TabTree: A TSS-assisted Bit-selecting Tree Scheme for Packet Classification with Balanced Rule Mapping

Wenjun Li*†‡, Tong Yang‡, Yeim-Kuan Chang§, Tao Li¶ and Hui Li*†‡
*Peking University Shenzhen Graduate School, †Peng Cheng Laboratory, ‡Peking University, §NCKU, ¶NUDT
Email: {wenjunli, yang.tong}@pku.edu.cn, ykchang@mail.ncku.edu.tw, taoli_network@163.com, lih64@pkusz.edu.cn

***Abstract*—To support fast rule updates in SDN, the Open vSwitch implements Priority Sorting Tuple Space Search (PSTSS) for its packet classifications. Although it has good performance on rule updates, it has a performance concern on table lookups. In contrast, decision tree methods are being actively investigated for high throughput, but they are not able to support fast updates because of rule replications. CutSplit, the state-of-the-art decision tree scheme, provides a novel rule update mechanism by avoiding tree reconstructions. However, its average update time is still two orders of magnitude larger than PSTSS. Meanwhile, existing decision trees are not only unbalanced but also depth unbounded, making them difficult to be optimized on FPGA. In this paper, we present a new decision tree scheme called TabTree, which achieves high performance on both lookups and updates. By mapping rules into tree nodes dynamically, a very limited number of balanced trees with bounded depths can be generated without the trouble of rule replications. Experimental results show that, TabTree has comparable update performance to PSTSS, but it outperforms PSTSS significantly in terms of number of memory accesses for packet classification. Additionally, TabTree is more practical for implementations on FPGA.**

***Index Terms*—OpenFlow, OVS, packet classification, FPGA**

I. Introduction

OpenFlow switches are being 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 *match-action* table lookups, which is essentially a multi-field packet classification problem [1]. As an extensively studied bottleneck [2], hardware using expensive TCAM has been the dominant implementation of packet classification in commercial OpenFlow switches. Despite its capability for line-speed classifications, TCAM is not only area-inefficient [3]–[9] and power hungry [10]–[15], but also not suitable for representing rules with range fields [16]–[28]. Meanwhile, OpenFlow has a much higher demand on rule updates, while modern TCAM still suffers from high update complexity [29]–[34]. Thus, efficient algorithmic solutions using ordinary memories are becoming a revitalized demand.

This work is supported by NSFC (61671001, 61672061), Key Areas R&D Program of Guangdong (2019B010137001), National Keystone R&D Program of China (2017YFB0803204, 2016YFB1000304, 2018YFB1800402), PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001), Shenzhen Municipal Development and Reform Commission (Disciplinary Development Program for Data Science and Intelligent Computing) and Shenzhen Research Program (JCYJ20170306092030521). Corresponding authors Hui Li and Wenjun Li are also with Shenzhen Key Lab of Information Theory and Future Internet Architecture, Shenzhen, P.R.China.

Among algorithmic packet classifications, decision tree and TSS are two major approaches. Decision tree has been recognized as a promising alternative to TCAM-based solutions, because of the following reasons: 1) it is well suited for handling rules with more fields, such as OpenFlow; 2) it is inherently favorable for implementations on modern hardwares, such as FPGA. So, decision tree algorithms [35]–[47] and its practical hardware implementations [48]–[64] are being actively investigated in the past decade. However, these decision trees cannot support fast updates because of the notorious rule replications. Recently, CutSplit [44] provides a novel rule update mechanism by avoiding tree reconstruction, but it takes an average of 50*us*, which is still two orders of magnitude larger than TSS schemes. Besides, existing decision trees are not only unbalanced but also depth unbounded, making them difficult to be well optimized on FPGA.

In contrast, TSS (Tuple Space Search) partitions rules into a set of hash tables without any rule replications, thereby enabling an average of one memory access for each rule update [65]–[67]. Thus, the popular Open vSwitch implements a variant of TSS called PSTSS [66] for its flow table lookups, the primary reason is its good support for fast rule updates, which is an important metric for SDN switches [68]. However, TSS schemes have a performance concern because of tuple expansions for large tables with more fields. Although the recent OpenFlow specification puts forward the multiple match tables (MMT) model that allows multiple smaller flow tables to be matched in a pipeline of stages, MMT may introduce higher latency for packets [69].

In this paper, we present a new decision tree scheme called **TabTree** (**T**SS-**a**ssisted **b**it-selecting **Tree**), which achieves high performance on both table lookups and rule updates. Essentially, TabTree is a two-stage framework for packet classification. In the first stage, several balanced bit-selecting trees are constructed from rule subsets grouped with respect to their *small fields*. This grouping eliminates wildcard (*) at a set of most significant bits in *small fields*, thereby enabling efficient rule mapping without the trouble of rule replications. The second stage handles the terminated nodes from pre-mappings, where wildcards may lead to serious 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.

978-1-7281-4387-3/19/$31.00 © 2019 IEEE

The main contributions of this paper are as follows:

- A novel two-stage framework consisting of heterogeneous algorithms: decision tree, linear search and TSS.
- Two heuristic bit-selecting algorithms are proposed to build partial trees in the first stage, which can map rules into tree nodes as balanced as possible.
- A TSS based scheme is used to assist decision tree constructions in the second stage, which can bound the tree depth and avoid rule replications simultaneously.
- A novel range encoding scheme is proposed, so that the range fields can also be employed for bit-selecting in the first stage of decision tree constructions.

Using ClassBench [70], preliminary experimental results show that, a very limited number of balanced and depth bounded subtrees can be generated in TabTree. Compared to the PSTSS algorithm, TabTree has similar update performance as PSTSS, but achieves 4.3 times reduction on the number of memory accesses for packet classification. Compared with another two latest decision tree schemes: CutSplit [44] and PartitionSort [42], TabTree also outperforms these two schemes on both lookups and updates considerably. Besides, experimental results also show that, even for rule sets up to 100k entries, TabTree can still construct shallow decision trees in a few MBytes, making it favorable for implementations and optimizations on FPGA. Our implementation of TabTree will be publicly available in our website (http://www.wenjunli.com/TabTree).

The rest of the paper is organized as follows. Section II briefly summarizes background and motivation. Section III presents the technical details of TabTree. Section IV provides preliminary experimental results. Finally, Section V draws the conclusion and our future work.

II. Background & Motivation

In this section, we first review the background and some classic approaches about packet classification. After that, we briefly describe two related algorithmic approaches: tuple space and decision tree. Finally, we give the motivation.

A. The Packet Classification Problem

The purpose of packet classification is to enable differentiated packet treatment according to a predefined packet classifier. A packet classifier is a set of rules, with each rule containing multiple field values (exact value, prefix or range) and an action to be taken in case of a match. Table I shows 14 rules defined over two prefix fields, which was before used in SmartPC [13]. As an extensively studied problem [2], a lot algorithmic approaches have been proposed in its first decade, such as decision tree [71]–[76], decomposition [71], [77]–[82] and TSS [65], [83]. Since TSS and decision tree are related to our proposed TabTree, we next give more detailed review about these two approaches, as well as their latest progress in the last decade.

TABLE I
Example Rule Set with Two IPv4 Address Fields

rule id	priority	src_addr field	dst_addr field	*action*
R_1	14	228.128.0.0/9	0.0.0.0/0	*action1*
R_2	13	223.0.0.0/9	0.0.0.0/0	*action2*
R_3	12	0.0.0.0/1	175.0.0.0/8	*action3*
R_4	11	0.0.0.0/1	225.0.0.0/8	*action4*
R_5	10	0.0.0.0/2	225.0.0.0/8	*action5*
R_6	9	128.0.0.0/1	123.0.0.0/8	*action6*
R_7	8	128.0.0.0/1	37.0.0.0/8	*action7*
R_8	7	0.0.0.0/0	123.0.0.0/8	*action8*
R_9	6	178.0.0.0/7	0.0.0.0/1	*action9*
R_{10}	5	0.0.0.0/1	172.0.0.0/7	*action10*
R_{11}	4	0.0.0.0/1	226.0.0.0/7	*action11*
R_{12}	3	128.0.0.0/1	120.0.0.0/7	*action12*
R_{13}	2	128.0.0.0/2	120.0.0.0/7	*action13*
R_{14}	1	128.0.0.0/1	38.0.0.0/7	*action14*

B. Tuple Space based Solutions

The original TSS scheme [65] partitions rules into a set of hash tables (i.e., tuple space) without any rule replications, which is based on easily computed rule characteristics such as prefix length. For example, rules in Table I can be partitioned into the following seven tuple spaces: $\{(9,0)|R_1,\ R_2\}$, $\{(1,8)|R_3,\ R_4,\ R_6,\ R_7\}$, $\{(2,8)|R_5\}$, $\{(0,8)|R_8\}$, $\{(7,1)|R_9\}$, $\{(1,7)|R_{10},\ R_{11},\ R_{12},\ R_{14}\}$ and $\{(2,7)|R_{13}\}$. Thus, rules can be inserted and deleted from hash tables in amortized one memory access, resulting in high update performance. But in order to find the best matching rule for each packet, all these partitioned hash tables have to be searched exhaustively, resulting in low lookup performance. As an improvement, PSTSS [66] reduce average table lookups by introducing a pre-computed priority for each tuple space, so that each search can terminate as soon as a match is found. However, its worst-case performance is still the same as TSS. TupleMerge [67], a recently proposed tuple space scheme, improves upon TSS by relaxing the restrictions on which rules may be placed in the same tuple space. However, with more tuple spaces merged, its performance may be affected due to hash collisions.

C. Decision Tree based Solutions

In decision tree based schemes, the geometric view of the packet classification problem is taken and a decision tree is built. The root node covers the whole searching space containing all rules. They work by recursively partitioning the searching space into smaller sub-spaces, until the rules covered by each sub-space is less than the pre-defined bucket size called *binth*. In case a rule spans multiple sub-spaces, the undesirable rule replication happens (e.g., R_8 in Figure 1). When a packet arrives, the decision tree is traversed based on the key values in the packet header, to find a matching rule at a leaf node. According to the partitioning method on searching space, current decision trees can be categorized into the following two major approaches:

1) Bit-selecting based Cutting. Cutting based schemes, such as HiCuts [72] and HyperCuts [76], separate the searching space into many equal-sized sub-spaces using local optimizations. Figure 1 shows the decision tree constructed using HyperCuts for 14 rules shown in Table I.

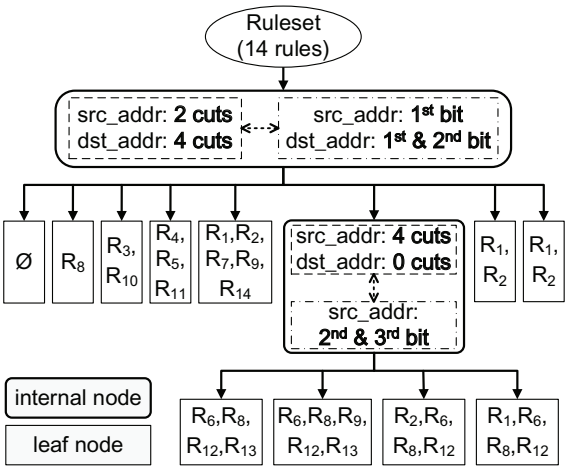


Fig. 1. An example of HyperCuts (*binth* = 5).

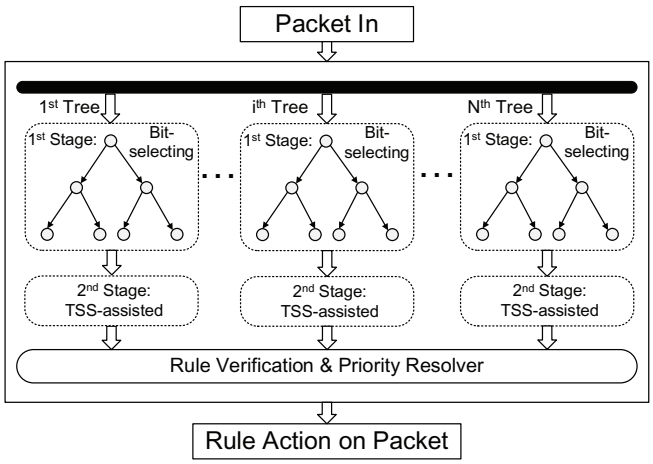


Fig. 2. The framework of TabTree.

From a binary bits point of view, the cutting operation can be treated as a bit-selecting problem equivalently, which can be illustrated from the internal nodes in Figure 1. According to the bit-selecting mechanism, these cuttings can be further categorized into two major approaches: 1) select orderly from the most significant field bits to the least significant field bits, such as HiCuts and HyperCuts; 2) select discretely among arbitrary field bits, such as ModularPC [74] and MC-SBC [62], as well as our proposed TabTree.

2) Point-comparing based Splitting. By selecting a few balanced points, splitting based schemes such as HyperSplit [35], divide the searching space into several unequal-sized subspaces while containing nearly equal number of rules. When a packet arrives, the decision tree is traversed by comparing with the balanced points stored in each internal node. Since our proposed TabTree falls into the category of bit-selecting based cutting trees, more detailed review and working examples about splitting trees can refer to [35], [44].

As reviewed in CutSplit, rule replication is the key trouble-maker for decision trees. To reduce rule replications, rule partition has been recognized as a common practice and a lot novel partition based decision trees have been proposed in the past decade, such as EffiCuts [36], ParaSplit [53], HybridCuts [37], SmartSplit [38], MC-SBC [62], PartitionSort [42], ByteCuts [43], CutSplit [44] and NeuroCuts [47]. But as far as we know, rule replication is still a performance hurdle against decision trees to achieve fast rule updates.

D. Why Yet Another Decision Tree?

Clearly, decision tree has been actively investigated in the past decade. But most of them achieve high-speed lookups but not fast updates. Recently, CutSplit provides a novel rule update mechanism by avoiding tree reconstructions in most cases. However, it still need to reconstruct sub-trees in post-splitting stages, leading to 10x-100x times for rule updates than PSTSS. Although PartitionSort, the latest splitting tree, achieves high update performance by avoiding rule replications in each partitioned subset, it suffers from too many memory accesses and a large number of separated subtrees.

Additionally, FPGA has been recognized as a promising alternative to TCAMs [84]–[88]. To fully exploit the potential of FPGA, a FPGA-friendly tree scheme should also meet the following demand: a controllable number of balanced and bounded subtrees with low memory footprints, while most existing trees are unbalanced and/or unbounded.

Thus, to achieve fast lookups and updates for packet classification at the same time, we propose TabTree, a novel bit-selecting tree which is also desirable for FPGA implementations and optimizations.

III. Our Proposed Approach

In this section, we first introduce the ideas and framework of TabTree. Then, we employ a scalable rule partition algorithm and reveal some novel observations on partitioned rule subsets. To exploit these characteristics of subsets, two heuristic bit-selecting algorithms are proposed to map rules into tree nodes as balanced as possible. Finally, a TSS based scheme is applied to facilitate tree constructions, which can bound the tree depth and avoid rule replications.

A. Ideas & Framework

According to above review, decision tree can support fast lookups but not updates, while TSS can support fast update but not lookups. Therefore, the idea directly perceived is to design a heterogeneous framework that can take advantage of both decision tree and TSS approaches: TSS-assisted tree. But, in order to design a FPGA-friendly decision tree which can achieve high performance on both lookups and updates, TabTree still faces several difficulties and challenges: 1) low memory footprint; 2) limited rule subsets; 3) low memory access; 4) easy for updates; 5) bounded tree depth.

To address the first two challenges, we adopt the existing partition methods based on *small fields*, so that a very limited number of decision trees can be generated in linear memory without the trouble of rule replications. To address the third challenge, we proposed two heuristic bit-selecting algorithms, which can build trees as shallow as possible. For the last two challenges, we employ a TSS based scheme to assist tree constructions. Figure 2 shows the framework of TabTree.

B. Partitioning & Observations

Before describing the partitioning and key observations, we first give the definition of an important concept: *small field*. More definitions and notations can refer to [37], [44].

1) Definition of Small Field:

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) Rule Partition based on Small Fields:

Based on the observations revealed in CutSplit that, even under very demanding thresholds, most rules still have at least one *small field*. Thus, similar to CutSplit, we can partition the vast majority of the rules into a very limited number of subsets without duplicates among each other, where rules in each subset share a common characteristic in the selected fields: *small field*. More details on partitioning and merging can refer to paper [44]. Besides, since the number of rules without any *small fields* is negligible, we can simply employ the TSS based scheme such as PSTSS to handle these rules in software simulations.

3) Key Observations on Small Fields:

By examining rules grouped with respect to their *small fields*, we identify a useful observation on rule fields, which is the key basis of the following proposed bit-selecting trees: For the *small field* of grouped rules with the type of prefix or exact value, there are a set of most significant bits which are indicated by either bit-0 or bit-1. More specifically, for a W-bit wide field F_i with the threshold value of 2^K, we can draw the conclusion that, F_i is a *small field* if and only if there are no wildcard (*) at its most significant $W-K$ bits. In this paper, we call these $W-K$ bits as *selectable bits*. For the *small field* of grouped rules with range type, we give a novel encoding scheme in next subsection, and show that the observation is still valid for its encoded prefixes.

C. False Range Encoding

Before describing the encoding scheme and key observations, we first give several definitions for a W-bit wide range $R_{ab} = [a, b]$. More definitions and notations can refer to [18], [22], [26], [28].

1) Definitions:

–Longest Common Prefix (LCP): LCP_{ab} is the longest prefix that covers the range R_{ab}. If prefixes are illustrated in a binary trie, LCP_{ab} is the lowest common ancestor of integer a and integer b.

–Splitting Endpoint: For each range R_{ab}, there are two *splitting endpoints* (one if $a = b$), where the value of these two endpoints are the middle two values of LCP_{ab} (i.e., m & n in Figure 3).

–Extremal Range: The range R_{ab} is called *extremal range*, if $a = 0$ or $b = 2^W - 1$.

–Generalized Extremal Range: More broadly, the *extremal range* R_{ab} is called *generalized extremal range*, if integer a is the leftmost value of LCP_{ab} or integer b is the rightmost value of LCP_{ab}.

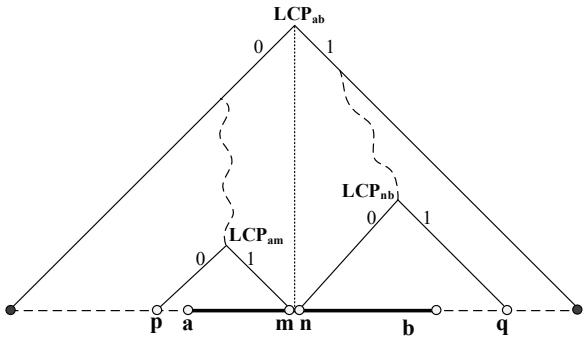


Fig. 3. False encoding sample ($R_{ab} \approx LCP_{am}$ & LCP_{nb}).

2) Encoding with Low False Positive:

Unlike traditional range encodings for TCAMs, which require to ensure absolute semantic equivalence, false positive is allowed in algorithmic decision trees. The reason is that, the main function of decision trees is to separate rules into subspaces, where linear search is still required for rules located in these subspaces. Thus, for each incoming packet, false negative can never happen in decision trees and the negative impact on correct search caused by false positive can also be eliminated by linear search in leaf nodes. Based on this observation, we describe a simple encoding scheme for ranges, which achieves low false positive and low range expansion simultaneously. In essence, the goal of our scheme is to find at most two longest prefixes, which can cover the range entirely. The encoding steps are as follows:

–Step 1: If R_{ab} is a *generalized extremal range*, skip to step 2. Otherwise, split R_{ab} into two *generalized extremal ranges* (i.e., R_{am} and R_{nb} in Figure 3);

–Step 2: For each split *generalized extremal range*, output its *LCP* as an encoded prefix (i.e., LCP_{am} for R_{am} and LCP_{nb} for R_{nb} in Figure 3);

–Step 3: If the two encoded prefixes are with the same prefix length, output LCP_{ab} as the final encoded prefix.

Note that there is a key difference between our encoding and traditional TCAM encodings: The encoded prefixes are only used during decision tree constructions, and rules stored in leaf nodes are still the original rules with ranges.

3) Narrower Ranges, Longer Perfixes:

Based on above encoding scheme, the range R_{ab} shown in Figure 3 can be encoded into two prefixes: LCP_{am} & LCP_{nb}. For these encoded prefixes, we have the following theorem:

–Theorem 1: For a W-bit wide range R_{ab}, the length of encoded prefixes using our encoding scheme are both $\geq L$, where $L = W - \lceil \log_2 (b - a + 1) \rceil$.

Theorem 1 can be easily proved based on previous literatures [18], [22], [26], but due to the consideration on its relevance and space limit, we do not elaborate on detailed proofs in this paper. Based on this theorem, we can draw the conclusion that, the observation described in above subsection is still valid for *small range fields*, that is to say: For the *small field* of grouped rules with range type, there are still a set of *selectable bits* in its encoded prefixes, which are indicated by either bit-0 or bit-1.

D. Balanced Bit-selecting

The rationale behind the above strategy of partitioning is simple: by grouping rules that are narrow in the same fields, we get a set of *selectable bits* among grouped rules without wildcard values. Thus, each *selectable bit* can map rules into at most two subsets without any rule replications. To exploit this favorable property, we build a multi-way tree by selecting a few *selectable bits* in each tree node recursively.

In order to build shallow decision trees, the key challenge is how to select a few most distinguishing *selectable bits* in each tree node, so that rules can be mapped into its children nodes as balanced as possible. Next, we introduce two heuristic bit-selecting strategies for this purpose. To control the width of the tree, we assume that at most *b* bits are allowed to be selected in each tree node.

1) Brute Force Strategy:

Assume that there are *M* rules and *B* unused *selectable bits* in current node, the brute force algorithm is to find the most distinguishing *b* bits, which partition the rules in current node into $n = 2^b$ subsets in the most balanced fashion. Thus, our goal is to find the best one from C_B^b bit combinations that minimize the value of the objective function defined in (1), that means the size of generated subsets are most similar. Here, x_i is the number of rules in the *i*-th subset.

$$costFunc(b\ bits) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x)^2}{n}}, where\ x = \frac{M}{n} \quad (1)$$

This bit-selecting strategy, though simple and optimal, is computationally expensive to calculate, as it involves comparing fully among all C_B^b bit combinations. As a compromise, we next give a greedy bit-selecting strategy, which has much low computational cost.

2) Greedy Strategy:

Unlike brute force algorithm which selects multiple bits at a time with global optimization, greedy algorithm tries to find a local optimal solution, where the "good" bits are selected one by one recursively. We assign an *imbalance* value for each current *selectable* and unused bit by using (2), where *#ruleLChild*/*#ruleRChild* is the number of rules mapped into the left/right child node (i.e., #bit-0/1s in *v*-th bit). The greedy algorithm is to choose at most *b* bits one by one, where each selected single bit is with the smallest *imbalance* value among current *selectable* and unused bits.

$$imbalance(bit\ v) = |\#ruleLChild - \#ruleRChild| \quad (2)$$

E. TSS-assisted Decision Tree

As reviewed in above section, a FPGA-friendly decision tree should also bound its tree depth, so that it can be easily optimized for FPGA pipeline stages. Meanwhile, to achieve high update performance, rule replication in the second stage has been the key performance metric of TabTree.

In order to bound tree depth and support fast rule updates, TabTree stops its first stage bit-selecting progress in one of the following cases: 1) the tree depth achieves the predefined maximum value; 2) the number of rules in the mapped tree node is less than a predefined threshold value (i.e., *binth*); 3) the remaining unselected rule bits share same values and cannot separate rules from each other; 4) the further bit-selecting will led to rule replications due to the wildcards. Then, TabTree resorts to other more effective methods for the following tree constructions.

A salient fact is that after balanced pre-mappings, the number of rules in the terminal nodes (i.e., leaf nodes) has been significantly reduced. To exploit this favorable property, we employ the linear search (#rules $\leq$ *binth*) or the PSTSS (#rules $>$ *binth*) to facilitate tree constructions.

TABLE II
PARTITIONED RULES WITH SMALL DST_ADDR FIELD

rule	src_addr (T_{src_addr}= 2^{25})	dst_addr (T_{dst_addr}= 2^{25})	
id	1-32th bits	**33-39th**	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**	**********************

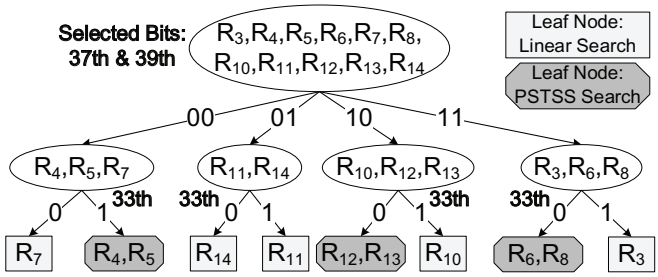


Fig. 4. TSS-assisted decision tree for rules in Table II.

F. A Working Example

To illustrate TabTree more clearly, we give a working example for 14 rules given in Table I. Assume that each internal tree node is allowed to select a maximum of two bits for rule mapping and the *binth* of the leaf node is one, the threshold value vector is T = (T_{src_addr}= 2^{25}, T_{dst_addr}= 2^{25}).

TabTree first partitions the 14 rules into the following two rule subsets: $(arbitrary_{src\_addr}, small_{dst\_addr})$ = $\{R_3, R_4, R_5, R_6, R_7, R_8, R_{10}, R_{11}, R_{12}, R_{13}, R_{14}\}$ and $(small_{src\_addr}, arbitrary_{dst\_addr})$ = $\{R_1, R_2, R_9\}$. Thus, for the *small field* of grouped rules in each subset, there are 32−25=7 *selectable bits*. After rule partition, a TSS-assisted decision tree is constructed for each rule subset. For example, Table II shows the partitioned rules in $(arbitrary_{src\_addr}, small_{dst\_addr})$ with the representation of ternary strings. Clearly, the middle 7 bits (i.e., 33-39th) in Table II are *selectable bits*. Based on the proposed bit-selecting and TSS-assisted methods, Figure 4 shows the TSS-assisted decision tree constructed for the rules shown in Table II.

IV. Preliminary Evaluation

Using ClassBench [70], we compare TabTree with PSTSS, CutSplit and PartitionSort, where the source codes of these three algorithms are downloaded from [42], [44]. 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 in ClassBench. Assume that each tree depth is limited to 5 and *binth* is 4. We next evaluate from the following key metrics respectively.

A. Number of Subsets

Table III shows the number of partitioned rule subsets in TabTree and another three algorithms. Clearly, TabTree and CutSplit produce a relatively stable number of subsets regardless of the type and size of rulesets, with an average of 3.7 subsets. Essentially, this partition result is consistent with existing observations on ClassBench rules: IP addresses are the most distinguishing rule fields. In contrast, the number of partitioned subsets in PSTSS and PartitionSort ranges from 10 to 230, with an average of 152 and 21 subsets respectively.

B. Memory Footprint

Table III also shows the memory footprint of TabTree and another three algorithms. Experimental results show that TabTree requires less space than other algorithms and the memory consumption of TabTree increases almost linearly with the rule set size. Even for rule sets up to 100k entries, TabTree can still construct decision trees in a few MBytes, small enough to be accommodated into the Block RAM (BRAM) of middle-end FPGAs, such as Xilinx Virtex-7 FPGAs.

C. Memory Access

We measure memory access by classifying all packets in trace files generated by ClassBench when it constructs the corresponding rule set. Figure 5 shows the memory access of TabTree and another three algorithms. For simplicity, we think traversing a decision tree node, a rule or a tuple table as one memory access in our experiments. It is obvious that TabTree is significantly better others, achieves an average of 4.3 times reduction compared with PSTSS. Compared to PartitionSort and CutSplit, TabTree also achieves 2.5 times and 1.3 times improvement on average.

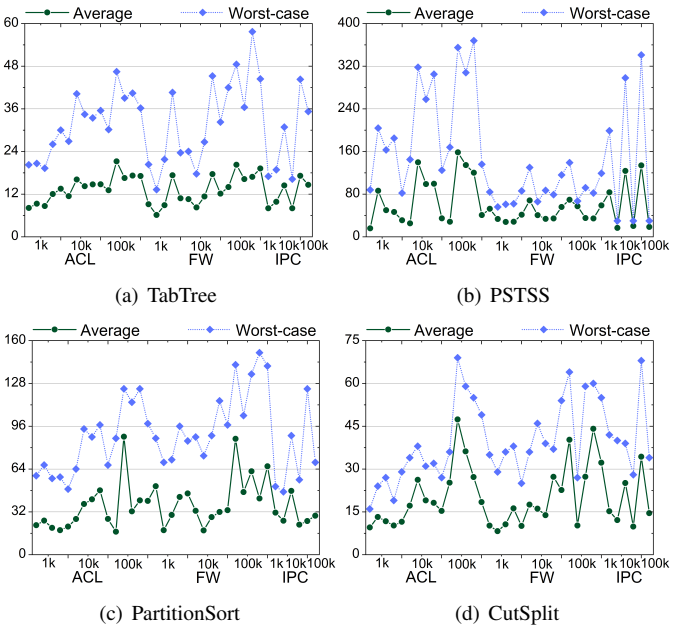
D. Update Performance

We measure incremental update time as the time required to performance one rule insertion or deletion. For each rule set, we shuffle rules randomly to generate a sequence of update operations, with half of insertions intermixed with half of deletions. Figure 6 shows that TabTree has comparable update performance to PSTSS, achieving an average of 2 MUPS (Millions of Update Per Second) in software simulations.

Thus, preliminary experimental evaluations show that, a very limited number of shallow trees can be generated with linear memory consumption in TabTree, which is also suitable for fast rule updates. More evaluations on FPGA will be given in our future work.

TABLE III
Average Subsets & Memory Footprint

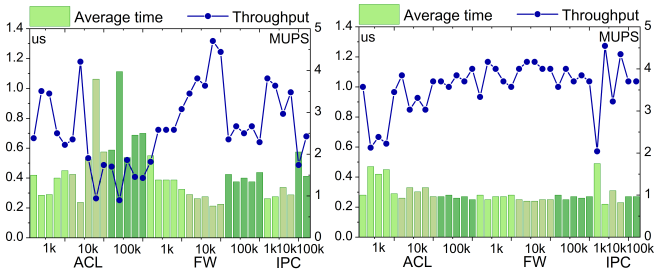
Algorithms	rules	#Subsets			Memory footprint (MB)		
		1k	10k	100k	1k	10k	100k
TabTree	ACL	3.8	3.8	3.4	0.03	0.25	2.34
	FW	4	4	4	0.03	0.21	2.44
	IPC	3.5	3.5	3.5	0.03	0.28	2.31
PSTSS	ACL	144.4	230.2	267	0.05	0.44	4.66
	FW	69.8	95.6	99.8	0.04	0.45	4.31
	IPC	114.5	164	185.5	0.04	0.48	4.92
PartitionSort	ACL	11	21.6	26.8	0.05	0.49	5.22
	FW	19.4	24.4	34.4	0.05	0.51	4.88
	IPC	10	11.5	12	0.05	0.54	5.57
CutSplit	ACL	3.8	3.8	3.4	0.04	1.28	11.52
	FW	4	4	4	0.04	4.17	18.29
	IPC	3.5	3.5	3.5	0.04	3.29	26.86



(a) TabTree (b) PSTSS

(c) PartitionSort (d) CutSplit

Fig. 5. Numbers of memory accesses.



(a) TabTree (b) PSTSS

Fig. 6. Update performance.

V. Conclusion and Our Future Work

To achieve fast lookups and updates at the same time, we propose a TSS-assisted decision tree scheme for packet classification with balanced rule mappings, which is also more friendly and practical for FPGA implementations. As our future work, we will improve TabTree from the following aspects: 1) self-adaptive rule partition instead of based on *small fields*; 2) design rule cache algorithm for TabTree. Besides, we will also implement our scheme on FPGA.

REFERENCES

[1] N. McKeown and et al, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.

[3] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness, and compactness," in *IEEE ICDCS*, 2004.

[4] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 490–500, 2010.

[5] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "Split: Optimizing space, power, and throughput for tcam-based classification," in *ACM/IEEE ANCS*, 2011.

[6] O. Rottenstreich and et al, "Compressing forwarding tables," in *IEEE INFOCOM*, 2013.

[7] E. Norige, A. X. Liu, and E. Torng, "A ternary unification framework for optimizing tcam-based packet classification systems," in *ACM/IEEE ANCS*, 2013.

[8] O. Rottenstreich and et al, "Lossy compression of packet classifiers," in *ACM/IEEE ANCS*, 2015.

[9] A. X. Liu, C. R. Meiners, and E. Torng, "Packet classification using binary content addressable memory," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1295–1307, 2016.

[10] E. Spitznagel, D. E. Taylor, and J. S. Turner, "Packet classification using extended tcams," in *IEEE ICNP*, 2003.

[11] F. Zane, G. Narlikar, and A. Basu, "Coolcams: Power-efficient tcams for forwarding engines," in *IEEE INFOCOM*, 2003.

[12] B. Vamanan and T. Vijaykumar, "Treecam: decoupling updates and lookups in packet classification," in *ACM CoNEXT*, 2011.

[13] Y. Ma and S. Banerjee, "A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification," in *ACM SIGCOMM*, 2012.

[14] X. Li, Y. Lin, and W. Li, "Greentcam: A memory-and energy-efficient tcam-based packet classification," in *IEEE International Conference on Computing, Networking and Communication*, 2016.

[15] W. Li, X. Li, and H. Li, "Meet-ip: Memory and energy efficient tcam-based ip lookup," in *IEEE ICCCN*, 2017.

[16] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *IEEE Hot Interconnects*, 2002.

[17] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary cams," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 193–204, 2005.

[18] Y.-K. Chang, "A 2-level tcam architecture for ranges," *IEEE Transactions on Computers*, vol. 55, no. 12, pp. 1614–1629, 2006.

[19] H. Che, Z. Wang, K. Zheng, and B. Liu, "Dres: Dynamic range encoding scheme for tcam coprocessors," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 902–915, 2008.

[20] A. Bremler-Barr and D. Hendler, "Space-efficient tcam-based classification using gray coding," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 18–30, 2010.

[21] A. Bremler-Barr, D. Hay, and D. Hendler, "Layered interval codes for tcam-based classification," *Computer Networks*, vol. 56, no. 13, pp. 3023–3039, 2012.

[22] O. Rottenstreich, R. Cohen, D. Raz, and I. Keslassy, "Exact worst case tcam rule expansion," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1127–1140, 2012.

[23] Y.-K. Chang, C.-C. Su, Y.-C. Lin, and S.-Y. Hsieh, "Efficient gray-code-based range encoding schemes for packet classification in tcam," *IEEE/ACM Transactions on Networking*, vol. 21, no. 4, pp. 1201–1214, 2012.

[24] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Sax-pac (scalable and expressive packet classification)," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 15–26, 2014.

[25] L. Schiff, Y. Afek, and A. Bremler-Barr, "Orange: Multi field openflow based range classifier," in *ACM/IEEE ANCS*, 2015.

[26] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "Optimal in/out tcam encodings of ranges," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 555–568, 2016.

[27] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Hel-Or, "Encoding short ranges in tcam without expansion: Efficient algorithm and applications," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 835–850, 2018.

[28] W. Li, X. Liu, W. Le, H. Li, and H. Zhang, "A practical range encoding scheme for tcams," in *ACM SIGCOMM Posters and Demos*, 2019.

[29] D. Shah and P. Gupta, "Fast updating algorithms for tcam," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.

[30] H. Song and J. Turner, "Fast filter updates for packet classification using tcam," in *IEEE GLOBECOM*, 2006.

[31] Y.-K. Chang, "Efficient multidimensional packet classification with fast updates," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 463–479, 2008.

[32] X. Wen and et al, "Ruletris: Minimizing rule update latency for tcam-based sdn switches," in *IEEE ICDCS*, 2016.

[33] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, "Partial order theory for fast tcam updates," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 217–230, 2018.

[34] K. Qiu and et al, "Fastrule: Efficient flow entry updates for tcam-based openflow switches," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 484–498, 2019.

[35] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *IEEE INFOCOM*, 2009.

[36] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "Efficuts: optimizing packet classification for memory and throughput," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 207–218, 2011.

[37] W. Li and X. Li, "Hybridcuts: A scheme combining decomposition and cutting for packet classification," in *IEEE Hot Interconnects*, 2013.

[38] P. He, G. Xie, K. Salamatian, and L. Mathy, "Meta-algorithms for software-based packet classification," in *IEEE ICNP*, 2014.

[39] S. Hager, S. Selent, and B. Scheuermann, "Trees in the list: Accelerating list-based packet classification through controlled rule set expansion," in *ACM CoNEXT*, 2014.

[40] G. Antichi, C. Callegari, A. W. Moore, S. Giordano, and E. Anastasi, "Ja-trie: Entropy-based packet classification," in *IEEE HPSR*, 2014.

[41] Z. Liu, X. Wang, B. Yang, and J. Li, "Bitcuts: Towards fast packet classification for order-independent rules," in *ACM SIGCOMM Posters and Demos*, 2015.

[42] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, "A sorted-partitioning approach to fast and scalable dynamic packet classification," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1907–1920, 2018.

[43] J. Daly and E. Torng, "Bytecuts: Fast packet classification by interior bit extraction," in *IEEE INFOCOM*, 2018.

[44] W. Li, X. Li, H. Li, and G. Xie, "Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification," in *IEEE INFOCOM*, 2018.

[45] S. Hager, P. John, S. Dietzel, and B. Scheuermann, "Rulebender: Tree-based policy transformations for practical packet classification systems," *Computer Networks*, vol. 135, pp. 253–265, 2018.

[46] H. Li, T. Huang, T. Yang, W. Li, and G. Zhang, "A fast flow table engine for open vswitch with high performance on both lookups and updates," in *ACM SIGCOMM Posters and Demos*, 2019.

[47] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," in *ACM SIGCOMM*, 2019.

[48] W. Jiang and V. K. Prasanna, "A memory-balanced linear pipeline architecture for trie-based ip lookup," in *IEEE Hot Interconnects*, 2007.

[49] Y. Qi and et al, "Towards high-performance flow-level packet processing on multi-core network processors," in *ACM/IEEE ANCS*, 2007.

[50] W. Jiang and V. K. Prasanna, "Field-split parallel architecture for high performance multi-match packet classification using fpgas," in *ACM SPAA*, 2009.

[51] Y. Qi and et al, "Multi-dimensional packet classification on fpga: 100 gbps and beyond," in *IEEE FPT*, 2010.

[52] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on fpgas," in *ACM/SIGDA FPGA*, 2009.

[53] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "Parasplit: A scalable architecture on fpga for terabit packet classification," in *IEEE Hot Interconnects*, 2012.

[54] W. Jiang and V. K. Prasanna, "A fpga-based parallel architecture for scalable high-speed packet classification," in *IEEE ASAP*, 2009.

[55] T. Ganegedara and V. K. Prasanna, "Stridebv: Single chip 400g+ packet classification," in *IEEE HPSR*, 2012.

[56] W. Jiang and V. K. Prasanna, "Scalable packet classification on fpga," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 20, no. 9, pp. 1668–1680, 2011.

[57] B. Yang, J. Fong, W. Jiang, Y. Xue, and J. Li, "Practical multituple packet classification using dynamic discrete bit selection," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 424–434, 2012.

[58] Y. R. Qu and V. K. Prasanna, "High-performance and dynamically updatable packet classification engine on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 197–209, 2015.

[59] Y. R. Qu, H. H. Zhang, S. Zhou, and V. K. Prasanna, "Optimizing many-field packet classification on fpga, multi-core general purpose processor, and gpu," in *ACM/IEEE ANCS*, 2015.

[60] S. Hager, D. Bendyk, and B. Scheuermann, "Partial reconfiguration and specialized circuitry for flexible fpga-based packet processing," in *IEEE ReConFig*, 2015.

[61] Y.-K. Chang and C.-S. Hsueh, "Range-enhanced packet classification design on fpga," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 214–224, 2015.

[62] C.-L. Hsieh and N. Weng, "Many-field packet classification for software-defined networking switches," in *ACM/IEEE ANCS*, 2016.

[63] A. Fiessler, S. Hager, B. Scheuermann, and A. W. Moore, "Hypafilter: A versatile hybrid fpga packet filter," in *ACM/IEEE ANCS*, 2016.

[64] A. Fiessler, C. Lorenz, S. Hager, B. Scheuermann, and A. W. Moore, "Hypafilter+: Enhanced hybrid packet filtering using hardware assisted classification and header space analysis," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3655–3669, 2017.

[65] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 135–146, 1999.

[66] B. Pfaff and et al, "The design and implementation of open vswitch," in *USENIX NSDI*, 2015.

[67] J. Daly and et al, "Tuplemerge: Fast software packet processing for online packet classification," *IEEE/ACM Transactions on Networking*, 2019.

[68] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *Springer International Conference on Passive and Active Network Measurement*, 2015.

[69] T. Yang and et al, "Fast openflow table lookup with fast update," in *IEEE INFOCOM*, 2018.

[70] D. E. Taylor and J. S. Turner, "Classbench: A packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499–511, 2007.

[71] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *ACM SIGCOMM*, 1998.

[72] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *IEEE Hot Interconnects*, 1999.

[73] A. Feldman and S. Muthukrishnan, "Tradeoffs for packet classification," in *IEEE INFOCOM*, 2000.

[74] T. Y. Woo, "A modular approach to packet classification: Algorithms and results," in *IEEE INFOCOM*, 2000.

[75] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to cams?" in *IEEE INFOCOM*, 2003.

[76] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *ACM SIGCOMM*, 2003.

[77] T. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 203–214, 1998.

[78] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 147–160, 1999.

[79] F. Baboescu and G. Varghese, "Scalable packet classification," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 199–210, 2001.

[80] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," in *IEEE INFOCOM*, 2005.

[81] F. Geraci, M. Pellegrini, P. Pisati, and L. Rizzo, "Packet classification via improved space decomposition techniques," in *IEEE INFOCOM*, 2005.

[82] W. Li, D. Li, Y. Bai, W. Le, and H. Li, "Memory-efficient recursive scheme for multi-field packet classification," *IET Communications*, vol. 13, no. 9, pp. 1319–1325, 2019.

[83] H. Lim and S. Y. Kim, "Tuple pruning using bloom filters for packet classification," *IEEE Micro*, vol. 30, no. 3, pp. 48–59, 2010.

[84] G. Brebner, "Packets everywhere: The great opportunity for field programmable technology," in *IEEE FPT*, 2009.

[85] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, 2014.

[86] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4-netfpga workflow for line-rate packet processing," in *ACM/SIGDA FPGA*, 2019.

[87] L. Linguaglossa and et al, "Survey of performance acceleration techniques for network function virtualization," *Proceedings of the IEEE*, vol. 107, no. 4, pp. 746–764, 2019.

[88] S. Pontarelli and et al, "Flowblaze: Stateful packet processing in hardware," in *USENIX NSDI*, 2019.