IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 24, 2022, accepted March 30, 2022, date of publication April 8, 2022, date of current version April 14, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3165157

MpFPC–A Parallelization Method for Fast Packet Classification

YUZHU CHENG[1] **AND QIUYING SHI**[2]
[1]School of Software, Changsha Social Work College, Changsha 410004, China
[2]School of Computer Science and Engineering, Central South University, Changsha 410083, China

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

This work was supported in part by the Natural Science Foundation of Hunan Province under Grant 2022JJ60099, in part by the Research Foundation of the Education Department of Hunan Province under Grant 21C1589, in part by the Doctoral Scientific Research Project of Changsha Social Work College under Grant 2020JB32, and in part by the National Natural Science Foundation of China under Grant 61877059.

ABSTRACT Packet classification is the core technology of network layer and an important means to ensure the security of network system. With the rapid development of network technology, higher requirements are put forward for the speed of network packet classification. This paper improves the traditional single thread package classification framework, A new parallelization method for fast packet classification (MpFPC) based on distributed computing is proposed, the method adopts the packet classification idea based on decision tree, but compared with the traditional algorithm, a rule mapping preprocessing process is added before constructing the classification decision tree, which effectively removes the rule redundancy and conflict, so as to avoid the rule replication problem of the traditional decision-tree-based method. In addition, the method can group the rules and data packets at the same time, which improves the packet classification efficiency. Experimental results show that MpFPC method has high classification efficiency and has obvious speed advantage compared with Uscuts method with time complexity of $O(k\log n)$. In addition, the test results also show that the classification speed of MpFPC will increase with the increasing number of computing nodes, which provides a new possible way to meet the classification wire-speed requirement.

INDEX TERMS Packet classification, cell space, decision tree, parallel computing.

I. INTRODUCTION

Packet classification is the core technology to Internet devices and network services implementation, it searches for the operation or task to be performed by the packet in a set of rules, according to a series of information carried by the specified packet (such as source address, destination address, source port, destination port and protocol, etc.) [10]. With the development of a series of cutting-edge network technologies, such as network function virtualization (NFV) and software-defined network (SDN), the Internet carries more and more application services, the scale of network traffic, backbone network routing table and firewall access control rules have increased explosively, which also puts forward higher requirements for the packet classification ability of network equipment [10]. With the continuous improvement of network communication cable manufacturing technology and the increasing link bandwidth, the optical fiber transmission rate of the current universal interface has reached or even exceeded 400Gbps [3]. In terms of 64 bytes per packet, in order to achieve wire-speed requirement, it is necessary to complete a packet classification within 1.28ns. However, existing software packet classification algorithms generally only have the ability to classify a particular dimension, especially the single dimensional classification and two dimensional classification algorithms. Some algorithms can be extended for the high dimensional classification, but it cannot meet the requirements in the space complexity and time complexity simultaneously.

Considering these requirements, we designed a novel parallelization method for fast packet classification (MpFPC) based on distributed computing. The proposed method has advantages of high efficiency without additional space requirement, and it is suitable for a large amount of data, especially for real-time packet classification which has the requirement of high speed. Our work contributes a new parallelization method for fast packet classification to improve the classification speed. The main innovations of this method

The associate editor coordinating the review of this manuscript and approving it for publication was Victor S. Sheng.

are as follows: Firstly, the decision tree is grouped into several sub-decision trees for parallel packet classification, and the improved grouping process can avoid the rule replication problem of the traditional decision-tree-based method; Secondly, during classification, each data packet will be compared according to the corresponding interval value of the root node of the decision sub-tree. After finding the matching interval, it will be distributed to the corresponding decision subtree for classification. However, the packet that cannot match any decision subtree can directly be given the classification decision of *discard*, which also improves the packet classification efficiency to a certain extent.

The rest of this paper is organized as follows: the related work is presented in Section 2; then we give the problem description, and present the algorithms of classification decision tree constructing, rule grouping and distributed packet classifying in Section 3. In Section 4, we give the classification results of MpFPC method and Uscuts method, followed by the comparison and analysis of experimental results. Finally, the conclusion is drawn in Section 5.

II. RELATED WORK

Packet classification is a hard problem with high complexity. From a geometric point of view, packet classification can be treated as a point location problem, which has been proved that the best bounds for locating a point are either $O(\log n)$ time with $O(n^k)$ space, or $O((\log n)^{k-1})$ time with $O(n)$ space for n non-overlapping hyper-rectangles in k-dimensional space [4]. Therefore, the worst case mathematical complexity of algorithmic packet classification is extremely high, which makes it impractical to achieve a wire-speed requirement within the capabilities of current memory technology. However, packet classification rules in real-life applications have some inherent characteristics that can be exploited to reduce the complexity.

A. HARDWARE-BASED METHODS

In industry, large routers and high-end classifiers use hardware devices, such as ternary content addressable memory (TCAM) [5], field programmable gate array (FPGA) [6] and specialized network processor chips, to achieve high-performance packet classification. The hardware-based classification method can carry out high-speed classification, but it is expensive, consumes a lot of power, has low flexibility and scalability [5]. Therefore, in academia, researchers are more willing to seek solutions based on software for packet classification [7].

According to the existing research, packet classification algorithms can be roughly divided into algorithms based on dimension decomposition and algorithms based on space division [8]. The algorithm based on space division usually partitions the whole rule space into several subspaces, then divides the rule set into several groups and puts them into each subspace. This type of method can be further divided into two main subcategories: tuple space-based method [9]–[11] and decision tree-based method [12], [13].

B. DIMENSION-DECOMPOSITION-BASED METHODS

The algorithm based on dimension decomposition decomposes each rule into multiple dimensions in a certain number of bytes or bits. Each dimension is searched separately, and then combined to obtain the final search result, representative classical algorithms include BV(Bit vector), ABV(Aggregated bit vector), RFC(Recursive flow classification) [14]–[16], etc. These methods are fast, but with the increase of the size of the rule set, the space consumption will increase exponentially in the worst case, and the space consumption is high.

C. TUPLE-SPACE-SEARCH-BASED METHODS

The algorithm based on tuple space constructs a hash table for each different prefix length, and the rule components with the same prefix length are stored in the same hash table. When classifying packets, all hash tables are accessed sequentially until the longest matching prefix is found. Representative algorithms include multi-dimensional packet classification algorithms TSS(Tuple Space Search) [17], PartitionSort [18], and TupleMerge [12], etc.

D. DECISION-TREE-BASED METHODS

The algorithm based on decision tree recursively decomposes the multidimensional space where the rules are located in a certain way, and establishes a decision tree. The root node of the decision tree represents the whole multidimensional space, and the non-root node represents the subspace. The end condition of recursive decomposition is that the number of rules associated with nodes is less than or equal to the preset threshold . When receiving a packet, the method first access the root node of the decision tree, and then decide how to access the next node according to the space division. Usually, the packet classification algorithm based on decision tree will eventually access the leaf node. When accessing a leaf node, the algorithm sequentially accesses at most rules associated with the leaf node to determine the rule that the packet can match. Representative classical algorithms include Hicuts [19], HyperCuts [20], etc.

Generally speaking, the methods based on tuple space have the problems of uneven subspace distribution and rule replication, which affect the classification performance of the methods to a certain extent. Although traversing the decision tree can achieve logarithmic time complexity, it need to perform sequence matching within rules after finding the leaf node of the decision tree. Obviously, the execution time efficiency of sequence matching is linear with the number of rules, which reduces the classification efficiency of the method to a great extent. In addition, when dividing the multidimensional space where the rule is located, a rule may need to be copied to multiple rule groups, resulting in a large amount of storage space for the method. Efficuts, SmartSplit and other methods [21]–[23] have some improvements in the depth of constructing decision tree compared with classical methods, but they do not fundamentally solved the problem

of sequential matching after rule replication and matching to leaf nodes. From the perspective of solving these two problems, a decision tree packet classification method Uscuts based on cell space division is proposed. The classification time complexity of this method is $O(k\log n)$ [24]. However, for the classification of massive data packets, its classification speed may still become a bottleneck.

At present, compared with other classification methods based on software, the method based on decision tree has the most advantages in classification speed. Especially in the data center network environment, there may be massive data packets to be classified instantaneously. At this time, the classification speed is the most important. However, by analyzing the current packet classification algorithm, there is still a long distance from the wire-speed requirements, which directly affects and limits the application scalability of network devices in the new generation Internet. Considering that the industry mostly uses parallelization methods to process massive data, if both the data packets and rules can be grouped, the packet classification problem may be handled in a parallel manner. Based on this idea, this paper proposes a new parallelization method for fast packet classification based on distributed computing.

III. THE PROPOSED APPROACH

A. PROBLEM DESCRIPTION

According to the analysis of previous studies, the biggest obstacle to the direct grouping of rules is that rules are often entangled, and there are overlaps and conflicts between rules [24]. Therefore, if the rules are divided into several groups and the data packets are distributed to these groups for classification, the classification results will generally be inconsistent with that of the original rules. As shown in Figure 1, the classification rule set contains nine rules, which are divided into three rule groups $Rg1$, $Rg2$ and $Rg3$. Suppose that packet $\boldsymbol{e}(5, 3)$ passes through policy R and matches nine rules from top to bottom, it will first match r_2, and its classification result is *discard*; Accordingly, $Rg1$ and $Rg2$ can match packet $\boldsymbol{e}$ too, but their classification decisions are different, namely *discard* and *accept*. However, there are no rules in $Rg3$ can match the data packet $\boldsymbol{e}$. It can be seen that if there are conflicts in classification rules, the direct division of rules will lead to ''inconsistency'' errors [25] in classification decisions. Moreover, copying data packets to each rule group for classification will increase the memory consumption due to replication on the one hand, and the ''incomplete'' error [25] of the rule will occur because the grouping rules cannot match the data packets on the other hand.

Therefore, the first problem to be solved is how to deal with the original classification rules so that the rules are independent of each other, but the rule semantics remain unchanged. The second problem is how to group these rules and divide them equally to each parallel computing node, and the semantics of the whole rules divided to each node must be the same as that of the original rules. In addition, generally speaking, if each node has the same computing power, the more average the number of rules deployed to each node, the higher the computing efficiency. Finally, the method also needs to consider how to properly group the massive data packets to be classified, and send the grouped data packets to each computing node for classification based on decision tree.

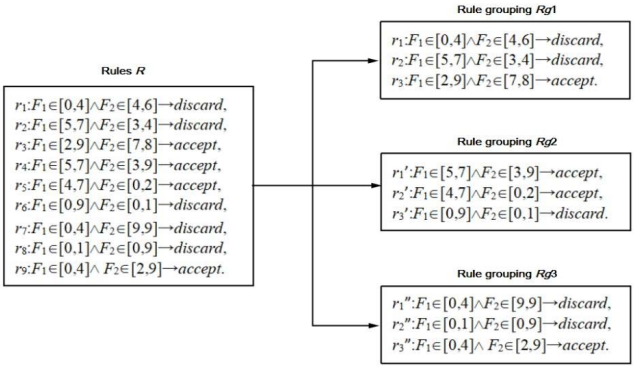

FIGURE 1. An intuitive example of rules direct grouping.

Firstly, we briefly introduce the solutions to the above problems, and then explain the specific steps of the method in detail with examples. To solve the problem of how to make the rules independent of each other while the semantics of the rules unchanged, we put forward a solution based on multidimensional matrix mapping (FDM) in our previous work [26]. The method can preprocess the original rules so that the target rules have the same semantics as the original rules, but the rules are independent of each other. The result of rule mapping in this method is a series of independent cell spaces, an example of mapping process is shown in Section 3.2. How to divide all rules evenly into pre-deployed computing nodes will be more complex. We need to comprehensively consider the efficiency of the subsequent packet classification process, and it is difficult to achieve absolute average in most cases. Let the mean value equals to the total number of rules divided by the number of computing nodes, we design a division method based on greedy strategy. Each time we partition the rules close to the mean value to the existing nodes until all rules are divided. Finally, we consider how to distribute the massive data packets to be classified to each computing node.

In this paper, we specify the distribution of massive data packets according to the root node interval of the subtree corresponding to each grouping rule, so as to avoid replication in the process of packet distribution. In addition, when constructing the classification decision tree, because the rules are non-conflict and redundancy-free after FDM mapping, there is no rule replication in the decision tree, which not only reduces the memory consumption, but also improves the classification efficiency of the decision tree.

For ease of understanding, we will supplement the division method based on greedy idea with Figure 2. Let the decision

tree T be composed of several subtrees in the F_1 dimension. As shown in Fig. 2, the decision tree T has six subtrees corresponding to $T_1, T_2, \ldots, T_6$ respectively. Suppose the number of branches in each subtree (we define the path from the root node to the leaf node of the subtree as a branch, and the number of branches corresponds to the number of rules) are {7, 5, 3, 4, 2, 4}, then the total number of rules is $N = 7 + 5 + 3 + 4 + 2 + 4 = 25$. Assuming that the number of deployed parallel computing nodes $\boldsymbol{n}$ equals '4,' then the average number of rules to be divided by each node is $N/n = 25/4 = 6.25$. The partition method based on greedy idea is to take the subtree with the number of rules as close as possible to N/n and deploy it to the corresponding computing node, while the subtree is not split. According to this method, all rules in the subtree are divided into each computing node.

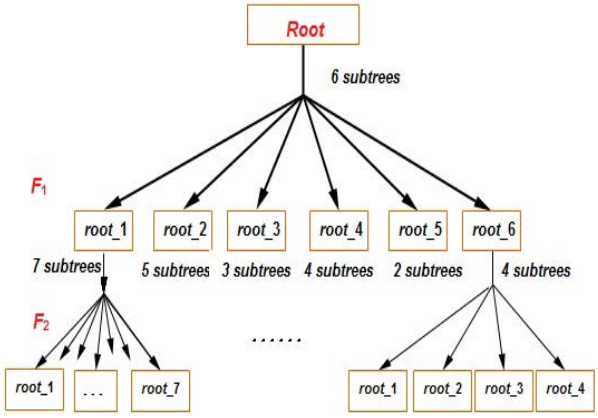

FIGURE 2. An decision tree T with six subtrees.

As shown in Fig. 3, subtree T_1 contains seven rules and T_2 contains five rules. Obviously, compared with $T_1 + T_2$, the number of rules contained in T_1 is closer to the average number of rules '6,' that is, $|7 - 6| < |(7 + 5) - 6|$, so the seven rules in T_1 are divided into node1, similarly, the five rules in T_2 are divided into node2, and the seven rules in $T_3 + T_4$ are divided into node3. Finally, all the remaining six rules in $T_5 + T_6$ are divided into node4.

Next, we briefly explain the execution steps of the algorithm. For the convenience of description, we use a small example of 2-tuple rule set shown in Figure 5 for subsequent discussions.

B. MpFPC ALGORITHM

The implementation process of the algorithm includes the following four steps: (1) Preprocessing the original rules, mapping the rules in the multidimensional matrix space through the rule mapping method to form a series of independent cell spaces; (2) In each dimension, the multidimensional matrix space where the rules are located is divided in turn to build a classification decision tree; (3) Based on the number of deployed parallel computing nodes and the number of branches in each subtree, the subtrees are distributed to each computing node as evenly as possible; (4) According to the

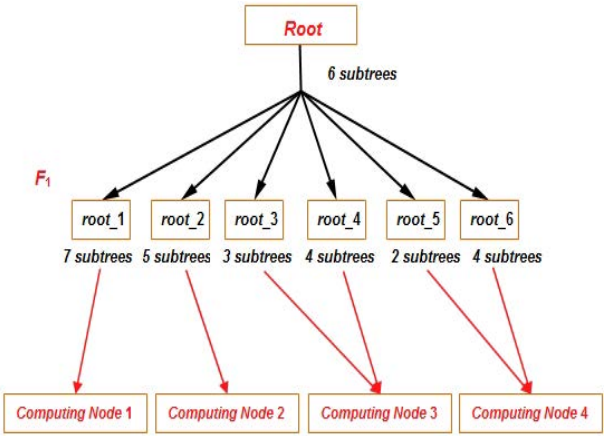

FIGURE 3. Partitioning T into four computing nodes.

packet classification method based on decision tree, large-scale data packets are distributed and classified in parallel.

The input of the algorithm is the nodes number n and the original k-dimensional classification rules R, after the rule preprocessing and decision tree constructing in the above step 1 and step 2, several $(k-1)$-level classification decision subtrees are output. Step 3 shows how to distribute these subtrees to each computing node as evenly as possible. Finally, step 4 describes the process of distributing and parallel classifying massive packets according to the distributed subtrees.

STEP 1: Rule Preprocessing

For the input classification rules R, the k-dimensional rules are mapped to the k-dimensional matrix space in reverse order by using the rule mapping method, and a series of independent cell spaces are formed after mapping. Generally speaking, classification rules can be expressed in the form of interval such 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 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, decision represents the decision (*accept* or *discard*) of rules, and k is the dimension. M_k can describe a k-dimensional matrix space. The coordinates of each dimension are expressed by F_i, and the corresponding coordinate interval is $[0, D(F_i)]$, $1 \leq i \leq k$. Fig. 4 defines a two-dimensional matrix space, and the coordinate intervals of both dimensions are [0, 9]. According to the idea of rule mapping based on FDM design model, any rule with the form of $\langle F_1 \in D(F_1) \wedge F_2 \in D(F_2) \wedge \ldots \wedge F_k \in D(F_k) \rangle \rightarrow \langle decision \rangle$ can be mapped to k-dimensional matrix space M_k. In the mapping process, we use the cell space $\boldsymbol{cs}$ (corresponding to a k-dimensional rectangle in the k-dimensional matrix space) to represent the region that is finally decided to *accept*: $[(l_1, l_2, \ldots, l_k)(d_1, d_2, \ldots, d_k)]$, where l_i and d_i refer to the minimum boundary value and range of the region in each dimension respectively. According to reference [26], if the number of rules is n, the time complexity of rule preprocessing (corresponding to the rule mapping method in FDM design model) is $O(kn)$, where k refers to the rule dimension.

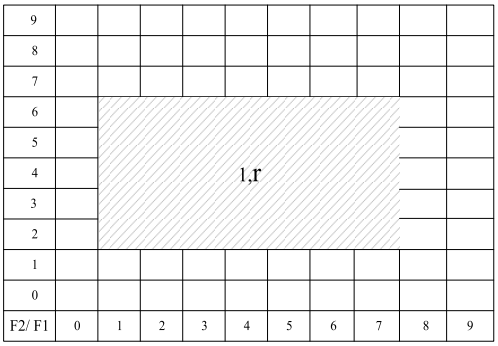

FIGURE 4. An intuitive mapping example.

r_1:$F_1 \in [0,4] \wedge F_2 \in [4,6] \rightarrow discard$,
r_2:$F_1 \in [5,7] \wedge F_2 \in [3,4] \rightarrow discard$,
r_3:$F_1 \in [2,9] \wedge F_2 \in [7,8] \rightarrow accept$,
r_4:$F_1 \in [5,7] \wedge F_2 \in [3,9] \rightarrow accept$,
r_5:$F_1 \in [4,7] \wedge F_2 \in [0,2] \rightarrow accept$,
r_6:$F_1 \in [0,9] \wedge F_2 \in [0,1] \rightarrow discard$,
r_7:$F_1 \in [0,4] \wedge F_2 \in [9,9] \rightarrow discard$,
r_8:$F_1 \in [0,1] \wedge F_2 \in [0,9] \rightarrow discard$,
r_9:$F_1 \in [0,4] \wedge F_2 \in [2,9] \rightarrow accept$.

FIGURE 5. An intuitive example of rules direct grouping.

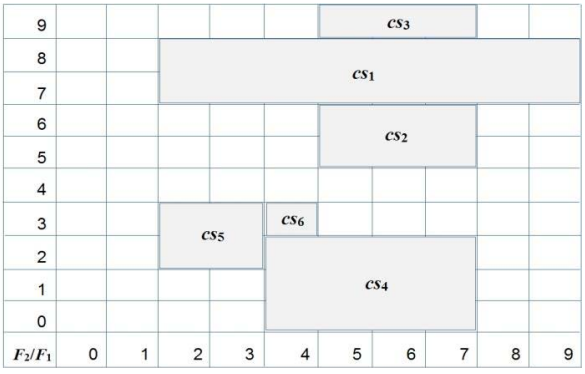

FIGURE 6. An example of rules mapping.

Fig. 4 shows the result of mapping the rule "r: $F_1 \in [1, 7] \wedge F_2 \in [2, 6] \rightarrow accept$." to the two-dimensional matrix space M_2, at this time, M_2 contains a cell space, expressed as [(1, 2) (7, 5)].

Suppose the original classification strategy contains nine rules, as shown in Fig. 5:

r_1: $F_1 \in [0, 4] \wedge F_2 \in [4, 6] \rightarrow discard$,
r_2: $F_1 \in [5, 7] \wedge F_2 \in [3, 4] \rightarrow discard$,
r_3: $F_1 \in [2, 9] \wedge F_2 \in [7, 8] \rightarrow accept$,
r_4: $F_1 \in [5, 7] \wedge F_2 \in [3, 9] \rightarrow accept$,
r_5: $F_1 \in [4, 7] \wedge F_2 \in [0, 2] \rightarrow accept$,
r_6: $F_1 \in [0, 9] \wedge F_2 \in [0, 1] \rightarrow discard$,
r_7: $F_1 \in [0, 4] \wedge F_2 \in [9, 9] \rightarrow discard$,
r_8: $F_1 \in [0, 1] \wedge F_2 \in [0, 9] \rightarrow discard$,
r_9: $F_1 \in [0, 4] \wedge F_2 \in [2, 9] \rightarrow accept$.

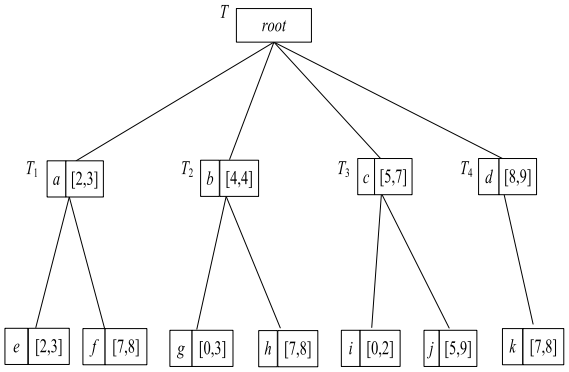

FIGURE 7. An example of decision tree construction.

Taking the original classification strategy shown in Fig. 5 as the input, the final mapping result can be obtained through the mapping operation, as shown in Fig. 6. At this time, the six cell spaces are:

$cs1$: [(2, 7) (8, 2)], $cs2$: [(5, 5) (3, 2)], $cs3$: [(5, 9) (3, 1)],
$cs4$: [(4, 0) (4, 3)], $cs5$: [(2, 2) (2, 2)], $cs6$: [(4, 3) (1, 1)].

STEP 2: Constructing Classification Decision Tree

The purpose of this step is to construct a classification decision tree according to the cell space obtained by the rule preprocessing process in Step 1 (as shown in Fig. 7).

Generally speaking, according to the definition of coordinate projection interval $\boldsymbol{P}(u, v, F_i)$ of cell spaces $\boldsymbol{u}$ and $\boldsymbol{v}$ on a given dimension F_i, the spatial relationship $\boldsymbol{R}(u, v, F_i)$ of any two cell spaces $\boldsymbol{u}$ and $\boldsymbol{v}$ must satisfy one of the six relationships {*crossed*, *covered*, *included*, *disjunctive*, *adjacent* and *equivalent*} [24].

Definition 1: 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 $P(u, v, F_i) = [l_i^{(u)}, l_i^{(v)}], [l_i^{(v)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(u)} + d_i^{(u)}, l_i^{(v)} + d_i^{(v)}]$;
(2) $R(u, v, F_i) =$ '***covered***,'
then $P(u, v, F_i) = [l_i^{(u)}, l_i^{(v)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}], [l_i^{(v)} + d_i^{(v)}, l_i^{(u)} + d_i^{(u)}]$;
(3) $R(u, v, F_i) =$ '***included***,'
then $P(u, v, F_i) = [l_i^{(v)}, l_i^{(u)}], [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(u)} + d_i^{(u)}, l_i^{(v)} + d_i^{(v)}]$;
(4) $R(u, v, F_i) =$ '***disjunctive***,'
then $P(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}]$;
(5) $R(u, v, F_i) =$ '***adjacent***,'
then $P(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}], [l_i^{(v)}, l_i^{(v)} + d_i^{(v)}]$;
(6) $R(u, v, F_i) =$ '***equivalent***,'
then $P(u, v, F_i) = [l_i^{(u)}, l_i^{(u)} + d_i^{(u)}]$.

Taking the two-dimensional case as an example, the spatial relationship and coordinate projection interval of two cell spaces $\boldsymbol{u}$ and $\boldsymbol{v}$ on dimension F_1 are shown in Fig. 8. Based on Definition 1, the coordinate projection interval of n cell spaces on dimension F_1 can be recorded

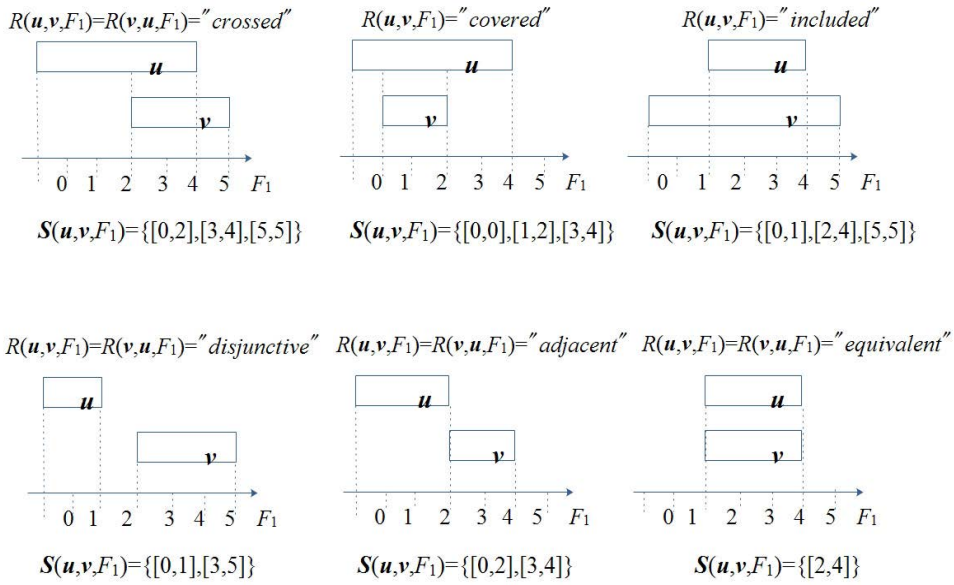

FIGURE 8. The spatial relations and the coordinate projection intervals of two cell spaces u and v in the dimension F_1.

as $\boldsymbol{P}(cs_1, \ldots, cs_N, F_1)$. Next, we give the process steps of constructing classification decision tree $\boldsymbol{T}$, as shown in Algorithm 1.

Algorithm 1 Constructing Classification Decision Tree

Input: n cell spaces $\boldsymbol{us}_1 \sim \boldsymbol{us}_n$.
Output: the classification decision tree T.
Begin
1. ***while*** $(t \leq k)$ {
2. $T \leftarrow \boldsymbol{P}(cs_1, \ldots, cs_n, F_1)$;
//add $P(cs_1, \ldots, cs_n, F_1)$ as the child nodes of root to T in turn to form a subtree
3. $t++$;
4. find all the cell spaces associated with the subspace corresponding to the root node of $T_i (i := 1 \text{ to } n)$ (recorded as $\{\boldsymbol{cs}\}$);
5. seek the coordinate projection interval $\boldsymbol{P}(\{\boldsymbol{cs}\}, F_t)$ of $\{\boldsymbol{cs}\}$ on dimension F_t;
6. $T_i \leftarrow \boldsymbol{P}(\{\boldsymbol{cs}\}, F_t)$; //add $P(\{cs\}, F_t)$ as a child node to T
7. }
8. return T;
End

Taking Fig. 7 as an example, in the initial case, $\boldsymbol{T}$ is a one-level decision tree, including only the root node *root*. Six cell spaces ($\boldsymbol{cs}_1 \sim \boldsymbol{cs}_6$) in the two-dimensional matrix space form four coordinate projection intervals {[2, 3], [4, 4], [5, 7], [8, 9]} on the F_1 dimension. The root node $\{a\}$ of the subtree T_1 corresponds to the interval [2, 3], and all the associated cell spaces $\boldsymbol{cs}_1$ and $\boldsymbol{cs}_5$ form two coordinate projection intervals [2, 3] and [7, 8] in the F_2 dimension. Two projection intervals are respectively added to the subtree T_1 as the child nodes $\{e, f\}$ of the root node $\{a\}$. Similarly, the projection intervals [0, 3] and [7, 8] are added to the subtree T_2 as child nodes $\{g, h\}$. Projection intervals [0, 2] and [5, 9] are added to the subtree T_3 as child nodes $\{i, j\}$; The projection interval [7, 8] is added to the subtree T_4 as a child node $\{k\}$, and finally the decision tree T is formed, as shown in Fig. 7.

It can be seen from Fig. 7 that the decision tree T is composed of four subtrees, and the corresponding interval of each subtree root presents an increasing relationship. Similarly, the corresponding interval of each leaf node in the subtree are also increases. The analysis shows that the branches formed from the root node of T to each leaf node are independent of each other, so these branches corresponding to the rules can be directly divided and grouped. It should be pointed out that in order to facilitate the subsequent data packets to be divided into each group, we regard each subtree as a whole and do not cut the subtree.

STEP 3: Dividing the Decision Subtree into Computing Nodes

Suppose there are n parallel computing nodes, this step aims to divide all subtrees obtained in step 2 into these n nodes approximate evenly according to the number of branches. Assuming that the decision tree T has m subtrees, the number of branch in each subtree is $l_1, l_2, \ldots, l_m$ respectively, let $L = l_1 + l_2 + \ldots + l_m$, then the division principle is to make the number of branches divided by each node as close to L/n as possible.

In the sequence $\{l_1, l_2, \ldots, l_m\}$, the first $t+1$ values are successively taken from l_1; if

$$|\sum_{i=1}^{t}(l_i) - \frac{L}{n}| \leq |\sum_{i=1}^{t+1}(l_i) - \frac{L}{n}| \quad (1)$$

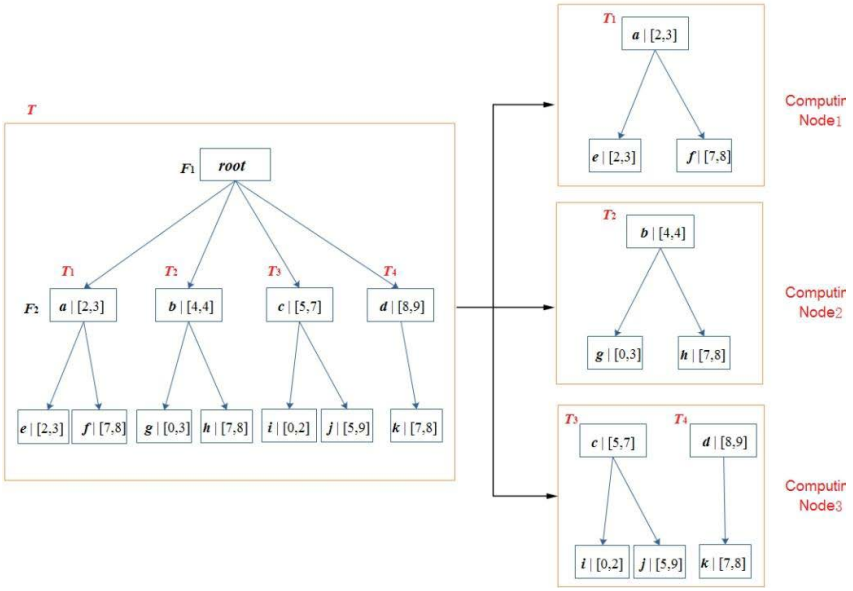

FIGURE 9. An example of decision subtree grouping.

the first t subtrees corresponding to $l_1, \ldots, l_t$ are taken and divided into the first computing node; Continue starting from l_{t+1}, take several subtrees and divide them into the second computing node according to the same method as above. Finally, divide all the remaining subtrees into the last computing node. The specific subtree grouping process is described in Algorithm 2.

Intuitively, Algorithm 2 divides all subtrees of decision tree T into groups according to the number of pre-deployed computing nodes and the number of branches in each subtree. The principle of division is that the number of branches in each group is as identical as possible. When classifying data packets, first determine whether the value of the data packet in the corresponding dimension matches the corresponding interval of the subtree root node. If so, send the data packet to the computing node corresponding to the subtree for further classification processing; if any interval cannot be matched, the packet decision is directly determined as *discard*.

As shown in Figure 9, the decision tree T has four subtrees $T_1 \sim T_4$, and the number of corresponding branches in each subtree is {2, 2, 2, 1} respectively, the total number of branches can be calculated as '7.' Assuming that the number of computing nodes is '3,' we divide the seven branches into three computing nodes as evenly as possible, then the number of branches in each node is approximately $7/3 \approx 2.33$.

The corresponding four interval values of each subtree root node are arranged as {[2, 3], [4, 4], [5, 7], [8, 9]} from small to large, and the corresponding branch number sequence is $\{l_1, l_2, l_3, l_4\} = \{2, 2, 2, 1\}$, starting from $l_1 = 2$. Because $|2 - 2.33| < |2 + 2 - 2.33|$, the first subtree is taken and divided into the first computing node; Continue to consider the second number in the sequence, similarly, divide [4, 4] into the second node. At this time, there is only one node left, all the remaining two intervals {[5, 7], [8, 9]} are divided into the third node.

Algorithm 2 Grouping Decision Subtree

Input: the number of branches of m subtrees in decision tree **T**, denoted as $l_1, l_2, \ldots, l_m$.
Output: the grouping result of subtrees (n groups, denoted as Group(1) ... Group(n)).
Begin
1. $s = 1, k = 1$;
2. $L = l_1 + l_2 + \ldots + l_m$;
3. ***for*** $(t := 0$ *to* $m - k)$ ***do***{
4. ***if*** $(|\sum_{i=k}^{k+t}(l_i) - L/n| \leq |\sum_{i=k}^{k+t+1}(l_i) - L/n|)$ ***do***{
5. $Group(s) \leftarrow Merge(l_k, \ldots, l_{k+t})$;
   //take t subtrees and divide them into computing nodes
6. $s++$;
7. $k += (t + 1)$;
8. ***if*** (s >= n-1) ***do***{
   //divide all remaining subtrees into the last computing node
9. $Group(n) \leftarrow Merge(l_k, \ldots, l_m)$;
10. *break*;
11. } ***else*** *continue*;
12. } ***else*** *continue*;
13.}
14. *return* $Group(1) \sim Group(n)$.
End

STEP 4: Classifying Packets in Parallel

Different from the traditional packet classification method based on decision tree, this method first divides the packets to be classified according to the corresponding interval value of

each subtree root node, which is approximately equivalent to reducing the packet classification scale to $1/n$ of the original one (n is the number of deployed computing nodes); After the data packets are divided into parallel computing nodes, they are classified according to the subtree (decision tree) divided in advance. According to the division process in step 3, the scale of the classification decision tree in each node is also approximately reduced to $1/n$ of T.

As far as data packet classification based on decision tree is concerned, its essence is a query operation. For the decision tree or any of its subtrees, the interval coordinate values corresponding to the child nodes of the root node are strictly increasing, so the binary search method can be directly applied in the process of classification. As shown in Fig. 10, the root node of the decision tree T has four child nodes, and the corresponding interval coordinate values are [2, 3], [4, 4], [5, 7] and [8, 9] respectively, satisfying the strict increasing relationship.

Consider a k-tuple packet P:(e_1, e_2, ..., e_k). When classifying it, start from the root node of the decision tree, and first bisearch on all the child nodes of the root node to judge whether the first metadata $\boldsymbol{e}_1$ of the packet is included in the corresponding interval of a certain node. If any interval can not be matched, it can be determined directly that packet P cannot match any rule, and the packet decision is recorded as *discard*. Otherwise, continue searching on the subtree within the node. If each tuple $e_i(i := 1 \text{ to } k)$ of packet P matches the corresponding interval of each layer node of a subtree branch, it can be determined that the packet P matches the subtree, and the decision of P is *accept*.

The classification process is shown in Figure 10. Firstly, the method groups the data packets and divides them into corresponding computing nodes according to the value of the first dimension of the data packets. Then, at each computing node, the packet decision is determined based on the decision tree method. Taking the packet to be classified $E = \{p_1, p_2, p_3, p_4, p_5, p_6\} = \{(2, 5), (1, 8), (4, 7), (6, 3), (2, 8), (8, 8)\}$ as an example, first, determine the value $\boldsymbol{e}_1$ of the first dimension of the packet, and compared it with the interval value of the child node of the root. It can be seen that p_2:(1, 8) cannot match any interval value, which means that p_2 cannot match any rule in the decision tree, so it can be directly determined that the packet p_2 decision is *discard*. In addition, p_1 and p_5 can be divided into the first node; p_3 is divided into the second node; p_4 and p_6 are divided into the third node.

Continue the packet classification based on decision tree at each node: in the subtree corresponding to the first node, because the second dimension of p_1:(2, 5) is '5,' it cannot match any interval [2, 3] or [7, 8] corresponding to the two child nodes, so it can be determined that the classification decision of p_1 is *discard*, While the second dimension of p_5:(2, 8) is '8,' which matches the interval [7, 8], so the decision of p_5 can be determined as accept. Similarly, the other packets can be classified at the second and third nodes, the decision of p_3:(4, 7), p_4:(6, 3) and p_6:(8, 8) is *accept*, *discard* and *accept* respectively. Here, it should be noted that, because the interval value of each child node is strictly increasing, the bisearch method can be used in packet matching, and the specific process of packet classification is described in Algorithm 3.

Algorithm 3 Classifying Packets in Parallel

Input: the packet P:(e_1, e_2, ..., e_k), the decision tree T.
Output: the packet decision ('*accept*' or '*discard*').
Begin
/* bisearch on all the child nodes of the root to judge whether e_1 is included in the interval of the s^{th} subtree *Child*(*root, s*). If it is true, send P to the computing node where the sub-tree is located; otherwise, determine that the decision of P is *discard.* */
1. ***if*** (*Bisearch* (*root*, e_1) == *true*) ***do***{
2. send P to the computing node where the sub-tree *Child*(*root, s*) located;
3. } ***else*** $P \rightarrow$ *discard*;
4. *break;*
/* at the computing node where the packet P located, start from the second dimension and search for e_i on the sub-tree *Child*(*root, s*) (denoted as *root_s*). if e_i is included in the interval of the t^{th} sub-tree of *root_s*, continue to search for e_{i+1} on the sub-tree *Child*(*root_s, t*). */
5. ***for*** ($i := 2$ to k) ***do***{
6. ***if*** (*Bisearch* (*root_s*, e_i) == *true*) ***do***{
7. *root* = *Child*(*root_s, t*);
8. *continue*;
9. ***if*** ($i \geq k$)
10. $P \rightarrow$ *accept.*
11. } ***else***
12. $P \rightarrow$ *discard*;
13. *break;*
14.}
End

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The proposed algorithms were implemented in Java JDK 1.7, our experiments were carried out on a desktop PC running Windows 10 with 16G memory and Intel(R) Core(TM) i7-10510U Processor of 1.80 GHz. In order to realize the distributed processing of packet classification,we built a Hadoop platform with a master and eight slave nodes. The implementation of the algorithm includes the following steps: Firstly, the rules are mapped by FDM [26] to generate independent cell spaces in multi-dimensional space; Then, the classification decision tree is constructed based on the spatial relationship of cell spaces, and the decision tree is divided and deployed to each computing node as evenly as possible according to the number of nodes; When classifying data packets, we first group the data packets according to the root node interval coordinate values of the subtree of each node, and then distribute them to each computing node for classification. The classification results of all data packets are directly output in each computing node.

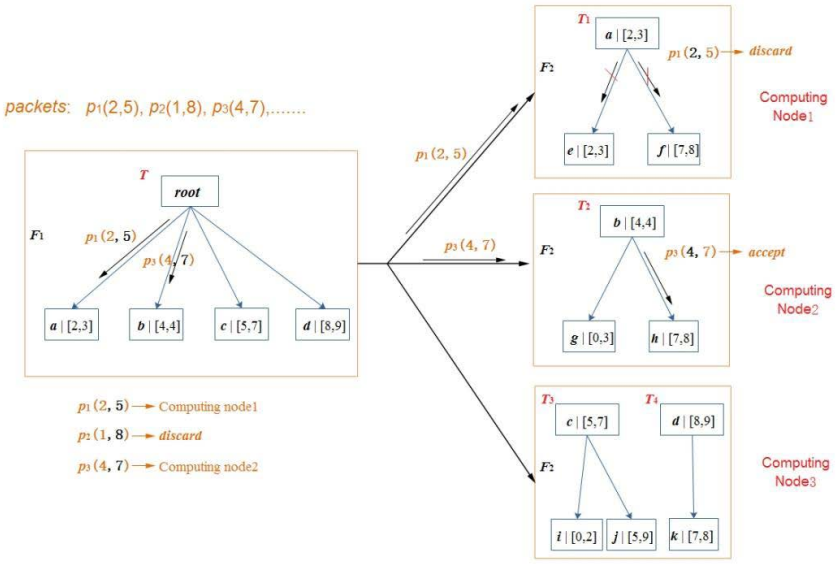

FIGURE 10. An example of parallel packet classification.

It should be pointed out that the FDM mapping of rules and the construction of decision tree can be carried out offline in advance. After each classification decision subtree is deployed to the computing node, the data packets can be classified in a distributed manner. Therefore, this method can be used for the parallel classification of large-scale data packets.

To verify the efficiency of classification method MpFPC, we choose three classification rules of different number (100, 1000 and 10000 rules respectively) and four data sets of different sizes (10KB, 1MB, 100MB and 200MB respectively) to test the time that required to classify data packets with MpFPC and Uscuts [24]. The test results are shown in Table 1, Table 2 and Table 3 respectively.

Take Table 1 as an example, when the rule number is 100 and the packet size is 100MB, the classification speed and average value of each node are mapped to Figure 11. It can be seen that the classification time on each node fluctuates slightly up and down near the average value, which also shows that the algorithm has good performance in dividing the decision tree as evenly as possible according to the number of nodes, this performance is important to improve the classification efficiency of the algorithm.

Next, taking the number of classification rules as the abscissa and the packet classification speed as the ordinate, the MpFPC and Uscuts methods are used to classify the packet of four different sizes when the rules number is 100 and 10000 respectively, and the classification time is mapped in Fig. 12 and Fig. 13. It can be seen that with the increase of packet size, the classification speed advantage of MpFPC over Uscuts becomes more obvious. For example, when the packet is 100MB, the classification time of MpFPC algorithm is about 1/3 of that of Uscuts algorithm.

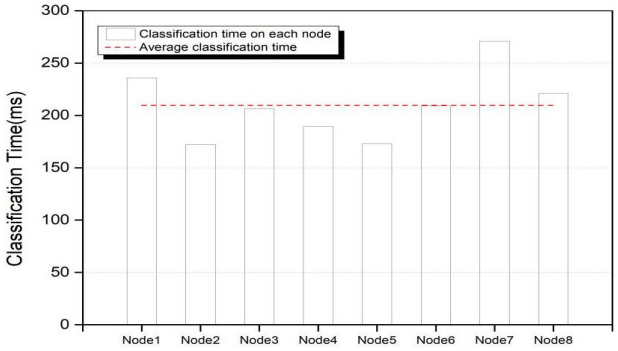

FIGURE 11. Classification time and average value at each node.

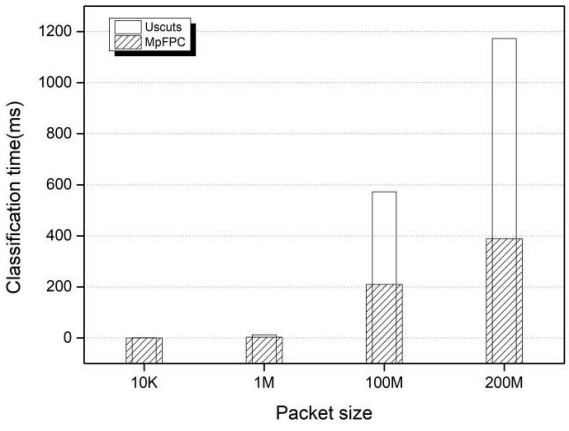

FIGURE 12. Classification time when rule size is 100.

In order to further test the relationship between MpFPC method performance and the number of nodes, we compare the classification speed when the rule number is 1000 and

TABLE 1. Packets classification time when rule size is 100.

Packet Size	*Uscuts*	*MpFPC*								
		Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8	average
10KB	0.50	0.17	0.14	0.09	0.12	0.16	0.11	0.13	0.16	0.14
1MB	12.11	4.29	2.77	3.66	2.90	3.155	3.07	4.93	3.57	3.54
100MB	573.10	235.91	172.43	206.34	189.39	172.87	209.26	270.97	221.00	209.77
200MB	1173.27	481.39	366.96	371.22	321.68	315.79	359.72	502.41	389.99	388.65

TABLE 2. Packets classification time when rule size is 1000.

Packet Size	*Uscuts*	*MpFPC*								
		Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8	average
10KB	0.81	0.39	0.20	0.32	0.22	0.45	0.44	0.35	0.45	0.35
1MB	26.35	8.40	4.84	5.59	4.77	5.05	7.55	8.77	9.69	6.83
100MB	1719.04	628.05	578.49	667.13	525.32	495.51	724.63	646.02	675.76	617.62
200MB	3125.52	1090.02	990.19	1174.86	968.10	926.55	1337.56	1204.03	1266.49	1119.73

TABLE 3. Packets classification time when rule size is 10000.

Packet Size	*Uscuts*	*MpFPC*								
		Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8	average
10KB	2.55	0.74	0.63	0.56	0.52	0.52	0.56	0.66	0.53	0.59
1MB	51.03	13.63	12.81	12.87	11.82	11.08	12.74	14.11	15.99	13.13
100MB	2927.63	701.76	781.04	748.06	755.43	747.28	859.06	843.62	736.0	771.53
200MB	5324.89	1402.56	1669.11	1449.82	1446.12	1433.12	1788.94	1861.99	1940.54	1624.03

TABLE 4. Packets classification time (ms) with 3 nodes.

Packet Size	*Uscuts*	*MpFPC*			
		node1	node2	node3	average
100MB	1719.04	905.15	618.49	1322.57	948.74
200MB	3125.52	1776.15	1384.46	2128.34	1762.98

TABLE 5. Packets classification time (ms) with 5 nodes.

Packet Size	*Uscuts*	*MpFPC*					
		node1	node2	node3	node4	node5	average
100MB	1719.04	905.15	618.49	1322.57	803.73	845.76	720.75
200MB	3125.52	1776.15	1384.46	2128.34	1580.87	1774.56	1408.97

the packet size is 100MB and 200MB in the case of three computing nodes, five computing nodes and eight computing nodes respectively. The classification speeds are shown in Table 4, Table 5 and Table 6.

For the sake of intuition, we comprehensively compare the classification speeds of Uscuts method and MpFPC method in three node cases and map them to Fig. 14. As shown in the figure, compared with Uscuts, the classification speed of MpFPC is significantly improved, and as the number of nodes increases, the classification speed of MpFPC method also increases. For example, when the packet to be classified is 200MB, the time required to classify using Uscuts method

TABLE 6. Packets classification time (ms) with 8 nodes.

Packet Size	*Uscuts*	*MpFPC*								
		node1	node2	node3	node4	node5	node6	node7	node8	average
100MB	1719.04	905.15	618.49	1322.57	525.32	495.51	724.63	646.02	675.76	617.62
200MB	3125.52	1776.15	1384.46	2128.34	968.10	926.55	1337.56	1204.03	1266.49	1119.73

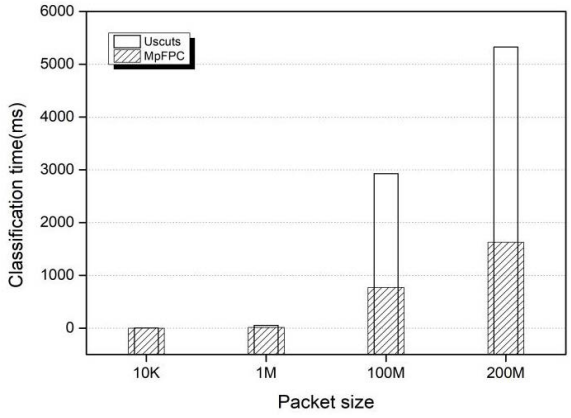

FIGURE 13. Classification time when rule size is 10000.

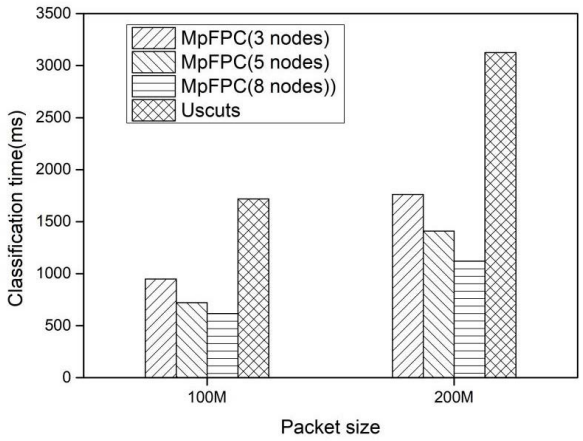

FIGURE 14. Classification time of Uscuts method and MpFPC method in different node cases.

is about 3000*ms*; while the average time required to classify using MpFPC method are 1762*ms* at three nodes, 1408*ms* at five nodes and only about 1000*ms* at eight nodes. Therefore, by adding the number and improving the computing performance of nodes, it is expected to further improve the packet classification speed and meet the wire-speed requirement.

V. CONCLUSION

With the development of network applications, higher requirements are put forward for the speed of network packet classification. In this paper, the traditional single thread packet classification framework is improved, and the parallel method of distributed computing is used to classify data packets based on decision tree. The algorithm proposed in this paper has two innovations: firstly, the decision tree is divided into several sub-decision trees for distributed packet classification, and the mapping method FDM based on multi-dimensional matrix is adopted before constructing the decision tree to remove rule conflict and redundancy, so as to avoid the rule replication problem of the traditional decision tree method; Secondly, during packet classification, each packet is compared according to the corresponding interval value of the root node of the decision subtree. After finding the matching interval, it is distributed to the corresponding decision subtree for classification, which realizes the distributed processing of packets and further improves the efficiency of packet classification. The experimental results also show that the classification speed of MpFPC is faster than that of Uscuts, and the advantage of classification speed is more obvious with the increase of packet size. In addition, the experimental results also show that the classification speed will increase with the addition of the nodes number, which provides a new possible way to meet the classification wire-speed requirement in the new generation network.

The design method discussed in this paper is not only limited to packet classification, but also can be extended to other applications, such as OpenFlow switch, Firewall, Security gateway, and so on. With the continuous updating of network applications, in our next work, we will provide new methods according to the OpenFlow requirements of high speed and fast update classification.

REFERENCES

[1] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *Proc. 28th Conf. Comput. Commun. (INFOCOM)*, Apr. 2009, pp. 648–656.
[2] S. Greenberg, T. Sheps, D. A. Leon, and Y. Ben-Shimol, "Packet classification using GPU and one-level entropy-based hashing," *IEEE Access*, vol. 8, pp. 80610–80623, 2020.
[3] (2017). *Force I.P.T. 400 Gigabits Per 12Second Ethernet Standard*. [Online]. Available: http://www.ieee802.org/3/bs
[4] 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.
[5] A. X. Liu, C. R. Meiners, and E. Torng, "Packet classification using binary content addressable memory," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1295–1307, Jun. 2016.
[6] C. Li *et al.*, "Memory optimization for bit-vector-based packet classification on FPGA," *Electron*, vol. 8, no. 10, pp. 1–16, 2019.
[7] P. He, G. Xie, and L. Mathy, "Meta-algorithms for software-based packet classification," in *Proc. IEEE ICNP*, Oct. 2014, pp. 308–319.
[8] W. Li and X. Li, "HybridCuts: A scheme combining decomposition and cutting for packet classification," in *Proc. IEEE 21st Annu. Symp. High-Performance Interconnects*, Aug. 2013, pp. 41–48.

[9] J. Daly *et al.*, "TupleMerge: Fast software packet processing for online packet classification," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1–15, Aug. 2019.
[10] 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.
[11] C. Zhang and G. Xie, "MultilayerTuple: A general, scalable and high-performance packet classification algorithm for software defined network system," in *Proc. IEEE IFIP Netw.*, Espoo Helsinki, Finland, Jun. 2021, pp. 1–9.
[12] 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-Performance Interconnects*, Santa Clara, CA, USA, Aug. 2012, pp. 1–8.
[13] 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. (INFOCOM)*, Honolulu, HI, USA, Apr. 2018, pp. 2645–2653.
[14] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203–214, Oct. 1998.
[15] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 147–160, Oct. 1999.
[16] U. Trivedi and M. L. Jangir, "An optimized RFC algorithm with incremental update," in *Proc. ICACCI*, New Delhi, India, Sep. 2014, pp. 120–127.
[17] 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.
[18] 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.
[19] P. Gupta and N. Mckeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. Hot Interconnects*, 1999, pp. 34–41.
[20] S. Singh *et al.*, "Packet classification using multidimensional cutting," *Comput. Commun. Rev.*, vol. 33, no. 4, pp. 213–224, 2003.
[21] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: Optimizing packet classification for memory and throughput," in *Proc. ACM SIGCOMM Conf. SIGCOMM (SIGCOMM)*, New Delhi, India, 2010, pp. 207–218.
[22] Y. F. Li, J. Wang, X. Chen, and J. Wu, "SplitTrie: A fast update packet classification algorithm with trie splitting," *Electronics*, vol. 11, no. 2, pp. 1–13, 2022.
[23] 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.
[24] Y. Cheng *et al.*, "A fast firewall packet classification algorithm using unit space partitions," *Adv. Eng. Sci.*, vol. 50, no. 4, pp. 144–152, 2018.
[25] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 9, pp. 1237–1251, Sep. 2008.
[26] 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.

YUZHU CHENG received the B.S. degree from the Hunan University of Science and Technology, in 2002, the M.S. 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. He joined Changsha Social Work College, in 2006, 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.

QIUYING SHI received the B.S. degree from Hunan Normal University and the M.S. degree in computer science and technology from Central South University. Her research interests include cyber security and pattern recognition.