Received 30 April 2023, accepted 28 May 2023, date of publication 1 June 2023, date of current version 16 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3281838

METHODS

PcmSU-A Packet Classification Method Supporting High-Speed Search and Fast Update

YUZHU CHENG[1], YIHANG XU[2], AND QIUYING SHI[1]
[1]School of Software, Changsha Social Work College, Changsha 410004, China
[2]Department of Resource and Safety Engineering, Central South University, Changsha 410083, China

Corresponding author: Yuzhu Cheng (vogue21ct@qq.com)

This work was supported in part by the Hunan Provincial Natural Science Foundation of China under Grant 2022JJ60099, in part by the Research Foundation of the Education Department of Hunan Province under Grant 21C1589, and in part by the Project of the Hunan Social Science Achievement Evaluation Committee under Grant XSP2023GLC124.

ABSTRACT Packet classification is the core technology that underpins software-defined networking (SDN). With the rapid development of network applications, the increasing complexity of flow tables in SDN brings challenges to update and classification time. In order to achieve fast search and update at the same time, this article proposes a high-performance packet classification method PcmSU based on decision tree. The method optimizes the search and update process in the traditional decision tree-based classification method, and the classification efficiency of the constructed decision tree has been greatly improved compared with that before optimization. Based on the characteristics of the small interval, the reconstruction of the decision tree is greatly reduced and the rule update efficiency is improved. In addition, based on the characteristics of the decision tree constructed by this method, there is no need to consider the specific location and sequence number of rule updates when adding rules, which can avoid introducing rule conflicts. Experimental results show that the PcmSU method not only supports high-speed packet classification and linear memory consumption, but also has fast rule update speed.

INDEX TERMS Packet classification, decision tree, SDN, rule update.

I. INTRODUCTION

In the traditional network architecture, routers as core devices contain functions such as routing and packet forwarding, however, the rapid expansion of the network has led to a blurring and bloating of the functions and structures they carry. The core idea of SDN technology is to realize the decoupling between the control plane of network devices and the data forwarding plane. OpenFlow, the standard protocol of SDN, supports the controller to dynamically manage the actions of each switch, and the core function of OpenFlow switches is to quickly classify each packet flowing through them based on the principle of first-rule-precedence. The classification needs to satisfy two conditions: high-speed packet classification (to meet the real-time needs of the client) and fast update (OpenFlow rules are frequently updated dynamically by the controller). Rule updating generally includes inserting rules and deleting rules. Each OpenFlow switching node has one or more flow tables, and each flow table is composed of data items including matches, counters and action sets. The switching node determines the processing mode of packets (forwarding, discarding, etc.) according to the matching items and action sets in the flow table. In addition, OpenFlow preliminarily realizes the prototype design idea of SDN, the controller adds, deletes and modifies the flow tables in switching nodes as required, so as to achieve the purpose of controlling the network (such as link load balancing, dynamic networking, etc.).

As a widely studied problem, packet classification in physical switches still relies on expensive TCAM (Ternary Content Addressable Memory), because the algorithm solution implemented in software is difficult to meet the wire-speed forwarding in traditional network infrastructure. Especially with the advent of SDN and NFV (Network Function Virtualization), efficient algorithmic solutions using ordinary memory such as DRAM/SRAM are becoming attractive again.

The associate editor coordinating the review of this manuscript and approving it for publication was Ali Kashif Bashir.

TABLE 1. An example of 2-tuple classification rules.

Rule ID	F_1	F_2	Action
r_1	[3,9]	[3,3]	accept
r_2	[7,9]	[0,7]	discard
r_3	[0,0]	[0,9]	discard
r_4	[0,4]	[6,6]	discard
r_5	[2,3]	[1,5]	discard
r_6	[0,9]	[8,8]	accept
r_7	[0,9]	[7,9]	discard
r_8	[0,3]	[0,6]	accept
r_9	[5,6]	[0,9]	accept

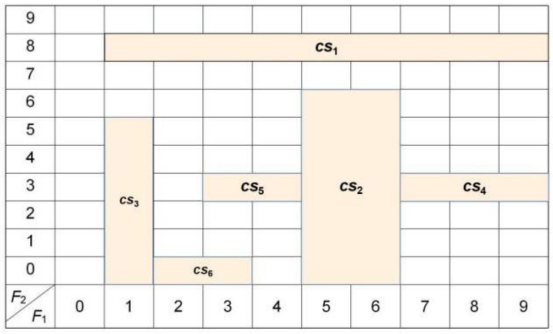


FIGURE 1. Rule mapping forms a set of cell spaces.

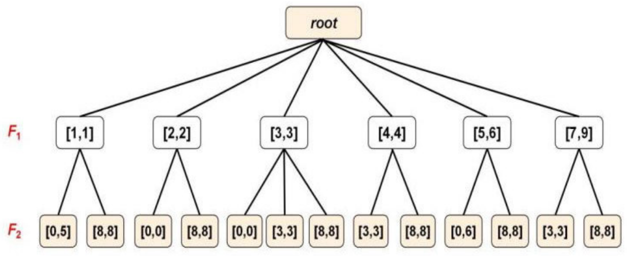


FIGURE 2. Constructing classification decision tree based on PCMIgr method.

In the existing packet classification algorithms, decision tree and tuple space search (TSS) are two main methods. In decision tree-based schemes, the geometric view of the packet classification problem is adopted and the decision tree is constructed. They work by recursively dividing the search space into smaller subspaces until each subspace contains fewer rules than a predefined threshold. However, if the rule spans multiple subspaces, there will be a rule replication problem, and the rules need to be replicated for each overlapping subspace. This rule replication problem becomes particularly serious in small-scale cutting operations. Therefore, the decision tree-based scheme achieves high-speed lookup on packet classification, but cannot support fast update due to rule replication issue. Because when constructing a decision tree, the rules in the original rule list will be copied to all intersecting areas, and the rule update needs to reconstruct the decision tree, which consumes a lot of time, resulting in slow and complex rule update based on the decision tree.

Different from traditional packet classification, the rule update process usually needs to be completed quickly online to ensure the continuous availability of applications such as VoIP and E-commerce. OpenFlow has higher requirements for rule update, which makes most decision tree algorithms unsuitable for this application. On the contrary, TSS divides the rules into a set of hash tables (i.e., tuple space) according to the prefix length. In the TSS-based scheme, rule replication does not occur, and each rule update can only access memory once on average. Therefore, TSS is the fastest rule update method at present. It divides the original rule set into several rule subsets according to the prefix hash value of the rule. These rule subsets satisfy the easy-to-compute property, and the rules can be quickly inserted and deleted based on the hash table lookup method. However, due to the large number of rule subsets, for each incoming packet, its final match is the matching rule with the highest priority among all tuples, so TSS needs to search for each tuple, that is, to calculate each rule subset, so the classification speed of TSS is slow.

Some of the existing packet classification methods focus on reducing update time, such as TupleMerge [1], or minimizing classification time, such as SmartSplit [2], or simultaneously supporting fast update and classification, such as PartitionSort [3]. TupleMerge is a fast update method that focuses on minimizing update time while sacrificing classification time. SmartSplit is a high-speed classification method based on decision tree, which focuses on minimizing classification time while sacrificing memory consumption and update time. PartitionSort divides the rule set into smaller sortable rule sets, and then stores each sortable rule set into a multidimensional interval tree (MITree) to support both fast update and high-speed classification. PartitionSort is superior to TSS and SmartSplit in classification speed and update time, respectively, which means that the decision tree method can also achieve rapid update, and points out a new research idea for the decision tree method.

Based on SDN performance requirements, supporting fast update and high-speed classification has become an important research direction in the field of packet classification. In order to achieve high-speed classification, we proposed a decision tree-based packet classification method PCMIgr [4] in our previous work, which achieved high-speed packet classification with logarithmic time complexity. Firstly, according to the rule mapping method of multi-dimensional cell space [5], the original rules are mapped to a multi-dimensional matrix in reverse order to obtain the cell space set with the same semantics as the original rules. Then, based on the greedy idea, when we need to select a feature as an internal node to split, we choose the feature with the highest "information gain ratio" to construct the decision tree. Taking the nine classification rules shown in Table 1 as an example, six cell spaces and corresponding classification decision tree (as shown in Fig. 2) can be obtained after mapping.

As can be seen from Fig. 2, the interval values of the subnodes corresponding to the roots of the decision tree or any

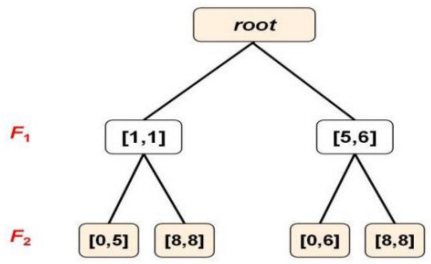


FIGURE 3. An example of decision tree with small interval value of sub-nodes.

sub-tree satisfy the orderly increasing relationship, so the binary-search method can be applied to packet classification based on the decision tree. Moreover, each leaf node of the decision tree uniquely corresponds to a cell space (according to the definition of cell space [5], the decision is accept). Therefore, when a packet matches the leaf node, the classification result of the packet can be clearly determined as accept. This property avoids the need for sequential matching in the rule group associated with the leaf nodes in the traditional decision tree-based classification method, which effectively improves the efficiency of packet classification.

However, when updating rules based on the decision tree shown in Fig. 2, the cost of decision tree reconstruction is relatively high under frequent rule set changes. For example, to insert a rule "r: F_1∈[2,4]∧ F_2∈[3,6]→ *accept*" in the decision tree, the interval value of the rule on dimension F_1 is [2,4], and the six interval values of the corresponding decision tree on this dimension are [1,1], [2,2], [3,3], [4,4], [5,6] and [7,9]. Obviously, the interval [2,4] covers or intersects with the three node intervals [2,2], [3,3] and [4,4] of the decision tree. Therefore, corresponding to the decision tree shown in Fig. 2, inserting the rule r needs to modify three decision subtrees at the same time, resulting in low efficiency of rule update.

However, if the interval value of the sub-node corresponding to the decision tree root is relatively small compared with the domain value $D(F_1)$=[0,9], as shown in Fig. 3, [1,1] and [5,6] correspond to the first layer node interval of the two decision subtrees, respectively. Let's still add the rule "r: F_1∈[2,4]∧F_2∈[3,6]→ *accept*", obviously, the interval value [2,4] of rule r on dimension F_1 and the two decision tree node intervals [1,1] and [5,6] on this dimension are independent of each other. Therefore, when inserting r, it can be directly added to the decision tree as a new subtree of the root, without any modification to other subtrees in the decision tree, which effectively improves the efficiency of rule update.

Comparing the rule update process in Fig. 2 and Fig. 3, we find that if the packet classification method based on decision tree wants to meet the requirements of high-speed packet classification and rapid update at the same time, a feasible design idea is that the rule interval and the node interval of the decision tree should be as independent as possible.

Based on the above observations, we propose a high-performance packet classification method PcmSU based on decision tree. The method first divides the original rule mapping space to form a series of independent cell space subsets, each of which has the same characteristic: the cell space mapping interval on a certain dimension F_i is a F_i-small interval (as defined in Section III-B), on this basis, multiple classification decision trees are constructed, which can perform packet classification and rule update in parallel. We evaluated our algorithm using ClassBench [25], and the results show that the PcmSU method not only supports high-speed packet classification and linear memory consumption, but also has a fast rule update speed. Compared with the TupleMerge [1] algorithm, PcmSU achieves similar rule update performance, but significantly better than TupleMerge in classification efficiency, improving by nearly an order of magnitude on average.

The main innovation of this method include the following aspects:

1) Constructing the classification decision tree based on projection interval mapping. Because the node interval values follow the increasing relationship, the binary-search method can be used in classification, which has high classification efficiency;

2) Dividing the cell space set to construct multiple decision subtrees, which realizes the parallel processing of packet classification and improves the classification efficiency;

3) Using the characteristics of the small interval to reduce the reconstruction of decision trees and improve the rule update efficiency; when adding rules, there is no need to consider the specific location and sequence number of rule updates, which avoids introducing rule conflicts.

The rest of this paper is organized as follows: the related work is introduced in Section II; The third section gives a description of the problem and the solution ideas, and elaborates on the technical details of the PcmSU method. Section IV presents the classification and update results of PcmSU, TupleMerge, HyperSplit and CutTSS, and followed by the analysis of experimental results; Finally, the conclusion is drawn in Section V.

II. RELATED WORK

Packet classification has been actively studied for more than two decades. But as far as we know, most of them can achieve high-speed packet classification, but not fast updates, which seriously limits their scalability in the SDN era. In contrast, tuple space-based schemes have become the de facto choice in software switches, because they support fast updates of linear memory consumption. However, these schemes have the problem of low classification performance and can not meet the demand of high-speed classification in the fast-growing networks.

Generally speaking, the research on packet classification can be broadly divided into four categories: decision tree-based methods, partition-based methods, integrated decision tree and partition-based methods, and hardware-based methods.

The algorithms based on decision tree mainly include two categories: one is based on Trie tree; the other is to construct a decision tree by cutting the multidimensional space where the classification rule set is located. The basic idea of Trie-based algorithm [6] is to establish a hierarchical binary tree according to the rule set, which divides each dimension of the rule into one layer, and extends the one-dimensional tree structure recursively to generate a k-dimensional hierarchical tree. The advantage of the algorithm is simple and straightforward, easy to implement in hardware. The disadvantage is that the backtracking time is long, is not conducive to the expansion of rule dimensions, and cannot directly support range matching. SplitTrie [7] improves on the basic trie algorithm to support multi-field search and avoid backtracking, but the algorithm still does not support range matching.

There are many algorithms for constructing decision trees by reasonably cutting multidimensional spaces. Such as HiCuts [8] and HyperCuts [9], which use local optimization to divide the search space into multiple subspaces of equal size until the number of rules in each subspace is less than the predefined threshold, showing excellent search performance. But equal-scale cutting will lead to huge storage requirements. H. Lim et al. reduced the memory consumption of the algorithm by boundary-based cutting [10]. HybridCuts [11] divides rules on a single rule field instead of all fields, reducing the number of subsets, thereby reducing the frequency of memory access. BitCuts [12] and PCMIgr [4] cut based on bit and cell space, respectively, for a better balance of speed and space. ByteCuts [13] intelligently divides the rule list into multiple trees by byte slicing, reducing rule replication. The MBitCuts method [14] reduces the space consumption and memory access by changing the bit selection method when cutting the geometric space model of each tree node.

Compared with the cutting-based method, the splitting-based method divides the search space into multiple isodensity subsets. "Isodensity" means that the number of rules in each subset is almost identical. HyperSplit [15] is a classical splitting method, which divides the search space into two equally dense subspaces. But this method increases the memory consumption as the number of rules increases. As an improved version of HyperSplit, ParaSplit [16] uses a new partitioning algorithm to reduce the complexity of the ruleset, and its memory consumption is reduced. CutSplit [17] combines the advantages of cutting and splitting to improve packet classification performance. However, the performance varies greatly for different rulesets, which is a common problem faced by most decision tree-based algorithms except "rule replication". Therefore, the decision tree-based approach can have fast classification speed, but the disadvantage is that rules may be replicated to many areas when dividing, resulting in high memory consumption, and rule updates are slow and complex, because when inserting or deleting rules, all replicated areas need to be operated on at the same time.

The partition-based method refers to quickly narrowing down the scope of multi-field search by partitioning the rule set by tuples. A tuple defines the number of bits for a specific bit in each field of the rule, that is, the bit characteristics of each domain. According to the tuple, the initial packet classification rule set can be divided into multiple rule subsets, and each rule subset establishes a corresponding hash table based on the bit of each rule domain. The most typical is the TSS algorithm, which divides the packet classification rule set into corresponding rule subsets according to the number of valid bits of each field prefix and stores them in hash table. When a packet is received, the corresponding rule subset is first searched by hash key, and then matched in the subset to obtain the best matching rule. For each group, TSS can update the rules (inserting or deleting rules) quickly based on the hash table lookup method, but its main disadvantage is that the number of partitions or tables will increase over time to a large number. When classifying packets, each partition needs to be calculated, resulting in slow packet classification. Representative algorithms include TSS [18], CNP3 [19], Dynamic Tuple [20], etc.

Hybrid methods combining partition and decision tree, such as EffiCuts [21] and SmartSplit [2], are designed to reduce the linear space requirements for rule replication and decision tree management. Specifically, EffiCuts and SmartSplit first place small or large rules in a field into the same partition, and then generate a HyperCuts or HyperSplit tree for each partition. Although these two methods significantly reduced the space storage consumption of rule replication and decision tree, they do not eliminate rule replication. Therefore, the rule update performance of these two methods is not ideal. By dividing the rules into several sortable subsets and constructing a MITree for each subset, PartitionSort [3] simultaneously achieves logarithmic time classification and update for each subset. However, due to the strict restriction on partitions, PartitionSort requires more trees than SmartSplit, resulting in slower classification speed.

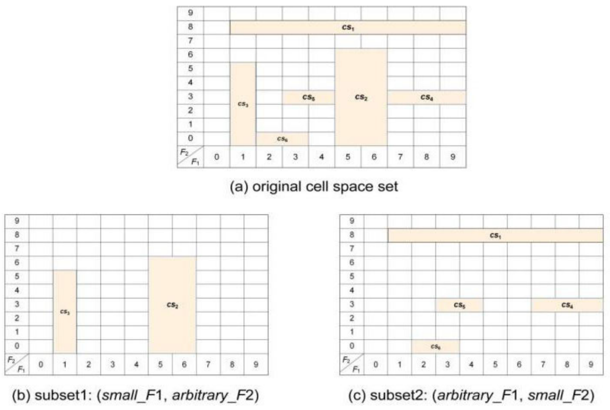
Finally, there are a number of hardware-based methods for packet classification, the core idea of which is to exhaustively search all the rules in the rule set to get a matching result. Typical hardware-based solutions include ternary content addressable memory (TCAM) [22], field programmable gate array (FPGA) [23] and dedicated network processor chips. The data structure of exhaustive search algorithm is simple, but the classification efficiency based on hardware processing is high. For example, the packet classification algorithm based on TCAM adopts parallel search scheme, and the time complexity of the algorithm is $O(1)$. At present, large routers and high-end classifiers mainly use hardware devices for packet classification based on exhaustive search. However, the expensive price, long development time and high energy consumption of dedicated hardware limit their application and scalability. In particular, hardware-based classifiers have a long construction time and cannot be used in dynamic OpenFlow environments. Therefore, whether hardware-based classifiers can be well extended to OpenFlow packet classification remains to be studied and verified.

In summary, one of the fundamental challenges of packet classification is achieving both high-speed classification and fast update. Currently, most methods, such as SmartSplit, focus on minimizing classification time while sacrificing update time and memory occupation. Some methods, such as tuple space search(TSS), sacrifices classification time for fast update. The final result is that high-speed classification methods are not competitive in update time, and rapid update methods do not have high-speed classification efficiency. CutTSS [24] combines the advantages of decision tree method and TSS method, proposes a two-stage framework to adaptively utilize the different characteristics of rule sets at different scales. It is one of the few decision tree packet classification methods that can use linear space and support fast update. However, this method is highly sensitive to the order of rules. Before inserting rules, it is necessary to clarify the order of position in which rules are inserted. Otherwise, it is easy to bring rule conflicts to the rule set, thus introducing security vulnerabilities. In addition, the method is more suitable for the rules represented by prefixes, and when querying the leaf node during classification, it is necessary to continue to perform sequential matching in the rule groups associated with the subspace, which reduces the classification efficiency to some extent.

It should be noted that there is a ‘‘prefix explosion’’ problem when using prefix form to represent rules. For example, when a k-dimensional rule expressed in the form of address interval is converted into a prefix form, there are $(2w–2)^k$ rules in the worst case, w is the domain length of the dimension where the address interval is located. To this end, we designed a rule representation model based on address interval in the preliminary work [5], and proposed a new classification method based on decision tree [4]. The method achieves logarithmic time classification speed, and completely eliminates rule conflicts, so there is no need to consider the order of rule insertion position when inserting accept rules, which lays a good foundation for rapid rule update. This is also the premise of this article to propose a packet classification method with both high-speed classification and fast update.

III. THE PROPOSED APPROACH

In this section, we first introduce the idea behind the design of PcmSU. Then, we propose a cell space set partition algorithm based on F_i-small interval, which divides all the cell spaces obtained by the original rules into multiple cell space subsets according to the corresponding dimensions of F_i-*small interval*. For each cell space subset, all the cell spaces it contains have a common characteristic: the cell space mapping intervals on a certain dimension F_i are small intervals. On this basis, we design a decision tree construction method based on projection interval. The corresponding decision tree is formed by each cell space subset, and this decision tree can achieve high-speed packet classification with logarithmic



cs1
cs2
cs3
cs4
cs5
cs6
F2
F1
0 1 2 3 4 5 6 7 8 9
(a) original cell space set
(b) subset1: (small_F1, arbitrary_F2)
(c) subset2: (arbitrary_F1, small_F2)

FIGURE 4. **Illustration of dividing cell space set into subsets.**

time efficiency; At the same time, a rapid rule update method based on spatial relation comparison is proposed. Finally, the effectiveness of PcmSU method is discussed theoretically and experimentally.

A. DESIGN IDEAS

According to the analysis of the rule update process based on decision tree, we know that if the decision tree-based packet classification method wants to achieve fast rule update, a design idea is to make the rule interval to be updated and the corresponding node interval of the decision tree as independent as possible. Specifically, the original decision tree can be divided into several decision sub-trees with the structural feature as shown in Fig. 3. In these decision subtrees, the node interval with one or more dimensions is F_i-*small interval*. Therefore, when updating rules, if the rule to be updated is a small interval on a certain dimension F_i, or the rule interval and the corresponding node interval of the decision tree are independent of each other on dimension F_i, the rule update operation is preferentially performed from F_i.

As shown in Fig. 4 (for ease of understanding, this article takes two-dimensional rules as an example). Firstly, the cell space in F_1 and F_2 dimensions that conforms to the definition of F_i-small interval is calculated. According to the calculation results, the entire cell spaces are divided into two subsets according to the dimension F_i to which the small interval belongs: $(\,small\_F_1, arbitrary\_F_2) = \{\text{subset1}\}$ and $(arbitrary\_F_1,\ \ small\_F_2) = \{\text{subset2}\}$, here $arbitrary\_F_2$ can be either $small\_F_2$ or big_F_2.

With reference to the decision tree construction method described in PCMIgr [4], the decision subtrees are respectively constructed according to the cell space subsets in Fig. 4, as shown in Fig. 5. It can be seen that all subtrees have F_i-small interval in one or more dimensions, and the rule spaces corresponding to each branch are independent of each other, which avoids the rule replication problem of traditional classification decision trees. However, in order to achieve high-speed classification and fast update of rules at the same time, there are still several issues to be considered:

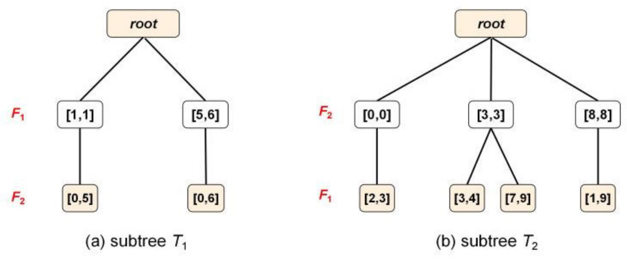


FIGURE 5. Constructing classification decision tree based on cell space subset.

TABLE 2. Notations used in this manuscript.

Notation	Paraphrase
F_i	the i-th dimension
$D(F_i)$	domain of F_i
cs	cell space
M_k	k-dimensional matrix
p	packet
R	classification rule
T	decision tree
T_i	the i-th decision subtree

(1) What criteria is used to define F_i-small interval? can we assume that most rules in a ruleset have at least one small field?

(2) The packet classification method based on decision tree can realize high-speed classification, but the decision tree needs to be reconstructed when updating rules, and the reconstruction is complicated and slow due to rule replication. Therefore, how to reduce the reconstruction operation of the decision tree as much as possible when updating rules, so as to achieve fast update?

Solving the above problems is the design goal of this article. Our solutions can be summarized as follows:

(1) The domain value characteristics of rules and cell space are analyzed, the definition of F_i-small interval is given, and the observation conclusion is drawn based on the statistical results of the proportion of small intervals.

(2) The rule mapping method based on multi-dimensional matrix eliminates rule conflicts, and divides the cell space into several subsets according to the corresponding dimension of F_i-small interval. Each subset has a common characteristic: the cell space mapping interval on a certain dimension F_i is a small interval. Therefore, the node interval of the decision tree constructed by this subset must be a F_i-small interval.

(3) The decision subtree is constructed from the subset of cell space according to the projection interval method. Because each decision subtree has a node with small interval in a certain dimension, this characteristic can be used to accelerate the rule update speed. In addition, because the cell spaces are independent of each other, there is no need for rule replication when constructing a decision tree. Therefore, parallel processing can be adopted for packet classification and rule update based on decision tree, which will help to improve the efficiency of packet classification and rule update.

B. CONCEPT PREPARATION

In this section, we first briefly explain some of the notations used in this article (as shown in Table 2), give the definition of F_i-small interval, and perform statistical experiments on several rule sets from ClassBench, and give statistical results and observation conclusions on the proportion of small intervals.

In the following, we give the definition of small interval based on the domain value characteristics of the rule and cell space:

Definition 1: Given a k-dimensional rule $R = (F_1, F_2, \ldots, F_k)$ and threshold vector $T = (T_1, T_2, \ldots, T_k)$, the interval scale of each dimension coordinate $F_i(i = 1, 2, \ldots, k)$ of the rule is defined as follows:

(1) If the interval length of F_i satisfies: $|F_i| > T_i$, the interval is called F_i-big interval;

(2) If the interval length of F_i satisfies: $|F_i| \leq T_i$, the interval is called F_i-small interval;

Accordingly, the definition of interval scale of cell space is given as follows:

(1) If for any $i = 1, 2, \ldots, k$, the interval of cs in each dimension is F_i-big interval, the cell space is called a big cell space;

(2) If for any $i = 1, 2, \ldots, k$, cs contains a total of n F_i-small intervals, the cell space is called an n-small cell space,here $n = 1, 2, \ldots, k$.

For the classic IPv4 five-tuple rule, since the protocol fields is limited to a small number of values (such as TCP, UDP, etc.), this article only considers the four fields of source address, destination address, source port and destination port. The corresponding threshold vector is denoted by $T = (T_{SA}, T_{DA}, T_{SP}, T_{DP})$, for example, take 1/2 of the domain value interval of each dimension as the threshold vector, then $T_{1/2} = (2^{16}, 2^{16}, 2^8, 2^8)$.

Based on the above definition, we conducted statistical experiments on several rule sets from ClassBench. Because the actual statistics is the ratio of big cell space, we first use the FDM method [5] to map each rule set before statistics. Table 3 gives the specific statistical results for each cell space set under the preset threshold. It can be seen that when the threshold vector $T_{1/2} = (2^{16}, 2^{16}, 2^8, 2^8)$ is taken, the proportion of big cell space is less than 0.003, which indicates that most cell space sets have at least one small cell space satisfying the threshold $T_{1/2}$. In practical applications, if there are a few big cell spaces, we can choose any dimension F_i and divide the cell space into two sub-cell spaces in this dimension on average. According to the definition of the threshold vector $T_{1/2}$, the two cell spaces after dividing must be F_i-small cell space.

C. METHODOLOGY

Based on the problem analysis and the statistical results of Table 3, we propose a new decision tree-based packet classification method, PcmSU, which can simultaneously meet the requirements of high-speed packet classification and fast update. Next, we will introduce the implementation process of PcmSU in detail, including the following five steps:

TABLE 3. Statistical results for cell space sets under preset threshold.

#cell spaces	Threshold vector	#big cell spaces	#n-small cell spaces				
			n=1	n=2	n=3	n=4	n>4
173		0	173	167	82	1	0
366	$T_{1/2}=(2^{16},$	0	366	354	147	5	0
453	$2^{16},2^{8},2^{8})$	1	452	432	218	2	0
624		0	624	606	295	7	0

STEP 1: Rule Preprocessing

For the input k-dimensional classification rule set R, the rule mapping method is used to map the rules to the k-dimensional matrix space M_k in reverse order, and a series of independent cell spaces are formed after mapping. In general, classification rules can be expressed as "$F_1 \in D(F_1) \wedge F_2 \in D(F_2) \wedge \ldots \wedge F_k \in D(F_k) \rightarrow$ *decision*", where each dimension coordinate $F_i (1 \leq i \leq k)$ represents the source address, destination address, source port and destination port, etc., $D(F_i)$ represents the corresponding domain value interval, and "decision" represents the rule's decision (accept or discard). According to the idea of rule mapping [5] based on multi-dimensional matrix, any k-dimensional classification rule set R can be mapped to the k-dimensional matrix space M_k to form a series of regions whose decision is accept (also known as the cell space ***cs*** corresponds to an independent k-dimensional rectangle in k-dimensional matrix space), which can be expressed as $[(l_1, l_2, \ldots, l_k)\ (d_1, d_2, \ldots, d_k)]$, where l_i and d_i refer to the minimum boundary value and range of the region in each dimension, respectively. Fig. 1 shows the six cell spaces obtained by preprocessing the nine classification rules shown in Table 1.

STEP 2: Partition Cell Space Set based on F_i*-small Interval*

Based on the above experimental statistical results, we propose a cell space set partitioning algorithm based on F_i-small interval, which divides all cell spaces into multiple cell space subsets. For each subset, all contained cell spaces have a common characteristic: the cell space mapping interval on a certain dimension F_i is a small interval. Therefore, the node interval of the decision tree constructed by this subset must be a F_i-small interval. The specific execution process of the method is described in Algorithm 1.

Taking the cell space set of Fig. 1 as an example, we calculate the number of n-small cell spaces on two different dimensions of F_1 and F_2, respectively. Because the whole set of cell spaces does not contain big cell space, that is, $(big\_F_1, big\_F_2) = \{\emptyset\}$, we can divide the set of cell spaces into two subsets: $(small\_F_1, arbitrary\_F_2) = \{cs_2, cs_3\}$ and $(arbitrary\_F_1,\ small\_F_2) = \{cs_1, cs_4, cs_5, cs_6\}$. The partitioning result of cell space set is shown in Fig. 4.

STEP 3: Construct Decision Tree based on the Projection Interval

The purpose of this step is to construct a classification decision tree based on each cell space subset. In general, on a given dimension F_i, the spatial relation $R(u, v, F_i)$ of any two cell spaces ***u*** and ***v*** must satisfy one of the six

Algorithm 1 Partition Cell Space Set Based on F_i-*Small Interval*

```
Input: A cell space set {S} consisting of n cell spaces cs
    [(l_1, l_2, ..., l_k) (d_1, d_2, ..., d_k)].
Output: k cell space subsets S_1, S_2 ..., S_k.
Begin
    1. while($cs \in \{S\}$ & $\{S\} \neq \emptyset$)
    2.   for($t := 1$ to $k$) do
    3.     $v_t = \frac{d_t}{D(F_t)}$;
    4.   end for
    5.   $v_i = min\{v_1, v_2, \ldots, v_k\}$;
    6.   if ($v_i \leq 1/2$), send cs to the cell space subset $S_i$ and remove cs from {S};
    7.   elseif ($v_i > 1/2$), divide cs into two cell spaces in half on the $F_i$ dimension, send them to the subset $S_i$, and remove cs from {S};
    8. end while
    9. return $S_t (t := 1$ to $k)$;
End
```

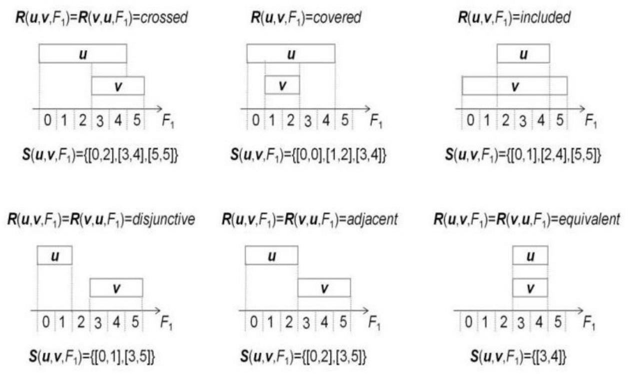

R(u,v,F1)=R(v,u,F1)=crossed
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[0,2],[3,4],[5,5]}
R(u,v,F1)=covered
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[0,0],[1,2],[3,4]}
R(u,v,F1)=included
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[0,1],[2,4],[5,5]}
R(u,v,F1)=R(v,u,F1)=disjunctive
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[0,1],[3,5]}
R(u,v,F1)=R(v,u,F1)=adjacent
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[0,2],[3,5]}
R(u,v,F1)=R(v,u,F1)=equivalent
u
v
0 1 2 3 4 5 F1
S(u,v,F1)={[3,4]}

FIGURE 6. The spatial relation and coordinate projection interval of two cell spaces u and v on dim F_1.

relations {*crossed*, *covered*, *included*, *disjunctive*, *adjacent* and *equivalent*} [26], as shown in Fig. 6. Next, we give the definition of the coordinate projection interval $S(u, v, F_i)$ of the cell spaces ***u*** and ***v*** on the dimension F_i.

Definition 2: Let cell spaces $u = (l_1^{(u)}, \ldots, l_k^{(u)})(d_1^{(u)}, \ldots, d_k^{(u)})$, $v = (l_1^{(v)}, \ldots, l_k^{(v)})\ (d_1^{(v)}, \ldots, d_k^{(v)})$, *if*

(1) $R(u, v, F_i) = $ *'crossed'*, then $S(u, v, F_i) = [l_i^{(u)}, l_i^{(v)}], [l_i^{(v)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(u)} + d_i^{(u)}, l_i^{(v)} + d_i^{(v)}]$;

(2) $R(u, v, F_i) = $ *'covered'*, then $S(u, v, F_i) = [l_i^{(u)}, l_i^{(v)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}], [l_i^{(v)} + d_i^{(v)}, l_i^{(u)} + d_i^{(u)}]$;

(3) $R(u, v, F_i) = $ *'included'*, then $S(u, v, F_i) = [l_i^{(v)}, l_i^{(u)}], [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(u)} + d_i^{(u)}, l_i^{(v)} + d_i^{(v)}]$;

(4) $R(u, v, F_i) = $ *'disjunctive'*, then $S(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}]$;

(5) $R(u, v, F_i) = $ *'adjacent'*, then $S(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}]$;

(6) $R(u, v, F_i) = $ *'equivalent'*, then $S(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}]$.

Based on Definition 2, we will briefly describe the general process of constructing classification decision trees based on the subset of cell space, as shown in Algorithm 2. Let the k-dimensional cell space set contain n cell spaces $\boldsymbol{cs}_i(i := 1$

Algorithm 2 Decision Tree Construction Based on Projection Interval

```
Input: c cell spaces $cs_i$ ($i$: =1 to $c$), and the dimension $F_t$ where the small cell space is located.
Output: the classification decision tree $T$.
Begin
    1. Calculate $n$ coordinate projection intervals $S(cs_1, \ldots, cs_c, F_t)$ of $c$ cell spaces on the dimension $F_t$;
    2. Add $n$ coordinate projection intervals as child nodes $node_i$ ($i$: =1 to $n$) to the root in turn, each child node constitutes a subtree $T_i$($i$: =1 to $n$) of $T$, and $node_i$ is the root corresponding to the subtree $T_i$;
    3. for ($s$: =1 to $k$($s \neq t$)) do
    4. Search all the cell spaces (denoted as $\{cs\}$) associated with the root of $T_i$($i$: =1 to $n$), calculate its coordinate projection interval $P(\{cs\}, F_s)$ on the dimension $F_s$, and denote the number of projection intervals as $n_i$.
    5. Add $n_i$ projection intervals $P(\{cs\}, F_s)$ to $T_i$ as child nodes of the root of $T_i$. Similarly, each child node constitutes a subtree of $T_i$ and becomes the root of the corresponding subtree.
    6. end for
    7. return $T$;
End
```

to n), and the dimension of the small cell space is denoted as F_t. Initially, the decision tree T only contains the root node.

Taking the cell space subset shown in Fig. 4(b) as an example, two cell spaces (cs_2, cs_3) in the two-dimensional matrix space form two coordinate projection intervals {[1,1], [5,6]} on the F_1 dimension. Each projection interval is added to the decision tree as a child node of *root* in turn, and each child node constitutes a subtree of T. For the corresponding interval [1,1] of the subtree root, the associated cell space cs_3 forms a coordinate projection interval [0,5] on the F_2 dimension. Add the projection interval to the subtree as a child node of the root. Similarly, the projection interval [0,6] is added as a child node to another subtree, and finally the decision tree T_1 is formed, as shown in Fig. 5(a).

Compared with classical decision tree methods such as HiCuts and HyperCuts, PcmSU has the following characteristics in the details of constructing decision trees: (1)PcmSU does not divide the rule space evenly, but adopts a division method based on the cell space boundary, which can effectively reduce the number of rule subspaces and save storage space; (2)PcmSU first uses the rule mapping method based on multidimensional matrix space to preprocess the original rules, so that the target rules and the original rules have the same semantics, but the rules are independent of each other and do not overlap. This is reflected in the classification decision tree based on target rules, where each branch of the tree uniquely corresponds to an independent multidimensional rule subspace. Intuitively, each leaf node of the decision tree has only one association rule with the decision "accept", different from the traditional packet classification method, which needs to continue to perform sequential matching within the rule group associated with the leaf node. This property significantly improves the speed of packet classification. (3) Each branch in the decision tree corresponds to the cell space one by one, and the interval value of the same layer nodes in the decision tree is strictly increasing, so the binary-search method can be applied when searching, which can ensure high classification efficiency. (4) Finally, several independent decision subtrees with the same semantics as the original decision tree are constructed, which can process packet classification and rule update based on decision tree in parallel, and help to improve the speed of packet classification and rule update.

STEP 4: Classification Packet based on Decision Tree

Packet classification can be regarded as the problem of "point location in multidimensional space" in computational geometry [27]: given some disjoint areas in multidimensional space, to locate the area containing the specified "point". A classifier is a hypercube set with priority, and the packet header represents a point in k-dimensional space. The packet classification problem corresponds to the decision tree model constructed in STEP 3 of this article, which is essentially the matching process between the packet and the corresponding node intervals of the decision tree. Because the rule decision associated with each leaf node of the decision tree is accept, if the corresponding packet is matched in any decision subtree (that is, D_i={*accept*}), the packet decision can be determined as accept; When all decision subtrees cannot match, the packet decision is discard.

Taking the decision tree shown in Fig. 5 as an example, when a packet P is received, it is sent to two decision subtrees T_i (i = 1,2) for matching, and the matching result is recorded as D_i={*accept* or *discard*}, i = 1,2. Then, the final decision of the packet is determined based on the matching result D_i (i = 1,2) of the packet on each subtree. The specific decision is based on the following criteria:

(1) when $\sum_{i=1}^{2}(D_i = accept) \geq 1$, the decision is accept;

(2) when $\sum_{i=1}^{2}(D_i = accept)=0$, the decision is discard.

It can be seen that the PcmSU method reduces the size of the decision tree after dividing the original decision tree into multiple subtrees, and the packet classification operation can be performed in parallel, which obviously improves the packet classification efficiency. In the packet classification method based on decision tree, the essence of packet classification is a query operation. Especially for the decision tree or any subtree shown in Fig. 5, the interval value of the same layer nodes in the decision tree is strictly increasing, so the binary-search method can be applied when searching, which is expected to achieve logarithmic time classification efficiency.

As shown in Fig. 7, the root node of decision tree T has two sub-nodes, and the corresponding interval coordinates are {[1,1], [5,6]}, which satisfies the strict increasing relationship. Let $\{p_1, p_2\}$={(1,6), (5,3)} be the packet to be classified. Firstly, we determine the packet p_1:(1,6). e_1=1 and e_2=6. Starting from the root node, e_1=1 matches the interval of the first node $\boldsymbol{a}$[1,1], so continue to search on the subtree with $\boldsymbol{a}$ as the root. Because e_2=6 cannot match the interval [0,5] corresponding to the child node of the subtree root node $\boldsymbol{a}$, the classification decision of p_1 can be determined as discard. For the packet p_2:(5,3), it is known that

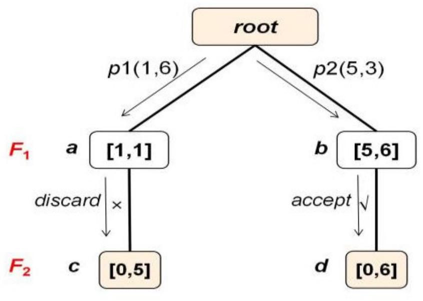


FIGURE 7. Fast packet classification based on binary search.

Algorithm 3 Packet Classification Based on Binary Search

Input: the packet P:(e_1, e_2, ..., e_k), the decision tree T.
Output: the packet decision ('*accept*' or '*discard*').
Begin
/* Assuming that the dimension matrix corresponding to the k-level nodes of decision tree T is L[k], and the root node of T is *root*. Bisearch the *i-th* metadata $\boldsymbol{e}_i$ of P in all the child nodes of the *root*, if $\boldsymbol{e}_i$ is included in the interval corresponding to the *s-th* child node *Child*(*root,s*) of *root*, then continue to search for $\boldsymbol{e}_{i+1}$ in the subtree with *Child*(*root,s*) as the root node. Otherwise, the decision of P is determined as discard. */

```
1. for (i: = 1 to k) do
2.   if (Bisearch (root, e_{L[i-1]})==true) do
3.     root = Child( root,s);
4.     continue;
5.   end if
6.   elseif P→discard;
7.     break;
8    end elseif
9. end for
10.  if (i>k)
11.    P→accept.
12.  end if
```
End

e_1=5, e_2=3. Starting from the root node, it can be quickly determined that e_1 matches the coordinate interval of the second child node ***b***[5,6]; Then continue to search on the subtree with ***b*** as the root node, e_2=3, which is included in the corresponding interval of the child node of ***d***[0,6]. So it can be determined that the classification decision of packet p_2 is accept. The specific process of packet classification is described in Algorithm 3.

STEP 5: Rule Update based on Decision Tree

In the model of rule mapping and decision tree construction based on cell space, only rules with a decision of accept are expressed as cell space, and correspondingly, each branch of the decision tree constructed based on cell space only corresponds to a rule whose decision is accept. Under this model, rule updates can be summarized into two cases: adding accept rules and adding discard rules. Because deleting a discard rule is equivalent to adding an accept rule in the corresponding area of the rule; deleting an accept rule is equivalent to adding a discard rule in the corresponding area; while modifying a rule can be regarded as deleting the part that needs to be modified before adding a new rule. In the following, we discuss the two cases of rule update respectively.

Case 1: Adding an accept rule (or deleting a discard rule)

According to the packet classification process and classification decision criteria of PcmSU method (if a matching node is found in any decision subtree, it can be determined that the packet decision is accept, and when all decision subtrees fail to match, the packet decision can be determined as discard). Therefore, different from the traditional packet classification method that must lookup the entire decision tree, PcmSU only needs to perform rule update operations in a specific single decision subtree for the case of adding accept rules (or deleting discard rules).

Determine whether there is a F_i-small interval in the accept rule r to be added, if so, send the rule to the decision subtree with the nodes on the corresponding dimension F_i are small intervals, and compare the spatial relationship between the rule interval of r and the node interval of the decision subtree on the dimension F_i. If they are independent of each other, directly add the rule r to the decision subtree as a new subtree of the root node. If it is intersection or other relationship, the decision tree reconstruction process as described in Algorithm 4 is performed. In addition, if the rule r is F_i-big interval on all dimensions $F_i(i = 1, 2, \ldots, k)$ (although the probability is very small), then we divide r into two rules on the halsection on the F_i dimension, according to the definition standard of threshold vector $T_{1/2}$, the two rules must be small intervals on dimension F_i, and they can be sent to the subtree T_i to perform similar operations as described above.

The foregoing experimental observation has confirmed that the vast majority of rules contain at least one F_i-small interval. If the node intervals of the decision subtree on the F_i dimension are also small interval, the probability that the rule interval of the rule to be added on the F_i dimension and the decision tree node interval are independent is very large, so the accept rule to be added (or the discard rule to be deleted) can be directly inserted, and the rule update efficiency is very high. Of course, it cannot be completely ruled out that the rule interval and the decision tree node interval are not independent, and the decision tree reconstruction is also performed according to Algorithm 4.

Case 2: Adding a discard rule (or deleting an accept rule)

When classifying packets based on the PcmSU method, the packet is determined as discard only when all decision subtrees cannot match the packet. Therefore, intuitively, the rule update in case 2 is equivalent to pruning the branches of the decision tree, and the update process needs to be performed simultaneously in all decision subtrees. Its essence is to search and delete the overlapping part of the discard rule to be added and the accept rule corresponding to the decision tree.

Next, we take the classification decision tree shown in Fig. 5 as an example to illustrate the two rule update cases. The two decision subtrees are F_1-small interval and F_2-small interval respectively.

First, consider "case 1": adding rule "r_1: $F_1 \in[9,9] \wedge F_2 \in[3,6] \rightarrow$ *accept*". Because the rule decision is accept, and it is a small interval on the F_1 dimension, we send it to

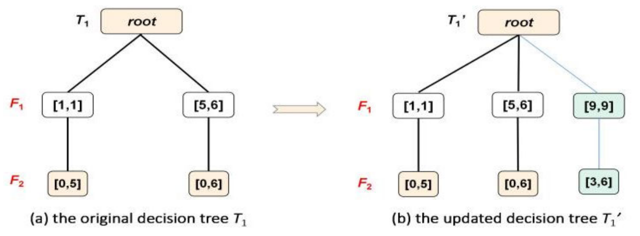


FIGURE 8. An example of rule update (adding rule "r_1: $F_1 \in [9, 9] \wedge F_2 \in [3, 6] \rightarrow$ accept").

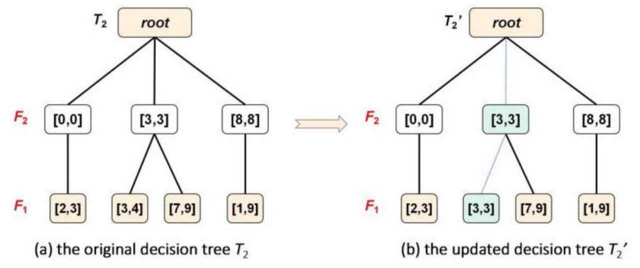


FIGURE 9. An example of rule update (adding rule "r_1: $F_1 \in [4, 4] \wedge F_2 \in [3, 6] \rightarrow$ discard").

the decision subtree T_1. Then compare the spatial relationship between the rule interval of r_1 on the F_1 dimension and the node interval of subtree T_1, because the rule interval of r_1 on the F_1 dimension is [9,9], which is independent from the node intervals [1,1] and [5,6] of subtree T_1 on the F_1 dimension, therefore, the rule r_1 can be directly added as a new subtree to the root node of T, as shown in Fig. 8.

Then analyze "case 2": adding rule "r_2: $F_1 \in [4,4] \wedge F_2 \in [3,6] \rightarrow$ *discard*". Because the rule decision is discard, it is necessary to search and delete the overlapping part of the discard rule to be added and all the accept rules corresponding to each subtree in Fig. 5. First, we search the decision subtree T_1, the rule interval of r_2 on the F_1 dimension is [4,4], which is independent of the node intervals [1,1] and [5,6] of subtree T_1 on the F_1 dimension. Therefore, it can be determined that there is no overlap between them, so no pruning operation is required for the subtree T_1.

Next, we search the decision subtree T_2, the rule interval of rule r_2 on the F_2 dimension is [3,6]. Compared with the node interval {[0,0],[3,3],[8,8]} of subtree T_2 on the F_2 dimension, there is a "*covered*" relationship at the interval [3,3]. The rule interval [3,6] is divided into two parts that are identical and adjacent to the subtree node interval: [3,3] and [4,6]. At [3,3], we continue to compare with the next dimension F_1, where the rule interval [4,4] on the F_1 dimension intersects with the subtree node interval [3,4], we delete the equivalent interval [4,4], and the subtree node interval is updated to [3,3] after deleting the equivalent interval [4,4]. The update result is shown in Fig. 9.

It can be seen that when adding a discard rule (or deleting an accept rule), the PcmSU method only needs to perform up to two rounds of pruning operations on the decision tree,

Algorithm 4 Rule Update

```
Input: the $k$ decision subtrees $T_i (i = 1, 2, \ldots, k)$, and the rule $R_u$ to be updated.
Output: the $k$ updated decision subtrees $T_i' (i = 1, 2, \ldots, k)$.
/* The rule updating can be adding an accept rule or adding a discard rule*/
Begin
    1. if $R_u$. decision='accept' then
    2.     for($i := 1, 2, \ldots, k$) do
    3.         if $R_u$ has a $F_i$-small interval, then PruningTree($R_u$, $T_i$, 1), break;
    4.         elseif for each dimension $F_i$, $R_u$ is $F_i$-big interval, then half-divide $R_u$ into two rules $R_{u1}$ and $R_{u2}$ on the $F_i$ dimension, do PruningTree($R_{u1}$, $T_i$, 1) and PruningTree($R_{u2}$, $T_i$, 1); break;}
    5.     end for
    6. end if
    7. elseif $R_u$. decision='discard' then
    8.     for($i := 1, 2, \ldots, n$) do PruningTree($R_u$, $T_i$, 0);
    9. end elseif
    10. return $T_i' (i = 1, 2, \ldots, k)$;
End
PruningTree($R$, $T$, $d$){
/* Let the dimension array corresponding to the decision tree $T$ from the first layer child nodes to the leaf node be denoted as dim[k]. */
    11. for(i: =dim[0], dim [1],..., dim[k−1]) do
    12.     c=Relation(R, T, dim[i]); //c is the spatial relationship between Rrule interval and T node interval on the dimension dim[i].
    13.     switch c{
    14.     "disjunctive":
    15.     if d='1' then add(R,T,i); //Add rule R to the i-th layer of T from dim[i] to dim[k−1] (denoted by R[dim[i], dim[k−1]]) as a new subtree of T's [k−i] layer.
    16.     if d='0' then ignore R;
    17.     "adjacent":
    18.     if d='1' then add(R,T,i);
    19.     if d='0' then ignore R;
    20.     "equivalent":
    21.     if d='1' then continue until i=dim[k-1], ignore R;
    22.     if d='0' then continue until i=dim[k-1], do delete(R,T); //Remove the nodes with the same interval as R from T; and if the node has no sibling, delete its parent node. The operation is backtracked until i=dim[0].
    23.     "covered":
    24.     Divide R into (1) the part that is equivalent with the T node interval is recorded as R.eq; (2) the part adjacent to the T node interval is recorded as R.ad.
    25.     if d='1' then perform steps (21) in the R.eq interval; perform steps (18) in the R.ad interval;
    26.     if d='0' then perform steps (22) in the R.eq interval; perform steps (19) in the R.ad interval;
    27.     "included":
    28.     Divide the T node interval into (1) the part that is equivalent with the T node interval is recorded as R.eq; (2) the remaining part of the T node interval after cutting out R.eq is recorded as R.rm. The original node interval of T is modified to R.rm, and the subsequent nodes on all dimensions remain unchanged.
    29.     if d='1' then perform steps (21) in the R.eq interval;
    30.     if d='0' then perform steps (22) in the R.eq interval;
    31.     "crossed":
    32.     Divide R into (1) the part that is equivalent with the T node interval is recorded as R.eq; (2) the part adjacent to the T node interval is recorded as R.ad; At this time, T is also divided into two parts, one of which is R.eq, and the other part is the remaining interval after dividing out R.eq, which is recorded as R.rm; The original node interval of T is modified to R.rm, and the subsequent nodes in all dimensions remain unchanged.
    33.     if d='1' then perform steps (21) in the R.eq interval; perform steps (18) in the R.ad interval;
    34.     if d='0' then perform steps (22) in the R.eq interval; perform steps (19) in the R.ad interval;
} //End of PruningTree(R, T, d)
```

and if there is no overlap between the rule interval and the node interval of the decision tree on a certain dimension, then no pruning operation is required in subsequent dimensions. In addition, compared with the traditional decision tree-based classification method, the PcmSU method has another advantage: as the decision tree is divided into several subtrees, its scale must be reduced to a certain extent, which on the one hand reduces the complexity of decision tree reconstruction when rules are updated, and on the other hand, it also provides the foundation for distributed packet classification method [28], and improves the efficiency of packet classification. The execution process of the above rule update can be extended to the general k-dimensional rule case, as described in Algorithm 4.

D. PERFORMANCE EVALUATION AND ANALYSIS

1) THEORETICAL ANALYSIS

Essentially, the PcmSU method framework consists of the following three stages: (1)rule preprocessing, (2)cell space set partitioning and decision tree construction, (3)packet classification and rule update based on decision tree. Because the two stages of rule preprocessing, cell space set partitioning and decision tree construction can be carried out offline, we will not discuss the time efficiency of these two stages in detail, only briefly explain their memory consumption. In the experimental verification part, we will introduce some experimental results and observation analysis of cell space set partitioning and decision tree construction. This part focuses on the time efficiency of PcmSU in packet classification and rule update.

In the process of algorithm implementation, we use the linked list structure to store each node in the decision tree. Specifically, set a child linked list for each node in the tree, and store these nodes and the head pointer of the corresponding child linked list in a vector, where each linked list element corresponds to the node of the decision tree. Assuming that the number of nodes in the decision tree is N_{i}, when the linked list structure is used to store the corresponding decision tree, it is necessary to store N_{i} pointer information and $2 \cdot N_{\mathrm{i}}$ node information. On 32-bit machines, the pointer size is 4 bytes, and each node is represented by an address interval, corresponding to a maximum of two 32bit addresses. Therefore, the storage space required for the corresponding PcmSU method is not greater than $N_{\mathrm{i}} \cdot 20 / 2^{20}$ MB.

In general, for decision tree-based classification methods, the storage structure of all dimensional nodes is the same except for the leaf node in the last dimension. However, we notice that the traditional decision tree-based classification method not only stores the node information, but also stores all rules in the rule group corresponding to the leaf node. Let N_{s} be the sum of the number of rules corresponding to the leaf nodes of the decision tree, and a single firewall rule represented in the form of interval corresponds to 28 bytes of storage space, so the total rule storage space is $N_{\mathrm{s}} \cdot 28/2^{20}$ MB. Comparatively speaking, in PcmSU method, there are only one rule associated with each leaf node in the decision tree. Therefore, this method does not need additional space to store grouping rules, which reduces the storage space requirement to some extent.

In the third stage of this method, assuming that the number of original rules is n, according to the rule mapping method [5] based on multi-dimensional matrix, the number of formed cell spaces is generally not more than n, and the projection interval of n cell spaces on any dimension is at most $2n$-1. According to the decision tree construction process of the PcmSU method, the number of child nodes of the decision tree or any of its subtrees will not exceed $2n$, so the search time corresponding to the worst-case is $log_2(2n)$. Therefore, the worst case time complexity of packet classification on the k-layer decision tree is $O(T_{\mathrm{worst}}) = O(k \cdot log_2(n))$. In addition, according to the PcmSU method, the cell space set will be divided before the decision tree is constructed, so the number of cell spaces contained in each subset will be less, calculated according to the average division into k subsets, and the number of cell spaces in each subset is approximately n/k. Therefore, from theoretical analysis, it can be seen that the PcmSU method can achieve logarithmic packet classification time efficiency.

For the above rule update "case 1", if the rule R_{u} to be updated contains a F_{i}-*small interval*, R_{u} is sent to the corresponding subtree T_{i} to perform $PruningTree(R_{\mathrm{u}}, T_{\mathrm{i}},1)$ operation. For $PruningTree(R_{\mathrm{u}}, T_{\mathrm{i}},1)$, firstly, the spatial relationship between the rule interval on *dim*[i] and each node interval of T is obtained. if it is "*disjunctive*" or "*adjacent*", the rule R_{u} is directly added to the i-th layer of T_{i} from *dim*[i] to *dim*[k−1] dimension as a new subtree of T_{i}'s (k-i)-layer, and its time complexity is related to the number of nodes, but because the node interval value is increasing, the binary-search method can be used. If the number of node intervals is n, the complexity of this operation is $O(log_2(n))$. If the spatial relationship between the two intervals is "*equivalent*", then continue to determine the next dimension until *dim*[k−1], and if the spatial relationship is still "*equivalent*", then ignore the rule. The process requires k comparison operations, the complexity is $O(k \cdot log_2(n))$. If it is "*covered*", "*included*", or "*crossed*" relationship, the rule interval or node interval is cut into sub-intervals with only "*equivalent*" or "*adjacent*"s relationship, and then the above search and update operations are performed, and the operations such as cutting and updating are performed up to k rounds. In addition, compared with "case 1", which only needs to update rules on one decision subtree, "case 2" needs to perform updates in all decision subtrees, but this process can be operated in parallel on multiple computing nodes.

Therefore, from the perspective of theoretical analysis, the effectiveness of PcmSU depends on the spatial relationship between the rule interval and the decision tree node interval. Although the rule update still needs to reconstruct the decision tree, some characteristics based on the small interval can be used to reduce the complexity of rule update. Next,

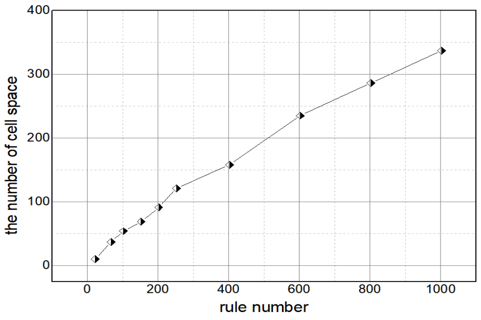


FIGURE 10. Comparison between the number of rules vs. the number of mapped cell spaces.

we give more observations from the perspective of experimental analysis.

2) EXPERIMENTAL VERIFICATION

Firstly, in response to the conclusion mentioned in the theoretical analysis section: "According to the rule mapping method based on multi-dimensional matrix, the number of formed cell spaces is generally not more than n", we conducted rule mapping experiments on 10 rule sets (including 2 real rule sets and 8 rule sets generated by ClassBench). Fig. 10 shows the comparison results between the rule number and the number of cell spaces after mapping.

As can be seen from Fig. 10, the number of cell spaces is much smaller than the number of original rules. This is because the cell space only reflects the rules of decision-making as accept; On the other hand, the rule mapping method based on multidimensional matrix will eliminate the redundancy among the original rules.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we introduce some experimental results of this method. We first summarize our experimental methods. Then, we evaluate our algorithm from several key aspects, such as decision tree construction, packet classification and rule update. Our experiments were carried out on an Intel (R) Core (TM) i7-10700 machine running Windows 10 with 16G memory and 2.90GHz processor.

We use ClassBench to generate five-tuple firewall (FW) rule sets, which range in size from 1k to 100k. Rule sets are named by their type and size, such as FW_1k referring to a firewall rule set with about 1000 rules. At the same time, in order to realize the parallel processing of packet classification, we build a Hadoop platform with one primary node and four secondary nodes, because this article only considers dividing small cell space and constructing decision subtrees on the four fields of source address, destination address, source port and destination port. We measure the classification time by classifying the packets generated when ClassBench constructs the rule set, and evaluate the rule update performance by the time required to insert or delete a rule. In order to reduce the CPU jitter error, we average the results by running five times for each evaluation cycle. It should be pointed out that rule preprocessing and decision tree construction can be carried out offline in advance. After each classification decision subtree is deployed to the computing node, online packet classification and rule update can be performed in parallel.

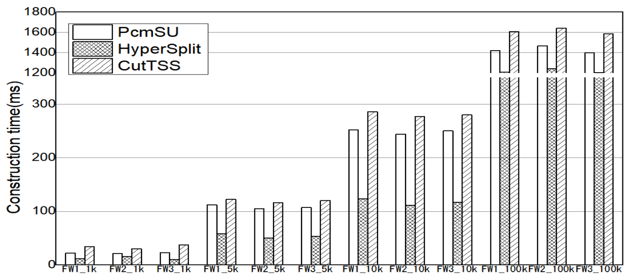


FIGURE 11. Comparison of decision tree construction time.

We compare PcmSU with three algorithms: TupleMerge, HyperSplit, and CutTSS. TupleMerge is a TSS-based optimization algorithm with high update performance. HyperSplit is a classic decision tree method with excellent classification performance. CutTSS is an advanced classification method based on split tree, which has a good performance trade-off between classification and update.

A. EVALUATION ON DECISION TREE CONSTRUCTION

1) CONSTRUCTION TIME

Fig. 11 shows the decision tree construction time for PcmSU, HyperSplit, and CutTSS. In contrast, PcmSU takes a little more time than HyperSplit, because it needs to go through the process of rule mapping and cell space division before constructing the decision tree. but this process can be carried out offline in advance, and does not affect online classification and rule update. However, even for rule sets up to 100k, PcmSU can still construct decision trees in less than 1.5 seconds, and the decision tree construction time increases linearly with the size of the rule set, which makes it suitable for large classifiers.

2) MEMORY CONSUMPTION

Fig. 12 shows the memory consumption of PcmSU, HyperSplit and CutTSS. Experimental results show that PcmSU requires less memory consumption compared to other algorithms, for example, when tested on FW-10K, CutTSS successfully constructs a decision tree that occupies 426KB of memory, while PcmSU only consumes 168KB of memory. We also found that as the ruleset size increased, the subset of PcmSU and CutTSS were not greatly affected, especially since PcmSU was always divided into only four decision subtrees. However, with the increase of the number of rule sets, HyperSplit's memory consumption increases dramatically.

B. EVALUATION ON PACKET CLASSIFICATION

To evaluate the efficiency of packet classification, we selected three datasets of different sizes (1MB, 100MB, and 200MB, respectively) to compare the time required for packet classification using PcmSU, TupleMerge, HyperSplit, and CutTSS.

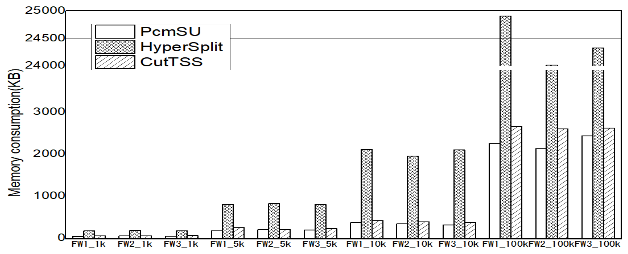


FIGURE 12. Memory consumption for PcmSU, HyperSplit and CutTSS.

TABLE 4. Classification time() when the packet size is 1MB.

FW rule size	FW-1k	FW-5k	FW-10k	FW-100k
TupleMerge	24.2	65.7	176.5	1114.8
HyperSplit	3.6	12.8	18.6	205.4
CutTSS	5.8	16.5	30.2	269.3
PcmSU	4.5	11.8	21.2	183.4

TABLE 5. Classification time() when the packet size is 100MB.

FW rule size	FW-1k	FW-5k	FW-10k	FW-100k
TupleMerge	2.0245	5.6131	22.0614	148.1506
HyperSplit	0.3824	1.4223	2.9815	24.2142
CutTSS	0.5282	1.4384	3.3083	27.0925
PcmSU	0.4115	1.6572	3.1412	19.3005

TABLE 6. Classification time() when the packet size is 200MB.

FW rule size	FW-1k	FW-5k	FW-10k	FW-100k
TupleMerge	3.3221	10.2156	29.1388	222.7685
HyperSplit	0.5376	1.1165	5.0082	48.6625
CutTSS	0.7264	1.7525	6.1328	52.2262
PcmSU	0.6056	1.3053	5.1445	30.3538

TABLE 4, TABLE 5, and TABLE 6 show the average classification time of each method. It can be seen that PcmSU takes less time to classify packets, and the average classification time of PcmSU improves by nearly an order of magnitude on average compared to TupleMerge.

The experimental results show that the average acceleration ratio of PcmSU is 1.95 times and 1.58 times that of HyperSplit and CutTSS, respectively. Especially with the increase of packet size, the classification speed advantage of PcmSU over TupleMerge and others becomes more and more obvious. For example, when the packet is 100MB, the PcmSU algorithm has a classification time of about 1/11 of the TupleMerge algorithm. The main reason is that when PcmSU classifies packets, on the one hand, it can use the binary-search method to classify packets, and on the other hand, it can further improve the packet classification efficiency by sending packets to each decision subtree for distributed lookup.

C. EVALUATION OF RULE UPDATE

Fig. 13 shows the average update time of PcmSU, TupleMerge, HyperSplit and CutTSS, and we can see that the time required for PcmSU to update rules is very small, with each

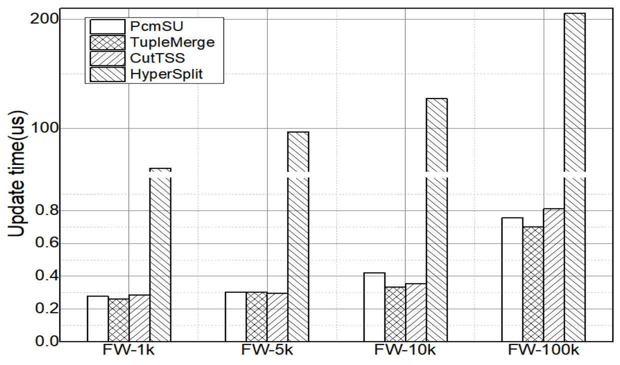


FIGURE 13. Comparison of rules update efficiency.

type of rule set reaching an average time of $0.322us$, $0.293us$, $0.381us$ and $0.727us$ respectively. In contrast, the HyperSplit method based on decision tree has lower update performance. As a result, PcmSU has the comparable update performance to TupleMerge and CutTSS.

The results show that PcmSU is close to the state-of-the-art solution in terms of update speed. Considering the parallel processing capability and fast classification speed of PcmSU, the method can be a better choice in some specific working scenarios, such as the changeable SDN environment.

V. CONCLUSION

In order to achieve fast search and update at the same time, this article proposes a high-performance packet classification method based on decision tree, PcmSU, which divides the rule mapping space to form a series of independent subsets of cell space, and each subset has the same characteristic: the mapping interval of cell space on a certain dimension F_i is small interval. On this basis, multiple classification decision trees are constructed, which can perform packet classification and rule update in parallel.

The method optimizes the searching and updating process in the traditional decision tree-based classification method, and the classification efficiency of the constructed decision tree is greatly improved compared with that before optimization. Based on the characteristics of the interval, the reconstruction of the decision tree is greatly reduced, and the rule update efficiency is improved. In addition, based on the characteristics of the decision tree constructed by this method, when adding rules, it unnecessary to consider the specific location and sequence number of rule updates, which can avoid introducing rule conflicts. Experimental results show that PcmSU not only supports high-speed packet classification and linear memory consumption, but also has fast rule updating speed.

In our future work, we will develop software-defined networking applications that utilized our classification method, and evaluate the feasibility of our method with corresponding experiments. In addition, we will explore more applications of deep learning [29], [30] in decision trees and provide reference for the study of packet classification.

REFERENCES

[1] J. Daly, V. Bruschi, L. Linguaglossa, S. Pontarelli, D. Rossi, J. Tollet, E. Torng, and A. Yourtchenko, "TupleMerge: Fast software packet processing for online packet classification," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1417–1431, Aug. 2019.
[2] P. He, G. Xie, K. Salamatian, and L. Mathy, "Meta-algorithms for software-based packet classification," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Raleigh, NC, USA, Oct. 2014, pp. 308–319.
[3] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, "A sorted-partitioning approach to fast and scalable dynamic packet classification," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1907–1920, Aug. 2018.
[4] Y. Cheng and Q. Shi, "PCMIgr: A fast packet classification method based on information gain ratio," *J. Supercomput.*, vol. 79, no. 7, pp. 7414–7437, May 2023.
[5] Y. Cheng, W. Wang, G. Min, and J. Wang, "A new approach to designing firewall based on multidimensional matrix," *Concurrency Comput., Pract. Exper.*, vol. 27, no. 12, pp. 3075–3088, Aug. 2015.
[6] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" in *Proc. 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, Oct. 2003, pp. 53–63.
[7] Y. Li, J. Wang, X. Chen, and J. Wu, "SplitTrie: A fast update packet classification algorithm with trie splitting," *Electronics*, vol. 11, no. 2, pp. 199–211, 2022.
[8] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. IEEE Annu. Symp. High-Perform. Interconnects (HOTI)*, 1999, pp. 34–41.
[9] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," *Comput. Commun. Rev.*, vol. 33, no. 4, pp. 213–224, 2003.
[10] H. Lim, N. Lee, G. Jin, J. Lee, Y. Choi, and C. Yim, "Boundary cutting for packet classification," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 443–456, Apr. 2014.
[11] W. Li and X. Li, "HybridCuts: A scheme combining decomposition and cutting for packet classification," in *Proc. IEEE 21st Annu. Symp. High-Perform. Interconnects*, Santa Jose, CA, USA, Aug. 2013, pp. 41–48.
[12] Z. Liu, S. Sun, H. Zhu, J. Gao, and J. Li, "BitCuts: A fast packet classification algorithm using bit-level cutting," *Comput. Commun.*, vol. 109, pp. 38–52, Sep. 2017.
[13] J. Daly and E. Torng, "ByteCuts: Fast packet classification by Interior bit extraction," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)* Honolulu, HI, USA, Apr. 2018, pp. 16–19.
[14] M. Abbasi, S. V. Fazel, and M. Rafiee, "MBitCuts: Optimal bit-level cutting in geometric space packet classification," *J. Supercomput.*, vol. 76, no. 4, pp. 3105–3128, Apr. 2020.
[15] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 648–656.
[16] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "ParaSplit: A scalable architecture on FPGA for terabit packet classification," in *Proc. IEEE 20th Annu. Symp. High-Perform. Interconnects*, Santa Clara, CA, USA, Aug. 2012, pp. 1–8.
[17] 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.*, Honolulu, HI, USA, Apr. 2018, pp. 2645–2653.
[18] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 135–146, Oct. 1999.
[19] H. Alimohammadi and M. Ahmadi, "Common non-wildcard portion-based partitioning approach to SDN many-field packet classification," *Comput. Netw.*, vol. 181, Nov. 2020, Art. no. 107534.
[20] C. Zhang, G. Xie, and X. Wang, "DynamicTuple: The dynamic adaptive tuple for high-performance packet classification," *Comput. Netw.*, vol. 202, Jan. 2022, Art. no. 108630.
[21] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: Optimizing packet classification for memory and throughput," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 207–218, Aug. 2010.
[22] V. S. M. Srinivasavarma and S. Vidhyut, "A TCAM-based caching architecture framework for packet classification," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 1, pp. 1–19, Jan. 2021.
[23] C. Li, T. Li, J. Li, D. Li, and B. Wang, "Memory optimization for bit-vector-based packet classification on FPGA," *Electronics*, vol. 8, no. 10, pp. 1159–1174, 2019.
[24] W. Li, T. Yang, O. Rottenstreich, X. Li, G. Xie, H. Li, B. Vamanan, D. Li, and H. Lin, "Tuple space assisted packet classification with high performance on both search and update," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1555–1569, Jul. 2020.
[25] 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.
[26] Y. Cheng, W. Wang, J. Wang, and H. Wang, "FPC: A new approach to firewall policies compression," *Tsinghua Sci. Technol.*, vol. 24, no. 1, pp. 65–76, Feb. 2019.
[27] M. H. Overmars and F. A. van der Stappen, "Range searching and point location among fat objects," *J. Algorithms*, vol. 21, no. 3, pp. 629–656, Nov. 1996.
[28] Y. Cheng and Q. Shi, "MpFPC–A parallelization method for fast packet classification," *IEEE Access*, vol. 10, pp. 38379–38390, 2022.
[29] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," in *Proc. ACM Special Interest Group Data Commun.*, Beijing, China, Aug. 2019, pp. 256–269.
[30] N. Hubballi and P. Khandait, "KeyClass: Efficient keyword matching for network traffic classification," *Comput. Commun.*, vol. 185, pp. 79–91, Mar. 2022.

YUZHU CHENG received the B.Sc. degree from the Hunan University of Science and Technology, in 2002, the M.Sc. degree in software engineering from Hunan University, in 2005, and the Ph.D. degree in computer science and technology from Central South University, in 2018. In 2006, he joined the Changsha Social Work College, where he is currently an Associate Professor. He is the author of three books and has published more than 30 articles in refereed journals and conference proceedings. His research interests include natural language processing, pattern recognition, and network security.

YIHANG XU received the B.Sc. degree from Central South University, in 2021, where she is currently pursuing the M.Sc. degree with the Department of Resource and Safety Engineering. Her research interests include network security and software engineering.

QIUYING SHI received the B.Sc. degree from Hunan Normal University and the M.Sc. degree in computer science and technology from Central South University. She was a Researcher with the Changsha Social Work College. Her research interests include cyber-security and big data.