A fast flow table engine for Open vSwitch with high performance on both lookups and updates

Hui Li, Ting Huang
PKUSZ&Peng Cheng Laboratory
{lih64,huangting53}@pku.edu.cn

Tong Yang, Wenjun Li*
EECS/SECE, Peking University
{yang.tong,wenjunli}@pku.edu.cn

Gong Zhang
2012 Labs, Huawei
nicholas.zhang@huawei.com

ABSTRACT

To support fast rule updates in SDN, the Open vSwitch uses a variant of Tuple Space Search (TSS) for flow table lookups, which is less efficient than decision trees on packet classifications. In this poster, we present our latest work on building fast flow table engine in Open vSwitch, which achieves high-speed table lookups and fast rule updates simultaneously. By mapping rules into tree nodes dynamically, a very limited TSS-assisted balanced trees can be generated without the trouble of rule replications. Preliminary experimental results show that using ClassBench, our work has comparable update performance to the TSS algorithm in Open vSwitch, while achieving almost an order-of-magnitude improvement on lookup performance over TSS on average.

CCS CONCEPTS

• **Networks → Packet classification**.

KEYWORDS

SDN, Open vSwitch, OpenFlow, Packet classification

ACM Reference Format:
Hui Li, Ting Huang, Tong Yang, Wenjun Li, and Gong Zhang. 2019. A fast flow table engine for Open vSwitch with high performance on both lookups and updates. In *SIGCOMM '19: ACM SIGCOMM 2019 Conference (SIGCOMM Posters and Demos '19), August 19–23, 2019, Beijing, China.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3342280.3342331

1 INTRODUCTION

OpenFlow virtual switches are being widely deployed in SDN/NFV to enable a wide spectrum of non-traditional applications, such as flexible resource partitioning and real-time migration. The OpenFlow switch enforces forwarding policies with multiple *'match-action'* table lookups, which is essentially an extensively studied multi-field packet classification problem [4]. But unlike traditional algorithmic packet classification which focused on high-speed table lookups, OpenFlow has a much higher demand on rule updates, making most of existing algorithms inapplicable in this context [7].

Among packet classification techniques, decision tree has been actively investigated because of its capability for high-speed table lookups [1–3, 8]. However, decision tree based schemes cannot support fast rule updates due to the notorious rule replication problem. In contrast, TSS based schemes partition rules into a set of hash tables without any rule replications, thereby enabling fast rule updates, which is an important metric for SDN switches. As a result, the popular Open vSwitch implements a variant of TSS called Priority Sorting Tuple Space Search (PSTSS) for its flow table lookups [5]. However, TSS based schemes have a performance concern for large rule set due to the tuple expansion problem.

In this poster, we present our latest progress on algorithmic packet classifications especially in decision trees, which achieves high performance on both table lookups and rule updates. Intuitively, to achieve this goal, there are two major challenges must be carefully addressed when building decision trees. First, in order to improve classification performance, how to build short trees to reduce memory access for each lookup; second, in order to support fast rule updates, how to avoid rule replication in decision trees.

To address these challenges, we propose a two-stage heterogeneous framework, which can generate a very limited short decision trees without any rule replications. In the first stage, several balanced mapping trees are constructed from rule subsets grouped with respect to their *small fields*. This grouping eliminates wildcard (*) at super-bits of *small fields*, thereby enabling very efficient mapping without any rule replications. The second stage handles the terminated nodes from pre-mappings, where wildcards may lead to rule replications. A salient fact is that after pre-mappings, the number of rules in the terminal nodes has been significantly reduced, where the linear search or TSS approaches can be well applied for these subsets to facilitate tree constructions.

Overall, the goal of our project is to design a fast flow table engine for packet processing in Open vSwitch, which can adaptively exploit the benefits of tree and TSS techniques.

2 DESIGN & A WORKING EXAMPLE

Before describing the key steps of our algorithm design, we first give the definition of an important concept: *small field*.

2.1 Definition: Given an N-field rule R=(F_1, ..., F_i, ... F_N) and a threshold value vector T=(T_1, ..., T_i, ... T_N), we give a definition for field F_i as follows: if the range span length of field $F_i \leq$ threshold value T_i, we say that F_i is a *small field*.

2.2 Key Steps: Based on the definition of *small field*, we now list the key steps of our designed algorithm as follows.

*Wenjun Li (Corresponding author) and Hui Li are also with the Future Network PKU Lab of National Major Research Infrastructure. This work is supported by the National Keystone R&D Program of China (2018YFB1004403, 2017YFB0803204, 2016YFB1000304), PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001), NSFC (61672061, 61671001), Shenzhen Municipal Development and Reform Commission (Disciplinary Development Program for Data Science and Intelligent Computing) and Shenzhen Research Program (JCYJ20170306092030521). This work is conducted under the guidance of Tong Yang.

SIGCOMM Posters and Demos '19, August 19–23, 2019, Beijing, China

ACM ISBN 978-1-4503-6886-5/19/08...$15.00
https://doi.org/10.1145/3342280.3342331

Table 1: Example rule set with two IPv4 address fields

rule #	src_addr	dst_addr	*rule #*	src_addr	dst_addr
R_1	228.128.0.0/9	0.0.0.0/0	R_8	0.0.0.0/0	123.0.0.0/8
R_2	223.0.0.0/9	0.0.0.0/0	R_9	178.0.0.0/7	0.0.0.0/1
R_3	0.0.0.0/1	175.0.0.0/8	R_{10}	0.0.0.0/1	172.0.0.0/7
R_4	0.0.0.0/1	225.0.0.0/8	R_{11}	0.0.0.0/1	226.0.0.0/7
R_5	0.0.0.0/2	225.0.0.0/8	R_{12}	128.0.0.0/1	120.0.0.0/7
R_6	128.0.0.0/1	123.0.0.0/8	R_{13}	128.0.0.0/2	120.0.0.0/7
R_7	128.0.0.0/1	37.0.0.0/8	R_{14}	128.0.0.0/1	38.0.0.0/7

Table 2: Partitioned rules based on *small dst_addr field*

rule	src_addr ($T_{src\_addr} = 2^{25}$)	dst_addr ($T_{dst\_addr} = 2^{25}$)	
#	1-32th bits	**33-39th bits**	40-64th bits
R_3	0*********************************	**1010111**	1*********************
R_4	0*********************************	**1110000**	1*********************
R_5	00********************************	**1110000**	1*********************
R_6	1*********************************	**0111101**	1*********************
R_7	1*********************************	**0010010**	1*********************
R_8	*********************************	**0111101**	1*********************
R_{10}	0*********************************	**1010110**	*********************
R_{11}	0*********************************	**1110001**	*********************
R_{12}	1*********************************	**0111100**	*********************
R_{13}	10********************************	**0111100**	*********************
R_{14}	1*********************************	**0010011**	*********************

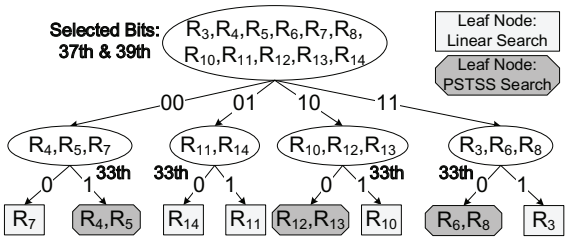


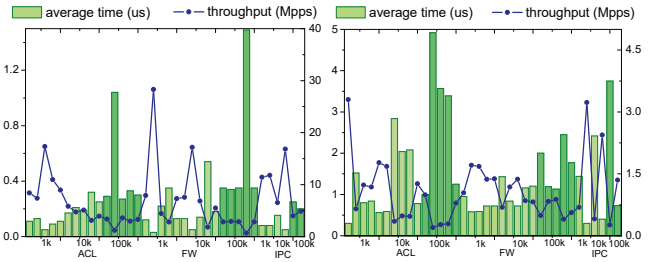
Figure 1: TSS-assisted decision tree for rules in Table 2.

–***Step 1***: *Rule Partitioning*. Based on the observations revealed in previous literatures [2, 3] that, even under very demanding thresholds, most rules still have at least one *small field*. Thus, similar to HybridCuts [2], we can partition the vast majority of the rules into at most N subsets without duplicates among each other, where rules in each subset share a common characteristic in the same single field: *small field*. Besides, since the number of rules without any *small fields* is negligible, we can simply apply PSTSS for these rules.

–***Step 2***: *Balanced Tree Mappings*. For each partitioned rule subset, we then build a multi-way tree from selective bits recursively, so that rules can be mapped into smaller subsets containing a very limited rules. Obviously, for the *small field* with the type of prefix/exact values, there is no wildcard at its corresponding superbits of the rules. To exploit this favorable property, we employ a greedy bit selection algorithm on rule bits especially in these superbits, to build a balanced mapping tree without any rule replications.

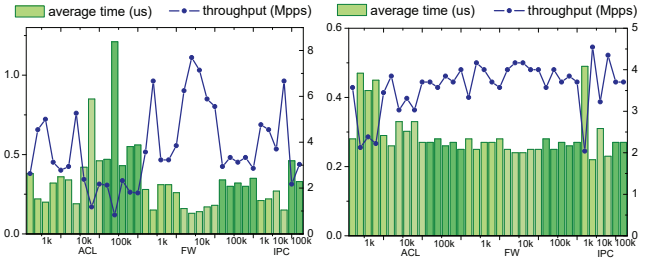
–***Step 3***: *TSS-assisted Decision Trees*. There are three conditions to stop the first stage mapping progress and resort to other more effective methods for the following tree constructions: 1) the number of rules in the mapped tree node is less than a predefined bucket size; 2) the remaining unselected rule bits share same values and cannot separate rules from each other; 3) the further bit mapping will led to rule replications due to the wildcards. Finally, for rules in these terminated mapping nodes (i.e., leaf nodes), we employ the linear search (#rules ≤ bucket size) or the PSTSS (#rules > bucket size) to facilitate tree constructions.

2.3 A Working Example: Suppose each internal tree node is allowed to select a maximum of two bits for rule mapping and the bucket size of the leaf node is one, the threshold value vector is T = ($T_{src\_addr} = 2^{25}$, $T_{dst\_addr} = 2^{25}$). Figure 1 shows the TSS-assisted decision tree constructed from the rules shown in Table 2, where these rules are partitioned based on the *small dst_addr field* from the rule set shown in Table 1.



(a) Our algorithm (b) PSTSS

Figure 2: Classification performance.



(a) Our algorithm (b) PSTSS

Figure 3: Update performance.

3 PRELIMINARY EVALUATION

Using ClassBench [6], we compare our algorithm with the PSTSS algorithm [5]. The source code of PSTSS is downloaded from PartitionSort [8]. There are three types of rule sets: ACL, FW and IPC, whose size varies from 1k to 100k. For each size, we generate 12 rule sets based on 12 seed files.

Figure 2 and Figure 3 show the average classification time and update time of our algorithm and PSTSS respectively, as well as their corresponding throughputs. Compared to PSTSS, experimental results show that our algorithm has similar update performance as the PSTSS, but achieves an average of 8.6 times faster than PSTSS on classification time.

4 CONCLUSION AND FUTURE WORK

Open vSwitch implements a variant of TSS instead of decision tree based algorithms with better performance on lookups. The primary reason is their poor support for fast rule updates, which is an important metric for SDN switches. To achieve fast lookups and updates at the same time, we introduce a TSS-assisted decision tree framework for packet classifications. Thanks for the clever partitioning and balanced mapping, a very limited short decision trees can be generated without any rule replications. As our future work, we will improve our scheme from the following aspects: 1) dynamic rule partitioning based on rule bits; 2) effective mapping for range fields. Besides, we will also implement our schemes in hardware platform such as FPGA.

REFERENCES

[1] Peng He, Gaogang Xie, Kavé Salamatian, and Laurent Mathy. 2014. Meta-algorithms for software-based packet classification. In *ICNP 2014*. IEEE, 308–319.

[2] Wenjun Li and Xianfeng Li. 2013. HybridCuts: A scheme combining decomposition and cutting for packet classification. In *Hot Interconnects 2013*. IEEE, 41–48.

[3] Wenjun Li, Xianfeng Li, Hui Li, and Gaogang Xie. 2018. CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification. In *INFOCOM 2018*. IEEE, 2645–2653.

[4] Nick McKeown and et al. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM CCR* 38, 2 (2008), 69–74.

[5] Ben Pfaff and et al. 2015. The design and implementation of open vswitch. In *NSDI 2015*. USENIX, 117–130.

[6] David E Taylor and Jonathan S Turner. 2007. Classbench: A packet classification benchmark. *IEEE/ACM ToN* 15, 3 (2007), 499–511.

[7] Tong Yang and et al. 2018. Fast OpenFlow Table Lookup with Fast Update. In *INFOCOM 2018*. IEEE, 2636–2644.

[8] Sorrachai Yingchareonthawornchai, James Daly, Alex X Liu, and Eric Torng. 2018. A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification. *IEEE/ACM ToN* 26, 4 (2018), 1907–1920.