International Journal of Innovative
Computing, Information and Control
Volume **17**, Number **4**, August **2021**

ICIC International ©2021 ISSN 1349-4198
pp. 1345–1361

AN IMPROVED PACKET CLASSIFICATION ALGORITHM SUPPORTING ADAPTIVE RULE SET PARTITIONING BASED ON SMALL-BIG FIELD

Chuanhong Li[1,2], Lei Song[1,2,*] and Xuewen Zeng[1,2]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
[*]Corresponding author: songl@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100190, P. R. China

Received January 2021; revised May 2021

Abstract. *Packet classification is a core feature for various services based on packet forwarding, attracting lots of researchers' attention. The decomposition-based packet classification algorithm is a promising method since it is more suitable for different rule sets. However, as the number of the rule increases, the pre-processing time to deal with the rule set and the memory consumption to store the rule set are increasing greatly, which is an unsolved problem. In this paper, we propose an improved packet classification algorithm supporting adaptive rule set partitioning based on small-big field. Through modeling the above issues, we propose some improvements including boundary-based rule traversal, the adaptive rule set partitioning based on small-big field, and rules adjustment to reduce the pre-processing time and memory consumption. Experimental results show that the proposed algorithm achieves an average memory reduction of 33.4% for large rule sets as well as 4.1 times improvement on pre-processing time on average for all rule sets in comparison with the PCIU algorithm. Compared with CutSplit and HybridCuts, the proposed algorithm also outperforms them in pre-processing time.*
Keywords: Packet classification, Decomposition, Rule set partitioning, Small-big field

1. **Introduction.** Packet classification plays a significant role in various network applications based on packet forwarding, such as Quality of Service (QoS) and network billing [1]. Packet classification is to find a matching rule from a rule set called classifier and perform an action corresponding to the matched rule, such as drop or forward for each packet. As the network traffic increases explosively, the performance of packet classification has gradually become the bottleneck of various network applications based on packet forwarding and has become a hotspot in recent years [1,2].

According to [3], we can simply categorize packet classification algorithms into the following four kinds: exhaustive search-based, decision tree-based, decomposition-based, and tuple space search-based. Packet classification algorithms based on exhaustive searching, generally rely on some special hardware to achieve line-rate packet classification, such as Field Programmable Gate Array (FPGA) [4,5] and Ternary Content Addressable Memory (TCAM) [6-9]. However, it is expensive and power-hungry, thus limiting its scalability [1,10]. Decision tree-based packet classification algorithms, construct one or more decision trees to cover the whole packet classifier according to the characteristics of the classifier [1,2,11-17]. Unfortunately, most of them suffer from rule replication problems, which

DOI: 10.24507/ijicic.17.04.1345

result in memory explosion as the rule sets become large. Besides, almost all of them are only suitable for some special rule sets. As for the tuple space search-based packet classification algorithm [18], the rule sets are divided into different partitions according to the prefix length of each field, and rules in the same partition are stored using the hash table since they have the same prefix length. All these partitions form the tuple space. The main drawback of tuple space is that the number of partitions/tables is large, which results in slow packet classification due to the fact that many tables need to be searched [19].

Decomposition-based packet classification algorithms use the idea of divide and conquer. By decomposing the multi-dimensional packet classification problem into multiple one-dimensional ones, these methods combine all of the results of these one-dimensional packet classification problems to get the final matching result. Bit Vector (BV) [21], the earliest algorithm, is proposed to use the bit vector to represent whether the rules are met; however, the length of BV is decided by the number of rules in the classifier, leading to a great increment in memory consumption as the number of rules increases. Besides, since it uses an 8-bit lookup, the BV intersection needs lots of memory accesses, which in turn leads to a decrease in search performance [22]. As an improved version, Baboescu and Varghese propose Aggregated Bit Vector (ABV) [23]. The proposed method makes use of bit aggregation and so on to reduce the memory access, thus speeding up classification speed. However, the memory consumption problem is still left, and even becomes more serious since it needs to store extra information. The classic decomposition-based packet classification algorithm, crossing-producing [24], constructs only one big cross-product table to cover the rule set, and the operation of Class Bitmap (CBM) intersection is performed in one stage. The number of rules increase will result in memory explosion, which limits its usability. To alleviate the above issue, Gupta and McKeown propose an improved algorithm, named Recursive Flow Classification (RFC) [25]. They perform bitwise AND of all possible CBMs in the pre-processing stage and store the (intermediate) results in Equivalent Class Tables (ECT). Using multi-stage mapping, RFC combines a few (two or three) ECTs from the previous stage to generate the new one in the current stage until there is only one table left in the final stage. The procedure of packet classification is only some table lookups; therefore, it achieves an excellent classification performance. However, the generation of intermediate ECTs not only makes the pre-processing time longer but also increases memory consumption. In recent years, many optimizations have been done to reduce memory consumption and pre-processing time, such as [20,22,26]. Unfortunately, due to the inherent complexity of the RFC, it is still difficult to satisfy various requirements. PCIU [27] is another decomposition-based packet classification algorithm, similar to the RFC phase 0, but different in the following aspects: firstly, PCIU uses 8-bit chunks while RFC uses 16-bit chunks; what is more, none of the intermediate ECTs is needed for PCIU; lastly, the intersection of the CBMs is performed in the classification stage, not in the pre-processing stage. The mentioned differences make PCIU sacrifice classification performance to some extent in exchange for a reduction in pre-processing time and memory consumption. However, as the number of rules increases, it still takes a long time to pre-process, and memory consumption is still very high. In our previous work [29], we propose an improved PCIU algorithm to reduce memory consumption and pre-processing time. However, more partitions harm the classification performance.

Among these algorithms, decomposition-based packet classification algorithms are promising due to the following two reasons. On the one hand, parallelism offered by modern hardware can be used to speed up the classification performance [20]. On the other hand, they are not dependent on the characteristics of the rules, which makes them more suitable for various rule sets to satisfy various service requirements. However, the

increased memory consumption and preprocessing time caused by the increase in the number of rules in the rule set are still unsolved.

To alleviate these issues, in this paper, we propose an improved packet classification algorithm supporting adaptive rule set partitioning based on small-big field. Firstly, by the memory consumption model we give, we construct an in-depth analysis of the real reasons behind the above issues. Then, we apply a programming technique called boundary-based rule traversal to avoiding unnecessary comparisons to speed up the pre-processing. What is more, a rule set partitioning based on small-big field is used to reduce the number of rules that need to be processed at one time, which is not only beneficial for shortening pre-processing time but also for reducing memory consumption. To further accelerate the pre-processing stage, we apply rules adjustment to solving the problem of the unbalanced distribution of rules in each subset, incurred by the rule set partitioning. Lastly, based on the memory consumption model we give, an adaptive rule set partitioning is considered as a supplement to avoid partitioning small rule sets, which would lead to an increase in memory consumption.

The rest of the paper is organized as follows. Section 2 presents an introduction to the Packet Classification algorithm with an Incremental Update capability (PCIU) algorithm and analyzes the issues of PCIU. Section 3 describes the proposed packet classification algorithm. In Section 4, the experimental results are given. At last, we conclude our paper with a discussion of future work in Section 5.

2. PCIU Introduction and Problem Statement.

In this section, we introduce the original PCIU algorithm in detail and conduct an in-depth analysis of its shortcomings in advance since our proposed packet classification algorithm is regarded as an improved version of the PCIU algorithm.

2.1. PCIU introduction.

For convenience, we use a classic 5-tuple rule set with three rules as an example to describe the procedures of pre-processing stage and classification stage, which makes it easy to understand the problems of PCIU. These three rules are selected from the ACL_10k rule set, generated by ClassBench [30], shown in Table 1. For each rule, we only extract the five-tuple information. The IP field and protocol field in Table 1 are represented in value/mask format, while the port field is a range format. For instance, the Source IP 186.79.91.0/24 has a 24 bits mask which can represent a range with the low part (186.79.91.0) and the high part (186.79.91.255). The IP address field with a 32-bit mask represents an exact value.

Table 1. A three-rule classifier

No.	IP (bits)		Port (32 bits)		Protocol (8 bits)
	Source (32 bits)	Destination (32 bits)	Source (16 bits)	Destination (16 bits)	
1	0.0.0.0/0	0.0.0.0/0	0:65535	0:65535	0/0
2	186.79.91.0/24	64.0.0.0/6	0:65535	0:65535	0/0
3	212.144.34.0/24	0.55.17.230/31	0:65535	1433:1433	6/ff

2.1.1. *The pre-processing procedure.* When loading rules, rules indicated in prefix format are converted to a range representation, which has a minimum part and maximum part. For instance, the source IP field of the third rule in Table 1 has a 24 bits mask, meaning the higher 24 bits are a fixed value while the lower 8 bits are varying. We change it to range representation with a low part (212.144.34.0) and a high part (212.144.34.255). Similar operations are performed for the destination IP field and protocol field. There

Table 2. Range representation of three rules

No.	IP (64 bits)				Port (32 bits)				Protocol (8 bits)	
	Source (32 bits)		Destination (32 bits)		Source (16 bits)		Destination (16 bits)			
	L	H	L	H	L	H	L	H	L	H
1	0.0.0.0	255.255.255.255	0.0.0.0	255.255.255.255	0	65535	0	65535	0	255
2	186.79.91.0	186.79.91.255	64.0.0.0	67.255.255.255	0	65535	0	65535	0	255
3	212.144.34.0	212.144.34.255	0.55.17.230	0.55.17.231	0	65535	1433	1433	6	6

is nothing to do with port fields since they are expressed in range representation. After translating, the results are shown in Table 2.

For each rule, all the fields will be divided into 8-bit chunks in a range representation. For example, according to the result in Table 2, the source IP field of the third rule is split into four chunks, $[0, 255]$, $[34, 34]$, $[144, 144]$, and $[212, 212]$. The total length of a 5-tuple rule is 13 bytes; therefore, there are 13 chunks generated. Table 3 shows the results of partitioning.

Table 3. The range format of the three rules

0		1		2		3		4		5		6		7		8		9		10		11		12	
L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255
0	255	91	91	79	79	186	186	0	255	0	255	0	255	64	67	0	255	0	255	0	255	0	255	0	255
0	255	34	34	144	144	212	212	230	231	17	17	55	55	0	0	0	255	0	255	153	153	5	5	6	6

For each chunk, a lookup table of size 2^8 is assigned. The lookup table is also called an index table since the value stored in the lookup table is the index pointing to a certain Bit Vector (BV) in the Equivalent Class Table (ECT) assigned to the corresponding chunk. The terms lookup table and index table are used interchangeably in this paper.

According to the pre-processing algorithm provided in [27], for each chunk, we enumerate each value in the chunk to check whether the value is in the range of the chunk of the rule. The result is expressed as a BV, the length of which is equal to the number of rules in the rule set. Each bit in the BV is pointing to a rule in the rule set. The corresponding bit is set to 1 if the value is in the range of the chunk of the rule; otherwise, it is set to 0. We use the generation of chunk#0 as an example to show the procedure of pre-processing. The index table size is 2^8; thus, there are 256 values of it. For the value 0, since the range of the chunk#0 of all three rules is $[0, 255]$, the value 0 is located in their range, meaning the value 0 satisfying three rules. Finally, a new BV-111 is generated and it will be stored in the ECT of chunk#0 with an identifier 0. Then, the identifier of BV-111 is filled in the index table of chunk#0 as the index for the value 0. For the remaining value of chunk#0, the processing is the same. However, since each value of chunk#0 satisfies all three rules, the same BV-111 is generated repeatedly. We only store the unique BVs in ECT; therefore, all these repeated BVs are removed, leading to only one BV-111 left in ECT. Correspondingly, we fill the identifier of the BV-111 in the index table for each value.

After pre-processing, the index tables and equivalent class tables are shown in Figure 1. For more details, please refer to [27].

2.1.2. *Classification stage.* When it comes to the classification stage, the 5-tuple is extracted from the header of the packet being classified. The 5-tuple information is divided into 8-bit values, each of which is an index to the corresponding lookup table to get the index to the BV in the ECT. Finally, all the BVs are intersected to get the matching

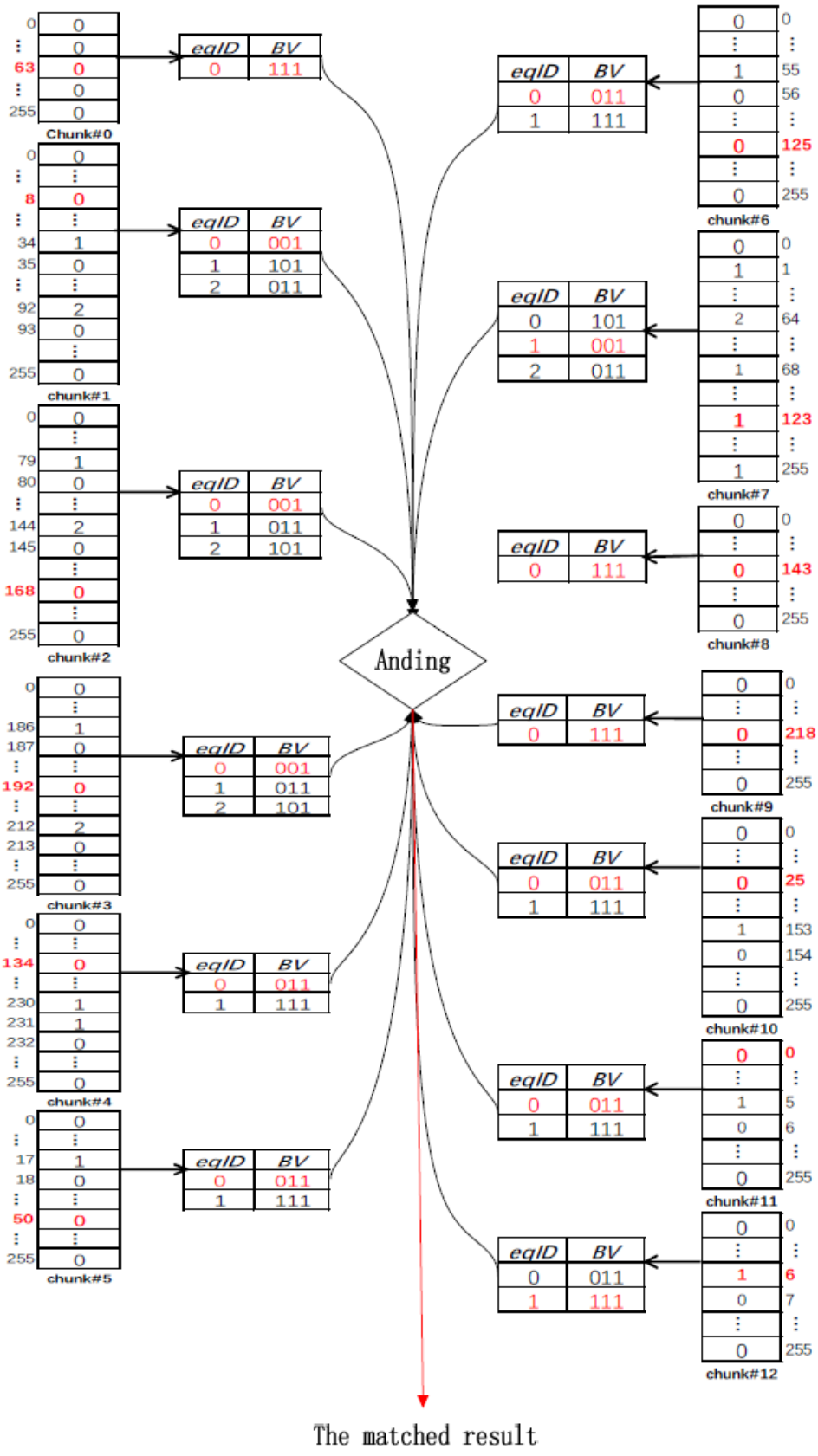

FIGURE 1. The chunk tables and equivalent class tables for the rule set shown in Table 1

result. For example, given a packet the source IP, destination IP, source port, destination port, and protocol code are 192.168.8.63, 123.125.50.134, 55951, 25, and 6 correspondingly. According to the pre-processing stage, we divide the 5-tuple into 13 bytes, using each byte as an index to get the eqID in the index tables, just as the red part in Figure 1. Then we get 13 BVs, 111, 001, 001, 001, 011, 011, 011, 001, 111, 111, 011, 011, and 111, the intersection of which is 001, showing that the packet matches the first rule in the rule set.

2.2. **Problem statements.** In this subsection, we follow the ideas in [29] and perform a more detailed analysis of the problems faced by the original PCIU algorithm.

2.2.1. *Memory consumption.* The memory consumption of the PCIU algorithm consists of two parts. One is used to store the index tables, and the other is used to store equivalent class tables. Supposed that M is the total memory consumption. M_{index} and M_{ect} are the memory consumption of lookup tables and equivalent class tables, respectively. Then we have the following equation:

$$M = M_{index} + M_{ect}. \tag{1}$$

As Figure 1 shows, each entry in the index table only contains an identifier, which is an index pointing to a BV in the ECT. Besides, the number of entries in each index table is fixed, written as K. Therefore, M_{index} can be represented as Equation (2):

$$M_{index} = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} m_{i,j}, \tag{2}$$

where N is the number of chunks and $m_{i,j}$ is the size of the jth entry in the ith chunk.

In each ECT as illustrated in Figure 1, it stores all the unique BVs and their identifiers for the chunk. Supposed that the size of the BV is len_{bv}, and the size of the identifier is len_{id}. Let n_i be the number of unique BVs in the ECT_i. M_{ect} can be expressed as Equation (3):

$$M_{ect} = \sum_{i=0}^{N-1} n_i \times (len_{bv} + len_{id}). \tag{3}$$

As we know, the size of the BV is equal to the number of rules in the rule set; therefore, we replace len_{bv} with n_{rules} in Equation (3), where n_{rules} is the number of the rules. Finally, the total memory consumption can be rearranged as Equation (4):

$$M = \sum_{i=0}^{N-1} \sum_{j=0}^{K-1} m_{i,j} + \sum_{i=0}^{N-1} n_i \times (n_{rules} + len_{id}). \tag{4}$$

With the number of rules in the rule set increasing, it is clear that the memory consumption increases drastically since the n_{rules} in Formula (4) increases rapidly. For the PCIU algorithm, the 5-tuple is fixedly divided into 13 chunks regardless of the number of rules, which means the memory consumption of the index tables is a fixed value. Meanwhile, as the number of rules increases, BVs account for the majority of memory consumption of ECTs. Therefore, when the rule set is large, Equation (4) can be simplified as follows:

$$M \sim \sum_{i=0}^{N-1} n_i \times n_{rules}. \tag{5}$$

We use a variety of rule sets, generated by ClassBench [30], with sizes ranging from 0.1k to 10k to verify the correctness of the above analysis. Figure 2 shows the results. As the figure shows, with the number of rules increasing, the memory consumption of ECTs accounts for the majority, for example, for rule sets with 10k rules, the memory

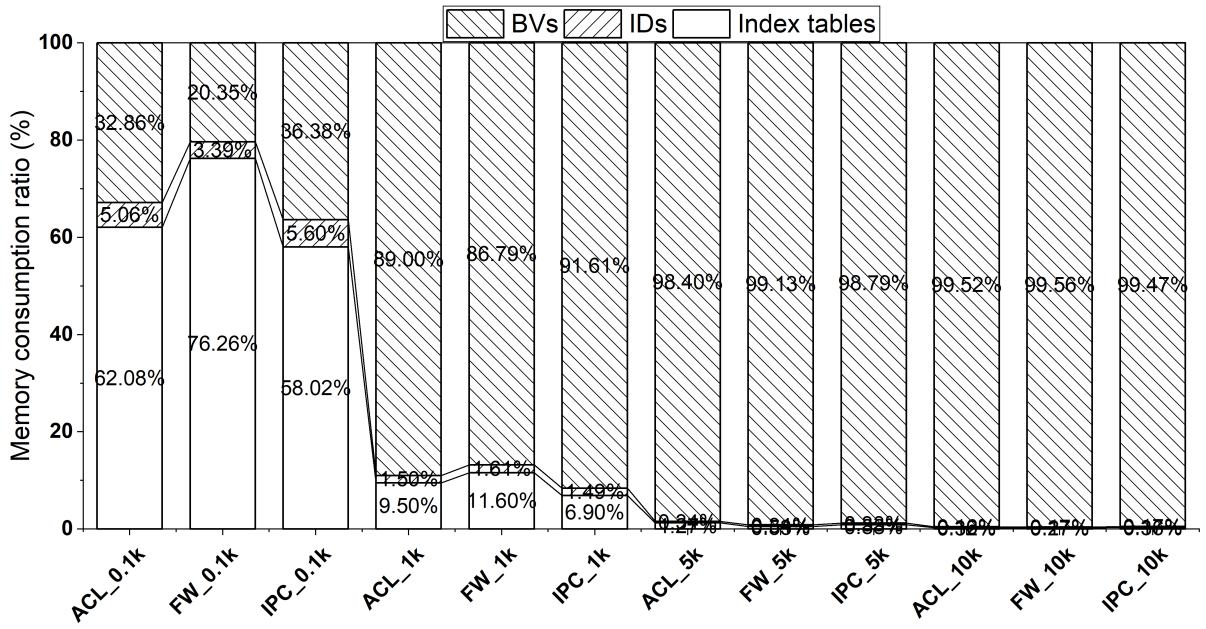

FIGURE 2. The memory consumption ratio for different rule sets

consumption ratio exceeds 99.4%, which is consistent with our analysis. More precisely, it is the memory consumption of BVs that accounts for the majority of that of ECTs.

From Equation (5), we can see that not only the number of rules but also the number of unique BVs in each ECT has an effect on memory consumption. As mentioned in [29], the BVs stored in the ECT are dependent on the characteristic of the rule set, which is out of control; however, through the rule set partitioning, we can reduce the number of rules, which in turn reduce the size of BVs.

In summary, to reduce memory consumption, the rule set partitioning is not a so bad idea.

2.2.2. *Pre-processing stage.* We use the generation of the chunk#0′ equivalent class table as an example, shown in Table 3 to describe the problem in the pre-processing stage. Using the pre-processing algorithm provided by PCIU, for each value in the first index table, a new BV is produced. At last, there are 256 BVs in total. However, only one of them is present in the equivalent class table since all of them are the same. The remaining one is 111, indicating that each value in the chunk#0 satisfies all three rules. Repeated generation of BVs needs additional comparisons to confirm the uniqueness of them. There is no doubt that these unnecessary operations will slow down the pre-processing, especially when the BV is large.

For 13 chunks, there are 3328 BVs generated, but only 27 BVs are stored in the ECTs, as Figure 1 shows. For each BV, at least one comparison is needed to determine its uniqueness. It is obvious that the pre-processing algorithm can be optimized to improve pre-processing performance.

3. **The Proposed Packet Classification Algorithm.** Based on the above analysis, we propose our packet classification algorithm with an adaptive rule set partitioning based on small-big field. Some improvements such as boundary-based rule traversal, the adaptive rule set partitioning based on small-big field, and rules adjustment are applied in our proposed method to reducing the pre-processing time as well as memory consumption. The detailed information is as follows.

3.1. **Boundary-based rule traversal.** This programming technology is first proposed in our previous work [29]. Using the boundary value of the rules in the rule set and a flag identifier, it makes sure that the unique BV is generated only once to avoid unnecessary comparisons as much as possible. The flag identifier is used to represent whether the BV has changed, whose value is 1 if the BV has done and 0 if not. Figure 3 shows the modified pre-processing algorithm.

```
/* Boundary-based pre-processing state*/
1 FOR each of the 13 chunks do
2  eqID := 0
3  ECT[eqID] := 0
4  eqID++
5  BV := 0
6  flag := 0
7  FOR I := 0, I < 256, I++
8    FOR ID := 1, ID <= NumberRule, ID++
9      IF I == Rule[ID].START
10       set rule bit ID to 1 in BV
11       Flag := 1
12     ELSE IF I == Rule[ID].M_END
13       set rule bit ID to 0 in BV
14     ENDIF_ELSE
15   ENDFOR
16   IF flag == 1
17     ECT[eqID] := BV
18     eqID++
19     Flag := 0
20   ENDIF
21  ENDFOR
22 ENDFOR
```

FIGURE 3. The modified pre-processing algorithm [29]

We continue to apply this technology in our packet classification algorithm since it achieves a nice performance improvement as the experimental results show in the next section.

Different from the original PCIU algorithm, we add a default BV 0 as the first entry of each ECT, indicating that none of the rules is matched, just as line 3 shows in Figure 3. The purpose of the default BV is as the initial value of the index tables. For each value of the index table, the goal of lines 8-15 is traversing the rule sets to produce the corresponding BVs. We use the $START$ as the start value of the current chunk of the rule and M_END represents the minimum integer larger than the end value of the same chunk of the same rule. These are the boundary values we use. Only when the current value is equal to the $START$, will we set the flag to 1, meaning a new BV is produced since the corresponding bit in the BV is changed from 0 to 1. The values in the range of $[START, M\_END)$ of the current chunk of the rule share the same bit value in the BV. When line 12 is true, it indicates that the value is out of the range; however, we do not change the flag value to 1 due to the fact that the BV must have already appeared in the ECT. The purpose of lines 16-20 is to add the new BV in the ECT. Compared with the original PCIU algorithm, we use the boundary value and the flag identifier to avoid the repeated generation of the BVs; hence, redundant comparisons are removed. As a result, the pre-processing time is shortened.

For example, we use the generation of the chuck#1′ ECT to depict the modified algorithm. For value 0, which is equal to the Rule[1].*START*, the corresponding bit in the BV is set to 1, which is 001. As for the second and third rules in Table 3, both the start values are larger than 0; thus ECT[1] is 001. For any value within $[1, 33]$, no new BV is generated. When the value is 34, the BV is changed to 101, making that ECT[2] is 101. When the value is 35, which is equal to the Rule[3].*M_END*, the BV becomes 001, which is already in the ECT. When the value is 91, the new BV is 011, which in turn leads to ECT[3] being 011. When the value is 92, the BV is changed to 001. As for the value in $[92, 255]$, the BV is always 001. Therefore, the ECT for chunk#1 has four BVs, whose values are 000, 001, 101, and 011. The same result can get using the original PCIU algorithm in addition to the BV, 000. However, in the PCIU pre-processing stage, there are 256 BVs in total for chunk#1, and at least one comparison is required to determine the uniqueness of it for each BV. It is obvious that pre-processing performance can be improved since all comparisons are removed in the modified algorithm.

The chunk tables and equivalent class tables generated by the modified algorithm are almost the same as those of the original PCIU algorithm, except the BV, 000 in each ECT, which are not shown in this paper.

The modified pre-processing algorithm proposed in [29] is beneficial for reducing pre-processing time; however, the classification performance has decreased due to excessive partitioning of the rule set. Different from the partition method used in [29], we propose an adaptive rule set partitioning based on small-big field, reducing the number of partitioning, thus reducing the passive effect on classification performance. Besides, we propose to use rules adjustment to solve the issue of the unbalanced distribution of rules in each subset, which can further shorten the pre-processing time.

3.2. **Rule set partitioning based on small-big field.** The concept of small-big field was first proposed by Li and Li in [15] and was also adopted in [1] and [20]. The definition is as follows [15]: A rule R with an n-dimension, written as $R = (F_1, F_2, \ldots, F_n)$. $Diff_i$ represents the range length of field F_i and $T = (T_1, T_2, \ldots, T_n)$ represents a threshold value vector. F_i is called a small field if $Diff_i \leq T_i$; otherwise, it is a big field, which means $Diff_i > T_i$.

In our paper, the definition of small-big field is a little different. A lot of works [8,9,15,20,25,28] point out that the IP address is the most distinguishing field for packet classification. Thus, we select the IP address field to partition rule sets, just as follows: For a rule R with n-dimension, its IP address can be divided into N 8-bit chunks, written as $(C_1, C_2, \ldots, C_N)$, where N is the sum of the length of source IP and destination IP in bytes. Supposed that $|C_i|$ is the range length of each byte of IP address and $T = (T_1, T_2, \ldots, T_n)$ is a threshold value vector. Then, we have the following definitions:

- The source or destination IP address field is a small field: the range length $|C_i| \leq T_i$, $\forall i \in \{1, 2, \ldots, N_m\}$;
- The source or destination IP address field is a big field: $\exists i \in \{1, 2, \ldots, N_m\}$, the range length $|C_i| > T_i$.

Here, N_m is the length of source IP or destination IP in bytes. Based on the above definitions, the rule set can be partitioned into four subsets, (Src_{big}, Dst_{big}), (Src_{small}, Dst_{big}), (Src_{big}, Dst_{small}), and $(Src_{small}, Dst_{small})$.

For example, if we set $T = (24, 24, 24, 24, 24, 24, 24, 24)$, the third rule in Table 1 will fall into the (Src_{big}, Dst_{small}). The range length of chunk#0 of the source IP field is 256, which is larger than 24; thus the source IP field is a big field. For the destination field, since the length of each chunk of the destination IP field is lower than 24, the destination field is a small field.

For each subset, the modified pre-processing algorithm, proposed in 3.1, is applied. By the rule set partitioning, we reduce the number of rules that need to be processed at one time and thus reduces the length of BVs; as a result, the memory consumption is reduced.

3.3. **Rule set partitioning with rules adjustment.** With partitioning, the original rule set is divided into four subsets. We use three rule sets, namely ACL_10k, FW_10k, IPC_10k, to show the result of the division, which is shown in Figure 4. As we can see, partitioning can indeed reduce the number of rules that need to be processed at one time; however, the distribution of rules among the four subsets is extremely unbalanced. Although the parallelism offered by modern hardware can be leveraged to speed up the pre-processing stage, the pre-processing time is still dependent on the rule set, which has the maximum number of rules among these four subsets.

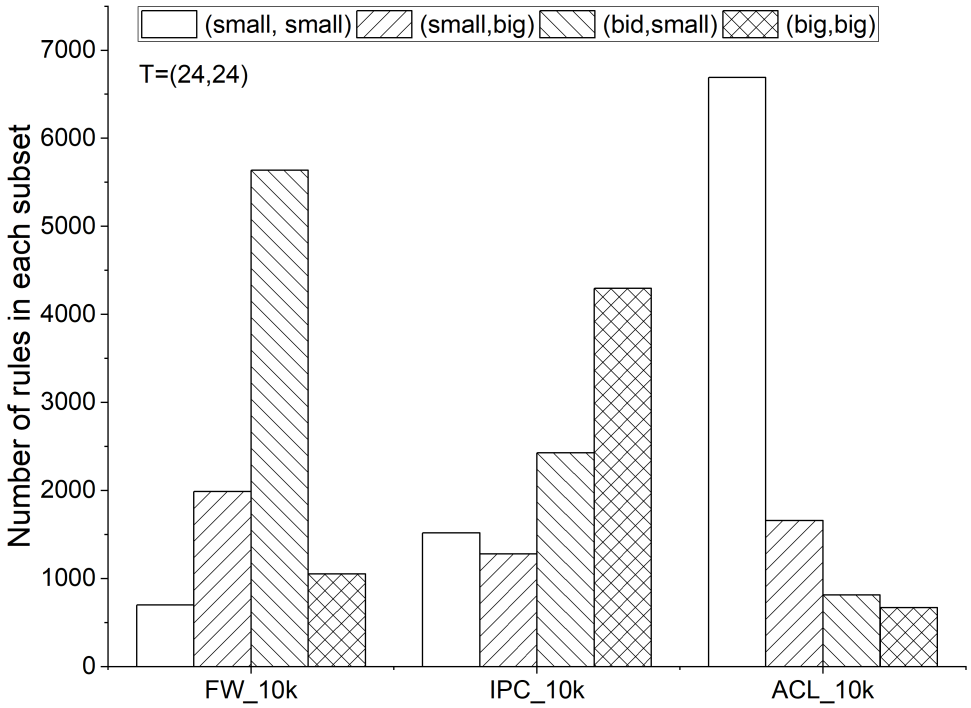

FIGURE 4. Number of rules in each subset

Here, we also test different threshold value vectors to see if there exists one best choice of the threshold vector, which can make the number of rules in each subset as balanced as possible. The results are shown in Table 4.

From Table 4, we can see that for the same rule set, threshold vectors have little effect on the number of rules in each subset. What is more, for different rule sets, the same threshold vector leads to a different number of rules in each subset, which means this partitioning based on small-big field does depend on the characteristics of the rule sets. Since different rule sets have different characteristics, no matter how the threshold vector is selected, all rule sets cannot be considered. In this paper, we use the threshold vector $(24, 24)$.

Based on the above observation and analysis, we think that finding a suitable threshold vector is not an easy job. Therefore, we propose that the adjustment of the rule set is to keep the number of rules in each subset as balanced as possible to achieve the purpose of reducing the pre-processing time. The adjustment used in this paper is very easy when the number of rules in a subset exceeds the $\lceil \frac{n_{rules}}{4} \rceil$, and we just add this rule to the subset that has the minimum rules.

TABLE 4. The number of rules in each subset for different threshold vectors

Rule sets	Threshold vector	Number of rules in each subset			
		$(Src_{small}, Dst_{small})$	(Src_{small}, Dst_{big})	(Src_{big}, Dst_{small})	(Src_{big}, Dst_{big})
FW_10k (9380)	16,16	700	1990	5638	1052
	24, 24	700	1990	5638	1052
	32, 32	763	2053	5623	941
	64, 64	768	2134	5640	838
	128, 128	768	2166	5641	805
ACL_10k (9524)	16, 16	1520	1280	2428	4296
	24, 24	1520	1280	2428	4296
	32, 32	1690	1320	2536	3978
	64, 64	2093	1283	2648	3500
	128, 128	2883	1326	2787	2528
IPC_10k (9832)	16, 16	6689	1659	814	670
	24, 24	6689	1659	814	670
	32, 32	6689	1659	856	628
	64, 64	6705	1643	871	613
	128, 128	6925	1497	827	583

3.4. **Adaptive rule set partitioning.** Based on the analysis of memory consumption of the original PCIU algorithm, as shown in Equation (4) and Figure 2, we found that when the number of rules in the rule set is not large, the memory consumption of chunk tables occupies the majority. If we perform rule set partitioning for small rule sets, the number of chunk tables will increase by 3 times. At the time, memory consumption increases instead. Figure 5 shows the results that are completely consistent with our analysis. Obviously, it is not a wise choice.

An adaptive version of the rule set partitioning is needed to avoid the mentioned problem. As stated above, we need a threshold to guide us to determine when we need to do a partition on the rule set. From Figure 5, we also find that there is an intersection

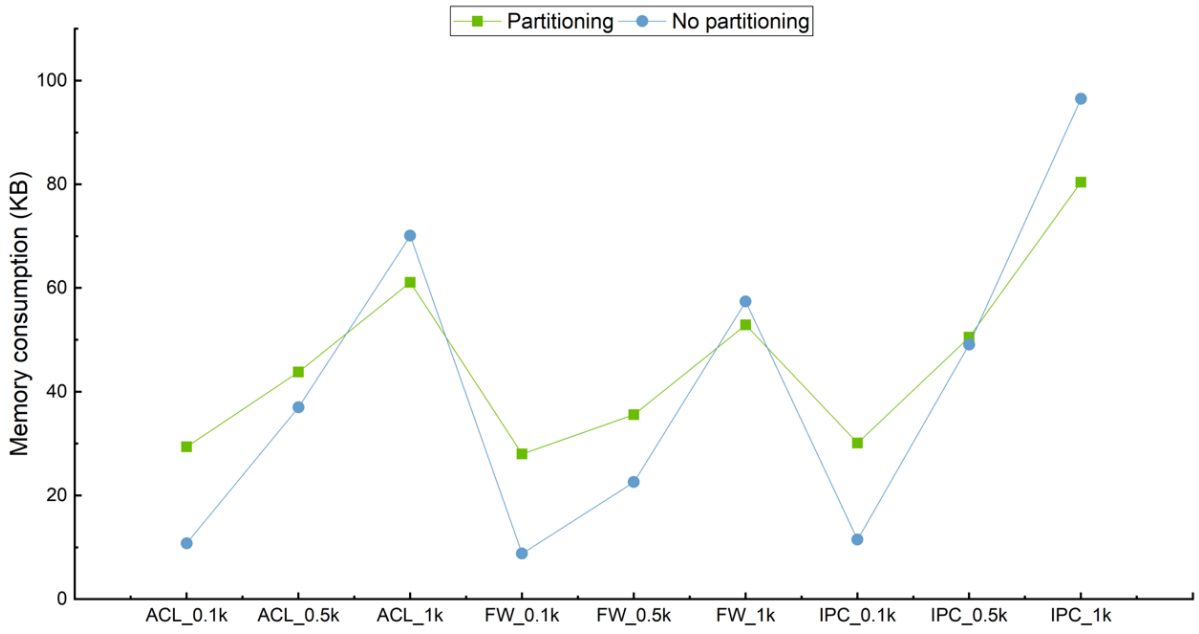

FIGURE 5. Memory consumption of various small rule sets with partitioning and without partitioning

point of memory consumption, from where memory consumption of partitioning is lower than that of the no-partitioning rule set. This intersection point is the threshold value we want. Unfortunately, the value is varying with rule sets. How to choose a precise threshold value is also interesting but beyond the scope of this article. The choice of the threshold we use follows our previous work [29], whose value is 1000.

4. **Experimental Results.** We evaluate our proposed packet classification algorithm with the original PCIU algorithm in terms of memory consumption and pre-processing time in this section. Besides, we also compare our method with CutSplit [1], HybridCuts [15], and RFC [25]. We use ClassBench [30] to generate rule sets, the size of whom are from 0.1k to 10k. The seeds used to generate rule sets are acl1_seed, fw1_seed, and ipc1_seed. All tests are performed on an Intel server with two quad-core Intel Xeon E5-2609 processors running at 2.40GHz with 16G of DDR3 RAM.

4.1. **Memory consumption.** Since the adaptive characteristic of our proposed method, we show the memory consumption in Table 5 and Table 6 separately, where Table 5 shows the memory consumption for larger rule sets while Table 6 shows that for small ones.

Table 5. Memory consumption for larger rule sets (KB)

Rule sets	PCIU	P_PCIU	Proposed	CutSplit	HybridCuts
FW_5k	1250.18	705.26	896.02	221.68	142.37
FW_10k	2463.58	1529.03	1948.26	652.86	299.71
ACL_5k	525.56	367.41	339.97	180.64	278.45
ACL_10k	2091.97	1557.19	1502.84	403.50	555.09
IPC_5k	758.81	507.04	550.29	76.95	270.13
IPC_10k	1833.14	1289.21	1464.22	238.65	466.09
Average reduction		33.4%	26.8%		

Table 6. Memory consumption for smaller rule sets (KB)

Rule sets	PCIU	P_PCIU	Proposed	CutSplit	HybridCuts	RFC
FW_0.1k	8.73	27.98	8.88	1.83	1.58	862.93
FW_1k	57.37	48.43	57.83	15.17	44.78	1061.04
ACL_0.1k	10.72	29.88	10.99	1.57	0.85	803.74
ACL_1k	70.06	67.12	70.58	38.37	27.24	60922.57
IPC_0.1k	11.47	29.75	11.79	4.62	0.97	1037.08
IPC_1k	96.41	77.35	97.12	22.85	33.68	96174.92
Average increment		86.6%	1.6%			

As we can see, our modified method with rule set partitioning can achieve an average memory reduction of 33.4% with a maximum of 43.6% for large rule sets. The purpose of rules adjustment is to further shorten the pre-processing time, but we also show the memory consumption for it. Compared with methods based on decision-tree, such as CutSplit and HybridCuts, the memory consumption of our proposed method is still very large since lots of BVs need to be stored. As for RFC, memory consumption is too large, which is not shown here.

From the analysis in Section 2.2, the reduction in the number of rules leads to the reduction of BVs' length, thus reducing memory consumption. The experimental results are consistent with our analysis, which shows the correctness of our analysis of the problem of the original PCIU algorithm.

As for the memory consumption for small rule sets, the experimental results are shown in Table 6. If partitioning is done for small rule sets, the memory consumption increases instead, just as the third column shows. There is an average increment of 86.6%. The results are consistent with the analysis in 3.4, which again demonstrates that an adaptive rule set partitioning is necessary.

The fourth column in Table 6 shows the result of the adaptive rule set partitioning based on small-big field. As we can see, it is negligible that there is a slight increase in memory consumption due to the default BV-000 in each ECT table, which shows the effectiveness of the adaptive rule set partitioning. Compared with CutSplit and HybridCuts, the result is consistent with that for large rule sets. Here, we also show the memory consumption of RFC, which tells us that even for small rule sets, memory consumption is prohibitively high.

4.2. **Pre-processing time.** As shown in Figure 6, the pre-processing time of our proposed method is lower than the original PCIU, CutSplit, and HybridCuts. For example, for the rule set FW_10k, our method only consumes 51.9 milliseconds (ms) to preprocess, while PCIU, CutSplit, and HybridCuts consume 355.9 ms, 1800.1 ms, and 3407 ms respectively. Compared with the PCIU, our method has a 4.1 times improvement in terms of pre-processing time on average for all rule sets. Compared with CutSplit and HybridCuts, the average improvement is 240.8 times and 376.6 times respectively. As for RFC, for instance, the pre-processing time is about 235084 ms for the ACL_1k rule set, which is extremely high so that we do not show it in Figure 6.

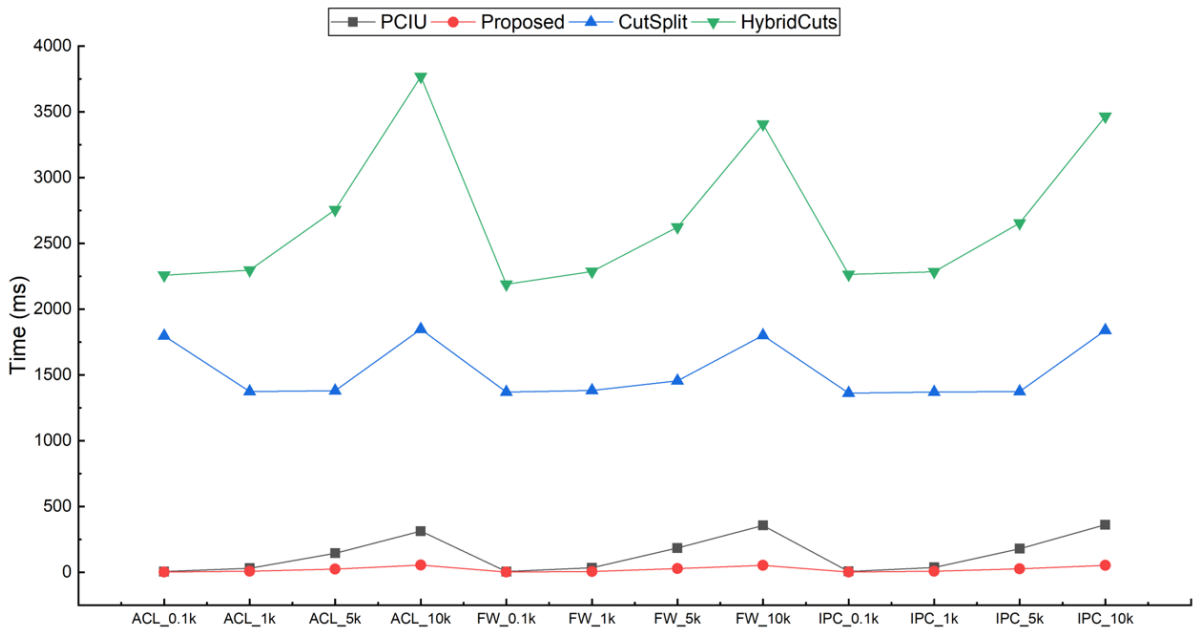

FIGURE 6. Pre-processing time

As mentioned in Section 3, using boundary-value based rule traversal, we remove unnecessary comparisons to speed up the pre-processing stage. What is more, the rule set partitioning based on small-big field reduces the number of rules that need to be processed at one time, thus shortening the pre-processing time. Lastly, we use rule adjustment to further accelerate the pre-processing stage. Using these improvements, the pre-processing time is shortened sharply, just as Figure 6 shows. The experimental results again demonstrate the correctness of the problem analysis and the effectiveness of the optimization measures we proposed.

4.3. **Memory access and throughput.** Figure 7 shows the average memory access of our method as well as PCIU, CutSplit and RFC. For the original PCIU and RFC algorithm, the procedure of classification is some table lookups; therefore, the memory access for each classification is fixed, whose values are both 13. As for our proposed method, the sequent access of each subset is applied, the process will end if it finds the best-matched rule. Compared with the other three methods, our proposed one needs more memory accesses for each classification, which would harm the classification performance. For all the rule sets, the average memory access is 23, 16, 13, and 13 for our method, CutSplit, PCIU, and RFC respectively.

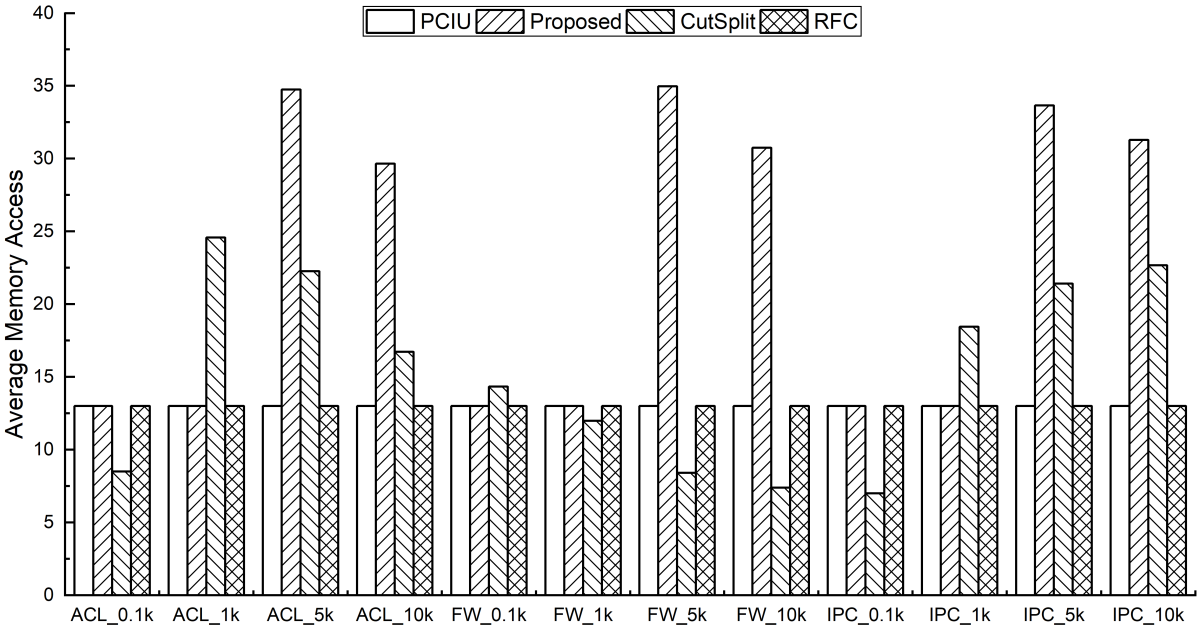

FIGURE 7. Average memory access

We also test the classification throughput to gain more insights into the influence of partitioning on classification speed. The results are shown in Figure 8.

Due to the Anding operation of the classification procedure, the classification speed of the PCIU is far behind CutSplit, but better than the one we proposed. For all rule sets tested, the throughput is 1.52 Mpps (Million packets per second), 1.67 Mpps, and 4.67 Mpps for our proposed method, PCIU, and CutSplit respectively.

In fact, the procedure of classification can be implemented in parallel to remove the passive effect of partitioning on classification speed [15,20], which is left as our future work.

4.4. **Effectiveness of different improvements used in this paper.** To obtain more insights, we verify the effectiveness of the three improvements used in our proposed method. The results are shown in Figure 9. B_PCIU means the improved method with only boundary value-based rule traversal; BP_PCIU means the improved method using the boundary value-based rule traversal and rule set partitioning based on small-big field; BPA_PCIU means the method applying these three improvements. For example, for the rule set ACL_10k, the pre-processing time is 312.3 ms for the original PCIU algorithm while the time is 190.2 ms, 138.7 ms, and 54.2 ms for B_PCIU, BP_PCIU, and BPA_PCIU respectively. Compared with the original PCIU algorithm, the pre-processing time is reduced by 46.6%, 57.5%, and 76.7% for B_PCIU, BP_PCIU, and BPA_PCIU

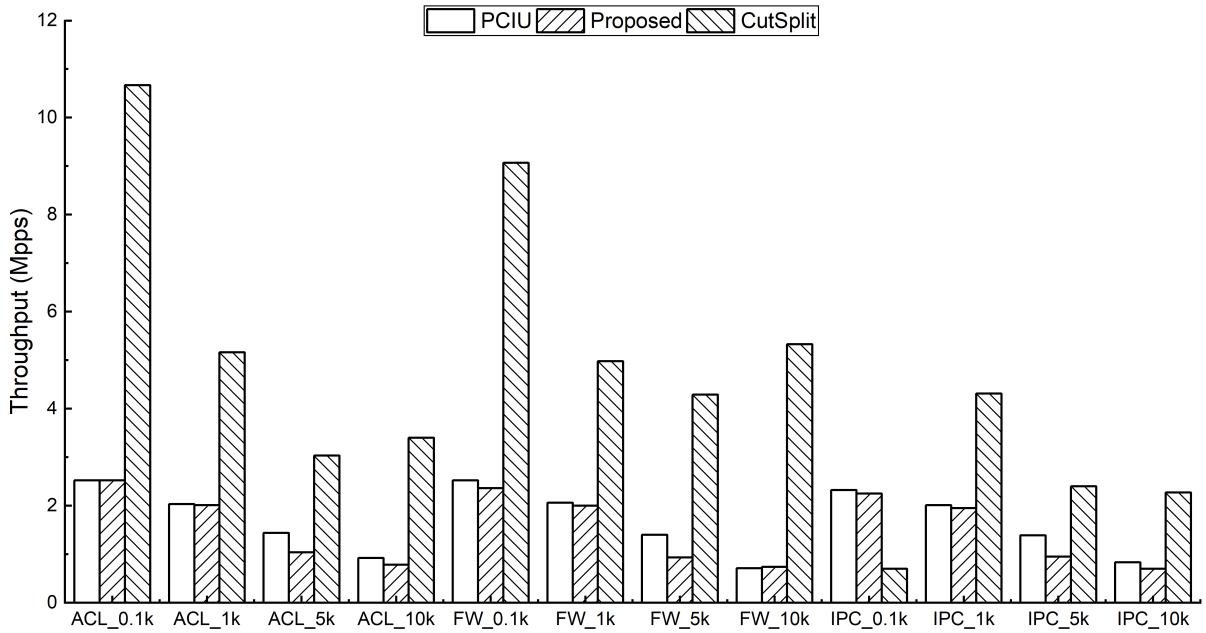

FIGURE 8. Classification throughput

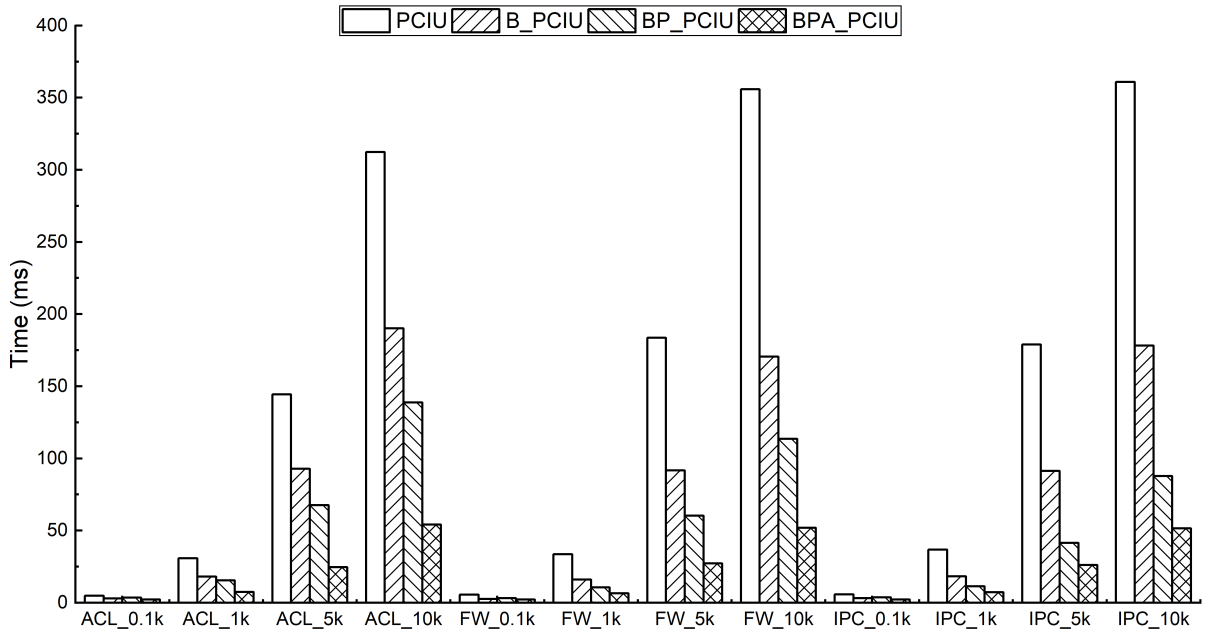

FIGURE 9. Pre-processing time of different improvements

respectively for all rule sets tested. The results show that applying different improvements, the pre-processing time has different degrees of reduction, demonstrating each of the improvements proposed in this paper is effective.

5. **Conclusions and Discussions.** In this paper, we propose an improved packet classification algorithm to alleviate the problems of the increased memory consumption and pre-processing time as the number of rules increases. Some improvements such as boundary-based rule traversal, the adaptive rule set partitioning based on small-big field, and rules adjustment are applied to reducing the pre-processing time as well as memory consumption. Compared with the original PCIU algorithm, experimental results show that the

proposed algorithm achieves an average memory reduction of 33.4% with a maximum of 43.6% for large rule sets as well as 4.1 times improvement in terms of pre-processing time on average for all rule sets. Compared with CutSplit and HybridCuts, the average improvement in terms of pre-processing time is 240.8 times and 376.6 times respectively, but the memory consumption of our method is still higher than CutSplit and HybridCuts, which should be improved in our future work. Besides, the classification performance in parallel should also be evaluated and some improvements in the classification procedure should also be considered to improve the classification speed to handle the ever-increasing network traffic. Lastly, extending our method to solve multi-field packet classification problems will also be an interesting direction.

Acknowledgment. This work is funded by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No. XDC02070100). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] W. Li, X. Li, H. Li and G. Xie, CutSplit: A decision-tree combining cutting and splitting for scalable packet classification, *Proc. of IEEE INFOCOM*, pp.2645-2653, 2018.

[2] L. N. Jing, Z. X. Ye and X. Chen, Packet classification algorithms based on decision tree, *Journal of Network New Media*, vol.7, no.2, pp.1-11, 2018.

[3] D. E. Taylor, Survey and taxonomy of packet classification techniques, *ACM Comput. Surv.*, vol.37, no.3, pp.238-275, 2005.

[4] W. Jiang and V. K. Prasanna, Scalable packet classification on FPGA, *IEEE Trans. Very Large Scale Integr. Syst.*, vol.20, no.9, pp.1668-1680, 2012.

[5] Y. R. Qu and V. K. Prasanna, High-performance and dynamically updatable packet classification engine on FPGA, *IEEE Trans. Parallel Distrib. Syst.*, 2016.

[6] K. Lakshminarayanan, A. Rangarajan and S. Venkatachary, Algorithms for advanced packet classification with ternary CAMs, *Comput. Commun. Rev.*, vol.35, no.4, pp.193-204, 2005.

[7] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan and E. Porat, On finding an optimal TCAM encoding scheme for packet classification, *Proc. of IEEE INFOCOM*, pp.2049-2057, 2013.

[8] X. Li, Y. Lin and W. Li, GreenTCAM: A memory- and energy-efficient TCAM-based packet classification, *2016 International Conference on Computing, Networking and Communications (ICNC 2016)*, 2016.

[9] Y. Ma and S. Banerjee, A smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification, *Comput. Commun. Rev.*, vol.42, no.4, pp.335-346, 2012.

[10] Y. Qi, L. Xu, B. Yang, Y. Xue and J. Li, Packet classification algorithms: From theory to practice, *Proc. of IEEE INFOCOM*, pp.648-656, 2009.

[11] P. Gupta and N. Mckeown, Packet classification using hierarchical intelligent cuttings, *Proc. of 2003 Conf. Appl. Technol. Archit. Protoc. Comput. Commun.*, pp.213-224, 2003.

[12] S. Singh, F. Baboescu, G. Varghese and J. Wang, Packet classification using multidimensional cutting, *Comput. Commun. Rev.*, vol.33, no.4, pp.213-224, 2003.

[13] S. Yingchareonthawornchai, J. Daly, A. X. Liu and E. Torng, A sorted partitioning approach to high-speed and fast-update OpenFlow classification, *Proc. of Int. Conf. Netw. Protoc. (ICNP)*, 2016.

[14] B. Vamanan, G. Voskuilen and T. N. Vijaykumar, EffiCuts: Optimizing packet classification for memory and throughput, *Comput. Commun. Rev.*, vol.40, no.4, pp.207-218, 2010.

[15] W. Li and X. Li, HybridCuts: A scheme combining decomposition and cutting for packet classification, *Proc. of IEEE the 21st Annu. Symp. High-Performance Interconnects (HOTI2013)*, pp.41-48, 2013.

[16] J. Fong, X. Wang, Y. Qi, J. Li and W. Jiang, ParaSplit: A scalable architecture on FPGA for Terabit packet classification, *Proc. of 2012 IEEE the 20th Annu. Symp. High-Performance Interconnects (HOTI2012)*, pp.1-8, 2012.

[17] C. Zhang, S. Chai, L. Cui and B. Zhang, Road condition recognition in self-driving cars based on classification and regression tree, *ICIC Express Letters, Part B: Applications*, vol.10, no.12, pp.1115-1122, 2019.

[18] V. Srinivasan, S. Suri and G. Varghese, Packet classification using tuple space search, *Comput. Commun. Rev.*, vol.29, no.4, pp.135-146, 1999.
[19] J. Daly et al., TupleMerge: Fast software packet processing for online packet classification, *IEEE/ACM Trans. Networking*, vol.27, no.4, pp.1417-1431, 2019.
[20] W. Li, D. Li, Y. Bai, W. Le and H. Li, Memory-efficient recursive scheme for multifield packet classification, *IET Commun.*, vol.13, no.9, pp.1319-1325, 2019.
[21] T. V. Lakshman and D. Stiliadis, High-speed policy-based packet forwarding using efficient multi-dimensional range matching, *Comput. Commun. Rev.*, vol.28, no.4, pp.203-214, 1998.
[22] U. Trivedi and M. L. Jangir, EQC16: An optimized packet classification algorithm for large rule-sets, *Proc. of 2014 Int. Conf. Adv. Comput. Commun. Informatics (ICACCI2014)*, pp.112-119, 2014.
[23] F. Baboescu and G. Varghese, Scalable packet classification, *IEEE/ACM Trans. Networking*, vol.13, no.1, pp.2-14, 2005.
[24] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, Fast and scalable layer four switching, *Comput. Commun. Rev.*, vol.28, no.4, pp.191-202, 1998.
[25] P. Gupta and N. McKeown, Packet classification on multiple fields, *Comput. Commun. Rev.*, vol.29, no.4, pp.147-158, 1999.
[26] W. Pak and S. Bahk, FRFC: Fast table building algorithm for recursive flow classification, *IEEE Commun. Lett.*, vol.12, no.3, pp.1082-1084, 2010.
[27] O. Ahmed, S. Areibi and D. Fayek, PCIU: An efficient packet classification algorithm with an incremental update capability, *Proc. of 2010 Int. Symp. Perform. Eval. Comput. Telecommun. Syst. (SPECTS'2010)*, pp.81-88, 2010.
[28] P. He, G. Xie, K. Salamatian and L. Mathy, Meta-algorithms for software-based packet classification, *Proc. of International Conference on Network Protocols (ICNP)*, 2014.
[29] C. Li, X. Zeng, L. Song and Y. Jiang, A fast, smart packet classification algorithm based on decomposition, *J. Control Sci. Eng.*, 2020.
[30] D. E. Taylor and J. S. Turner, ClassBench: A packet classification benchmark, *IEEE/ACM Trans. Networking*, vol.15, no.3, pp.499-511, 2007.