FastUp: Compute a Better TCAM Update Scheme in Less Time for SDN Switches

Ying Wan*, Haoyu Song†, Hao Che‡, Yang Xu§, Yi Wang††, Chuwen Zhang*, Zhijun Wang‡, Tian Pan**, Hao Li¶, Hong Jiang‡, Chengchen Hu¶‖, Zhikang Chen* Bin Liu*
* Tsinghua University, China † Futurewei Technologies, USA ‡ University of Texas at Arlington, USA
§ Fudan University, China †† Southern University of Science and Technology, China ‖ Xilinx, Singapore
¶ Xi'an Jiaotong University, China ** Beijing University of Posts and Telecommunications, China

***Abstract*—While widely used for flow tables in SDN switches, TCAM faces challenges for rule updates. Both the computation time and interrupt time need to be short. We propose *FastUp*, a new TCAM update algorithm, which improves the previous dynamic programming-based algorithms. Evaluations show that *FastUp* shortens the computation time by 40$\sim$100$\times$ and the interrupt time by 1.2$\sim$2.5$\times$. In addition, we are the first to prove the NP-hardness of the optimal TCAM update problem, and provide a practical method to evaluate an algorithm's degree of optimality. Experiments show that *FastUp*'s optimality reaches 90%.**

I. Introduction

Software-Defined Networking (SDN) uses flow tables to enforce flexible policies [1]. While using TCAM for flow tables is great for lookup speed and rule flexibility, it suffers from high update cost. Since the rules in TCAM need to be arranged in order of priority, inserting a new rule to TCAM may require relocating existing rules. Unfortunately, lookups and updates in TCAM usually share the same interface, forcing the lookup process to pause while TCAM is being updated. Therefore, it is important to optimize the TCAM update process by shortening the TCAM interrupt time. Meanwhile, to achieve the low update delay required by many applications, we also need to keep the update computation time short.

Most existing works focus on reducing the required number of rule moves per new rule insertion (*i.e.*, interrupt time optimization). *RuleTris* [2] and *Pot*$_2$ [3] achieve the best-in-class interrupt time performance. However, their average computation time reaches hundreds of milliseconds for a 4K-entry TCAM, falling short of application requirements [4]. Although the follow-up works *Pot*$_1$ [3] and *FastRule* [4] shorten the computation time of *Pot*$_2$ and *RuleTris*, respectively, the gains are at the cost of prolonged interrupt time.

In this paper, we propose *FastUp*, a new TCAM update algorithm, which shortens the interrupt time and the computation time simultaneously. Instead of using the Dynamic Programming Algorithm (DPA) adopted by *RuleTris* and *Pot*$_2$, *FastUp* uses a Sequential Stack-based Algorithm (SSA), which reduces both the time complexity and the space complexity.

To the best of our knowledge, we are the first to prove that the problem of Optimal TCAM Update (OTU) (*i.e.*, computing the minimum number of rule moves for an update) is NP-hard. Alternatively, we develop a Branch-and-Bound Algorithm (BBA) to evaluate the degree of optimality of a practical optimizer.

This work is supported by NSFC (61432009, 68172213, 61702049, 61872420), "FANet: PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (No. LZC0019)", Guangdong Key Research and Development Program (No.2019B121204009), and Guangdong Basic and Applied Basic Research Foundation (2019B1515120031). Corresponding Author: Bin Liu (lmyujie@gmail.com).

II. Algorithm Design

A. Problem Statement

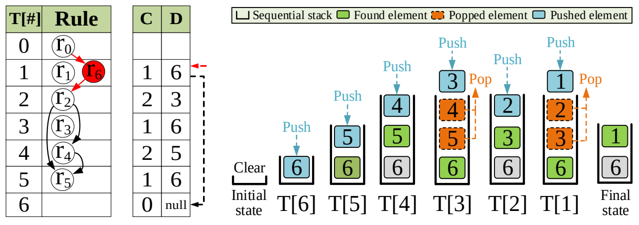
Rules in a flow table may overlap, so a packet may match multiple rules. Among these rules, the one with the highest priority is taken. Since TCAM only returns the first match, the overlapping rules must be placed in TCAM in decreasing priority order. For two overlapping rules, the one with a higher (lower) priority is called the parent (child) of the one with a lower (higher) priority. If we represent each rule as a node and add a directed edge from each parent to each of its children, the TCAM update problem can be formulated as placing a new node in topological order in a Directed Acyclic Graph (DAG). Fig. 1(a) shows such an example, where r_6 is the update rule.

B. Principle of DPA

A new rule r_u can be placed in any TCAM entry that is below r_u's youngest parent (*i.e.*, the parent with the highest address) and above r_u's oldest child (*i.e.*, the child with the lowest address). These entries are all candidates. However, placing r_u in an occupied candidate entry causes a chain effect. The original rule has to be kicked out and moved downward to another entry, which can lead to other rule moves recursively until a rule is finally settled in an empty entry. Therefore, the basic strategy is to evaluate the moving cost C[$\cdot$] (*i.e.*, the number of rule moves) for inserting r_u in each candidate entry. The best moving scheme is the one with the smallest moving cost.

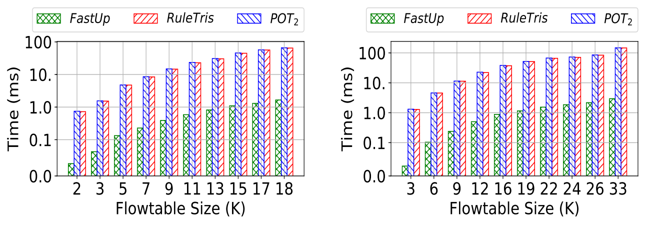
For the i-th TCAM entry T[i], C[i] is equal to the smallest moving cost among all its candidates plus 1, as formulated in Equation 1, where oldchd(i).addr indicates the entry address of the oldest child of the rule in T[i].

$$\mathrm{C}[i] = \min_{j \in (i,\, \mathrm{oldchd}(i).\mathrm{addr}]} \{\mathrm{C}[j]\} + 1 \qquad (1)$$



(a) TCAM. (b) *DPA*. (c) *FastUp*.

Fig. 1. Comparison between two cost-based approaches to insert a new rule.



(a) ACL (b) FW

Fig. 2. Compute time comparison on different Flow tables.

Meanwhile, D[·] is used to track moving sequence. D[i] = j' records the best-candidate T[j'] that contributes to the smallest C[i], which means T[j'] is the target entry for the original rule in T[i]. After the moving cost computation, the moving sequence can be inferred by accessing D[·] recursively.

The cost calculation can be tackled by *DPA* starting upwards from the bottom entry until the moving cost of all candidates of r_u is calculated, as shown in Fig. 1(b). *DPA* needs to process $O(m)$ entries, and for each entry, finding its best-candidate needs $O(m)$ comparisons. Therefore, the time complexity of *DPA* is $O(m^2)$. Due to the introduction of C[·] and D[·], *DPA*'s space complexity is $O(m)$.

C. *Principle of FastUp*

FastUp's approach is more efficient. Fig. 1(c) shows how *FastUp* inserts r_6 to TCAM with the help of an array-based sequential stack $\mathbb{S}$. $\mathbb{S}$[0] indicates the stack bottom. In the beginning, $\mathbb{S}$ is empty. For the empty TCAM entry (*e.g.*, T[6]), *FastUp* directly pushes its address into $\mathbb{S}$. For the next non-empty entry (*e.g.*, T[5]), *FastUp* finds the first candidate of the rule in the entry in $\mathbb{S}$. Then, *FastUp* pops all the elements above the found element (*e.g.*, $\mathbb{S}$[0]) and pushes the current entry address (*e.g.*, "5") into $\mathbb{S}$. After processing each entry upwards from bottom entry (*e.g.*, T[6]) in this way until all candidate entries of r_u are processed (*e.g.*, T[1]), *FastUp* finds the first candidate of r_u in $\mathbb{S}$ (*e.g.*, $\mathbb{S}$[1] is found for r_6). The moving sequence to insert r_6 is exactly recorded in $\mathbb{S}$ (*e.g.*, $\mathbb{S}$[0:1] for inserting r_6). That is, $r_6 \to$ T[$\mathbb{S}$[1]] $\to$ T[$\mathbb{S}$[0]].

The result of *FastUp* is identical to that of *DPA*. However, $\mathbb{S}$ is in strict descending order and has at most h elements, where h is the number of unique priorities. A binary search on it takes $O(\log h)$ time. Meanwhile, the element popping is done by simply resetting the size of $\mathbb{S}$, which is an $O(1)$ operation. Overall, *FastUp*'s time complexity is $O(m \log h)$ and its space complexity is $O(h)$.

D. *NP-hardness of OTU*

The solution given by *FastUp* and *DPA* only allows moving rules downward. If rules can move bidirectionally, a better solution is possible. We have proved that the OTU problem is NP-hard. For a special case, the OTU problem can be formulated as follows: When inserting a new rule to TCAM, among all topological sequences of the rule set including the new rule, find the one with the minimum number of rules that have changed their positions. Finding such a topological sequence is NP-hard, which can be deduced from a proved NP-hard problem, Colored Token Swapping on Undirected Complete Graph. To evaluate the degree of optimality of a practical optimizer, we develop a Branch-and-Bound Algorithm (BBA) which can efficiently acquire OTU for up to 1,000 rules.

TABLE I
INTERRUPT TIME COMPARISON ON ACL FLOW TABLES

Size(k)	**Average time** (ms)			**Maximal time** (ms)		
	FastUp	*Pot*$_2$,*RuleTris*	*Pot*$_1$	*FastUp*	*Pot*$_2$,*RuleTris*	*Pot*$_1$
3.4	0.72	0.86	3.16	2.4	3.6	10.8
7.4	0.62	0.65	2.31	1.2	1.8	6.6
11.4	0.64	0.78	4.57	3	4.2	10.8
15.4	0.70	0.95	4.78	2.4	6	18
18.6	0.65	0.85	4.23	2.4	4.8	11.4

III. PERFORMANCE EVALUATION

A. *Experimental Setup*

We compare *FastUp* with *RuleTris* and *Pot*$_2$, because they achieve the shortest up-to-date interrupt time. We also include *Pot*$_1$ because it represents the works that shorten the computation time by relaxing the interrupt time. They are implemented with C++ and tested by extending the firmware on ONetSwitch [5]. Two types of flow tables, Access Control List (ACL) and Firewall (FW), are generated by ClassBench [6].

B. *Experimental Results*

Fig. 2 (a) and (b) show the computation time per rule insertion for different table sizes. *FastUp* is two orders of magnitude better than *RuleTris* and *Pot*$_2$. Table I shows the results about interrupt time on ACL. *FastUp* again shows much better performance than the others. Compared with the result of *BBA*, *FastUp*'s interrupt time performance is within 90% of the optimality.

REFERENCES

[1] Nick McKeown *et al.* Openflow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2):69–74, 2008.
[2] Xitao Wen *et al.* RuleTris: Minimizing rule update latency for TCAM-based SDN switches. In *ICDCS*, pages 179–188. IEEE, 2016.
[3] Peng He *et al.* Partial order theory for fast TCAM updates. *IEEE/ACM Transactions on Networking*, 26(1):217–230, 2018.
[4] Kun Qiu *et al.* Fast lookup is not enough: Towards efficient and scalable flow entry updates for TCAM-based OpenFlow switches. In *ICDCS*, pages 918–928. IEEE, 2018.
[5] Chengchen Hu *et al.* Design of all programable innovation platform for software defined networking. In *Presented as part of ONS*, 2014.
[6] David E Taylor and Jonathan S Turner. Classbench: A packet classification benchmark. *IEEE/ACM TON*, 15(3):499–511, 2007.