



PCMIgr: a fast packet classification method based on information gain ratio

Yuzhu Cheng¹ · Qiuying Shi^{1,2}

Accepted: 10 November 2022 / Published online: 28 November 2022
© The Author(s) 2022

Abstract

To solve the problem of ambiguous attribute selection in existing decision tree classification algorithms, a decision tree construction method based on information entropy, PCMIgr, is proposed. PCMIgr is a heuristic method based on greedy strategy. At each decision tree node, when it is necessary to select classification attributes for division, the attribute with the highest information gain ratio is selected. The main innovation of this method is that the attribute selection in the traditional classification method based on decision tree is optimized, and the classification efficiency of the constructed decision tree is improved compared with that before optimization. At the same time, the decision tree ensures that each leaf node is only associated with one rule, which avoids the common problem of "rule replication" in the process of traditional decision tree construction, and effectively saves memory and calculation time. The experimental results show that the application of this method to the construction of classification decision tree can further improve the efficiency of packet classification method based on decision tree, and can be applied to high-speed real-time packet classification.

Keywords Packet classification · Decision tree · Information gain ratio

1 Introduction

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

✉ Yuzhu Cheng
vogue21ct@qq.com

¹ School of Software, Changsha Social Work College, Changsha 410004, China

² School of Computer Science and Engineering, Central South University, Changsha 410083, China

explosively, which puts forward higher requirements for packet classification and processing capacity of network devices.

Packet classification is the core technology to Internet devices and network services implementation. According to a series of information carried by the specified packet (such as source address, destination address, source port, destination port and protocol, etc.), it searches for the operation or task to be performed by the packet in a set of rules based on the principle of highest priority matching [1]. The rules in the classifier are generally expressed in the form of prefix or address range, and the two forms are semantically equivalent. In order to represent the structure of decision tree more intuitively, the classification rules are expressed in the form of address range in this paper, as shown in Fig. 1.

According to the basic principle of packet classification technology, it is well known that the most intuitive packet classification method is sequence matching. However, the time complexity of sequence matching algorithm is linearly related to the number of rules. With the increase in the scale and dimension of classification rules, sequence matching takes more time, which leads to the decrease in packet classification efficiency and becomes the bottleneck of network performance. Fortunately, the rules in the actual classifier have some inherent characteristics, which can be used to reduce the complexity of packet classification. Under this background, researchers have proposed many excellent packet classification algorithms, including the method based on dimension decomposition, the method based on tuple space search and the method based on decision tree, etc. [2].

Taking HiCuts [3], a method based on decision tree, as an example. It recursively divides the rule search space into multiple subspaces with equal size by using the equal-scale cutting method, until the number of rules in each subspace is less than the predefined threshold τ (τ is equal to 6 in the following example). According to HiCuts method, it first maps all rules to a two-dimensional space according to the address range. As shown in Fig. 2a, the area r_1 marked with backslashes is one rule mapping area, while areas with multiple mappings are marked with cross lines, such as areas $r_1 \hat{\wedge} r_5$, $r_4 \hat{\wedge} r_6 \hat{\wedge} r_9$, et al.

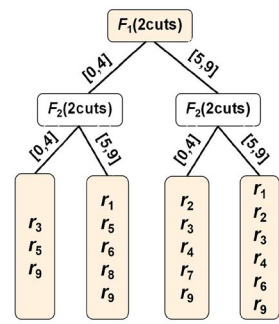
After all the rules are mapped to the two-dimensional space, the decision tree can be constructed. First, divide all rule spaces into two subspaces on the F_1 dimension equally. In this example, the address range of each dimension is [0,9], so the address

Fig. 1 An example of two-dimensional classification rules

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

9	r_1	$r_1 \wedge r_5$	r_1	$r_1 \wedge r_4$	r_1	$r_1 \wedge r_2$	r_1			
8					r_4		r_2			
7										
6	r_8	$r_5 \wedge r_6$	r_6							
5				$r_6 \wedge r_5$	$r_4 \wedge r_6 \wedge r_5$	$r_6 \wedge r_5$	$r_2 \wedge r_4 \wedge r_5$			
4		r_5		r_5	$r_1 \wedge r_5$	$r_4 \wedge r_1 \wedge r_5$	$r_1 \wedge r_5$			
3										
2	r_3	$r_3 \wedge r_5$	r_3	$r_3 \wedge r_1$	$r_3 \wedge r_4 \wedge r_1$	$r_3 \wedge r_1$	r_3			
1										
0					r_4					
$F_2 \setminus F_1$	0	1	2	3	4	5	6	7	8	9

(a) Mapping rules to M_2



(b) Constructing decision tree

Fig. 2 An example of HiCuts method

ranges of these two subspaces are $[0,4]$ and $[5,9]$, respectively. Then a first-level decision tree with two leaf nodes can be constructed from the root node. However, according to the idea of HiCuts method, it is required that the number of rules associated with each leaf node should not exceed the specified threshold τ . Therefore, on the basis of the first-level decision tree division, it is necessary to continue to divide the subspace equally in the F_2 dimension, as shown in Fig. 2b, the subspace $F_1 \in [0, 4] \wedge F_2 \in [0, 4]$ is only associated with three rules r_3 , r_5 and r_9 , which meet the threshold requirements. Continue to divide the remaining leaf nodes according to the same process until each leaf node meets the requirement that the number of association rules is not greater than the threshold value of 6, and the decision tree construction process ends.

It can be seen that HiCuts method can effectively improve the efficiency of packet classification by preprocessing the original classification rules and transforming the classification operation from sequence matching to search based on decision tree. This method provides a good reference for accelerating packet classification, but with the increase in link bandwidth, packet classification speed gradually becomes the bottleneck of network performance, and the existing packet classification algorithms still have room to further improve the classification speed.

If the packet is classified according to the decision tree in Fig. 2b, the sequence matching needs to be continued after accessing the leaf node from the root node, which reduces the efficiency of packet classification to some extent. In addition, as shown in the cross line marked area in Fig. 2a, there are often a lot of mutual "entanglement" between rules. These entanglements form the rule replication in the decision tree [4]. As shown in Fig. 2b, rule r_3 is copied twice and rule r_9 is copied three times, thus increasing the storage space consumption. In addition, most decision trees do not deal with rule conflicts during the construction process. For example, the rules r_1 and r_4 match the region $F_1 \in [5, 5] \wedge F_2 \in [9, 9]$ concurrently, but their decisions are different, which may lead to the erroneous discarding of legitimate packets or the acceptance of malicious packets, thus bringing security vulnerabilities to the network.

To solve the above problems, we put forward an improved package classification method based on decision tree, Uscuts [4]. First, the original rules are mapped to a multi-dimensional matrix M_k (where M_k stands for a k -dimensional matrix) in reverse order, and the unit spaces with the same semantics as the original rules but independent of each other in space are obtained [5]. Figure 3 shows the result of mapping rule $r: "F_1 \in [1, 7] \wedge F_2 \in [2,6] \rightarrow accept."$ to a two-dimensional matrix M_2 . Here M_2 contains a unit space, represented as [(1,2) (7,5)]. Because it is a two-dimensional space, there are only two attributes: F_1 dimension and F_2 dimension.

Based on the mapping method described in reference [5], mapping the rules shown in Fig. 1 to a two-dimensional matrix in reverse order can obtain seven unit spaces named cs_1-cs_7 , which correspond to seven rectangles in the two-dimensional space as shown in Fig. 4.

This method divides the rule space corresponding to cs_1-cs_7 according to the attribute order of F_1 dimension first and then F_2 dimension, and constructs the classification decision tree as shown in Fig. 5. It can be seen that each branch of the decision tree is in one-to-one correspondence with the rule subspace, that is, each leaf node of the decision tree is just associated with a rule. Therefore, when a packet matches a leaf node, the Uscuts method can directly determine that the packet classification decision is "accept", which is different from the traditional packet classification method based on decision tree, which needs to perform sequence matching in the rule group associated with the leaf node. Therefore, the speed of packet classification is effectively improved.

However, further research shows that for the unit spaces shown in Fig. 4, if they are divided according to the attribute order of F_2 dimension first and then F_1 dimension, the construction of decision tree is shown in Fig. 6. Obviously, if the selection order of classification attributes is different, the constructed decision tree and classification efficiency will also be different. For example, when the decision tree in Fig. 5 is used for packet classification, we first perform packet matching on eight nodes of F_1 dimension. Because the range value of each node is strictly increasing,

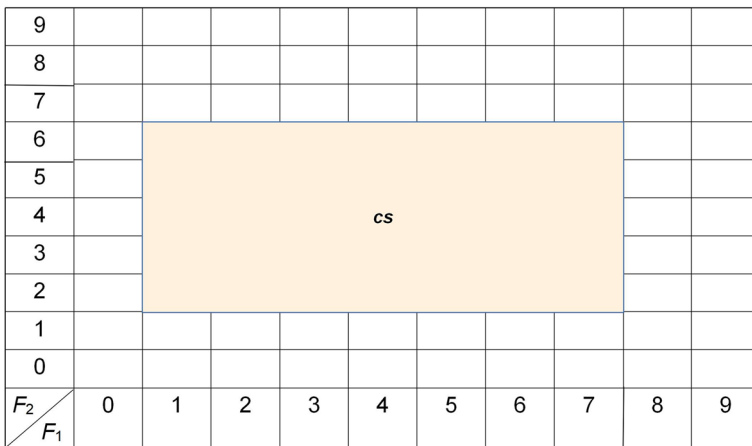


Fig. 3 An example of rule mapping, $r: F_1 \in [1,7] \wedge F_2 \in [2,6] \rightarrow accept$

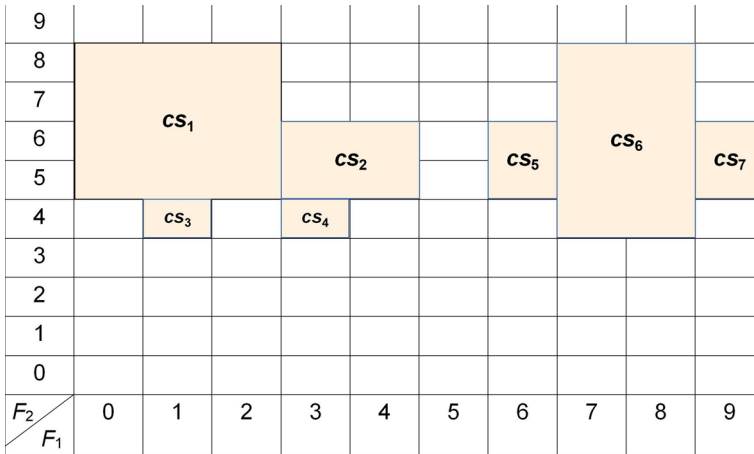


Fig. 4 Rules mapping forms unit spaces

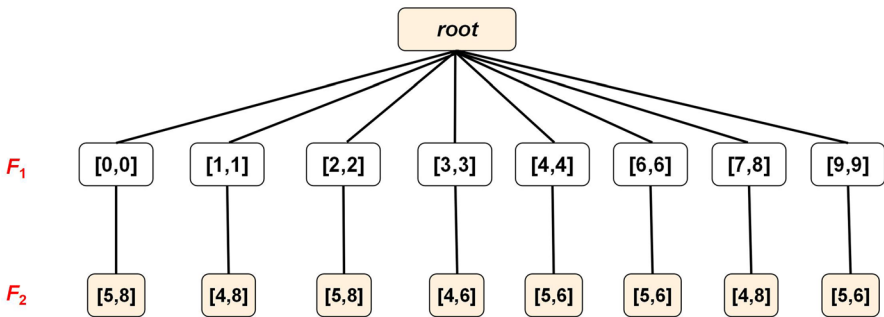


Fig. 5 The decision tree constructed according to Uscuts method

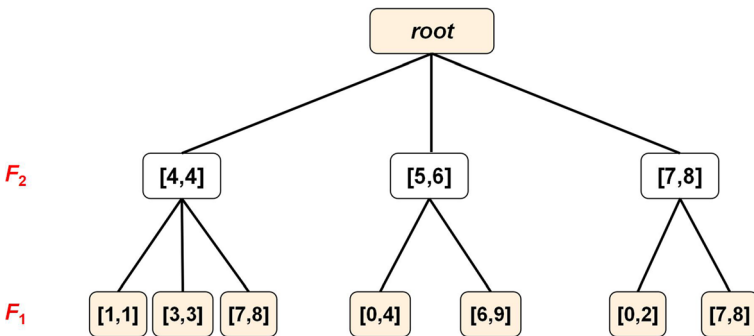


Fig. 6 The decision tree constructed according to attribute order selection

the Binary Search algorithm can be adopted, and the time complexity is $\log_2(8)=3$. However, when the decision tree shown in Fig. 6 is used for packet classification, the time complexity of matching on the first layer is $\log_2(3)\approx 1.58$. Combing the matching time on the second layer, it can be easily calculated that the efficiency of packet classification according to the decision tree shown in Fig. 6 is higher than that in Fig. 5.

Through the analysis of the above examples, it can be inferred that when constructing a decision tree according to the specified classification rules, the selection of attribute order will affect the efficiency of package classification. Therefore, the problem we are faced with is how to determine the order of attribute selection when constructing the decision tree, so as to make the package classification based on the decision tree more efficient.

Generally speaking, the most intuitive way to deal with this problem is to construct all decision trees according to different arrangement of attributes, and then carry out package classification and efficiency comparison. However, for k -dimensional classification rules, to determine the decision tree with the highest classification efficiency, it is necessary to construct k factorial (e.g., 5 factorial equals 120) decision trees, and then classify packets based on them, respectively, and compare the classification efficiency. This is a global optimized solution, but it is obviously too time-consuming and difficult to realize. Therefore, we designed a heuristic decision tree construction method, PCMIgr. Based on the idea of greedy strategy, this method selects the attribute with the highest "information gain ratio" at each decision tree node. According to Informatics theory, information entropy can be used to measure the uncertainty of data samples and categories with different characteristics. The greater the information entropy of the eigenvector, the greater the uncertainty, so we should give priority to dividing from the eigenvector.

In this paper, the information gain ratio is used to select attributes, which overcomes the disadvantages of selecting attributes with more values when using information gain [29]. The experimental results show that the decision tree based on information gain ratio has higher classification efficiency, and does not require additional memory. It is suitable for high-speed real-time packet classification.

The main innovation of this method is that the attribute selection in the traditional classification method based on decision tree is optimized, and the classification efficiency of the constructed decision tree is improved compared with that before optimization. At the same time, the decision tree ensures that each leaf node is associated with only one rule, which avoids the deficiency that the traditional decision tree needs to continue sequence matching after the packet matches the leaf node. More importantly, the decision tree avoids the common problem of "rule replication" in traditional decision tree construction, and effectively saves memory and calculation time.

The rest of this paper is organized as follows: the related work is introduced in Sect. 2; In the third section, the problem description is given, and the classification decision tree construction algorithm based on information gain ratio is elaborated in detail. The fourth section gives the classification results of PCMIgr, Uscuts, Hicuts and HyperSplit, and followed by the analysis of experimental results; Finally, the conclusion is drawn in Sect. 5.

2 Related work

Packet classification can be regarded as the problem of "point location in multidimensional space" in computational geometry: 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. Assuming that these areas do not intersect, and the dimension k is greater than 3, it has been proved that for n non-overlapping hyper-rectangles in k -dimensional space, the best bounds is either $O(\log_2 n)$ time with $O(n^k)$ space, or $O((\log_2 n)^{k-1})$ time with $O(n)$ space [6]. In addition, in packet classification, hyper-rectangles may overlap, which makes packet classification more difficult than point location, it may require too much memory or take a long time.

For the rules in the classifier, with the increase in rule dimensions, the performance of packet classification algorithm will drop sharply. Moreover, with the rapid expansion of Internet services, the scale of classification rules is also increasing, which brings severe challenges to packet classification. Therefore, the massive packets to be classified and the large-scale classification rules bring more severe challenges to the packet classification problem. In a word, although packet classification has been studied for decades, with the emergence of new network services and new requirements, there are still many technical barriers to break through. Especially considering the real-time and rapid classification requirements of massive packets, how to further improve the speed of packet classification, so as to meet the requirements of high-speed packet classification in the new generation network environment still needs to be explored and studied.

At present, large routers and high-end classifiers mainly use hardware devices to classify packets based on exhaustive search. Its core idea is to directly traverse all the rules in the rule list and get the matching result. Typical hardware-based solutions include ternary content addressable memory (TCAM) [7], field programmable gate array (FPGA) [8], and dedicated network processor chip. The exhaustive search algorithm has simple data structure and high classification efficiency. For example, the packet classification algorithm based on TCAM adopts parallel search scheme, and the time complexity of the algorithm is $O(1)$. However, dedicated hardware has some disadvantages, such as high price, long development time and high energy consumption, which limit their application and scalability to some extent. At present, in academia, researchers have put forward many general solutions based on software for packet classification.

(1) Dimension-decomposition-based methods.

The algorithm based on dimension decomposition decomposes each multidimensional rule into multiple dimensions in a certain number of bytes or bits. Each dimension is searched separately, and then the final search result is obtained by combination. Cross-Producting [9] is an early classical algorithm, which firstly matches in each dimension respectively, then combines the results of each dimension to form a cross-product, and finally maps it to a product table to get the best match. The

algorithm makes full use of the idea of sacrificing memory for speed to achieve fast matching of k -dimensional classification rules. The cross product algorithm has a short search time, but in the worst case, the space complexity is $O(n^k)$ (n is the number of rules).

ABV [10] reduces memory access and improves classification speed by aggregating bit vectors. However, its memory consumption is high because it needs to store additional information, such as aggregated bit vectors. Through the modular BV architecture, StradBV [11] eliminates the rule expansion caused by the conversion from range to prefix. Different from the Cross-Producing method, RFC (Recursive flow classification) [12] uses multilevel mapping to transform packet classification into table lookup process, which has good classification performance. However, due to the long preprocessing time, extral class tables need to be stored, which consumes a lot of memory. To sum up, these methods are fast, but as the scale of classification rules increases, the memory consumption will increase exponentially in the worst case.

(2) Tuple-space-search-based methods.

The algorithm based on tuple space constructs a hash table for each different prefix length, and the subsets of rules 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.

A classic algorithm is TSS (Tuple space search) [13], which divides the classification rules into multiple rule subsets according to the prefix bits of each field and stores them in hash tables. When a packet is received, TSS first finds the corresponding rule subset through the hash key, and then searches the subset for the most matching rule. The main disadvantage of tuple space search method is that the number of hash tables will greatly increase with time, which leads to slow packet classification. The representative algorithms are TupleMerge, PartitionSort [14–17], and so on.

(3) Decision-tree-based methods.

Algorithms based on decision tree can be divided into two categories: one is based on trie, whose basic idea is to establish a hierarchical binary tree according to the classification rules, divide each dimension of the rules into one layer, then recursively expand the one-dimensional tree structure, and finally generate a k -dimensional hierarchical tree. The advantages of the algorithm based on trie are simplicity, directness and easy hardware implementation. Its disadvantage is that it takes a long backtracking time, which is not conducive to the expansion of the rule dimension and cannot directly support range matching. SplitTrie [18] improved the basic Trie-based algorithm, supporting multi-field search and avoiding backtracking, but the algorithm still does not support range matching.

Another algorithm based on decision tree is to build a decision tree by recursively decomposing multidimensional space. Typical classical algorithms are HiCuts,

HyperCuts [19] and EffiCuts [20], etc. These algorithms divide the search space into several subspaces of equal size by using local optimization until the number of rules in each subspace is less than the predefined threshold τ . It shows excellent search performance, but equal-scale cutting will lead to huge storage requirements. H. Lim et al. reduced the memory consumption of the algorithm through boundary-based cutting [21]. Hybridcuts [22] divide rules on a single rule field instead of all fields, which reduces the number of subsets and the frequency of memory access. Bitcuts [23] and Uscuts cut rules based on bit and unit space, respectively, achieving a better balance between classification speed and space consumption. Bytecuts [24] intelligently divides classification rules into multiple trees through byte segmentation, thus reducing rule duplication. Mbitcuts [25] reduces the space consumption and memory access in the algorithm by changing the bit selection mode when cutting the geometric space model of each tree node.

Compared with the cutting-based method, the segmentation-based method divides the search space into multiple equal-density subsets. "Equal-density" means that the number of rules in each subset is almost the same. HyperSplit [26] is a classical segmentation method, which divides the search space into two equal dense subspaces. However, this method will lead to the increase in memory consumption as the number of rules increases. As an improved version of HyperSplit, ParaSplit [27] uses a new partitioning algorithm to reduce the complexity of classification rules and its memory consumption. CutSplit [28] combines the advantages of cutting and segmentation to improve the performance of packet classification. However, for different rules, their performance varies greatly, which is a common problem faced by most decision tree-based algorithms except "rule replication".

At present, compared with other software-based classification methods, the method based on decision tree has more advantages in classification speed. Therefore, this paper continues to discuss the classification method based on decision tree, and optimizes the construction process of decision tree on the basis of existing methods, so as to further improve the classification speed.

3 The proposed approach

3.1 Problem description

According to the research and analysis of previous packet classification methods based on decision tree, the existing algorithms focus on how to transform the representation of classification rules from access control list to decision tree. Its core idea is to construct one or more decision trees covering all rules according to the characteristics of rules, including scale cutting, density splitting and boundary division. However, few methods consider the order of each layer when constructing decision trees.

Generally speaking, the key to construct a decision tree is to measure attribute selection, then construct different branches according to the different division of a certain attribute at a node, and determine the topology between each attribute. The attribute selection metric is to divide the data of the training set marked by a given

category into the "best" category, which determines the choice of topology and split position. So we need to consider how to choose an attribute as the root node of the decision tree from the data set composed of multidimensional attributes. That is, how to choose the attribute with the largest disorder degree in the attribute set as the dividing node every time?

According to the previous analysis, when the dimension k is high (e.g., greater than 4), the optimal scheme will be very time-consuming. Therefore, our idea is to find a heuristic local optimization solution, that is, to choose the "best" attribute every time we choose the attribute. At this time, the key to the problem is what criteria are used to measure the "best" attribute?

In Informatics, the concept of information entropy is introduced to measure the order (or disorder) of an object's attribute value. Information entropy is used to measure the expected value of random variables. The greater the information entropy of a variable, the more information it contains, that is, more information is needed to fully determine the value of the variable.

For the set of random variables $X = \{x_1, x_2, \dots, x_m\}$, if the occurrence probability of any random variable $x_i (i = 1, 2, \dots, m)$ is p_i , then the information entropy of X is expressed as:

$$H = - \sum_{i=1}^m p_i \log_2 p_i \quad (1)$$

Considering that information entropy is used to measure the expected value of a random variable, information entropy can be used to measure the uncertainty of categories in decision trees. The greater the information entropy of the attribute, the greater the uncertainty of the vector. Therefore, it can be considered to divide based on this attribute vector first.

Dong, X et al. proposed an attribute selection method based on information entropy [29]. Although this method is only applicable to rules expressed in prefix form, it has been proved by a large number of experiments that using information gain to measure the priority of attribute selection is helpful to construct a decision tree with better classification performance.

In this paper, C4.5 algorithm is used to calculate the optimal eigenfunction. Compared with ID3 algorithm, it is easy to fall into the trap of selecting attributes with the most values. C4.5 algorithm uses information gain ratio instead of information gain to select attributes, which overcomes the deficiency that information gain tends to select attributes with more values. Next, we use a specific classification rule to illustrate the implementation steps of the algorithm. For simplicity and intuition, two-dimensional classification rules are adopted here.

3.2 Classification algorithm based on information gain ratio

The implementation process of the algorithm includes the following four steps: (1) Pre-processing the original rules, and mapping the rules into multidimensional matrix space by rule mapping method to form a series of independent unit spaces; (2) Constructing a data set according to the coordinate projection interval, wherein

the attribute set $C = \{F_1, F_2, \dots, F_k\}$, and calculating the information gain ratio of each attribute; (3) Using the top-down recursive divide-and-conquer method and the greedy strategy without backtracking, the attribute with the largest information gain ratio is selected as the partition node to construct the classification decision tree; (4) Classify data packets by decision tree.

(1) Rule pre-processing.

Using the rule mapping method, the input k -dimensional classification rules are mapped to the k -dimensional matrix space M_k in reverse order, forming a series of independent unit spaces. Generally speaking, classification rules can be expressed in the form of intervals, such as " $F_1 \in D(F_1) \wedge F_2 \in D(F_2) \wedge \dots \wedge F_k \in D(F_k) \rightarrow decision$ ", where F_i ($1 \leq i \leq k$) represents the source address, destination address, source port and destination port, etc., $D(F_i)$ indicates the corresponding domain value interval, and $decision$ represents the action (*accept* or *discard*) performed by the rule.

According to the rule mapping idea based on multi-dimensional matrix [5], any k -dimensional classification rule can be mapped to k -dimensional matrix space M_k . In the mapping process, we use unit space cs (corresponding to a k -dimensional rectangle in k -dimensional matrix space) to represent the area that is finally decided to *accept*: $[(l_1, l_2, \dots, l_k)(d_1, d_2, \dots, d_k)]$, where l_i and d_i refer to the minimum boundary value and range of the area in each dimension, respectively.

(2) Calculating the Information gain ratio.

The purpose of this step is to calculate the information gain ratio of attributes according to the unit space obtained in the rule preprocessing stage. Firstly, the data set is constructed according to the definition of coordinate projection interval, as shown in Table 1. In this example, the dataset has two attributes, corresponding to the attribute set $C = \{F_1, F_2\}$; In addition, there are two category labels, which constitute the category set $L = \{accept, discard\}$. FPC [30] introduces the construction process of data set in detail. For any two unit spaces u and v , if they satisfy these two

Table 1 Constructed data set

F_1	F_2	Decision
[0, 0]	[4]	Discard
[0, 0]	[5, 6]	Accept
[0, 0]	[7, 8]	Accept
[1]	[4]	Accept
[1]	[5, 6]	Accept
.....
[7, 8]	[7, 8]	Accept
[9]	[4]	Aiscard
[9]	[5, 6]	Accept
[9]	[7, 8]	Aiscard

conditions: (1) in dimension F_x , $R(u, v, F_x) = adjacent$, and (2) in any other dimension F_y , $R(u, v, F_y) = included$ or $R(u, v, F_y) = crossed$, we let the coordinate value of u in dimension F_y to be the secant, then cut u into two or three sub-unit spaces. This operation on unit spaces is iteratively conducted in a certain dimension until all the k dimensions are completed. In the follows, we calculate the information gain ratio of each attribute based on the C4.5 algorithm (as described in Algorithm 1) and the data set described in Table 1.

Algorithm 1 C4.5(S)

Input : the attribute-valued dataset S , the category label $C = \{c_1, c_2, \dots, c_m\}$, the attribute $A = \{a_1, a_2, \dots, a_k\}$.

Output : the information entropy gain ratio of k attribute $IGR [k]$.

Begin

1. divide S into $S_c = \{S_{c_1}, S_{c_2}, \dots, S_{c_m}\}$ with C , where $S_{c_i} = \{s | s.label = c_i, s \in S\}$ and $S_{c_i} \cap S_{c_j} = \emptyset, 1 \leq i \neq j \leq m$;
2. calculate the information entropy $I(S_c)$ with formula(2);
3. **for** ($i := 1$ to k) **do** {
4. calculate the attribute entropy $E(a_i)$ with formula(3);
5. calculate the information gain of attribute: $Gain(a_i) = I(S_c) - E(a_i)$;
6. calculate the split information of attribute $Split_{info}(a_i)$ with formula(6);
7. calculate the information entropy gain ratio of attribute: $IGR(a_i) = Gain(a_i) / Split_{info}(a_i)$;
8. **end for**
9. **return** $IGR [k]$.

End

Step 1: Using information entropy to measure the uncertainty of category label to the whole sample.

Let S be a set of data samples, and its category label $C = \{c_1, c_2, \dots, c_m\}$. Class c divides the data sample set S into $S_c = \{S_{c_1}, S_{c_2}, \dots, S_{c_m}\}$, where $S_{c_i} = \{s | s.label = c_i, s \in S\}$ and $S_{c_i} \cap S_{c_j} = \emptyset, 1 \leq i \neq j \leq m$, $s.label$ represents the label of sample s . The information entropy formula of sample classification is as follows:

$$I(S_c) = I(S_{c_1}, S_{c_2}, \dots, S_{c_m}) = - \sum_{i=1}^m p_i \log_2 p_i \tag{2}$$

where $p_i = \text{length}(S_{c_i}) / \text{length}(S)$ is the probability that the sample belongs to category c_i . $\text{length}(S_{c_i})$ indicates the number of elements of category c_i in sample set S ; $\text{length}(S)$ indicates the number of elements in the sample set S , that is, the total number of samples. Substituting the data set in Table 1 into Formula (2), the category information entropy can be calculated as follows:

$$I(S_c) = -15/24 * \log_2(15/24) - 9/24 * \log_2(9/24) = 0.955.$$

Step 2: Using information entropy to measure the uncertainty of different values of each attribute.

Assuming that the attribute $A = \{a_1, a_2, \dots, a_v\}$ has v different values, then the sample set S can be divided into v disjoint subsets $\{S_1^A, S_2^A, \dots, S_v^A\}$ by using attribute A , where $S_j^A = \{s | s \in S, s.A = a_j\}$ and $j = 1, 2, \dots, v$. If attribute A is selected as the optimal partition feature, then the partitioned subset is the branch of the decision tree that grows out the S node of the sample set. The information entropy of the subset divided by attribute A is shown in the following formula.

$$E(A) = \sum_{j=1}^v \frac{\text{length}(S_j^A)}{\text{length}(S)} I(S_j^A) \quad (3)$$

$$I(S_j^A) = - \sum_{i=1}^m p_{ij} \log_2 p_{ij} \quad (4)$$

In which $\text{length}(S_j^A)$ represents the number of elements in the subset S_j^A , p_{ij} is the probability that the samples in S_j^A belong to category c_i , and its value is equal to the ratio of the number of samples in category c_i in S_j^A to the number of S_j^A . In this example, the two categories are F_1 and F_2 , where F_1 has eight different values, namely $\{[0,0], [1,1], [2,2], [3,3], [4,4], [5,5], [6,6], [7,8], [9,9]\}$, F_2 has three different values, namely $\{[4,4], [5,6], [7,8]\}$. Therefore, the information entropy of each attribute can be calculated: $E(F_1) = 0.689$; $E(F_2) = 0.652$.

Step 3: Using information gain to determine the division basis of decision tree branches.

The formula for calculating the difference between the information entropy of the whole data set on a branch of the decision tree and the information entropy of the current node is:

$$\text{Gain}(A) = I(S_c) - E(A) \quad (5)$$

Thus, the information gain can be calculated:

$$\text{Gain}(F_1) = I(S_c) - E(F_1) = 0.955 - 0.689 = 0.266.$$

$$\text{Gain}(F_2) = I(S_c) - E(F_2) = 0.955 - 0.652 = 0.303.$$

Step 4: Calculating the split information $\text{Split}_{\text{info}}(S)$.

Split information is defined as:

$$\text{Split}_{\text{info}_A}(S) = - \sum_{j=1}^v \frac{|S_j|}{|S|} \log_2 \left(\frac{|S_j|}{|S|} \right) \quad (6)$$

where S_i ($1 \leq i \leq v$) is the division of sample set S on attribute A , and it is assumed here that attribute A has v different values. Thereby, the attribute splitting information metric can be calculated:

$$\text{Split}_{\text{info}}(F_1) = 3.0$$

$$\text{Split}_{\text{info}}(F_2) = 1.585.$$

STEP 5: Calculating the information entropy gain ratio $\text{IGR}(S)$.

$$\text{IGR}(S) = \text{Gain}(S) / \text{Split}_{\text{info}}(S) \quad (7)$$

Therefore, the information gain ratio can be calculated:

$$\text{IGR}(F_1) = \text{Gain}(F_1) / \text{Split}_{\text{info}}(F_1) = 0.266 / 3.0 = 0.089.$$

$$\text{IGR}(F_2) = \text{Gain}(F_2) / \text{Split}_{\text{info}}(F_2) = 0.303 / 1.585 = 0.191.$$

Based on STEP 1 to STEP 5 of the above process, each attribute (unselected attribute) in the sample set S can be calculated to obtain the information entropy gain ratio.

(2) Constructing a classification decision tree based on the information gain ratio.

The purpose of this step is to determine the priority of attribute selection and build a classification decision tree according to the information gain ratio calculated in the STEP 5. In this example, it can be seen that $\text{IGR}(F_2) > \text{IGR}(F_1)$, so when constructing the decision tree, the F_2 dimension with larger information gain ratio is selected as the preferred attribute. Assuming that there are k attributes in the data set, the information entropy gain ratio of each attribute is calculated, which is recorded as $\text{IGR}[k] = \{\text{IGR}(F_1), \text{IGR}(F_2), \dots, \text{IGR}(F_k)\}$. Initially, the decision tree T only contains the root node 'root'. Next, we briefly describe the general process of building a classification decision tree based on the information gain ratio, as described in Algorithm 2.

Algorithm 2 Constructing a Classification Decision Tree based on the Information Gain Ratio

Input : the n unit spaces $cs_1 \sim cs_n$; the information entropy gain ratio of k attributes $IGR[k] = \{IGR_{F_1}, IGR_{F_2}, \dots, IGR_{F_k}\}$.

Output : the decision tree T .

Begin

1. **while** ($IGR[k] \neq \emptyset$)
2. $IGR_{F_u} \leftarrow$ the maximum of $\{IGR_{F_1}, IGR_{F_2}, \dots, IGR_{F_k}\}$;
3. calculate the coordinate projection interval $P(cs_1, \dots, cs_n, F_u)$ of n unit spaces in the F_u dimension, the number of $P(cs_1, \dots, cs_n, F_u)$ is t ;
4. add $P(cs_1, \dots, cs_n, F_u)$ to the decision tree T as child node $node_i$ ($i=1$ to t), and each child node constitute a subtree T_i ($i=1$ to t) of T , where $node_i$ is the root node of the subtree T_i ;
5. delete IGR_{F_u} in $IGR[k]$;
6. $k--$;
7. **end while**
8. **return** T .

End

Taking Fig. 6 as an example. In the initial case, T is a decision tree containing only the root node 'root'. Because $IGR(F_2) > IGR(F_1)$, we choose the attributes in the order of F_2 before F_1 to construct the decision tree. In Fig. 4, seven unit spaces ($us_1 \sim us_7$) in the two-dimensional matrix space form three coordinate projection intervals $\{[4,4], [5,6], [7,8]\}$ in the F_2 dimension. These three intervals are added to the decision tree T as child nodes $Node_i$ ($i=1$ to 3) of the root, respectively. Each child node constitutes a subtree T_i ($i=1$ to 3) of T , and each node $Node_i$ is the root node of the corresponding subtree T_i .

The root node of the subtree T_1 corresponds to the interval $[4,4]$, and all the associated unit spaces us_1 and us_5 form three coordinate projection intervals $[1,1]$, $[3,3]$ and $[7,8]$ on the F_1 dimension, and then these three projection intervals are added to the subtree T_1 as child nodes, respectively. Similarly, projection intervals $[0,4]$, $[6,9]$ are added to the subtree T_2 as child nodes; $[0,2]$ and $[7,8]$ are added to the subtree T_3 , and finally form the decision tree T , as shown in Fig. 6.

(4) Classifying packets

In the packet classification method based on decision tree, packet classification is essentially a query operation. For the decision tree or any of its sub-trees, the interval coordinate values corresponding to the child nodes of the root are strictly increasing, so the binary search method can be directly applied to search. As shown in Fig. 6, the root node of the decision tree has three sub-nodes, and the corresponding interval coordinate values are $[4,4]$, $[5,6]$ and $[7,8]$, which meet the strict increasing relationship.

Consider the classification process of k -tuple packet $P:(e_1, e_2, \dots, e_k)$, the classification starts from the root node of the decision tree. Because the first layer of the

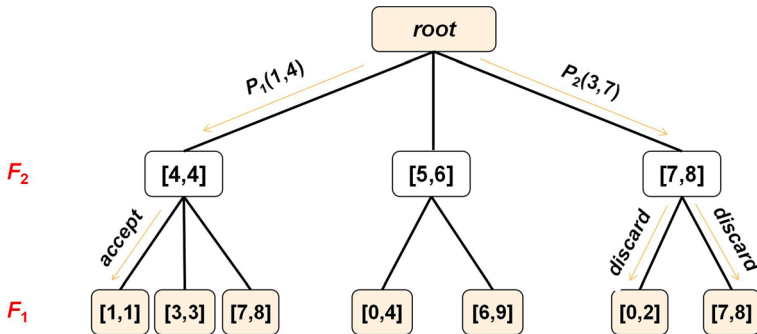


Fig. 7 Classifying packets based on decision tree

decision tree is divided based on the attribute F_2 , first, binary search is performed on all the child nodes of the *root* to determine whether the second metadata e_2 of the packet is included in the corresponding interval of a child node. If any interval cannot be matched, it can be directly determined that the packet P cannot match any node, and the packet is determined to be discarded; Otherwise, continue searching on the subtree with the node as the root. If each tuple $e_i (i = 1$ to $k)$ of the packet P matches the corresponding interval of each layer node of a subtree branch, it can be determined that the packet P matches the decision tree, and the decision of P is *accept*.

As shown in Fig. 7, assuming that the packets to be classified is $E = \{p_1, p_2\} = \{(1,4), (3,7)\}$, we first analyze the matching result of packet $p_1:(1,4)$, where $e_1 = 1$ and $e_2 = 4$. Starting from the root node, through binary search on all child nodes, it can be seen that $e_2 = 4$ matches the interval $[4,4]$ of the first node, and then continue searching on the subtree with this node as the root. Since $e_1 = 1$ can match one of the branch intervals $[1,1]$ in the subtree, it can be determined that the classification decision of p_1 is *accept*.

Next, continue to classify the packet $p_2:(3,7)$, where $e_1 = 3$ and $e_2 = 7$. Because e_2 matches the coordinate interval $[7,8]$ of the third child node, the search continues on the subtree with the child node as the root. However, the first dimension of $p_2:(3,7)$ is '3', which does not match any interval $[0,2]$ or $[7,8]$ corresponding to the two branches, therefore it can be determined that the classification decision of packet p_2 is *discard*. The specific process of packet classification is described in Algorithm 3.

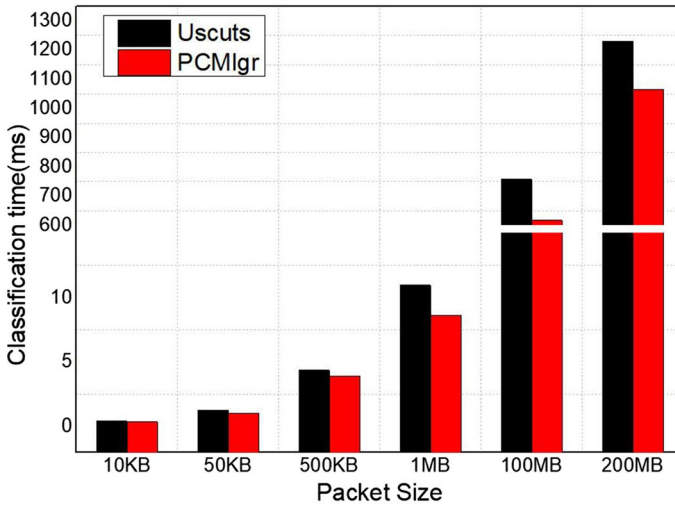


Fig. 8 Comparison of classification efficiency between PCMIgr and Uscuts

Table 2 Packet classification time (*ms*) under classification rules with different size (*n*)

Packet size	Packet classification time/ <i>ms</i>							
	<i>n</i> = 25	<i>n</i> = 100	<i>n</i> = 200	<i>n</i> = 400	<i>n</i> = 500	<i>n</i> = 1000	<i>n</i> = 1500	<i>n</i> = 1800
10 k	0.1945	0.3491	0.6406	0.7239	0.8041	0.9587	0.9638	1.0726
50 k	0.6505	1.0498	1.2024	1.3172	1.4276	1.8926	2.0894	2.2295
500 k	2.7596	3.8992	5.0105	5.8753	6.5245	8.9131	9.4701	9.8488
1 M	4.8022	8.6522	9.1253	10.0409	10.8632	14.8887	16.6322	17.4159
100 M	279.5787	616.8604	753.2394	908.9015	1109.1896	1387.5278	1477.5949	1592.4802
200 M	466.3585	1066.3291	1250.2795	1513.2574	1996.9552	2565.1483	2874.0079	3224.2313

Algorithm 3 Packet Classification**Input:** the packet $P:(e_1, e_2, \dots, e_k)$, 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 $root$. Bisearching the i^{th} metadata e_i of P in all the child nodes of the $root$, if e_i is included in the interval corresponding to the s^{th} child node $Child(root, s)$ of $root$, then continue to search for e_{i+1} in the sub-tree 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.   } else  $P \rightarrow discard$ ;
6.   break;
7. } end for
8.   if ( $i > k$ )
9.      $P \rightarrow accept$ .

```

End

4 Performance evaluation and analysis

4.1 Effectiveness

In order to test the effectiveness of PCMIgr method in constructing decision tree using C4.5 algorithm, according to the classification rules shown in Fig. 1, we use PCMIgr and Uscuts methods to classify packets with different sizes (from 10 KB to 200 MB). Unlike the PCMIgr method, the Uscuts method directly constructs the decision tree from the F_1 dimension to the F_k dimension. The experimental results of classification are shown in Fig. 8. It can be seen that the classification efficiency of PCMIgr method is higher than that of Uscuts method, which indicates that reconstructing the decision tree based on the information gain ratio can further improve the classification efficiency.

To further verify the classification efficiency of PCMIgr, we have generated eight classification rules of different number (ranging from 25 to 1800, respectively) and six, data packets of different sizes (10 KB, 50 KB, 500 KB, 1 MB, 100 MB, and 200 MB respectively) to test the time required to classify packets using PCMIgr algorithm. We generated these rules by ClassBench [31], which is a well-known benchmark that provides classifiers similar to real classifiers used in Internet routers and inputs traces corresponding to the classifiers. These algorithms were implemented in Java jdk1.7, and our experiment is conducted on a desktop PC running Windows10, which has 16G memory and 1.80 GHz Intel (R) core (TM) i7-10510u Processor. The test results are shown in Table 2.

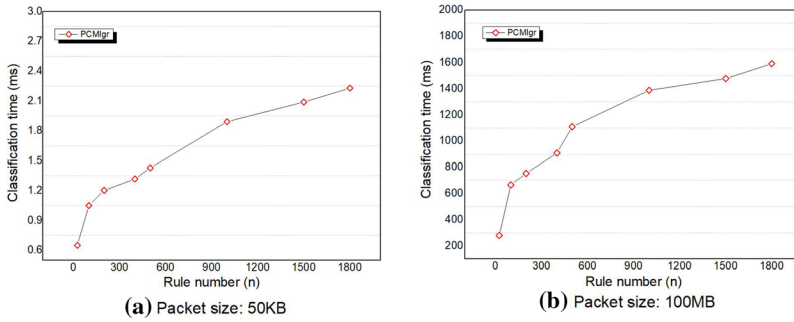


Fig. 9 Classification time of PCMIgr method under different size of rules and data packets

Table 3 Comparison of time complexity of algorithms in worst case

Algorithm	$O(T_{\text{worst}})$
TSS	$O(n^k)$
Linear search	$O(n)$
Grid-of-Tries	$O(2w)$
Hicuts	$O(w + \tau \cdot \text{Rulesize}/C)$
HyperSplit	$O(k \cdot \log_2(2n + 1))$
PCMIgr	$O(k \cdot \log_2 n)$

It can be seen that for the classification rules with the scale of 1800, when the data set reaches 200 MB, the time required to classify these packets is no more than 3500 ms. Moreover, the running time of the system also includes the process of rule preprocessing and decision tree construction, which can be carried out offline in advance in actual classification.

4.2 Efficiency

According to the decision tree constructed by PCMIgr method, only one rule is associated with each leaf node. Therefore, in the process of packet classification, when visiting the leaf nodes of the decision tree, it is not necessary to continue the sequential search in the rule grouping as the traditional packet classification method based on the decision tree, but to directly determine the classification decisions of packets. At the same time, for the decision tree or any of its subtrees, the interval coordinate values of each child node corresponding to the root of the tree are strictly increasing, so the efficient binary search method can be adopted.

Assuming that the original rule number is n , then according to the rule mapping method based on multidimensional matrix [5], the number of unit spaces formed will be less than n , and the projection interval of n unit spaces in any dimension will not be greater than $2n-1$. As described in the decision tree construction process of

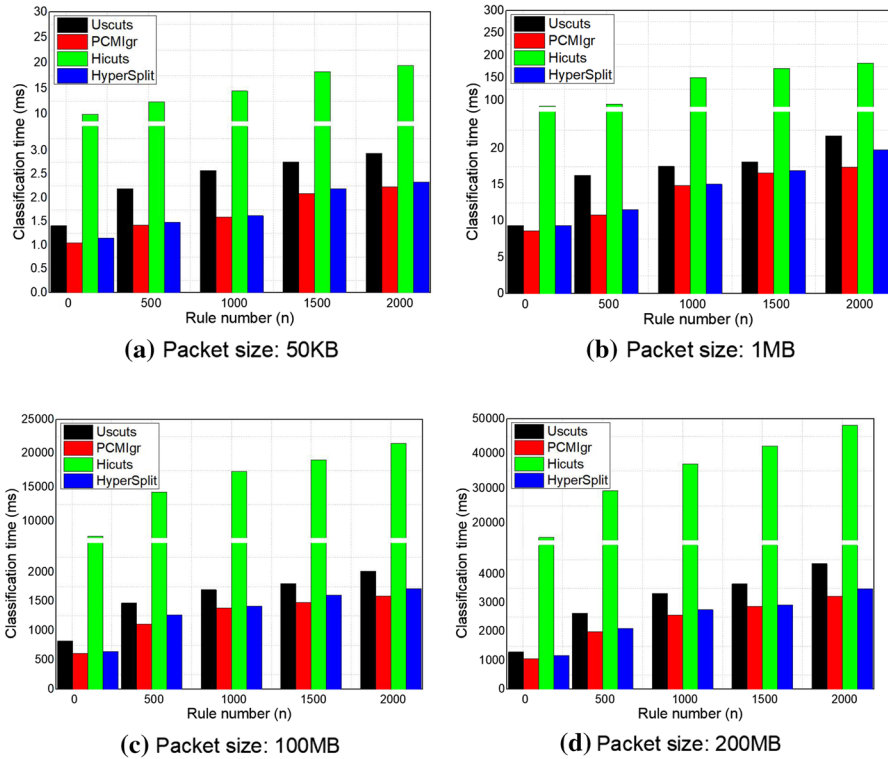


Fig. 10 Comparison of classification speed of Hicuts, Uscuts, HyperSplit and PCMIgr methods

PCMIgr method, the number of child nodes of the root of the decision tree or any of its subtrees will not exceed $2n-1$, so in the worst case, the corresponding search time is $\log_2(2n-1)$ and the time complexity is $O(\log_2(n))$. Therefore, when searching on a k -level decision tree, the time complexity in the worst case is $O(T_{\text{worst}}) = O(k \cdot \log_2 n)$, where n and k refer to the number and dimension of rules, respectively.

In practical network classification applications, the data packets and rules are dynamic. Two hypotheses are put forward and verified by experiments: (1) About 50% of the m data packets to be classified are classified as *discard*; (2) The packet classified as *discard* has about $1/k$ probability of not finding a matching node in the F_i ($i: = 1$ to k) dimension [4]. The experimental results show that if the number of data packets to be classified is m , about $m/2$ data packets will be classified as *accept*, and these $m/2$ data packets will need up to $\log_2(2n)$ access time in each dimension, so the total time required is $(m/2) \cdot [k \cdot \log_2(2n)]$; For the other $m/2$ packets whose decision is *discard*, there is a probability of $1/k$ in each dimension that no matching coordinate interval can be found. Therefore, when the matching coordinate interval cannot be found in the i ($i: = 1$ to k) dimension, the execution time is $(m/2 k) \cdot i \cdot \log_2(2n)$, and the time complexity of PCMIgr in average case, T_{avg} , is:

$$T_{\text{avg}} = \left(\frac{mk}{2} \cdot \log_2 n + \frac{m}{2k} \sum_{i=1}^k i \cdot \log_2 n \right) / m = \frac{3k+1}{4} \cdot \log_2 n \quad (8)$$

Considering the searching process of the k -layer decision tree, the time complexity of PCMIgr method in average case is $O((0.75k+1) \cdot \log_2 n)$. According to the classification results shown in Table 2, with the number of classification rules n as the abscissa and the classification time of packet as the ordinate, we randomly select the classification time of two packets with different sizes (50 KB and 100 MB, respectively) under different classification rules, and map them on the coordinate chart. The mapping results are shown in Fig. 9. It can be seen that the classification time trend approximately conforms to the logarithmic function curve, and basically consistent with the average time complexity analysis results of the PCMIgr method. Table 3 lists the time complexity of PCMIgr method and several classical packet classification methods in the worst case. In Table 3, n and k refers to the number and dimension of rules, respectively, and w is the length of rule domain (in IPv4 protocol: $w=104$); τ is the threshold of the number of rules associated with leaf nodes, and RuleSize is the number of bytes occupied by a single rule (generally $\text{RuleSize}=24.5\text{Byte}$ [21]). C is the number of bytes of CacheLine, and its size generally ranges from 16 to 256 bytes. Currently, the mainstream CacheLine is 64 Byte.

As shown in Table 3, in the worst case, the time complexity of TSS algorithm is $O(n^k)$, Grid-of-Tries algorithm is $O(2w)$ and HiCuts algorithm is $O(w + \tau \cdot \text{RuleSize}/C)$. Even if the number of classification rules n reaches 100,000, $k \cdot \log_2 n$ is still less than w (typical rule dimension $k=5$). Therefore, compared with Grid-of-Tries, HiCuts and other algorithms, the execution efficiency of PCMIgr classification algorithm has certain advantages, and it does not need extra space to store the classification rules, which reduces the requirement of storage space to some extent. For example, HyperSplit is close to PCMIgr in time complexity, but it requires more memory storage.

For the sake of intuition, we comprehensively compare the classification speed of Hicuts, Uscuts, PCMIgr and HyperSplit. The difference among the first three methods is that the order of attribute selection is different when constructing decision tree. The Hicuts and Uscuts methods are in the order from F_1 dimension to F_k dimension, while the PCMIgr method selects the corresponding dimensions to construct the decision tree according to the information gain ratio of each attribute from the largest to the smallest. Four data packet sets with different sizes, namely 50 KB, 1 MB, 100 MB and 200 MB, were selected in the experiment. The classification rules included five different sizes, ranging from 100 to 2000. The experimental results are shown in Fig. 10.

As shown in Fig. 10, when classifying data packets of the same size, the classification speed of the PCMIgr method is slightly higher than that of the HyperSplit method, and it has a certain improvement compared with the Uscuts method, while it has obvious advantages compared with the Hicuts method. The experimental results show that the PCMIgr method preprocesses the original rules and avoids the problem of "rule replication", which is of great significance to improve the classification speed. In particular, compared with the Uscuts method, PCMIgr chooses to

construct the decision tree based on the information gain ratio, and the comparison of classification results further proves that this idea has certain reference value for the improvement of classification efficiency based on decision tree.

5 Conclusions

With the development of network application, higher requirements are put forward for the speed of network packet classification. In this paper, a heuristic decision tree construction method, PCMIgr, is proposed. It is based on the greedy strategy. When each decision tree node needs to select the classification attribute, it selects the attribute with the highest information gain ratio for classification. This method optimizes the attribute selection in the traditional decision tree construction process, and the classification efficiency is greatly improved compared with that before optimization. This method also avoids the common problem of "rule replication" in traditional classification methods based on decision tree, and effectively saves storage space. The experimental results show that applying PCMIgr method to the construction of classification decision tree can further improve the efficiency of packet classification method based on decision tree. This idea also provides a new way for the research of packet classification.

Acknowledgements This work was supported by Hunan Provincial Natural Science Foundation of China (No. 2022JJ60099), the Research Foundation of the Education Department of Hunan Province (No.21C1589), the Doctoral Scientific Research Project of Changsha Social Work College (No. 2020JB32), and the National Natural Science Foundation of China (No.61877059). The authors would like to thank the anonymous reviewers and the editors for their helpful suggestions and comments.

Author contributions YC designed the experiments and wrote the paper; QS contributed the simulation experiments; all authors have read and approved the final manuscript.

Data Availability The datasets generated or analyzed during this study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Cheng Y, Shi Q (2022) MpFPC-A parallelization method for fast packet classification. *IEEE ACCESS* 10:38379–38390
2. Taylor D (2005) Survey and taxonomy of packet classification techniques. *ACM Comput Surv* 37(3):238–275
3. Gupta P, McKeown N (1999) Packet classification using hierarchical intelligent cuttings. In: *IEEE Annual Symposium on HOTI'99*, pp 34–41
4. Cheng Y, Wang W, Wang J et al (2018) A fast firewall packet classification algorithm using unit space partitions. *Adv Eng Sci* 50(4):144–152
5. Cheng Y, Wang W, Min G et al (2015) A new approach to designing firewall based on multidimensional matrix. *Concurr Comput Pract Exp* 27(12):3075–3088
6. Overmars M, Stappen F (1996) Range searching and point location among fat objects. *J Algorithms* 21(3):629–656
7. Norige E, Liu AX, Torng E (2018) A ternary unification framework for optimizing TCAM based packet classification systems. *IEEE/ACM Trans Netw* 26(2):657–670
8. Li C, Li T, Li J et al (2019) Memory optimization for bit-vector-based packet classification on FPGA. *Electron* 8(10):1–16
9. Srinivasan V, Varghese G, Suri S, et al (1998) Fast and scalable layer four switching. In: *Proceeding of ACM SIGCOMM'98*, pp 191–202
10. Baboescu F, Varghese G (2001) Scalable packet classification. In: *Proceeding of ACM SIGCOMM'01*, pp 2–14
11. Ganegedara T, Jiang W, Prasanna V (2014) A scalable and modular architecture for high-performance packet classification. *IEEE Trans Parallel Distrib Syst* 25(5):1135–1144
12. Gupta P, McKeown N (1999) Packet classification on multiple fields. In: *Proceedings of ACM SIGCOMM'99*, pp 147–160
13. Srinivasan V, Suri S, Varghese G (1999) Packet classification using tuple space search. In: *Proceedings of ACM SIGCOMM'99*, pp 135–146
14. Pak W, Choi Y (2015) High performance and high scalable packet classification algorithm for network security systems. *IEEE Trans Dependable Secur Comput* 14(1):37–49
15. Daly J, Bruschi V, Linguaglossa L et al (2019) TupleMerge: fast software packet processing for online packet classification. *IEEE/ACM Trans Netw* 27(4):1417–1431
16. Hsieh C, Weng N, Wei W (2019) Scalable many-field packet classification for traffic steering in SDN switches. *IEEE Trans on Netw Serv Manag* 16(1):348–361
17. Li W, Yang T, Rottenstreich O et al (2020) Tuple space assisted packet classification with high performance on both search and update. *IEEE J Sel Areas Commun* 38(7):1555–1569
18. Li Y, Wang J, Chen X et al (2022) SplitTrie: a fast update packet classification algorithm with trie splitting. *Electronics* 11(2):1–13
19. Singh S, Baboescu F, Varghese G, et al (2003) Packet classification using multidimensional cutting. In: *Proceeding of ACM SIGCOMM'03*, pp 213–224
20. Vamanan B, Voskuilen G, Vijaykumar T (2011) Efficuts: optimizing packet classification for memory and throughput. In: *Proceeding of ACM SIGCOMM'11*, pp 207–218
21. Lim H, Lee N, Jin G et al (2014) Boundary cutting for packet classification. *IEEE/ACM Trans Netw* 22(2):443–456
22. Li W, Li X (2013) Hybridcuts: a scheme combining decomposition and cutting for packet classification. In: *Proceedings of IEEE HOTI'13*, pp 41–48
23. Liu Z, Sun S, Zhu H et al (2017) BitCuts: a fast packet classification algorithm using bit-Level cutting. *Comput Commun* 109:38–52
24. Daly J, Torng E (2018) ByteCuts: fast packet classification by interior bit extraction. In: *Proceedings of IEEE INFOCOM'18*, pp 2654–2662
25. Abbasi M, Fazel S, Rafiee M (2020) MBitCuts: optimal bit-level cutting in geometric space packet classification. *J Supercomput* 76:3105–3128
26. Qi Y, Xu L, Yang B, et al (2009) Packet classification algorithms: from theory to practice. In: *Proceedings of IEEE INFOCOM'09*, pp 648–656
27. Fong J, Wang X, Qi Y, et al (2012) Parasplit: a scalable architecture on FPGA for terabit packet classification. In: *Proceedings of IEEE HOTI'12*, pp 1–8

28. Li W, Li X, Li H, et al (2018) Cutsplit: a decision-tree combining cutting and splitting for scalable packet classification. In: Proceedings of IEEE INFOCOM'18, pp 2645–2653
29. Dong X, Qian M, Jiang R (2020) Packet classification based on the decision tree with information entropy. *J Supercomput* 76:4117–4131
30. Cheng Y, Wang W, Wang J et al (2019) FPC: a new approach to firewall policies compression. *Tsinghua Sci Technol* 24(1):65–76
31. Taylor D, Turner J (2007) ClassBench: a packet classification benchmark. *IEEE Trans on Networking* 15(3):499–511

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.