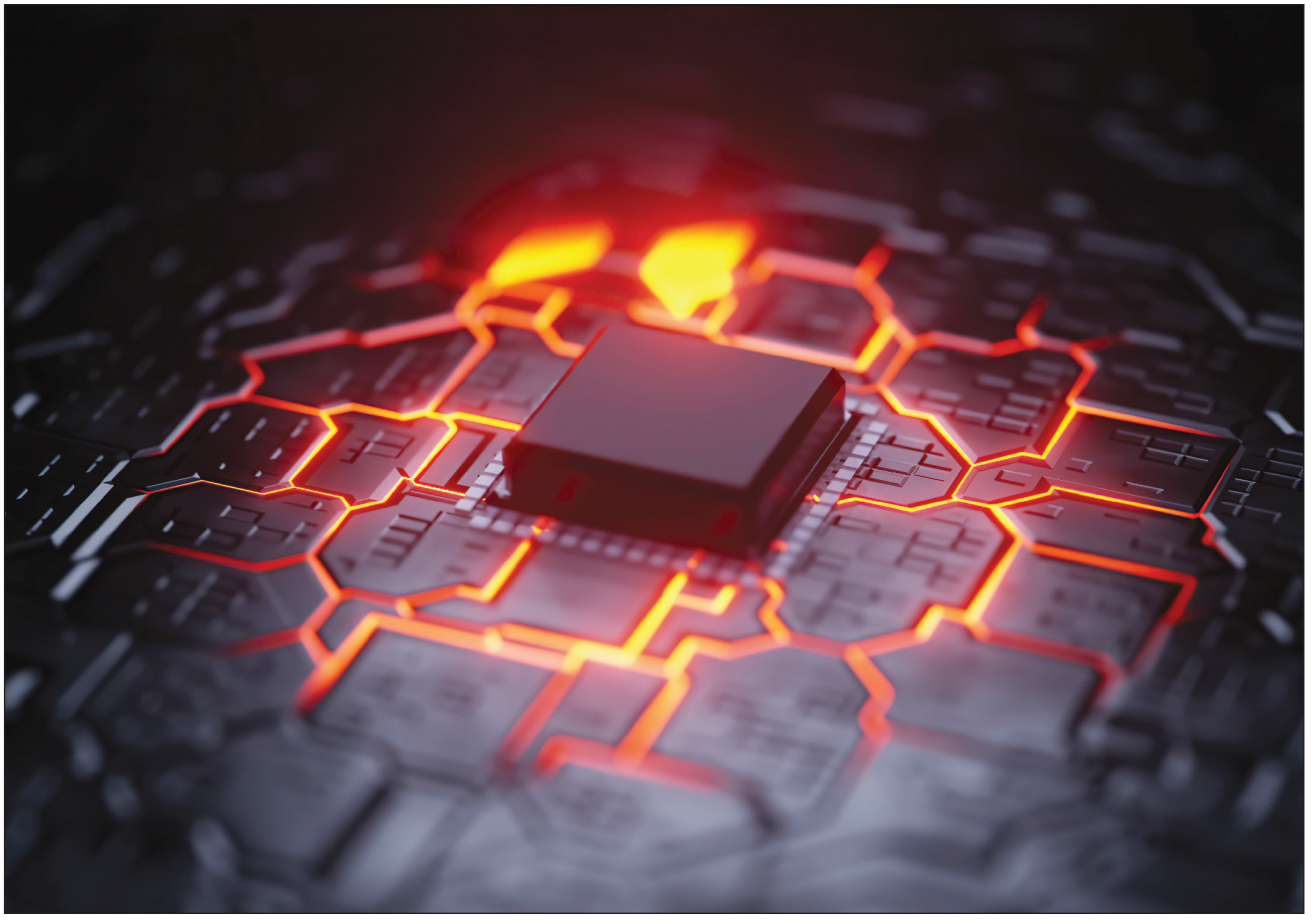
IEEE micro
The magazine for chip and silicon systems designers

VOLUME 43, NUMBER 2 MARCH/APRIL 2023



Mini-Theme: Emerging System Interconnects
Mini-Theme: Hot Interconnects 29

IEEE
IEEE COMPUTER SOCIETY
www.computer.org/micro

A Parallel and Updatable Architecture for FPGA-Based Packet Classification With Large-Scale Rule Sets

Yao Xin and Wenjun Li, *Peng Cheng Laboratory, Shenzhen, 518055, China*
Gaogang Xie, *CNIC, Chinese Academy of Sciences, Beijing, 100083, China*
Yang Xu, *Fudan University, Shanghai, 200433, China*
Yi Wang, *Southern University of Science and Technology, Shenzhen, 518055, China*

As a programmable hardware, field-programmable gate array (FPGA) provides more opportunities for algorithmic network packet classification. Despite more than 10 years of research, the most actively investigated pipeline architectures still struggle to support fast rule search and efficient rule update for large-scale rule sets. In this article, we design and implement a novel architecture for multitree-based packet classification on FPGA, where the search and update processes are decoupled. A strategy of multi-processing elements (PEs), parallel search, and serial update is adopted. The parsing of multiple tree search results adopts a modular and hierarchical design, supporting architecture with various tree numbers. In addition, incremental rule updates can be achieved simply by traversing all PEs in one pass, with little and bounded impact on rule searching. Compared with TcbTree, the state-of-the-art updatable classifier, the experimental results on FPGA show that the classification performance of our design improves 3.4× on average for various 100k-scale rule sets.

Network functions virtualization and software defined networking have emerged as a solution for data centers that can provide flexibility by implementing a software-based network over physical infrastructure.[1] Virtualization has enabled hundreds of virtual machines per server in data centers using multicore central processing unit (CPU) technology. However, x86 servers not optimized for packet processing are inefficient for software implementation. Under such circumstances, SmartNIC, a programmable hardware accelerator, has been increasingly adopted to offload the frequently updated network functions of the virtual network protocol stack. Among these, packet classification is a fundamental and essential task, which is to discriminate packets into separate "flows" and enable differentiated functionalities, so that all packets belonging to the same flow will be processed similarly.

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3238012
Date of publication 19 January 2023; date of current version 13 March 2023.

Field-programmable gate array (FPGA) has been regarded as promising hardware for line-speed packet classification in SmartNIC, due to its ability to reconfigure and offer massive parallelism. The majority of FPGA-related solutions are fully pipelined, which can benefit from the high throughput. However, the architectures of pipelines have the disadvantage of not supporting dynamic rule updates (i.e., without precomputing the memory content) well, especially for large-scale rule sets. By introducing a multicore architecture instead of a pipeline architecture for packet classification, the recently proposed TcbTree[2] can effectively support real-time rule updates. However, it has a performance concern for large-scale rule sets. Therefore, existing FPGA-based packet classification designs are still difficult to support fast rule search and efficient rule update for large-scale rule sets.

In this article, we propose an updatable FPGA-based packet classifier for large-scale rule sets, based on our recently proposed algorithmic packet classification scheme KickTree,[3] which is a multitree algorithm dedicated to FPGA. For high lookup throughput, we

TABLE 1. Example rule set with four IPv4 fields.

Rule id	Src IP addr.	Dst IP addr.	Src port	Dst port	Action
R_1	228.128.0.0/9	124.0.0.0/7	119:119	0:65535	action1
R_2	223.0.0.0/9	38.0.0.0/7	20:20	1024:65535	action2
R_3	175.0.0.0/8	0.0.0.0/1	53:53	0:65535	action3
R_4	128.0.0.0/1	37.0.0.0/8	53:53	1024:65535	action4
R_5	0.0.0.0/2	225.0.0.0/8	123:123	0:65535	action5
R_6	107.0.0.0/8	128.0.0.0/1	59:59	0:65535	action6
R_7	0.0.0.0/1	255.0.0.0/8	25:25	0:65535	action7
R_8	106.0.0.0/7	0.0.0.0/0	0:65535	53:53	action8
R_{19}	160.0.0.0/3	252.0.0.0/6	0:65535	0:65535	action9
R_{10}	0.0.0.0/0	254.0.0.0/7	0:65535	124:124	action10
R_{11}	128.0.0.0/2	236.0.0.0/7	0:65535	0:65535	action11
R_{12}	0.0.0.0/1	224.0.0.0/3	0:65535	23:23	action12
R_{13}	128.0.0.0/1	128.0.0.0/1	0:65535	0:65535	action13

adopt multiple processing elements (PEs) running in parallel to perform rule search, and multiple computing cores of the classifier enter into force at the top layer. To support incremental rule updates, we adopt a strategy of centralized memory and serial access in each PE rather than an entire pipeline. With respect to the organization of multiple PEs in each classifier, a method of parallel search and serial update is proposed to decouple the search and update process. Experimental results show that, for various 100k-scale rule sets, it can achieve an average classification throughput of 182.6 million packets per second (MPPS) and an update throughput of 3.1 million updates per second (MUPS). As an extension of Xin et al.,[4] our proposed design is compared with the state-of-the-art updatable classifier TcbTree,[2] the classification performance of this work is superior in terms of throughput and latency.

BACKGROUND

Packet Classification Problem

The purpose of packet classification is to classify network traffic based on a classifier that contains a set of predefined rules with priority. Each rule consists of a set of fields in exact value, prefix, or range representations, and the action to be taken when being matched. Table 1 shows an example rule set with four IPv4 header fields. As a widely studied bottleneck, packet classification has attracted extensive research attention in the past two decades.[5] Since our aim in this work is to design a novel architecture for packet classification on FPGA, based on our recently proposed algorithm KickTree, we next describe some related architecture designs and give some technical reviews of the KickTree algorithm. More related work on algorithmic and architectural solutions can be found in Xin et al.[4] and Li et al.[6]

Architecture Design Review

Generally speaking, current FPGA solutions can be divided into two categories: fully pipelined and multicore computing.

1) FPGA Solutions Based on Full Pipelines: Because the fully pipelined architecture can produce results every cycle, the classification throughput is directly related to the frequency and can achieve high values. Thus, the vast majority of FPGA designs adopt this scheme. However, limited by the algorithms they are based on, none of these architectures currently supports both large-scale and real-time rule updates. Specifically, the presence of rule duplication and irregular tree structure in decision tree-based FPGA architectures hinders their support for dynamic rule updates.[7] On the contrary, although decomposition-based designs can fully support dynamic updates,[8,9] the method is essentially exhaustive in listing all possible matching combinations for each bit. Therefore, the consumption of hardware resources is considerable, especially for rules with more wildcards. This feature always restrains the scale of rule sets accommodated by FPGAs.

2) FPGA Solutions Based on Multicore Computing: Another paradigm not utilizing full pipelines adopts a strategy of multiple computing cores or engines, which can leverage centralized memory to facilitate

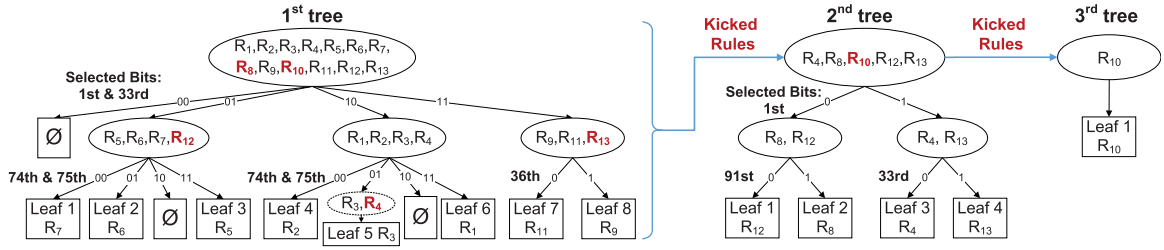


FIGURE 1. Working example of KickTree, with max depth = 2, max selecting bits = 2, binth = 1.

rule updates. The design in Kennedy and Wang[10] used multiple packet classification engines working in parallel with shared memory, allowing it to classify packets at high speed. However, it has no dynamic updating function, as the HyperCuts algorithm it adopts does not eliminate rule duplication. TcbTree[2] is able to support dynamic updates without rule replication, but it encounters a performance bottleneck in support of large-scale rules, due to the considerable tree depth and the large number of rules that cannot be included in the tree using hash lookups. The latest updatable architecture for KickTree,[4] which is the preliminary version of this article, achieves good performance for large-scale rules. However, it lacks peer-to-peer experimental comparisons with other updatable classifiers, which cannot prove the innovation and improvement of performance.

KickTree Algorithm Review

KickTree first converts each range field into a longest common prefix,[2] allowing each rule to be represented by a sequence of ternary strings (i.e., 0, 1, wildcard). After that, several balanced trees of bounded depth are constructed in a recursive manner, consisting of the following two key steps.

1) Tree building by bit selecting. Nonwildcard bits are dynamically extracted from all possible header fields using a locally greedy strategy for shallow and balanced tree purposes.
2) Rule sifting. Rules that do not meet the bit selection conditions (e.g., the rule value for the selected bit position is a wildcard) or rules that exceed binth (the threshold for the number of rules in a leaf node) in leaf nodes are "kicked out" from the current tree. If there are still rules left after building the current tree, the same method would be utilized to construct the decision tree continuously, and the kicked-out rules would be retained for building the next tree. This process continues until there are no more rules.

Figure 1 illustrates a working example for the rules given in Table 1.

TO FULLY SUPPORT DYNAMIC RULE UPDATES WITHOUT SACRIFICING LOOKUP PERFORMANCE FOR LARGE-SCALE RULE SETS, WE PREFER TO DESIGN A NEW HARDWARE ARCHITECTURE FROM SCRATCH RATHER THAN USE A CLASSIC PURE PIPELINE DESIGN SO THAT THE POTENTIAL OF THE KickTree ALGORITHM CAN BE FULLY UNLEASHED.

HARDWARE DESIGN

Design Overview

While the KickTree algorithm appears to be a good fit for FPGA, there are still many challenges in concrete FPGA implementation because the most widely studied pure-pipeline designs can hardly support dynamic rule updates without precomputing memory content. In addition, as a multitree scheme, how to efficiently collect and parse the results of multiple trees and how to cope with the relationship between classification and update remains challenging. Therefore, to fully support dynamic rule updates without sacrificing lookup performance for large-scale rule sets, we prefer to design a new hardware architecture from scratch rather than use a classic pure pipeline design so that the potential of the KickTree algorithm can be fully unleashed. Next, we will introduce the top-level architecture and storage organization dedicated to KickTree. After that, the detailed architectural design of each search tree PE is elaborated, followed by a hierarchical concurrent result

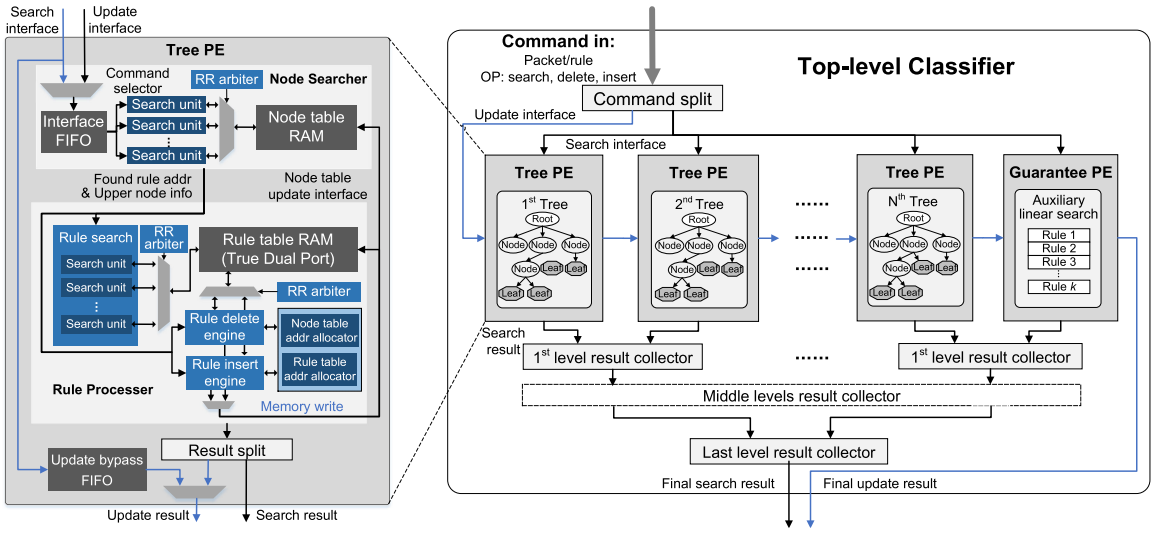


FIGURE 2. Architecture of the classifier.

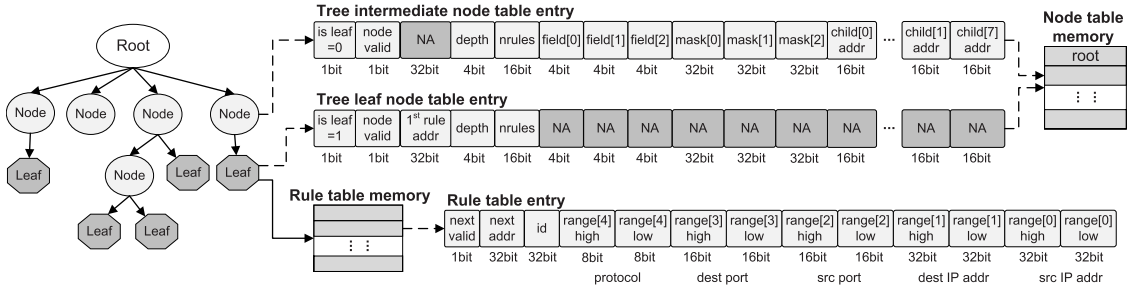


FIGURE 3. Storage organization of search tree.

collection scheme. Finally, we present the rule update mechanism.

Top-Level Architecture

The top-level architecture of each classifier adopts a parallel-search, serial-update strategy, as illustrated on the right-hand side of Figure 2. Each PE corresponds to a tree generated in KickTree. Each PE processes rule search and update separately. As a result, there are two interfaces for receiving commands for rule search/updates and two interfaces for exporting search/update results, respectively. The input command comprises a packet/rule along with an operation code of SEARCH (for packet) or DELETE/INSERT (for rule), while the result consists of the matched rule ID and result code of RULE_FOUND, RULE_NOT_FOUND, UPDATE_SUCCESS, UPDATE_FAILURE, etc.

The top-level classifier handles the two input commands differently: the search command is delivered to all PEs for parallel execution, whereas the update command is only distributed to the first PE and executed serially in subsequent PEs. Up until the last resolver provides the final result, the results of all PE searches are parsed and merged in pairs by the result resolver level by level.

The update results, however, go through each PE in turn. The rule update operation is continued in the current PE if the update in the preceding PEs fails, and the result is passed until the update is successful in a particular PE. From a high-level perspective, each classifier can operate independently. Thus, as long as hardware resources allow, multiple computing cores can enter into force on the FPGA to improve the overall performance.

Search Tree Storage Organization

As shown in Figure 3, the storage organization of each search tree is cached in two centralized memories: node table random access memory (RAM) and rule table RAM. The entries of these memories

constitute three types of unidirectional chained lists. The intermediate node table and leaf node table entries represent intermediate nodes (including the root node) and leaf nodes, respectively. Moreover, each rule table entry represents a rule. Each leaf node is associated with a rule subset, and the collection of subsets of all leaf nodes is cached in the rule table RAM.

Search Tree PE

Instead of a fully pipelined design, the architecture of the search tree PE adopts a centralized memory and serial access method, as illustrated on the left-hand side of Figure 2. The node searcher traverses the tree nodes from the root by reading linked node table entries from memory. When a valid leaf node is found, the rule subset address will be delivered to the rule processor, which will linearly search the rule table RAM and take the appropriate actions of search, delete, or insert, according to the operation code. To enhance memory utilization, we implement multiple search units to work in parallel since they can access the memory in different time slots with limited query time. Furthermore, round robin ensures the arbitration between multiple units in both modules as each unit has the same priority.

PEs are able to process rules and packets similarly by sharing hardware resources. In particular, the node searcher processes rules using the lower endpoint of the range identical to the packet format as input. Meanwhile, the rule processor handles the specific operations of packets (search) and rules (delete/insert) with different modules. In addition, each PE implements an update bypass first-in–first-out (FIFO), which caches successful update results and forward them, bypassing the tree. The node searcher, in contrast, processes only unsuccessful update results in preceding PEs. This mechanism avoids repeated updates of multiple PEs and reduces the delay of serial updates.

Hierarchical Concurrent Results Collection

For the purpose of rapid development and hardware implementation of a classifier with arbitrary tree numbers, we introduce the idea of modularity in the top-level architecture design since the number of trees can be random and unpredictable for any given set of rules. We employ a multilevel result resolver with a 2-input result resolver as the fundamental component to address this challenge. Accordingly, the results of the search trees are parsed hierarchically in pairs.

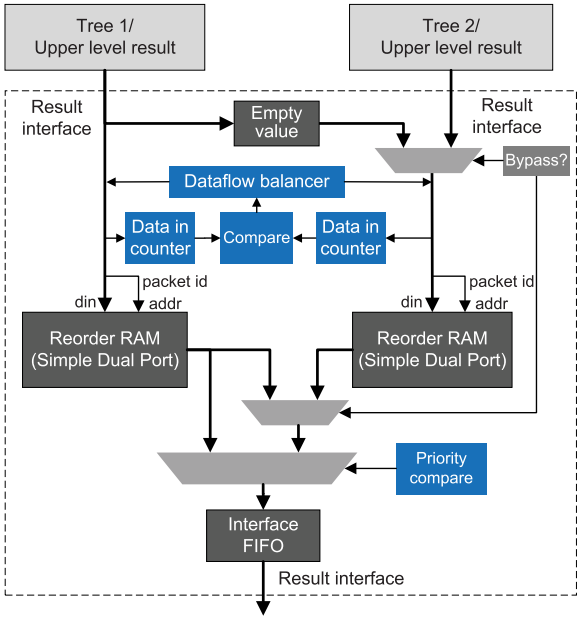


FIGURE 4. Modularized result resolver.

Empty trees are used to supplement the number of search trees to powers of 2.

The architecture of the elementary 2-input result resolver is shown in Figure 4. It aims to solve two issues as follows.

1) The classifier may produce out-of-order search results due to the existence of multiple search units and the different processing delays of each packet.
2) The individual trees produce results at different rates.

According to the first issue, each input channel has an independent RAM to reorder the out-of-order results, and the low-order bits of the packet ID act as the write address. Once a predetermined number of results have been written, the results of both channels are concurrently read in order and compared by priority. Rules with higher priority are output to the FIFO controlled by the bus interface to the next level module. The second issue is tackled by a dataflow balancer, which dynamically monitors the amount of data flowing into the two channels and controls the bus interface in real time so that the input speed of the results on both sides is roughly equal. For the purpose of resource saving, the second channel can be set to bypass mode, which is activated when the second channel is connected to an empty tree. In bypass mode, the reorder RAM will not be implemented.

Rule Update Mechanism

The architecture supports real-time incremental rule updates without precomputing memory contents as in pipelined designs. When performing rule updates in each PE, the node searcher caches the complete information of up to two levels of traversed nodes. Therefore, we can trace back two levels of nodes to modify the contents of related tables.

From the top-level perspective of the classifier, update commands and results pass sequentially through each PE. If the result of the previous PE has been successful, no command will be passed to the tree. Instead, it will be cached directly in the bypass FIFO along with the result and continue to pass down until they are read from the last PE. The packet classification procedure must be interrupted prior to the update operation. Moreover, it is also preferable to wait until the update command has traversed all PEs before sending the subsequent one for the sake of maintaining atomic consistency.

Due to the hardware's difficulty in achieving the exact dynamic tree reconstruction as in software, there may be an update failure: a rule cannot be inserted into any tree. To prevent this from happening, the last PE of each classifier employs a linear search without the restriction of binth to accommodate previously inserted failing rules. Figure 2 displays this feature. Nevertheless, this is merely a guarantee mechanism. In practice, it is rare for all previous PE updates to fail, especially when the number of PEs is large.

FPGA IMPLEMENTATION RESULT

Experimental Setup

ClassBench[11] is utilized to generate three types of rule sets (i.e., ACL, FW, and IPC). The 100 scale is generated as the large-scale rule set because it is widely recognized as large scale for packet classification work, and it is also challenging for resource-precious FPGA. The design is developed by Vivado 2021.2 tool and evaluated on a Xilinx UltraScale+ VU9P FPGA. The performance evaluations are conducted by simulation instead of testing on the actual FPGA board.

Parameter Customization

In the case of resource constraints, a set of parameters needs to be customized to maximize performance. Specifically, the maximum tree depth and binth would affect the number of generated trees. The smaller these values, the faster the search within the tree, but the more trees are generated, the more hardware resources are consumed, and vice versa. The number of selection bits is similar, because more bits correspond to more nodes in a tree level, less tree depth, and faster search speed, but more expansive node tables, more resource consumption, more complex multiplexers, and lower working frequency. In experiments, we first generate a series of parameter combinations and implement them to evaluate the performance. The parameter combination that achieves relatively high performance is selected as the final version, which is relatively balanced between performance and resource consumption.

The selection of the number of search units is in direct proportion to the memory access latency, since more search units can make better use of the time slots during the latency and improve memory utilization. When it reaches a certain value, the search performance will not be further improved by continuing to increase the number, as the time slots for reading memory are fully filled. This number is also determined by experiments, which is the minimum value that almost maximizes the storage read efficiency.

In the following evaluations, the number of selectable bits, maximum tree depth, and binth (the threshold for the number of leaf node rules) are set to 3, 8, and 10, respectively. The numbers of search units in node searcher and rule processor in each search tree PE are set to 5 and 6 separately. In addition, we further reduce the number of trees by increasing binth and the maximum depth settings for trees generated later. This adjustment is based on an observation: the first two trees concentrate most of the rules.

Storage Configuration

Because of the distinct characteristics of each rule set, the resulting storage organizations are different. To achieve the optimal performance for a specific rule set, hardware configurations for various rule sets have been customized and finely tuned. Because our hardware architecture supports real-time rule updates, sufficient spare storage space should be allocated for the search tree in each PE at the outset in case there are more insertions than deletions. Hardware configuration mainly involves how to allocate three kinds of storage resources in FPGA reasonably, namely, ultra RAM, block RAM, and distributed RAM.

The selection of the three kinds of RAM follows two principles.

1) They are selected mainly depending on the scale of the nodes or rules to be stored in each PE. Specifically, the minimum depth of ultra RAM, block RAM, and distributed RAM on FPGA is 4096, 512,

TABLE 2. Resource utilization for different rule sets.

Rule set	Core num	CLB LUTs (1,182,240)	CLB Registers (2,364,480)	BRAM (2160)	URAM (960)	Max frequency (MHz)
acl1_100k	6	607,596	819,039	1815	762	200.08
acl2_100k	6	944,525	1,263,251	2010	738	200.12
acl3_100k	6	589,951	802,639	1494	846	200.00
acl4_100k	6	573,805	741,532	1758	816	200.36
acl5_100k	6	326,950	275,717	1635	576	193.46
ipc1_100 k	6	333,575	472,205	1593	762	199.88
ipc2_100k	7	365,928	415,700	1365	684	200.56
fw1_100k	6	835,036	974,454	1716	936	192.83
fw2_100k	7	330,732	465,005	1519	812	200.20
fw3_100k	6	896,606	1,049,058	1170	564	200.68
fw4_100k	6	84,444	1,243,527	1536	600	201.01
fw5_100k	6	799,875	1,101,059	1524	744	200.76

and 64, respectively. When the number of entries is large, medium, or very small, the node table RAM or rule table RAM can be ultra RAM, block RAM, or distributed RAM, respectively.

2) The resource consumption of the three types of RAM for implementation should be balanced as far as possible. If one type of RAM is consumed too much, some of it will be replaced with other types, regardless of the actual depth requirements.

In summary, we first roughly allocate three kinds of memory according to Principle 1, and then fine-tune them according to Principle 2.

Resource Utilization

Table 2 summarizes the number of accommodated computing cores, resource usage, and maximum frequencies of hardware implementations of various rule sets after they are synthesized, placed, and routed. It is natural to note that memory, including ultra RAM and block RAM, is the most consumed FPGA resource.

Throughput Evaluation

We evaluate the performance of our FPGA implementation in terms of throughput, including packet classification throughput and rule update throughput, measured in MPPS and MUPS as a unit. We calculate both types of throughput by simulation. Figure 5(a) shows classification and update throughput with respect to the benchmark rule sets. Performance

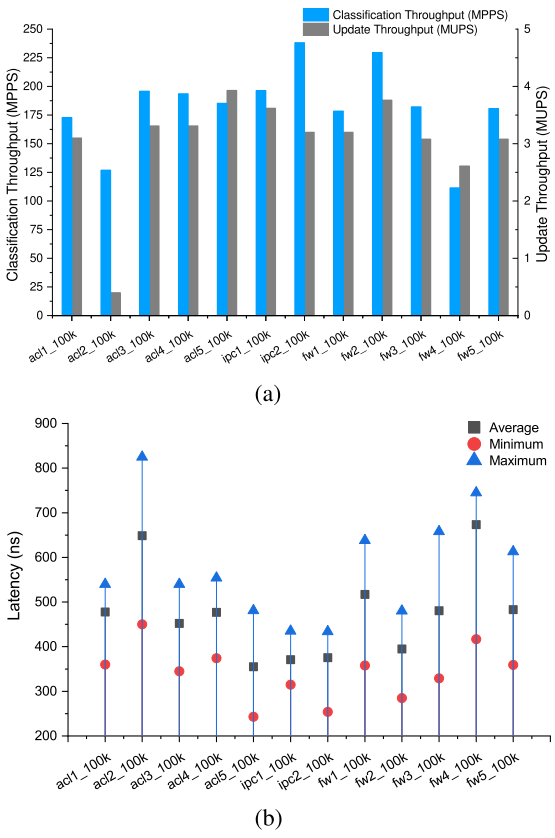


FIGURE 5. Throughput and latency evaluation for large-scale rule sets. (a) Classification and update throughput. (b) Classification latency.

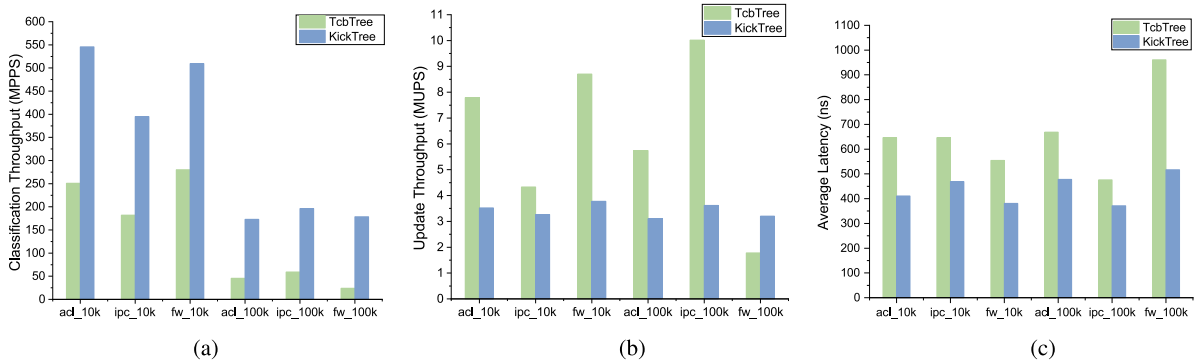


FIGURE 6. Comparison between TcbTree and the proposed KickTree architecture. (a) Classification throughput. (b) Update throughput. (c) Classification latency.

varies according to different rule types. The classification throughput is distributed between 102.3 and 238.3 MPPS, with an average of 182.6 MPPS. On the other hand, with the exception of the acl2_100k rule set, the range of update throughput does not fluctuate much, with an average of 3.1 MUPS. The extremely low update throughput of the acl2_100k rule set is due to the fact that it generates the largest number of trees, and the update is sequential, which results in a long delay.

Latency Evaluation

We have calculated different latency values per packet for benchmark rule sets, including average latency, worst case and best case latencies, and Figure 5(b) shows the results. Latency is closely related to several factors, such as the number of trees, the depth of the tree, the number of leaf node rules, and memory read latency. Although our latency is not very advantageous compared to some pure pipeline designs with only a few stages, it is a tradeoff to support dynamic rule updates. Nevertheless, we aim to reduce latency in our future work.

Comparison With Related Work

Most decomposition-based FPGA implementations only support rule set scales of no more than 5k,[8,9] as this kind of method requires a large amount of on-chip memory to implement bit vectors, although they can achieve high performance in classifying packets. As a result, these works are not comparable to our design regarding the rule set scale.

On the other hand, among multiple FPGA-based decision tree schemes, only Tan et al.[12] and Xin et al.[2] implement the evaluation of 100,000 rule sets. Although Tan et al.[12] have a higher classification performance than our implementation, it does not support rule updates, a common drawback of most decision tree based methods. Jiang and Prasanna[7] proposed the method of inserting write bubbles to pipeline memories to enable rule updates. However, the new contents of the memory are computed offline rather than changing dynamically according to the on-the-fly update orders as in our proposed method.

The recently proposed TcbTree[2] is able to support on-the-fly rule updates without the need to precompute the updated contents of the memory, similar to the preliminary version of this article.[4] These two designs adopt different subset partition strategies and have different performance. However, there is no comparison provided between the KickTree architecture and TcbTree.

To make up for this deficiency and make a comprehensive performance comparison between the two architectures, three 10k-scale rule sets are complemented and implemented by the KickTree architecture. Figure 6 shows a variety of performance comparisons between the two architectures for 10k and 100k rule sets with seed_1, in terms of classification throughput, rule update throughput, and classification latency. It can be noted that KickTree's architecture outperforms TcbTree's regarding classification throughput and latency, with 3.4× improvement and 0.5× reduction for 100k rule sets, respectively.

Although the update performance of our architecture generally lags behind that of TcbTree due to the sequential nature of our update mechanism, it is still at the level of MUPS. The exceptional low throughput of fw_100k rule set for TcbTree is due to the commonly generated big leaves (i.e., the leaf nodes contain a large number of rules that cannot be further

segmented), because the rule set has a large number of overlapping rules.

CONCLUSION

In this article, we designed and implemented an updatable hardware architecture based on the multi-tree algorithm KickTree for packet classification.

For various 100k-scale rule sets, experimental results showed that it could achieve high performance in classification because of the parallel-search design, and the update throughput could also reach the level of millions per second. In terms of classification throughput and latency for 10k- and 100k-scale rule sets, our architecture vastly outperformed the recently proposed updatable architecture TcbTree. Future work will focus on combining multicore with the fine-grained pipeline to improve classification and update performance while reducing resource consumption.

IN THIS ARTICLE, WE DESIGNED AND IMPLEMENTED AN UPDATABLE HARDWARE ARCHITECTURE BASED ON THE MULTITREE ALGORITHM KickTree FOR PACKET CLASSIFICATION.

ACKNOWLEDGMENTS

This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grants 2020B0101130003 and 2021B0101400001; in part by the National Key Research and Development Program of China under Grants 2022ZD0115303 and 2020YFB1806400; in part by NSFC under Grants 62102203, 61725206, 62150610497, and 62172108; in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2019B1515120031; in part by China Postdoctoral Science Foundation under Grants 2020TQ0158 and PC2021037; and in part by the Major Key Project of PCL under Grants PCL2021A02, PCL2021A08, and PCL2021A15.

REFERENCES

1. M. F. Bari et al., "Data center network virtualization: A survey," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 2, pp. 909–928, Apr.–Jun. 2013.
2. Y. Xin, W. Li, G. Tang, T. Yang, X. Hu, and Y. Wang, "FPGA-based updatable packet classification using TSS-combined bit-selecting tree," *IEEE/ACM Trans. Netw.*, vol. 30, no. 6, pp. 2760–2775, Dec. 2022.
3. Y. Xin, Y. Liu, W. Li, R. Yao, Y. Xu, and Y. Wang, "KickTree: A recursive algorithmic scheme for packet classification with bounded worst-case performance," in *Proc. IEEE/ACM Symp. Archit. Netw. Commun. Syst.*, 2021, pp. 23–30.
4. Y. Xin, W. Li, G. Xie, Y. Xu, and Y. Wang, "Updatable packet classification on FPGA with bounded worst-case performance," in *Proc. IEEE Symp. High-Perform. Interconnects*, 2022, pp. 21–28.
5. D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
6. W. Li, X. Li, H. Li, and G. Xie, "CutSplit: A decision-tree combining cutting and splitting for scalable packet classification," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 2645–2653.
7. W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.
8. Y. R. Qu and V. K. Prasanna, "High-performance and dynamically updatable packet classification engine on FPGA," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 197–209, Jan. 2016.
9. Y.-K. Chang and C.-S. Hsueh, "Range-enhanced packet classification design on FPGA," *IEEE Trans. Emerg. Topics Comput.*, vol. 4, no. 2, pp. 214–224, Apr.–Jun. 2016.
10. A. Kennedy and X. Wang, "Ultra-high throughput low-power packet classification," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 2, pp. 286–299, Feb. 2014.
11. D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, Jun. 2007.
12. J. Tan, G. Lv, Y. Ma, and G. Qiao, "High-performance pipeline architecture for packet classification accelerator in DPU," in *Proc. IEEE Int. Conf. Field-Programmable Technol.*, 2021, pp. 1–4.

YAO XIN is an assistant researcher with Peng Cheng Laboratory, Shenzhen, 518055, China. His research interests include high-performance VLSI design for networking and deep learning. Xin received a Ph.D. degree from the City University of Hong Kong, Kowloon Tong, Hong Kong. Contact him at xiny@pcl.ac.cn.

WENJUN LI is a postdoctoral fellow at Harvard University, Allston, MA, 02134, USA, and an associate researcher at Peng Cheng Laboratory, Shenzhen, 518055, China. His research interests include programmable network data plane and network algorithms. Li received a Ph.D. degree from Peking University, Beijing, China. He is the corresponding author of this article. Contact him at wenjunli@g.harvard.edu.

GAOGANG XIE is a professor with the Computer Network Information Center of the Chinese Academy of Sciences, Beijing, 100083, China. His research interests include Internet architecture, packet processing and forwarding, and Internet measurement. Xie received a Ph.D. degree from Hunan University, Changsha, China. Contact him at xie@cnic.cn.

YANG XU is with Peng Cheng Laboratory, Shenzhen, 518055, China. He is also the Yaoshihua Chair Professor at Fudan University, Shanghai, 200433, China. His research interests include software defined networking, dater center network, network function virtualization, and edge computing. Xu received a Ph.D. degree from Tsinghua University, Beijing, China. Contact him at xuy@fudan.edu.cn.

YI WANG is with Peng Cheng Laboratory, Shenzhen, 518055, China. He is also a research professor at the Southern University of Science and Technology, Shenzhen, 518055, China. His research interests include future network architectures and high-performance network devices. Wang received a Ph.D. degree from Tsinghua University, Beijing, China. Contact him at wangy37@sustech.edu.cn.