KickTree: A Recursive Algorithmic Scheme for Packet Classification with Bounded Worst-Case Performance

Yao Xin
Peng Cheng Laboratory
Shenzhen, China

Yuxi Liu
Southern University of Science and Technology, Shenzhen, China

Wenjun Li*
Peng Cheng Laboratory, China & Harvard University, MA, US

Ruyi Yao
Fudan University
Shanghai, China

Yang Xu
Fudan University
Shanghai, China

Yi Wang
Southern University of Science and Technology, Shenzhen, China

ABSTRACT

As a promising alternative to TCAM-based solutions for packet classification, FPGA has received increasing attention. Although extensive research has been conducted in this area, existing FPGA-based packet classifiers cannot satisfy the burgeoning needs from OpenFlow, which demands large-scale rule sets and frequent rule updates. As a recently proposed hardware-specific approach, TabTree avoids rule replication and supports dynamic rule update. However, it still faces problems of unbalanced rule subset partition, unevenly distributed subtrees and excessive TSS leaf nodes when implemented on FPGA. In this paper, we propose a hardware-friendly packet classification approach called KickTree, which is elaborated by considering hardware properties. To take advantage of intrinsic parallelism of FPGA, KickTree adopts multiple balanced decision trees which can run simultaneously. The bit selection is more flexible which breaks the restriction of rule subset. Moreover, each subset size is strictly limited, leading to bounded and evenly-distributed trees. Experimental results show KickTree outperforms TabTree significantly in terms of the number of memory accesses for each classification operation while providing a rule update performance comparable to TabTree. In summation, KickTree is more practical for implementations on FPGA.

CCS CONCEPTS

• **Networks** → **Packet classification**.

KEYWORDS

SDN, packet classification, decision tree, FPGA

ACM Reference Format:
Yao Xin, Yuxi Liu, Wenjun Li, Ruyi Yao, Yang Xu, and Yi Wang. 2021. KickTree: A Recursive Algorithmic Scheme for Packet Classification with Bounded Worst-Case Performance. In *Symposium on Architectures for Networking and Communications Systems (ANCS '21), December 13–16, 2021, Layfette, IN, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3493425.3502752

1 INTRODUCTION

Packet classification provides a way to discriminate packets into different "flows" and enables differentiated functionalities in various applications such as quality of service (QoS), security, and monitoring [40]. All packets belonging to the same flow obey a pre-defined rule and are processed in similar manner by the router. Although it has been widely studied in the last two decades, the well-known OpenFlow, being a foundation of Software-Defined Networking (SDN), puts forward higher requirements for packet classification [29]. Compared with conventional switches and routers, the packet classification in OpenFlow switches requires higher dimensions, larger rule sets, and a faster update rate.

At present, the mainstream hardware-based algorithms are principally TCAM-based solutions, but their shortcomings of limited capacity, high cost, and high power consumption make them difficult to be widely used in OpenFlow scenarios [30]. As a powerful hardware alternative to TCAM, the FPGA is gradually receiving attention [3, 8, 18, 31]. The existing FPGA-based packet classification designs are mainly

*Corresponding author: Wenjun Li (wenjunli@seas.harvard.edu). Yao Xin and Yuxi Liu contributed equally to this paper. This work is supported by National Key R&D Program of China (2019YFB1802600), Key-Area Research and Development Program of Guangdong Province (2021B0101400001, 2020B0101130003), NSFC (62102203, 62172108), the Major Key Project of PCL (PCL2021A08), Guangdong Basic and Applied Basic Research Foundation (2019B1515120031), Shanghai Pujiang Program (2020PJD005), Science and Technology Commission of Shanghai Municipality (20S31903800) and China Postdoctoral Science Foundation (2020TQ0158, 2020M682825, PC2021037).

ANCS '21, December 13–16, 2021, Layfette, IN, USA

ACM ISBN 978-1-4503-9168-9/21/12…$15.00
https://doi.org/10.1145/3493425.3502752

based on two algorithms: decision tree [9, 20] and decomposition method [10, 33]. The method based on decision tree faces the challenge of dynamic update due to rule replication, while the decomposition method has the dilemma of consuming too many resources so being unable to store large-scale rule sets. So almost none of the mentioned methods can meet the needs of OpenFlow scenarios.

TabTree [25] is a newly proposed decision tree based packet classification algorithm dedicated to FPGA. It uses *small fields* to divide rule set into subsets, and then uses Tuple Space Search (TSS) [38] to assist in the construction of decision tree for each subset, thus avoiding rule duplication and supporting dynamic updates. However, in the actual FPGA implementation, it still encounters several problems: i) The division of subsets heavily relies on empirical *small fields* characteristics of rules, and the number of subsets is the exponential size of the *small fields*. For instance, if K *small fields* are selected, 2^K subsets need to be generated, and the higher the rule dimension, the higher the probability of a larger K value. Therefore, the scalability is poor for high-dimensional rules; ii) The distribution of each rule subset partitioned based on the preset *small fields* is uneven, so that the depth of each decision tree is very different, which is not conducive to the convergence of concurrent results by FPGA; iii) There are a large number of TSS leaf nodes in the decision tree. Each TSS structure contains multiple hash tables, and the number of TSS is unpredictable for each rule set, which is not friendly to hardware implementation.

In response to the above problems, we propose KickTree in this paper, which fully takes the hardware characteristics into consideration, eliminates the disadvantages of rule partition based on *small fields* division, and takes advantage of parallel computing. The main contributions are as follows:

- KickTree does not adopt a TSS and builds multiple evenly distributed decision trees in a recursive manner, which can tap the advantages of FPGA inherent parallelism. Each tree is searched in parallel.
- This algorithm breaks the restriction of static subset partition, merges all header fields into the bit-selection range at the same time, and dynamically selects bits to establish decision trees without rule duplication.
- The maximum tree depth (i.e. number of intermediate node levels) and the number of rules contained in each leaf node are both strictly limited, so that each tree is constructed in an equalized manner, which balances the search time of each tree in the hardware and reduces the bottleneck effect.

Preliminary experimental results show that, a limited number of evenly distributed subtrees can be generated in KickTree. Compared with other latest decision tree schemes: CutSplit [24] and TabTree, KickTree has a significant reduction in the number of memory accesses for classification. Moreover, even for rule sets up to 100k entries, KickTree can still construct shallow decision trees with limited subsets number, which is hardware-friendly for FPGA implementation.

Table 1: Example rule set with four IPv4 header fields

rule id	*priority*	SA	DA	SP	DP	*action*
R_1	13	228.128.0.0/9	124.0.0.0/7	119:119	0:65535	*action1*
R_2	12	223.0.0.0/9	38.0.0.0/7	20:20	1024:65535	*action2*
R_3	11	175.0.0.0/8	0.0.0.0/1	53:53	0:65535	*action3*
R_4	10	128.0.0.0/1	37.0.0.0/8	53:53	1024:65535	*action4*
R_5	9	0.0.0.0/2	225.0.0.0/8	123:123	0:65535	*action5*
R_6	8	107.0.0.0/8	128.0.0.0/1	59:59	0:65535	*action6*
R_7	7	0.0.0.0/1	255.0.0.0/8	25:25	0:65535	*action7*
R_8	6	106.0.0.0/7	0.0.0.0/0	0:65535	53:53	*action8*
R_9	5	160.0.0.0/3	252.0.0.0/6	0:65535	0:65535	*action9*
R_{10}	4	0.0.0.0/0	254.0.0.0/7	0:65535	124:124	*action10*
R_{11}	3	128.0.0.0/2	236.0.0.0/7	0:65535	0:65535	*action11*
R_{12}	2	0.0.0.0/1	224.0.0.0/3	0:65535	23:23	*action12*
R_{13}	1	128.0.0.0/1	128.0.0.0/1	0:65535	0:65535	*action13*

The rest of the paper is organized as follows. Section 2 summarizes background and related work briefly. Section 3 presents the technical details of KickTree. Section 4 shows preliminary experimental results. Finally, Section 5 draws the conclusion and future work.

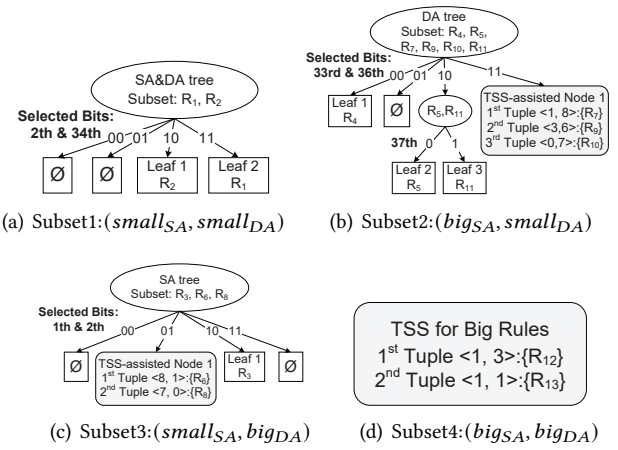
2 BACKGROUD AND RELATED WORK

2.1 The Packet Classification Problem

Packet classification is classifying network traffic in fine granularity according to multi-domain packet header information and a pre-established classifier which consists of a set of rules. Each rule r has d components represented by r_i. r_i is a regular expression on the i field of the packet header, which could be prefix, range or exact value. A packet $p = (p_1, p_2, ..., p_d)$ is said to match rule r if $\forall i, p_i \in r_i$. Table 1 shows an example rule set with four IPv4 header fields. Priority indicates the degree of importance, meaning that if a packet conforms to more than one rule, low priority rules would give way to high priority rules. Packet classification has been extensively researched in last two decades [40] with numerous algorithmic approaches proposed, such as decision tree [1, 4, 5, 7, 9, 14–17, 20, 23, 24, 32, 36, 36, 43, 46, 48], decomposition [2, 11, 13, 21, 39, 41, 47], and TSS [6, 28, 35, 37, 38]. Since KickTree is a decision tree algorithm for FPGA hardware design, the following subsections mainly review the decision tree algorithms and the related work of FPGA-based packet classification.

2.2 Decision Tree based Algorithms

Decision tree based methods involve cutting the search space recursively into several smaller sub-regions based on the information from one or more fields in the rule, until the number of rules in each region is lower than a certain threshold (i.e., *binth*). The key values in the packet header are used to search in the tree until the leaf of the decision tree is found,



(a) Subset1:($small_{SA}, small_{DA}$) (b) Subset2:($big_{SA}, small_{DA}$)

(c) Subset3:($small_{SA}, big_{DA}$) (d) Subset4:(big_{SA}, big_{DA})

Figure 1: Classifier based on TabTree (*binth* = 1)

which contains the rules or sub rule set that can match this packet. According to the partitioning method on space, current decision trees can be categorized into point-comparing based splitting such as HyperSplit [32] and bit-selecting based cutting such as HiCuts [14] and HyperCuts [36].

Although these methods can achieve high-speed packet classification, rule replication is the key trouble-maker for decision trees due to the case that a rule spans multiple sub-spaces. Rule replication not only causes a large amount of memory consumption, but also results in slow and complicated rule updates. To reduce rule replications, rule partitioning has been recognized as a common practice and plenty of novel partition based decision trees have been proposed in the past decade, such as EffiCuts [44], HybridCuts [23], SmartSplit [16], PartitionSort [49], CutSplit [24], NeuroCuts [27], CutTSS [26] and NeuvoMatch [34]. However, most of them do not take into account hardware characteristics and are unsuitable for FPGA implementation.

2.3 Hardware-specific Solutions

Although the FPGA has been increasingly recognized as a promising alternative to TCAM-based solutions in the last decade, existing FPGA-based packet classifiers cannot satisfy the burgeoning needs from OpenFlow switches such as large-scale rule sets and frequent rule updates. Specifically, although BV decomposition based method supports dynamic rule updates [10, 19, 22, 33], the scale of rule sets is restricted by FPGA logic resource, since it consumes a large amount of distributed RAMs. Most decision tree algorithms are designed for software with imbalanced and unbounded depth, which not only results in inefficient optimizations on FPGA but also makes the dynamic update difficult. Besides, the characteristics of hardware are different from software, so the migration and mapping process from software to hardware will sacrifice some intrinsic advantages.

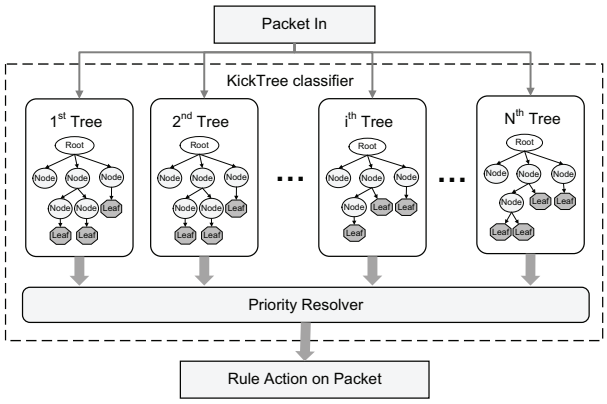


Figure 2: The framework of KickTree

To address this issue, the newly proposed TabTree [25] designs a controllable number of balanced subtrees with low memory footprints, which takes into account the characteristics of FPGAs. Figure 1 shows the classifier based on TabTree with rules in Table 1. Nevertheless, it still faces many challenges in actual implementation, which have been discussed in Section 1. Therefore, the demands of FPGA design for decision trees are refined as follows: i) shallow tree depth; ii) few number of rules in each leaf node; iii) multiple subtrees taking advantage of the multi-concurrency feature of hardware; iv) maximum balance among subtrees to reduce performance bottlenecks.

3 OUR PROPOSED APPROACH

3.1 Ideas & Framework

Decision tree algorithms generally have uncontrollable tree depth, or the number of rules in leaf nodes is not fixed. The depth determines the node search latency, while the latter determines the linear search latency in terminal nodes. These two variables interact and influence each other. For example, limiting the maximum tree depth will increase the number of terminal node rules, and conversely, limiting the number of terminal node rules will expand the levels of intermediate nodes. However, from the perspective of hardware, many decision trees with a fixed depth and a small leaf node rule number are more preferred than a small number of deep and bulky trees, since the hardware supports concurrent operations of multiple trees, and the tree with the worst performance will become the bottleneck of overall algorithm.

Based on this observation, KickTree adopts a balanced concept to build decision trees instead of empirical and static partitioning of rules subsets. In this approach, we break the restriction of partitioning rule subsets, gather all possible header fields together as a bit-selection pool, and dynamically extract valid bits (not wildcards) each time to build a decision tree in a recursive manner. Before building the tree, the maximum depth and the threshold for the number of rules in each leaf node (*binth*) are specified to make worst-case

bounded. In the process of building the tree, the local optimal principle is used to select bits sequentially, and the rules that do not meet the bit-selecting conditions (e.g. the value of the rule in the selection position is a wildcard) or exceed the leaf node rule threshold are removed from current tree. After the current tree is built, if there are remaining rules, we continue to build the decision tree in the same way and retain the rules of being kicked out for constructing the next level of tree. This process continues until there are no rules left. The framework of KickTree is shown in Figure2.

3.2 Bit-selecting Decision Tree

As each non-wildcard bit can map rules into at most two subsets without any rule replications. To exploit this favorable property, a multi-way tree could be built by selecting a few non-wildcard bits in each tree node recursively. Choosing more bits for each node to be divided can increase the number of forks and reduce the depth of the tree. However, too many bits will increase the logic and storage resources of the hardware and cause performance degradation. Therefore, the choice of the number of bits is a matter of trade-off.

To control the width of the tree, we assume that at most b bits are allowed to be selected in each tree node. In the process of tree construction, the heuristic local optimal strategy bit-selecting algorithm is utilized to select the most distinguishing non-wildcard bits, in order to build shallow and balanced decision trees.

Local Optimal Strategy: This method selects the "good" bits one by one and tries to find the most balance for each bit. An *imbalance* value is assigned for each current unused bit by using eq. (1), where *#L_Child*/*#R_Child* is the number of rules mapped into the left/right child node (i.e., #bit-0/1s in v-th bit). The local optimal algorithm selects at most b bits in each round, where each selected single bit has the minimum *imbalance* value among the currently unused bits.

$$imbalance(bit\ v) = |\#L\_Child - \#R\_Child| \quad (1)$$

3.3 Evenly-distributed Tree Construction

The construction process of KickTree is shown in Figure 3. The classifier construction starts from building the first tree with the complete rule set as the root node.

In one of the following situations, the method stops the bit selection process: 1) The tree depth reaches the predefined maximum value; 2) The number of rules in the tree node is less than the predefined threshold *binth*; 3) The remaining unselected rule bits share the same value and cannot further separate the rules from each other.

The rules would be "kicked" out of current tree in one of the following two cases: 1) when current node is dividable, the value of rules is a wildcard in the bit determined based on the local optimal strategy; 2) when the current node is indivisible which means the node would be a leaf node, and the number of associated rules exceeds *binth*, the rules whose priorities lag behind the top *binth* rules will be removed.

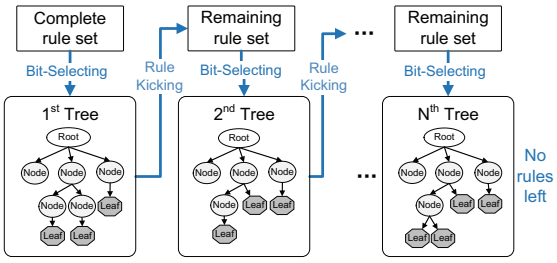


Figure 3: The construction process of KickTree

This recursive manner might result in multiple decision trees with evenly distributed depth and leaf node rule number. These trees could be implemented on FPGA which run simultaneously to perform packet search. By minimizing the search delay among different decision trees, this even feature can improve the overall classification result generation speed, thereby preventing the so-called bottleneck effect.

3.4 A Working Example

This subsection illustrates a KickTree classifier construction example for the 13 rules given in Table 1. Assume that the maximum tree depth is two, 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. Port fields are simply transformed to Longest Common Prefix (LCP) as in Table 2. The details of LCP can be referred to [25].

The process starts from building the first tree with the complete rule set. The selected bits for dividing root node are in 1st and 33rd, which would remove R_8 and R_{10} as their 33rd bit or 1st bit is wildcard. This cutting generates three valid nodes. The first valid node $\{R_5, R_6, R_7, R_{12}\}$ chooses 74th and 75th bits to generate two leaf nodes where R_{12} is removed since its corresponding bits are wildcards. The cutting bits for the second valid node $\{R_1, R_2, R_3, R_4\}$ are the same which generate three valid nodes including two leaf nodes. The intermediate node $\{R_3, R_4\}$ reaches the maximum tree depth and the number of rules exceeds *binth*, so the higher priority rule R_3 remains as a leaf node. With the rules removed from the first tree as the root node, the second tree is built and the rule R_{10} is removed to build the third tree. Then no rules are left and the process of classifier construction is done. The constructed KickTree classifier is illustrated in Figure 4.

3.5 Packet Classification & Rule Update

3.5.1 Classification. The classification mechanism for KickTree is similar to that of other multiple decision tree approaches. As shown in Figure 2, the incoming packet searches in all subtrees and the results are collected to choose the rule with the highest priority. The packet search procedure is serial in software and parallel in hardware implementation.

Table 2: Selectable bit for example rule set

rule id	src_addr (SA) 1-32th bits	dst_addr (DA) 33-64th bits	src_port (SP) LCP 65-80th bits	dest_port (DP) LCP 81-96th bits
R_1	111001001*********************************	0111110*********************************	0000000001110111	**********
R_2	110111110*********************************	0010011*********************************	0000000000010100	**********
R_3	10101111*********************************	0*********************************	0000000000110101	**********
R_4	1*********************************	00100101*********************************	0000000000110101	**********
R_5	00*********************************	11100001*********************************	0000000001111011	**********
R_6	01101011*********************************	1*********************************	0000000000111011	**********
R_7	0*********************************	11111111*********************************	0000000000011001	**********
R_8	0110101*********************************	*********************************	**********	0000000000110101
R_9	101*********************************	111111*********************************	**********	**********
R_{10}	*********************************	1111111*********************************	**********	0000000001111100
R_{11}	10*********************************	1110110*********************************	**********	**********
R_{12}	0*********************************	111*********************************	**********	0000000000010111
R_{13}	1*********************************	1*********************************	**********	**********

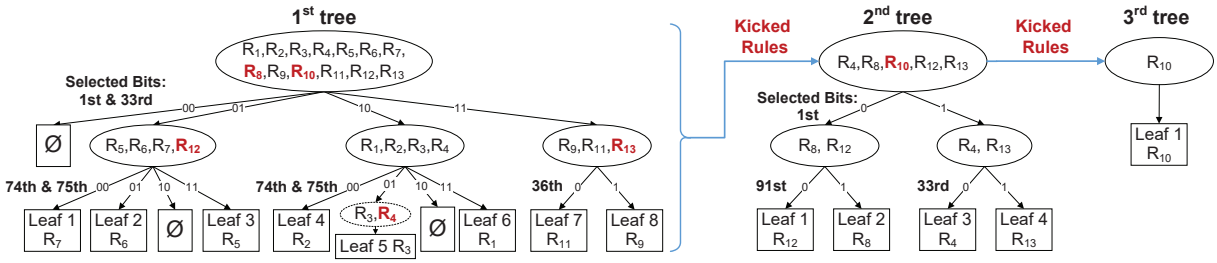


Figure 4: A working example of KickTree

3.5.2 Update. For rule deletion, the process is relatively simple. In software, the trees are traversed and searched from the first one. Thanks to the fact that rule replication does not exist in KickTree, once the tree where the rule is located is found, the rule would be deleted from this tree and the remaining trees do not need to be searched further. In hardware, all trees are traversed in parallel and only the tree where the rule is located would execute the delete operation. For insertion, search starts from the first tree. If the rule number of the leaf node to be inserted has already reached *binth*, or the current selecting bit is wildcard, then the next tree is entered to search until a suitable tree is found to insert. If existing trees do not meet the conditions, a new subtree will be created. The update in hardware could be in a cascaded manner as software.

4 PRELIMINARY EVALUATION

4.1 Experiment Methodology

In this section, we evaluate KickTree and two recently proposed decision tree based approaches: CutSplit [24] and TabTree [25]. Different schemes are evaluated from the following key aspects: number of rule subset, memory footprint, memory access and incremental update performance. For the last three evaluations, the maximum tree depth, selection bit number and *binth* are set to 10, 4, 8 respectively. All experiments are run on a PC with Intel Core i7 CPU@3.20GHz.

Three types of rule sets are generated by ClassBench [42] using default parameters, which are ACL, FW and IPC. The rule set size varies from 1k, 10k to 100k. For each size, 12 rule sets based on 12 seed parameter files (i.e, 5 ACL, 5 FW, and 2 IPC) are generated in ClassBench.

The source code of KickTree can be downloaded from the website [12], as well as the GitHub [45].

4.2 Subset Number

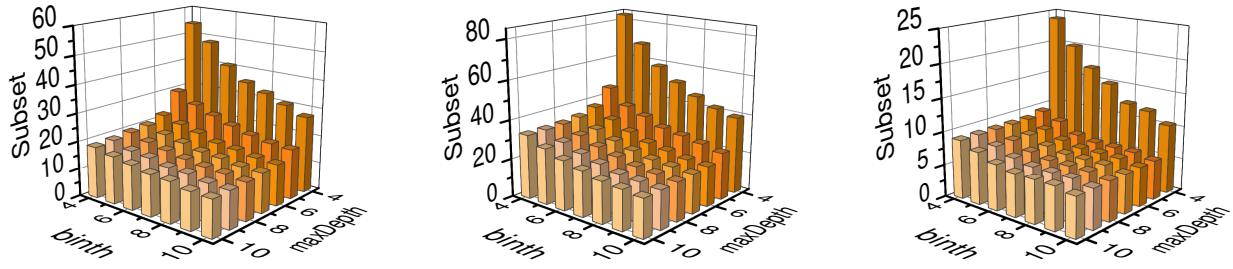
The subset number is the subtree number in KickTree, which is determined by a set of parameters, such as maximum tree depth and *binth*. The maximum selection bit number is fixed to 4. Figure 5 shows the generated subtree number in KickTree under different parameter combinations for 100k rules. Obviously, KickTree can produce a relatively stable number of subsets by adjusting the construction parameters for large size rule sets.

4.3 Memory Footprint

Figure 6 shows the memory footprint of KickTree and another two algorithms. Experimental results show that KickTree requires storage space comparable to other algorithms and the memory consumption increases almost linearly with the rule set size. Even for rule sets up to 100k entries, KickTree can still construct decision trees in a few MegaBytes, which is small enough to fit in the on-chip RAM (BRAM or URAM) of mid-range FPGAs.

4.4 Memory Access

Traversing a decision tree node, a rule or a tuple table is treated as one memory access in our evaluation. Memory access is classified into average memory access and worst-case memory access, the former refers to the tree average depth and the latter refers to the maximum tree depth plus leaf depth. Since each subtree runs concurrently in hardware, the overall performance mainly depends on the tree with



(a) ACL_100k (b) FW_100k (c) IPC_100k

Figure 5: Number of subsets for different rule sets

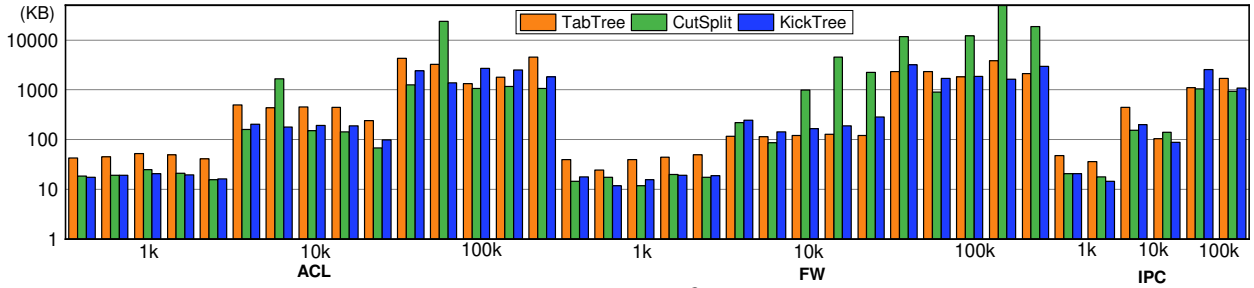


Figure 6: Memory footprint

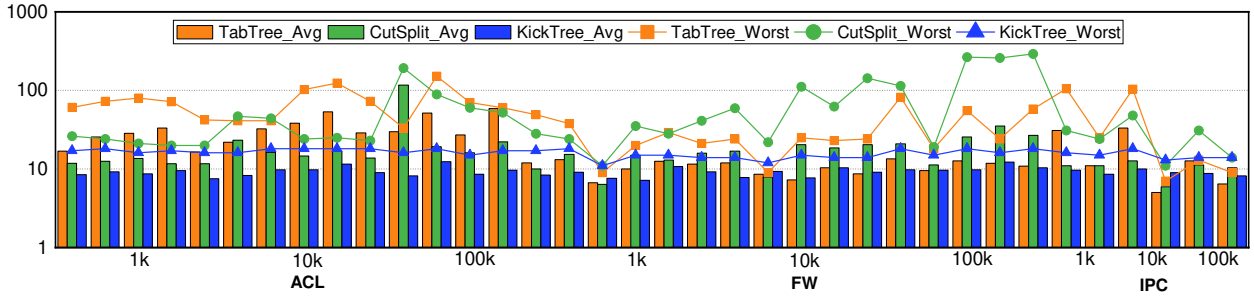
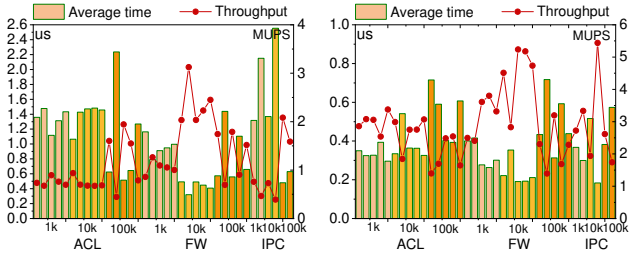


Figure 7: Average memory access and worst-case memory access

the worst performance. Thus we compare the trees with the worst memory access among the algorithms. Figure 7 shows the memory access of KickTree and another two algorithms. It is obvious that KickTree is significantly better than others in most rule sets, in terms of average access and worst-case access. Specifically, compared with TabTree and CutSplit, KickTree achieves 1.2 times and 0.99 times reduction in aspect of average access, and reaches 2.14 times and 3.09 times improvement in aspect of worst-case access, on average.

4.5 Incremental Update Performance

The incremental update time is measured as the time required to execute a rule insertion or deletion. For each rule set, we generate a series of update operations by randomly shuffling the rules. Since CutSplit does not support incremental rule update, KickTree is only compared with TabTree. Figure 8 shows that KickTree has achieved an average of 1.21 MUPS (Millions of Update Per Second) in software simulations, which is in the same order of magnitude as that of TabTree and is sufficient in actual application scenarios.



(a) KickTree (b) TabTree

Figure 8: Update Performance

5 CONCLUSION AND FUTURE WORK

Aiming at customizing an FPGA-friendly updatable packet classifier, we propose KickTree, an evenly distributed and worst-case bounded decision tree scheme designed by taking the hardware features into consideration. In the future, KickTree will be improved with more balanced rule mapping and a smaller number of subtrees by reinforcement learning. Moreover, the corresponding hardware architecture will be designed and implemented on FPGA.

REFERENCES

[1] Florin Baboescu, Sumeet Singh, and George Varghese. 2003. Packet classification for core routers: Is there an alternative to CAMs?. In *IEEE INFOCOM*.

[2] Florin Baboescu and George Varghese. 2001. Scalable packet classification. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 199–210.

[3] Gordon Brebner. 2009. Packets everywhere: The great opportunity for field programmable technology. In *IEEE FPT*.

[4] Yeim-Kuan Chang. 2008. Efficient multidimensional packet classification with fast updates. *IEEE Trans. Comput.* 58, 4 (2008), 463–479.

[5] Yeim-Kuan Chang and Chun-Sheng Hsueh. 2015. Range-enhanced packet classification design on FPGA. *IEEE Transactions on Emerging Topics in Computing* 4, 2 (2015), 214–224.

[6] James Daly and Eric Torng. 2017. TupleMerge: Building Online Packet Classifiers by Omitting Bits. In *IEEE ICCCN*.

[7] James Daly and Eric Torng. 2018. ByteCuts: Fast Packet Classification by Interior Bit Extraction. In *IEEE INFOCOM*.

[8] Andreas Fiessler, Sven Hager, Björn Scheuermann, and Andrew W Moore. 2016. HyPaFilter: A versatile hybrid FPGA packet filter. In *ACM/IEEE ANCS*.

[9] Jeffrey Fong, Xiang Wang, Yaxuan Qi, Jun Li, and Weirong Jiang. 2012. ParaSplit: A scalable architecture on FPGA for terabit packet classification. In *IEEE Hot Interconnects*.

[10] Thilan Ganegedara and Viktor K Prasanna. 2012. StrideBV: Single chip 400G+ packet classification. In *IEEE HPSR*.

[11] Filippo Geraci, Marco Pellegrini, Paolo Pisati, and Luigi Rizzo. 2005. Packet classification via improved space decomposition techniques. In *IEEE INFOCOM*.

[12] GitHub. 2021. https://github.com/wenjunpaper/KickTree.

[13] Pankaj Gupta and Nick McKeown. 1999. Packet classification on multiple fields. *ACM SIGCOMM Computer Communication Review* 29, 4 (1999), 147–160.

[14] Pankaj Gupta and Nick McKeown. 1999. Packet classification using hierarchical intelligent cuttings. In *IEEE Hot Interconnects*.

[15] P. Gupta and N. McKeown. 2000. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro* 20, 1 (2000), 34–41.

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

[17] Cheng-Liang Hsieh and Ning Weng. 2016. Many-field packet classification for software-defined networking switches. In *ACM/IEEE ANCS*.

[18] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4-NetFPGA Workflow for Line-Rate Packet Processing. In *ACM/SIGDA FPGA*.

[19] Weirong Jiang and Viktor K Prasanna. 2009. Field-split parallel architecture for high performance multi-match packet classification using FPGAs. In *ACM SPAA*.

[20] Weirong Jiang and Viktor K Prasanna. 2012. Scalable Packet Classification on FPGA. *IEEE Transactions on Very Large Scale Integration Systems* 20, 9 (2012), 1668–1680.

[21] TV Lakshman and Dimitrios Stiliadis. 1998. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *ACM SIGCOMM*.

[22] Chenglong Li, Tao Li, Junnan Li, Zilin Shi, and Baosheng Wang. 2020. Enabling Packet Classification with Low Update Latency for SDN Switch on FPGA. *Sustainability* 12, 8 (2020), 1–16.

[23] Wenjun Li and Xianfeng Li. 2013. HybridCuts: A scheme combining decomposition and cutting for packet classification. In *IEEE Hot Interconnects*.

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

[25] Wenjun Li, Tong Yang, Yeim-Kuan Chang, Tao Li, and Hui Li. 2019. TabTree: A TSS-assisted Bit-selecting Tree Scheme for Packet Classification with Balanced Rule Mapping. In *ACM/IEEE ANCS*.

[26] Wenjun Li, Tong Yang, Ori Rottenstreich, Xianfeng Li, Gaogang Xie, Hui Li, Balajee Vamanan, Dagang Li, and Huiping Lin. 2020. Tuple Space Assisted Packet Classification with High Performance on Both Search and Update. *IEEE Journal on Selected Areas in Communications* 38, 7 (2020), 1555–1569.

[27] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural Packet Classification. In *ACM SIGCOMM*.

[28] Hyesook Lim and So Yeon Kim. 2010. Tuple pruning using bloom filters for packet classification. *IEEE Micro* 30, 3 (2010), 48–59.

[29] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (mar 2008), 69–74.

[30] Chad R. Meiners, Alex X. Liu, and Eric Torng. 2010. *Hardware Based Packet Classification for High Speed Internet Routers* (1st ed.). Springer Publishing Company, Incorporated.

[31] Salvatore Pontarelli and et al. 2019. FlowBlaze: Stateful Packet Processing in Hardware. In *USENIX NSDI*.

[32] Yaxuan Qi, Lianghong Xu, Baohua Yang, Yibo Xue, and Jun Li. 2009. Packet classification algorithms: From theory to practice. In *IEEE INFOCOM*.

[33] Yun R Qu and Viktor K Prasanna. 2015. High-performance and dynamically updatable packet classification engine on FPGA. *IEEE Transactions on Parallel and Distributed Systems* 27, 1 (2015), 197–209.

[34] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein. 2020. A Computational Approach to Packet Classification. In *ACM SIGCOMM*.

[35] Tong Shen, Gaogang Xie, Xin Wang, Zhenyu Li, Xinyi Zhang, Penghao Zhang, and Dafang Zhang. 2018. RVH: Range-Vector Hash for Fast Online Packet Classification. *Techical Report of ICT* (2018).

[36] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. 2003. Packet classification using multidimensional cutting. In *ACM SIGCOMM*.

[37] Haoyu Song, Jonathan Turner, and Sarang Dharmapurikar. 2006. Packet classification using coarse-grained tuple spaces. In *ACM/IEEE ANCS*.

[38] Venkatachary Srinivasan, Subhash Suri, and George Varghese. 1999. Packet Classification using Tuple Space Search. In *ACM SIGCOMM*.

[39] Venkatachary Srinivasan, George Varghese, Subhash Suri, and Marcel Waldvogel. 1998. Fast and Scalable Layer Four Switching. In *ACM SIGCOMM*.

[40] David E Taylor. 2005. Survey and taxonomy of packet classification techniques. *Comput. Surveys* 37, 3 (2005), 238–275.

[41] David E Taylor and Jonathan S Turner. 2005. Scalable packet classification using distributed crossproducing of field labels. In *IEEE INFOCOM*.

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

[43] Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2010. EffiCuts: Optimizing Packet Classification for Memory and Throughput. In *ACM SIGCOMM*.

[44] Balajee Vamanan, Gwendolyn Voskuilen, and TN Vijaykumar. 2011. EffiCuts: optimizing packet classification for memory and throughput. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 207–218.

[45] Website. 2021. http://www.wenjunli.com/KickTree.

[46] Thomas YC Woo. 2000. A modular approach to packet classification: Algorithms and results. In *IEEE INFOCOM*.

[47] Yang Xu, Zhaobo Liu, Zhuoyuan Zhang, and H. Jonathan Chao. 2014. High-Throughput and Memory-Efficient Multimatch Packet Classification Based on Distributed and Pipelined Hash Tables. *IEEE/ACM Transactions on Networking* 22, 3 (2014), 982–995.

[48] Sorrachai Yingchareonthawornchai, James Daly, Alex X Liu, and Eric Torng. 2016. A sorted partitioning approach to high-speed and fast-update OpenFlow classification. In *IEEE ICNP*.

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