Bundle-Updatable SRAM-Based TCAM Design for OpenFlow-Compliant Packet Processor

Ding-Yuan Lee, Ching-Che Wang, and An-Yeu Wu

Abstract—Static random-access memory (SRAM)-based ternary content-addressable memories (TCAMs) emulate TCAM functions with high throughput at low cost. However, the implementation of SRAM-based TCAM as a rule table in network switches tends to prolong updating latency, which can cause a false packet routing. This brief proposes a novel low-latency bundle-updatable TCAM (BU-TCAM) scheme that uses binary tree-based prefix encoding (BPE) to support single- and multiple-rule updating in a software-defined networking/OpenFlow network. The proposed encoding method transforms the original ternary rule data into a binary code word and determines the range of overlap in SRAM addresses to facilitate updating. This greatly decreases latency in cases where multiple rules are required to update on an SRAM-based TCAM. We implemented an emulated 64×32-bit TCAM of the proposed design on a Xilinx ZC-706 field-programmable gate array. The proposed scheme reduced updating latency by 79.6%, compared with a conventional updating structure, which had only 9.8% and 23% increases in LUTs and registers overhead, respectively.

Index Terms—Field-programmable gate array (FPGA), prefix encoder, rule update, static random-access memory (SRAM)-based ternary content-addressable memory (TCAM).

I. INTRODUCTION

TERNARY content-addressable memory (TCAM) allows for rapid parallel content search and the ability to store three states of data: (0), (1), and Don’t care (X). TCAM is widely used in the packet processor engines of novel network architectures, including software-defined networking (SDN) and OpenFlow [1] [2]. However, the three-state storage and parallel searching of TCAM requires a large area and high-power consumption. Thus, static random-access memory (SRAM)-based TCAM has recently been developed to increase area and power efficiency [3]–[5]. It enables content search in a single cycle at speeds as fast as typical TCAM and a considerably lower cost.

OpenFlow is a novel network protocol for future 5G/SDN, which requires frequent rule updating in the network switches [6], [7]. An updating protocol, referred to as “bundle updating” or “group updating,” is defined to ensure consistency in the rules within the network [8]. In bundle updating, the network controller continuously inserts rules and decides whether to update all of the rules or drop them, after all of the rules have been sent. A delayed update can cause false routing in the network [7], [9]. Furthermore, since an SRAM-based TCAM has to access all the addresses on SRAM for a single-rule update, the processing latency linearly increases in this case. Fig. 1(a) illustrates an example of the conventional updating method in an SRAM-based TCAM. A variety of SRAM-based TCAM memory architectures have been developed [3]–[5], and a fast content updating algorithm has been proposed to decrease latency in the updating of SRAM-based TCAM cells [10]. However, bundle updating in OpenFlow is not supported by the current SRAM-based TCAM cells. In this brief, we propose a binary tree-based prefix encoder (BPE) for low-latency updating in an OpenFlow network environment while retaining the area efficiency of conventional SRAM-based TCAM. The proposed BPE transfers a ternary rule into a binary tree-based code word, which can be decoded and mapped directly within the writing address range of the SRAM. As shown in Fig. 1(b), when being applied to bundle updating, BPE can identify the overlap and ignore redundant ranges on bit vectors to minimize latency and set an upper bound in updating operations.

This brief is organized as follows. In Section II, we introduce the fundamentals of SRAM-based TCAM architectures and the updating requirements of OpenFlow. Section III describes the proposed architecture and encoding algorithm. Section IV presents the evaluation and analysis of the latency theoretically, experiments with network traffic, and resource utilization on the Xilinx ZC-706 field-programmable gate array (FPGA). The conclusions are drawn in Section V.

II. PREVIOUS WORKS AND PROBLEM FORMULATION

A. SRAM-Based TCAM Architecture

Typical $N \times L$ TCAM performs parallel comparisons between input data and $N$ rules and outputs an $N \times 1$ matching vector. The positions of 1s in the matching vector refer to the corresponding matched rule entries in the TCAM. SRAM-based TCAM expands all combinations of an $L$-bit input data into $2^L$ addresses and stores the matching results (i.e., $N \times 1$ vector) in each address. This means that a TCAM with size $N \times L$ can be mapped to an SRAM module with size $2^L \times N$. The relation of size between a conventional TCAM and an SRAM-based TCAM can be formulated as follows:

$$\text{TCAM}(N \times L) \leftrightarrow \text{SRAM}(2^L \times N).$$

Fig. 1. (a) Conventional address-by-address and rule-by-rule updating method of an SRAM-based TCAM. (b) Proposed bundle-updating structure with address relation judgment using BPE.
When performing rule insertion on a TCAM, new entries can be written within a few cycles using the exact address. Conversely, inserting a rule with entry \( n \), \( n \in \mathbb{N} \), in an SRAM-based TCAM requires updating the \( n \)th bit of all stored bit vectors. Thus, in general, a single-rule insertion generally requires \( 2^L \) cycles to modify the content stored in an SRAM cell [3], [11].

### B. Updating Requirements in OpenFlow and Related Works

SDN is an emerging network architecture characterized by frequent updating and large rule tables. These days, a rule entry can contain up to 773 bits [12]. When the rules are presented in ternary forms, the mask bits are always continuous and start from the LSB. The latency of rule updating of network switches can greatly affect the performance of dynamic routing [7]. The OpenFlow protocol defines bundle updating to synchronize packet forwarding paths in the network [8]. In bundle updating, rules are sent to the corresponding switches within a given period, and it is determined in the last cycle whether the rules should be installed or dropped [13]. For example, the network controller sends rule \( R_1, R_2, \ldots, R_m \) to a switch at time \( T_1, T_2, \ldots, T_m \). After sending \( m \) rules, a decision to cancel the update prompts a discard message at \( T_{m+1} \). Otherwise, the switch performs the regular updating operation. The large updating latency (\( m \times 2^L \) cycles for \( m \) rules) on an SRAM-based TCAM can seriously affect the rule consistency between switches, further resulting in routing faults.

The state-of-the-art scheme in [10] includes a fast updating algorithm, in which rules are inserted within a specified number of cycles, depending on the length of ternary bits \( w \). This scheme consumes \( 2^w \) clock cycles for the updating of a single TCAM word. However, the counters require an additional \( 2^w \) clock cycles to perform preprocessing. In cases where multiple-rule updating is required, processing latency increases linearly. In this brief, we propose a novel structure that enables the low-latency updating of single and group rules for OpenFlow networks.

## III. PROPOSED ARCHITECTURE

### A. Architecture of Proposed SRAM-Based TCAM

A proposed bundle-updatable SRAM-based TCAM (BU-TCAM) module comprises three submodules: an updating controller (UC) with BPE encoder and code-word buffer, an operation module (OM), and a dual-port SRAM. Fig. 2(a) illustrates the design of the BU-TCAM in detail. Since network rules require a large rule table [12], we refer to hybrid partitioning [14], in which a large table is divided into vertical and horizontal partitions to reduce the dimensionality of bit vectors. Fig. 2(b) illustrates the overall architecture of the proposed BU-TCAM. Search data or updating/insertion data are separated and sent by the string divider to the corresponding BU-TCAM modules. In data searching, the result of each module in a given row is ANDed prior to the priority encoder.

During updating, an \( i \)-bit ternary rule input is presented with two \( i \)-bits of data, an input word \( D \), and a mask word \( M \). We developed a binary-tree prefix encoding (BPE) scheme to meet the updating requirements of OpenFlow. In BPE, updating data are encoded to reduce the storage in the buffer. The UC also compares the range relation based on the parent/child relation via a binary tree with the aim of reducing the number of updating cycles by disregarding overlapping addresses during bundle updating. Finally, a dual-port SRAM can simultaneously read and write data, realizing the pipeline structure for the updating process. The proposed updating flow of BU-TCAM is shown in Fig. 3 and listed in the following.

1) When inserting an updating rule, the UC module sends the data to a corresponding group- or single-update buffer.

2) In the case of single updating, the UC encodes the data and sends the code word to the OM.

3) In cases where the network controller requires group updating, the UC module receives the data, performs encoding, and compares the range continuously.

4) Following the completion of group updating, the network controller determines whether the UC module should send the code words to the OM for the following update event or clear the buffer.

With the proposed design for OpenFlow protocol, the BU-TCAM is allowed to update the network rules within a minimum number of cycles.

### B. Binary Tree-Based Prefix Encoder

We also developed a binary tree-based prefix encoder (BPE) to save the storage size in the buffer and reduce latency during bundle...
First, we set a 1-bit value called a child index (CI) to indicate the using the characteristics of parent/child relations in the binary tree.

### C. Bundle Updating in BPE

Updating in OpenFlow. The BPE encoder is set within the UC module, and the decoder is set within the OM module of BU-TCAM. In implementing BPE, we utilize the characteristics of prefix data and a binary tree to encode the original ternary data. Prefix data in a network contain continuous ternary worlds; therefore, is mapped to the bit vector are presented as a range. Table I presents an example involving four rules. We can observe that the positions of 1s in the bit vector present a distribution that matches the position of the tree node in the corresponding level. When a binary tree is started from level 0, a rule with a j-bit binary prefix can be mapped to level j of the tree.

Fig. 4(a) presents an example illustrating the relationship among a ternary rule, a bit vector, and a binary tree. Rule $R_0$ has 1-bit binary word and 2-bit ternary words; therefore, it is mapped to level one of the tree. The prefix of $R_0$ is 0; therefore, it is mapped to the left node in the level of the tree. The distribution of 1s in the bit vector can be mapped to the corresponding level and the node of a binary tree. Ternary rules $R_1$ and $R_3$ can be mapped similarly. In this way, a k-bit ternary word can be transferred to a $k + 1$-level binary tree. The mapping and encoding method is shown in Fig. 4(b). Table II lists examples of the encoding in detail.

The pseudocode of BPE is shown in Algorithm 1. $D$ and $M$ are the input word and the mask word; $D[0]$ and $M[0]$ represent the most significant bit (MSB) in each word. The steps are summarized as follows. First, we initialize the code word $Y$ to 0 (the length of $Y$ is the length of the input data +1). Under ternary conditions, we set the first bit to 0 and calculate the size of the ternary bits of the input, $SumM$. We then assign a “1” to the corresponding bit position of code word $Y[SumM]$ (the $SumM$th bit from MSB). Next, we assign prefix data to the remaining code word after $Y[SumM + 1]$. If the data are the binary rule, we shall only add “1” before the original data, and the binary rule can be perfectly encoded and inserted exactly into the last level of the tree. Finally, a k-bit ternary rule can be encoded into a $k + 1$-bit binary code word. The fact that BPE uses unique decodable words allows the OM to perform the updating without performance degradation.

### IV. PERFORMANCE EVALUATION

#### A. Analysis of Updating Latency

We seek to reduce the redundant latency during bundle updating using the characteristics of parent/child relations in the binary tree. First, we set a 1-bit value called a child index (CI) to indicate the range of overlap during group updating. Following the insertion of every rule, we check the relation and perform a parallel comparison with all of the rules in the buffer. In this comparison, if the updating range of one rule covers the other, the CI of the rule in the child position is set to 1. After receiving all of the updating rules from the network controller, the UC simultaneously completes BPE encoding and CI labeling. While the range of the parent is updating, the rule with the child relation is inserted. For instance, if $R_0$ and $R_3$ are required for updating in a given group, then the CI of $R_3$ is set to 1 during the range comparison. When the UC sends the update data to the OM, the range of $R_3$ is skipped.

We can access the parent/child relation of the BPE code word based on the numerical characteristics of a binary tree, where each leaf node on the left is double the value of its parent. The distance between the first “1” and the MSB in a BPE code word indicates the number of ternary bits in the original data, which can be derived from the level of the node in the tree. For fast parent/child relation processing, we add $\log_2 k$-bit into the BPE code word to store the number of ternary bits of the code word. We then calculate the difference in level between the two code words $DeltaL$. Next, we only have to right shift the higher level code-word $DeltaL$ bits and compare it with another code word, making it possible to determine whether the range overlaps during updating. $R_0$ and $R_3$ in Fig. 4 and Table II are an example of the child relation. The detailed pseudocode is shown in Algorithm 2.
Algorithm 2 Child Relation Judgment

1: Given
2: Codeword C1[i], C2[j] // input data
3: Preprocessing
4: \( L_1, L_2 = \text{the level of } C_1, C_2 \)
5: \( \Delta L = L_1-L_2 \) // number of ternary X
6: Processing
7: if \( \Delta L > 0 \)
8: \( \text{Temp} = L_1+\Delta L \)
9: if \( (C_2==\text{Temp}) \) Child_index[C1] = 1
10: else Child Child_index = 0
11: else
12: \( \text{Temp} = L_2+\Delta L \)
13: if \( (C_1==\text{Temp}) \) Child_index[C2] = 1
14: else Child_index = 0

### Table III

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single rule</td>
<td>( O(2^L) )</td>
<td>( O(2^{w+1}) )</td>
<td>( O(2^w) )</td>
</tr>
<tr>
<td>Group rules</td>
<td>( O(m \times 2^L) )</td>
<td>( O(m \times 2^{w+1}) )</td>
<td>( O(2^Lw) )</td>
</tr>
<tr>
<td>OpenFlow discard support</td>
<td>×</td>
<td>×</td>
<td>0</td>
</tr>
</tbody>
</table>

\( w \) is the average value of \( w \)

C denotes the worst case

the SRAM cell can store \( 2^L \) addresses with \( N \)-bits data to form a \( N \times L \) general TCAM cell. In the single-rule updating scenario, the conventional scheme requires \( 2^L \) cycles to complete rule updating for every case [3], [11]. The scheme in [10] requires an additional \( 2^w \) cycles in their address generator stage for preprocessing and checking the boundary of output address and \( 2^w \) cycles for rule updating. The \( w \) represents the number of ternary bits in the rule. This results in the consumption of \( 2^{w+1} \) cycles. In the proposed work, the encoding and decoding steps can be completed in a short period, and the updating latency is only proportional to the number of ternary bits \( 2^w \).

Previous schemes do not take into account situations involving group updating; therefore, they impose \( m \) times the latency of single-rule updating for \( m \) rules. The proposed BPE allows us to complete updating within \( 2^L \) cycles, which is the upper bound for bundle updating. Table III presents a summary of the updating latency analysis.

### B. Synthesis Results: Single-Rule Update

We compare the updating latency of the prior art designs on the Verilog-HDL and analyze the updating latency in the case of various length of ternary bits in the updating rule. We implemented the related works and proposed work using \( 64 \times 8 \) bits emulated SRAM-based TCAMs. The number of ternary bits is set between 1 and 5.

Fig. 5 presents the results of updating latency in the experiments. The experiments show the consistent results with our analysis in Section IV-A. The updating latency of the conventional method is a constant [3], [11], which only depends on the size of the emulated TCAM. Compared with the state-of-the-art update scheme [10], the proposed design reduces 10%–30% of the updating latency when \( m \) is small due to the constant latency of the proposed encoding stage. When \( m \) is larger, we can earn up to 50% of the updating latency. Since the results show a low updating latency, we can further compare the performance under an SDN/OpenFlow environment.

### C. Bundle Update Analysis With Network Rules

ClassBench [15] is the most widely used network simulator and can generate routing rules under networking environments. We implemented the related works and proposed work using \( 64 \times 32 \) bits emulated SRAM-based TCAMs. We generated 1000 networking rules to simulate TCAM cells in single- and group-updating scenarios. Referring to the related research [16], we set the number of group update rules randomly between 2 and 8 with uniform distribution and implemented eight BPE code-word buffers with registers in the UC to support the bundle updates.

Fig. 6 presents the average results of updating latency in the experiments. Compared to the conventional method, the proposed BPE reduced the latency by 80.3% in single-rule updating and 79.6% in group updating. Although the state-of-the-art update scheme [10] consumes extra latency on counters, once the rule is not contained with all ternary bits, it can still reduce the latency by updating the 1s in the bit vectors only. The proposed CI labeling consumes a few cycles during the encoding stage, which can be ignored during the continuous rules input with a simple pipeline structure.

In the results, we further reduced the single- or group-updating latency by 50.4% and 74%, respectively, compared with the state-of-the-art updating scheme using the BPE encoding. By addressing more overlaps in the updating, the processing latency can be further reduced, regardless of the size of \( m \). These results demonstrate the efficiency of the proposed BPE in reducing the latency associated with updating bundles in an SDN/OpenFlow environment.

### D. Implementation Results on FPGA

Finally, we implemented the proposed works and related works on a Xilinx ZC-706 FPGA. The size of the BU-TCAM block was \( 64 \times 32 \) bits using two 36k block RAM (built-in dual-port SRAM) cells. The clock cycle was 100 MHz. Comparisons focused on
TABLE IV

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice LUTs</td>
<td>2,600</td>
<td>2,152</td>
<td>2,856</td>
</tr>
<tr>
<td>Slice Registers</td>
<td>156</td>
<td>144</td>
<td>192</td>
</tr>
<tr>
<td>BRAM (36K)</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Power (W)</td>
<td>0.293</td>
<td>0.253</td>
<td>0.313</td>
</tr>
</tbody>
</table>

resource utilization; the results of which are listed in Table IV. Due to the encoder and buffer design to store the code word for bundle updating, the proposed architecture had a tradeoff of an extra 9% and 23% for the registers and LUTs, respectively, compared with past works, while providing more comprehensive updating functions.

V. CONCLUSION

In this brief, we present a low-latency bundle-updatable SRAM-based TCAM (BU-TCAM) for OpenFlow-compliant packet processing. The proposed BPE structure minimizes the number of updating cycles for SRAM cells. Compared with the conventional updating architectures, the proposed BU-TCAM reduced the updating latency by 79.6% in bundle-updating scenarios. The proposed BU-TCAM design is ideally suited to emerging SDN/OpenFlow packet processing, requiring fast rule updates.

REFERENCES


