torng, "Topological transformation approaches to TCAM-based packet classification," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 237–250, Feb. 2011.
[6] Z. Ullah, M. K. Jaiswal, and R. C. C. Cheung, "Z-TCAM: An SRAM-based architecture for TCAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 2, pp. 402–406, Feb. 2015.
[7] N. Ayyildiz, E. G. Schmidt, and H. C. Curan, "S-DIRECT: Scalable and dynamically reconfigurable TCAM architecture for high-speed IP lookup," *Comput. J.*, vol. 58, no. 6, pp. 1443–1455, Jun. 2015.
[8] Xilinx. *TCAM Search IP for SDNet*. Accessed: Nov. 2017. [Online]. Available: www.xilinx.com/support/documentation/ip_documentation/tcam/pg190-tcam.pdf
[9] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, Jan./Feb. 2000.
[10] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. ACM SIGCOMM*, Karlsruhe, Germany, 2003, pp. 213–224.
[11] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM Trans. Netw.*, vol. 13, no. 1, pp. 2–14, Feb. 2005.
[12] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *Proc. IEEE INFOCOM*, Miami, FL, USA, 2005, pp. 269–280.
[13] D. Pao and C. Liu, "Parallel tree search: An algorithmic approach for multi-field packet classification," *Comput. Commun.*, vol. 30, no. 2, pp. 302–314, 2007.
[14] P.-C. Wang, "Scalable packet classification with controlled cross-producting," *Comput. Netw.*, vol. 53, no. 6, pp. 821–834, 2009.
[15] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "EffiCuts: Optimizing packet classification for memory and throughput," in *Proc. ACM SIGCOMM*, 2010, pp. 207–218.
[16] H. Lim, S. Lee, and E. E. Swartzlander, Jr., "A new hierarchical packet classification algorithm," *Comput. Netw.*, vol. 56, no. 13, pp. 3010–3022, 2012.
[17] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.
[18] A. Kennedy and X. Wang, "Ultra-high throughput low-power packet classification," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 2, pp. 286–299, Feb. 2014.
[19] D. Pao and Z. Lu, "A multi-pipeline architecture for high-speed packet classification," *Comput. Commun.*, vol. 54, pp. 84–96, Dec. 2014.
[20] Y.-K. Chang and H.-C. Chen, "Fast packet classification using recursive endpoint-cutting and bucket compression on FPGA," *Comput. J.*, vol. 62, no. 2, pp. 198–214, Feb. 2019.
[21] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, Jun. 2007.