Research Article

A Fast, Smart Packet Classification Algorithm Based on Decomposition

Chuanhong Li ^{1,2}, Xuewen Zeng^{1,2}, Lei Song ^{1,2} and Yan Jiang^{1,2}

¹National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China

²School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

Correspondence should be addressed to Chuanhong Li; lich@dsp.ac.cn and Lei Song; songl@dsp.ac.cn

Received 1 August 2020; Accepted 17 September 2020; Published 15 October 2020

Academic Editor: Daniel Morinigo-Sotelo

Copyright © 2020 Chuanhong Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Packet classification algorithms have been the focus of research for the last few years, due to the vital role they play in various services based on packet forwarding. However, as the number of rules in the rule set increases, not only the preprocessing time but also the memory consumption is increasing greatly. In this paper, we first model and analyze the above issue in depth. Then, a fast, smart packet classification algorithm based on decomposition is proposed. By boundary-based rule traversal and smart rule set partitioning, both the preprocessing time and memory consumption are reduced dramatically. Experimental results show that the preprocessing time of our method achieves 8.8-time improvement at maximum compared with the PCIU and achieves about 31.5-time improvement on average compared with CutSplit for large rule sets. Meanwhile, the memory overhead is reduced by 40% at maximum and 27.5% on average compared with the PCIU.

1. Introduction

Nowadays, more and more network services based on packet forwarding, such as policy routing, firewall, network billing and Quality of Service (QoS), are based on packet classification. For each incoming packet, packet classification is to find a matching rule from a set of rules, called a packet classifier, and decide on an action regarding the packet, such as forwarding or dropping, which is described by the corresponding rule [1]. Each rule in the packet classifier consists of a tuple of field values (exact value, prefix, or range) and an action to be taken in case of matching [2]. However, with the explosive growth of network traffic, the performance requirements for packet classification algorithms are getting higher and higher, motivating researchers to do a lot of research.

According to [3], packet classifications can be categorized broadly into four kinds: (1) exhaustive search, (2) decision tree, (3) decomposition, (4) tuple space. Among them, decomposition-based algorithms are considered very

promising since the rich features of the modern hardware [4], such as parallelism, can be used to speed up the lookup performance. In addition, they are not dependent on the characteristics of the rules, making them more suitable for packet classification to satisfy various services requirements. However, the increased memory consumption and preprocessing time caused by the increase in the number of rules in the rule sets are pending [5].

To alleviate the above issues, in this paper, a fast, smart packet classification algorithm based on decomposition is proposed to reduce the memory requirements; at the same time, the preprocessing time is also reduced drastically. The proposed method can be considered as an improved version of the PCIU algorithm [6]. The “fast” feature of our proposed method is reflected in the preprocessing time. We make use of the boundary value of rules to accelerate the preprocessing. What is more, by simply dividing the rule sets into multiple sub-rule sets, parallel preprocessing on these sub-rule sets is performed, which not only further reduces preprocessing time greatly, but also reduces the memory

overhead. The “smart” feature of our method is reflected in the division of the rule sets. Instead of dividing any size rule sets, only when the number of rules in the rule set reaches a threshold, will we divide the rule set into multiple sub-rule sets to speed up the preprocessing, since when the number of rules is small, the proposed method is fast enough. For small rule sets, if the division is performed, the reduction in classification performance may be more significant than the reduction in preprocessing time [2, 7], which shows the importance of the smart characteristic of the proposed method. The main contributions of our work include the following:

- (i) We model and analyze the reason why the memory overhead and the preprocessing time increase for large rule sets
- (ii) Boundary-based rule traversal is proposed to accelerate the preprocessing stage
- (iii) Smart rule set partitioning is applied to further shorten the preprocessing time as well as reduce memory consumption
- (iv) Comparative experiments are done to evaluate our proposed method

The rest of the paper is organized as follows. Section 2 describes some related work. After that, we provide some introductions to the PCIU algorithm in Section 3. The proposed packet classification algorithm is present in Section 4. In Section 5, the experimental results are given. At last, we conclude our paper with discussion of future work in Section 6.

2. Related Work

As the network traffic increases exponentially, packet classification has gradually become the bottleneck of advanced forwarding, attracting abundant research attentions in the last few years [2]. In this section, we will review the literature on packet classification algorithms according to the classification criteria in [3] and briefly summarize their advantages and disadvantages.

Packet classification algorithms based on exhaustive search usually depend on specialized hardware support to achieve excellent performance, such as FPGA [8–10], ASIC, and TCAM [11, 12]. However, the expensive price of the dedicated hardware, the longer development time, and the high energy consumption limit their scalability.

Decision-tree-based packet classification algorithms are considered as one of the promising approaches as they can achieve high classification performance by parallel processing and be applied to rules with more fields [2, 13]. According to the characteristics of the rule set, one or more trees are built to cover the whole rule set. There are two different methods to deal with the rule sets to construct the decision tree. One is equal-size cutting, and the other is equal-dense splitting. For methods based on cutting, HiCuts [14] and HyperCuts [15] are both excellent instances. Both of them divide the searching space into multiple equal-size subspaces using local optimizations until the number of

rules in each subspace is less than a threshold, called *binth*, which is defined in advance. The difference between HiCuts and HyperCuts is that the former allows cutting on only one field per step while the latter allows cutting on many fields at one time. Although HyperCuts adopts several optimizations, such as node merging, rule overlap, region compaction, and pushing common rule subsets upwards, both of them are suffering from the rule replication problem, especially for large rule sets. To alleviate it, EffiCuts [16] divides the rule sets into multiple subsets and builds a decision tree using HyperCuts for each subset separately based on the observation on real-life rules. However, this partitioning method will lead to plenty of extra memory access, resulting in degraded classification performance. HybridCuts [7] partitions rules on single rule field instead of on all fields, which greatly reduces the number of subsets, thus reducing the frequency of memory access. Compared with cutting-based schemes, methods based on splitting divide the search space into many equal-dense subsets. “Equal-dense” means that there is almost the same number of rules in each subset. HyperSplit [17] is a popular splitting method, which splits the search space into two equal-dense subspaces to bound the worst-case search performance. As the number of rules increases, the memory consumption blows up. As the improved version of HyperSplit, ParaSplit [18] uses a new partitioning algorithm to reduce the complexity of the rule set; as a result, the memory consumption is reduced.

As the state-of-the-art decision-tree-based packet classification algorithm, CutSplit [2] combines the benefits of cutting and splitting to boost up the performance of packet classification. However, for different rule sets, its performance varies widely, which is a common problem faced by all decision-tree-based algorithms apart from rule replication.

Decomposition-based packet classification algorithms are another well-known method, using the idea of “divide and conquer.” By decomposing the multidimensional 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) [19], an earlier algorithm, uses each field of the rule to match the packet being classified and uses a bitmap as the matching result for each field. Finally, all bitmaps are intersected to get the matching result. As the number of the rules increases, the memory consumption increases drastically since the bitmap length depends on the number of the rules; at the same time, since it uses 8-bit lookup, the total memory access required for BV intersection is very high, which in turn leads to the decrease in search performance [20]. Aggregated bit vector (ABV) [21] is proposed to reduce the memory access by bit aggregation, which improves the classification speed; however, the memory consumption becomes more serious since it needs to store extra information such as the aggregated bit vector. Recursive flow classification (RFC) is introduced in [22]. All the possible CBMs (class bitmaps) of RFC are intersected in the preprocessing stage, and the (intermediate) results are stored in tables, called equivalent class tables (ECTs). Unlike cross-producing [23], in which all the CBMs

are intersected in one stage, RFC uses multistage mapping, and a few (two or three) ECTs from the previous stage are combined to produce the new one in the current stage until there is only one table left in the final stage. The packet classification process is only some table lookups; therefore, it shows an excellent classification performance. However, since all CBMs intersected are in the preprocessing stage, the preprocessing time is very long. It may take several hours to preprocess large rule sets. Meanwhile, memory consumption is also very high due to the reason that extra ECTs need to be stored. Many optimizations have been done in the last few years to reduce not only the memory consumption but also the preprocessing time, such as those in [4, 20]. Due to the inherent complexity of the RFC, it is still difficult to satisfy various requirements. PCIU is a simplified RFC algorithm since it is very similar to the RFC phase 0, apart from the fact that it uses 8-bit chunks while RFC uses 16-bit chunks. Compared with RFC, PCIU sacrifices classification performance to some extent in exchange for a reduction in preprocessing time and memory consumption. However, as the number of rules increases, preprocessing still takes a long time, and memory consumption is still very high. Unlike BV and RFC, the parallel packet classification (P^2C) [5] algorithm proposed by van Lunteren and Engbersen applies a novel encoding scheme of the intermediate results, which greatly reduces the memory overhead. Besides, fast incremental updates are also supported by minimizing the dependencies within the search structures.

As for tuple space [24], according to the prefix length of each field, we divide the rule sets into different partitions and use the hash table to store these rules in the same partition 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 many tables needing to be searched [25].

Among the above work, exhaustive search-based packet classification algorithms usually depend on specialized hardware to achieve high classification performance; however, the expensive price of the dedicated hardware, the longer development time, and the high energy consumption limit their scalability. Decision-tree-based packet classification algorithms have an excellent classification performance for some special rule sets, not for all, which also limits its usability. Decomposition-based packet classification algorithms, such as PCIU, are applied to various rule sets and are more suitable for a variety of services requirements; however, the long preprocessing time and the high memory consumption as the number of rules increases are pending, which motivates us to design a new packet classification algorithm to alleviate these problems.

3. PCIU Introduction and Problem Statements

The proposed fast, smart packet classification algorithm is considered as an improved version of the original PCIU algorithm; therefore, in this section, we first give a detailed description of the PCIU algorithm and then we analyze the

problems of the preprocessing stage and the memory consumption of the original PCIU algorithm.

3.1. PCIU Introduction. For convenience, we use a classic 5-tuple rule set with three rules as an example, shown in Table 1. For each rule, the IP field and protocol field are represented in value/mask format, while the port field is a range format. For instance, the source IP 192.168.8.0/24 has a 24-bit mask which can represent a range with the low part (192.168.8.0) and the high part (192.168.8.255). The IP address field with a 32-bit mask represents an exact value.

In the preprocessing stage, all fields of the rules are converted to a range representation, just as Table 2 shows. Each field is divided into 8-bit chunks; thus, there are 13 chunks for a 5-tuple rule. Table 3 shows all the chunks of the three-rule classifier. Meanwhile, a lookup table of size of 28 is assigned for each chunk. For each value in the lookup table, it will be checked whether it is in the range of the corresponding field of the rules in the rule sets, and the result is expressed with a bit vector (BV), whose length is equal to the number of rules. Every bit in the bit vector is pointing to a rule in the rule sets with a value of 1 if it satisfies the field of the rule; otherwise, the value is set to 0. All the unique bit vectors and corresponding identifiers (id) are stored in a corresponding table, called equivalent class table (ECT) for each chunk. The procedure of the preprocessing stage is described in Figure 1. For more details, please refer to [6].

To have a better understanding of algorithm 1, we use the generation of chunk #2 to describe the preprocessing stage of the PCIU. For each value in the lookup table, we travel the whole rule set to see if it is in the range which the rule represents. Lines 3–12 show the procedure. There are many BVs produced, but only the unique ones are stored in the ECT, as described in lines 13–20. For chunk #2, after the above processing, we have three BVs in the corresponding ECT, which are 001, 101, and 011.

Similar to the preprocessing stage, the classification stage produces values according to the header of the packet to be classified, 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 obtain the matching result.

3.2. Problem Statements. The previous section describes the PCIU algorithm, and now we analyze the problems of the PCIU in depth, including memory consumption and the time spent on the preprocessing stage.

3.2.1. Memory Consumption. The memory consumption of the PCIU algorithm is mainly composed of two parts. One is used to store the lookup tables while the other is used to store ECTs. Let M be the total memory overhead. M_{index} and M_{ect} are the memory consumption of lookup tables and equivalent class tables, respectively. Then, we have the following formula:

$$M = M_{\text{index}} + M_{\text{ect}}. \quad (1)$$

TABLE 1: A three-rule classifier.

No.	P (64 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	0:1024	18:21	0/ff
2	192.168.8.0/24	192.168.8.6/32	0:65535	20:30	17/ff
3	212.83.4.0/16	212.83.4.0/24	0:65535	21:21	0/0

TABLE 2: Range representation of three rules.

No.	P (64 bits)				Port (32 bits)				Protocol (8 bits)	
	Source		Destination		Source		Destination		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	1024	18	21	0	0
2	192.168.8.0	192.168.8.255	192.168.8.6	192.168.8.6	0	65535	20	30	17	17
3	212.83.4.0	212.83.255.255	212.83.4.0	212.83.0.255	0	65535	21	21	0	255

TABLE 3: The range format of the three rules.

Chunk #	0		1		2		3		4		5		6		7		8		9		10		11		12	
No.	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
1	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	255	0	4	18	21	0	0	0	0
2	0	255	8	8	168	168	192	192	6	6	8	8	168	168	192	192	0	255	0	255	20	30	0	0	17	17
3	0	255	4	255	83	83	212	212	0	255	0	0	83	83	212	212	0	255	0	255	21	21	0	0	0	255

```

/* The pre-processing stage */
1 FOR each of the 13 chunks do
2   GN := 0 // group number
3   For I := 0, I < 256.I++
4     VecPos := 1
5     BitVec := 0
6     For ID := 0, ID < numberrule, ID++
7       If I >= Rule [ID].low and
8         I <= Rule [ID].high then
9         BitVec := BitVec or VecPos
10      End if
11      VecPos = VecPos * 2
12    Endfor
13  G := 0
14  While G <= GN and BitVec != Group [G] Do
15    G++
16  Endwhile
17  If G >= GN then
18    Group [GN] := BitVec
19    GN := GN + 1
20  Endif
21 Endfor
22 Endfor

```

FIGURE 1: PCIU preprocessing stage [6].

Since each entry of the lookup table only contains an identifier, pointing to a BV in the ECT, the size of the lookup table can be obtained by using the number of entries

multiplied by the size of the entry, written as m_i . Then, M_{index} can be represented as

$$M_{\text{index}} = \sum_{i=1}^n m_i, \quad (2)$$

where n is the number of chunks.

Each ECT stores all the unique bit vectors and corresponding identifiers for the chunk. Suppose that the size of the bit vector is l_{bv} , and the size of the identifier is l_{id} . Let n_i be the number of unique bit vectors in ECT_i . Then, M_{ect} can be expressed as

$$M_{\text{ect}} = \sum_{i=1}^n n_i \times (l_{\text{bv}} + l_{\text{id}}). \quad (3)$$

As we know, the size of the bit vector is dependent on the number of rules in the rule set; therefore, l_{bv} can be replaced by n_{rules} , where n_{rules} is the number of the rules. Finally, the total memory overhead can be rearranged as

$$M = \sum_{i=1}^n m_i + \sum_{i=1}^n n_i \times (n_{\text{rules}} + l_{\text{id}}). \quad (4)$$

For the 5-tuple packet classification, Figure 2 shows the memory consumption of the PCIU algorithm for different rule sets, which are produced by ClassBench [26], including acl, ipc, and fw with sizes of 0.1 k, 1 k, 5 k, and 10 k rules. As we see, the memory consumption is increasing dramatically as the number of rules in the set increases. As for memory consumption, Figure 3 describes the memory consumption ratio of the index tables and equivalent tables for different rule sets. As the figure shows, with the number of rules

FIGURE 2: Memory consumption of the PCIU algorithm for different rule sets.

FIGURE 3: The memory consumption ratio for different rule sets.

increasing, the memory consumption of ECTs accounts for the majority; for example, for rule sets with 10k rules, the memory consumption ratio exceeds 99.6%. In fact, the first part of (4) is a certain value regardless of the number of rules in the rule set. However, the second part is not only related to

the number of rules but also the number of unique BVs in each ECT. The number of the unique BVs is decided by the characteristics of the rule sets, which is out of control; however, the number of rules in the rule set can be controlled by the division of the rule set, which in turn reduces the size

of BVs, motivating us to design a smart packet classification algorithm according to the number of the rules in the rule set to partition it into multiple subsets.

3.2.2. Preprocessing Stage. As Table 2 shows, the range representation of low 8 bits of the source IP field of the first rule is [0, 255]. Using the preprocessing algorithm provided by PCIU, for each value of the first lookup table, called chunk #0, line 7 and line 8 are both satisfied, and a new BV is produced. However, all these new BVs have the same value, which is 111, since each value in the chunk #0 satisfies all three rules. The BV is generated repeatedly, and additional comparisons are needed to confirm if the BV is unique (lines 13–16 of Figure 2), all of which are unnecessary since they would slow down the preprocessing.

Another observation is that as the number of rules increases, the time of the preprocessing stage is longer, just as Figure 4 shows. The rule sets are produced by ClassBench [26], including acl, ipc, and fw with sizes of 0.1 k, 1 k, 5 k, and 10 k.

Due to either the redundant operations in the preprocessing stage or the increase in the number of rules, the preprocessing time will increase, which inspired us to redesign the preprocessing algorithm to speed up the preprocessing stage.

4. Fast, Smart Packet Classification Algorithm

The above analysis shows us that the redundant operations should be reduced to accelerate the preprocessing stage. Meanwhile, reducing the number of rules that need to be processed at one time is also beneficial for shortening the preprocessing time. Therefore, our proposed packet classification algorithm is optimized based on the above two aspects.

4.1. Boundary-Based Rule Traversal. The idea behind this method is very simple; that is, the unique BV is generated only once to avoid unnecessary comparisons as much as possible. To achieve this goal, a flag is used to indicate whether a BV has changed, whose value is 1 if it has and 0 if not. The improved preprocessing algorithm is shown in Figure 5.

As Figure 5 shows, for each of the ECT, we add a default BV of the value 0, as the first entry, indicating that none of the rules is matched. For each value of the lookup table, the goal of lines 7–14 is traversing the rule sets to produce the corresponding BVs. The flag is set to 1 only when the value is equal to the start value of the range representation of the chunk of the rule. Since the value in the range of the rule is always satisfying the rule, the BV is not changed from the start to the end. When line 11 is true, this 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 operations of lines 15–19 are to add the new BV in the ECT. Compared with the original PCIU algorithm, we use the boundary checking and the flag

FIGURE 4: The preprocessing time for different rule sets with varying size for the PCIU algorithm.

```

/* Boundary-based preprocessing stage*/
1 FOR each chunk do
2   BV := 0
3   GN := 1 //group number
4   ECT[0] := 0
5   Flag := 0
6   For I := 0, I < 256, I++
7     For ID := 1, ID < numberrule, ID++
8       If I == Rule [ID].start
9         Set rule bit ID to 1 in BV
10        Flag := 1
11      Else I == (Rule [ID] . END + 1)
12        Set rule bit ID to 0 in BV
13      Endif_else
14    Endfor
15  If flag == 1
16    ECT [GN] := BV
17    GN := GN + 1
18    Flag := 0
19  Endif
20 Endfor
21 Endfor

```

FIGURE 5: Boundary-based preprocessing stage.

to avoid redundant comparisons to shorten the preprocessing time.

Again, we use chunk #2 to depict the modified PCIU algorithm. Chunk #2 ECT is produced as follows: for value 0, since it is equal to the first rule start value, the corresponding bit in the BV is set to 1, that is, 001. As for the second and third rule in Table 3, both of the start values are larger than 0, so ECT [1] = 001. For a value in the range [1, 82], no new BV is produced. When the value is 83, the BV is changed to 101; thus, ECT [2] = 101. When the value is 84, since 84 = (Rule [3]. END + 1), the BV is changed to 001, which is already in the ECT. When the value is 168, the new BV is 011, which in turn leads to ECT [3] = 011. When the value is 169, the BV is changed to 001. As for a value in [170, 255], the BV is 001. Therefore, the ECT for chunk #2 has four BVs, whose values are

000, 001, 101, and 011. The same result can be obtained using the original PCIU algorithm in addition to the BV whose value is 000. However, in the PCIU preprocessing stage, there are 256 BVs produced, and 259 comparisons are required to determine the uniqueness of them. In our proposed method, all comparisons are removed; hence, it is obvious that the time spent on the preprocessing stage is reduced.

4.2. Partitioning the Rule Sets According to the Number of the Rules. According to (4), as the number of rules increases, the memory overhead also increases. To partition the rule sets into multiple subsets may decrease the memory overhead intuitively, which is a popular technique used by decision-tree-based packet classification algorithms, such as EfficCuts [16] and HybridCuts [7]. Memory-efficient recursive scheme for multifield packet classification, which is an improved RFC algorithm, introduced in [4], also applies rule set partitioning. However, the authors divide the rule sets into four subsets according to the IP fields, which is related to the characteristics of the rule sets.

Different from the above methods, our partition method is very simple and smart. Based on the observation of the preprocessing time and the memory consumption of the original PCIU algorithm, just as Table 4 and Figure 3 show, we find that with different partition threshold, the preprocessing time and memory consumption differ greatly. When it is set to 2000, the time of the preprocessing time is doubled compared with the case where the threshold value is 1000; at the same time, the memory consumption is also increasing. However, when the threshold is set to 500, the preprocessing time or memory consumption is reduced a little compared with the case where the threshold is 1000. Meanwhile, with partitions increasing, the classification performance is also reduced regardless of parallel classification being used [2, 7]. Therefore, we use the value 1000 as the threshold. When the number of rules is larger than the threshold, we divide the rule sets into multiple subsets with the rule number being 1000 and the number of rules of the last subset is usually less than 1000. When the number of the rule set is less than 1000, no partitioning is done, since for small rule sets, if division is performed, the reduction in classification performance may be more significant than that in preprocessing time, which shows the importance of the smart characteristic of the proposed method.

In addition to the observation on the real rule set, the threshold value can be selected by the following analysis. Supposed that the time spent on handling a rule set is t , the number of rules in the rule set is n , and the memory overhead to store the rule set is m . For each threshold, there is a weight, written as w , which indicates the possibility of its selection. The weight is computed as follows:

$$w = p * \left(\frac{t}{n}\right) + q * \left(\frac{m}{n}\right), \quad (5)$$

where $p + q = 1$.

Intuitively, if it takes as little time as possible to process a rule, it will take very little time to process the entire rule set. Memory consumption will have a similar result. We adopt this original method to compute the weight. The parameters

p and q show the proportion of the processing time and memory overhead in calculating the weight, which can be configured. However, for each rule set, if we calculate the above weights online to choose the optimal threshold, the time consumption is obviously unacceptable.

The selection of the threshold not only determines the number of subsets but also affects the classification performance of the algorithm. If the influence of classification performance is considered in the calculation of the threshold weight, the above results will be more accurate, but this part is not considered in this article, leaving it as our future work.

For each subset, the boundary-based preprocessing algorithm is applied. All of the subsets are handled in parallel to shorten the whole preprocessing time.

4.3. Classification Stage. The classification stage is similar to the original PCIU algorithm. If the partitioning is done for large rule sets, the search will be done in all subsets in parallel. Under the assumption that the rules are sorted by priority, eventually, the rule with the highest priority is considered the best matching rule.

5. Experimental Results

The proposed fast, smart packet classification algorithm is evaluated in this section. We compare our method with CutSplit [2], which is the state-of-the-art decision-tree-based packet classification algorithm, and the original PCIU algorithm in terms of memory consumption and preprocessing time. The rule sets we use are generated by ClassBench [26], with sizes from 0.1 k to 10 k. The seeds used to generate the rule sets are `acl1_seed`, `acl2_seed`, `fw1_seed`, `fw2_seed`, `ipc1_seed`, and `ipc2_seed`. All tests are done on an Intel server with two quad-core Intel Xeon E5-2609 processors running at 2.40 GHz with 16 G of DDR3 RAM.

5.1. Memory Consumption. We use six rule sets with sizes from 0.1 k to 10 k, generated by ClassBench [26], to test the memory overhead of the original PCIU algorithm and our proposed method. The result is shown in Table 5. Only when the number of rules in the rule set is larger than 1000, will we partition the rule set into multiple subsets. For small rule sets, no partitioning is done. Although a flag is used to indicate whether the BV has changed for each BV in the ECTs, it is a temporary variable so that no extra memory is needed to store it. Therefore, the memory consumption of our packet classification algorithm is the same as the PCIU for small rule sets, which is not shown in Table 5.

For larger rule sets, whose number of rules exceeds 1000, as we can see, the reduction in memory consumption is very significant, with the maximum approaching 40% and the average reduction being about 27.5%, which demonstrates the effectiveness of the rule set partitioning. From the analysis in section 3, the reduction in the number of rules leads to the reduction of BVs, thus reducing the 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.

TABLE 4: The preprocessing time and memory consumption for different partition numbers in subsets (* means no partitioning is done).

	Partition number	Preprocessing time (ms)	Memory consumption (MB)
fw1_5000	500	23.1	0.71
	1000	25.8	0.92
	2000	52.2	1.07
	*	166.8	1.21
fw1_10000	500	32.9	1.36
	1000	37.2	1.73
	2000	53.7	2.12
	*	322.4	2.46

TABLE 5: Memory consumption for different rule sets with different sizes (kB).

	PCIU	FSPC	Partition (%)
acl1_5k	455.89	304.30	33.3
acl2_5 k	688.95	538.48	21.8
fw1_5 k	1230.89	920.07	25.3
fw2_5 k	1087.82	726.70	33.2
ipc1_5 k	690.86	531.01	23.1
ipc2_5 k	1103.77	793.90	28.1
acl1_10 k	1910.10	1148.14	39.9
acl2_10 k	2472.03	2095.11	15.2
fw1_10 k	2457.71	1730.59	29.6
fw2_10 k	2197.96	1431.24	34.9
ipc1_10 k	1728.16	1290.01	25.4
ipc2_10 k	2196.90	1744.68	20.6
Average reduction			27.5

5.2. Preprocessing Time. In this paper, we accelerate the preprocessing stage in two aspects: one is boundary-based rule traversal, and the other is rule set partitioning. Therefore, in this section, we also evaluate our method with the PCIU and CutSplit through two aspects, where the former is only using boundary-based rule traversal while the latter uses both.

5.2.1. Preprocessing Time of Boundary-Based Rule Traversal. The test results are shown in Figure 6. For all rule sets, the preprocessing time of our proposed method is lower than that of PCIU and CutSplit, except fw2_10 k in Figure 6(b). Compared with PCIU, with only boundary-based rule traversal, the preprocessing time is reduced by 32.6% on average. Compared with CutSplit, our method with only boundary-based rule traversal achieves 11-time improvement in the preprocessing time on average.

By using boundary values and a flag, a large number of unnecessary comparisons are avoided in processing the rule sets, thereby reducing the preprocessing time. Only through this programming technique, the preprocessing time has been greatly improved, just as the experimental results show.

Besides, as we can see, the preprocessing time of CutSplit fluctuates greatly for different rule sets, which is more obvious in Figure 6(b). For the rule set fw2_10 k, the preprocessing time is even lower than that of our method, while the preprocessing time for acl2_5 k and acl2_10 k is 1.3

seconds and 8.7 seconds, respectively, which is prohibitively high. The results also show that the decision-based packet classification algorithms surely depend on the characteristics of the rule sets, which limits their scalability.

5.2.2. Preprocessing Time of Boundary-Based Rule Traversal and Rule Set Partitioning. The preprocessing time of our smart packet classification with rule set partitioning is evaluated in this section. Figures 7(a) and 7(b) give the experiment results. There is no doubt that our proposed method has shorter preprocessing time for all rule sets than PCIU and CutSplit, just as Figure 7 shows.

For large rule sets, whose number of rules exceeds 1000, our method with rule set partitioning achieves about 6.9-time improvement on average compared with the PCIU. The maximum improvement is about 8.8 times for the acl2_10 k rule set shown in Figure 7(b). Compared with CutSplit, even excluding two special cases, in which the preprocessing time is prohibitively high, our proposed method still achieves 6.4-time improvement on average for large rule sets. For all large rule sets we use, the average improvement is about 31.5 times.

For small rule sets, since our smart packet algorithm does no partition them, the reduction in the preprocessing time is determined by the boundary-based rule traversal. Compared with the PCIU, the preprocessing time is reduced by 32.5% on average, while, compared with CutSplit,

FIGURE 6: Comparisons of preprocessing time for different rule sets.

FIGURE 7: Comparisons of preprocessing time for different rule sets.

our method achieves about 18-time improvement on average.

Thanks to the smart rule set partitioning and boundary-based rule traversal, our proposed method has achieved a good performance improvement on various rule sets.

6. Conclusions and Discussion

In this paper, we propose a fast, smart packet classification algorithm based on decomposition to solve the increased

memory consumption and preprocessing time as the number of rules increases. The “fast” feature of our method is reflected in the preprocessing time. By using boundary-based rule traversal and rule set partitioning, not only the preprocessing time but also the memory consumption is reduced greatly. The “smart” feature of our method is reflected in the division of the rule sets. Instead of dividing rule sets of any size, only when the number of rules in the rule set reaches a threshold, will we divide the rule set into multiple sub-rule sets to speed up the preprocessing. The experimental results show that, for the preprocessing time

of large rule sets, our smart packet classification algorithm achieves 8.8-time improvement at maximum and 6.9-time improvement on average compared with the PCIU. Compared with CutSplit, we can achieve 31.5-time improvement on average. For small rule sets, since only boundary-based rule traversal is used to accelerate the preprocessing stage, the preprocessing time of the proposed method is reduced by 32.5% compared with the PCIU, while, compared with CutSplit, it can achieve about 18-time improvement on average. As for memory consumption, for large rule sets, the proposed method reduces the memory overhead by about 40% at maximum and 27.5% on average.

In this paper, the threshold we use is based on the observation of the preprocessing time and memory consumption of the original PCIU algorithm for different rule sets. In fact, the choice of threshold is determined by the trade-off of memory overhead, preprocessing time, and classification speed. There are many complex heuristics that can be used to find a better threshold, which is our future work. In addition, the effect incurred by rule set partitioning on the classification performance should be evaluated in future work.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project no. XDC02070100).

References

- [1] 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.
- [2] W. Li, X. Li, H. Li, and G. Xie, "Cutsplit: a decision-tree combining cutting and splitting for scalable packet classification," in *Proceedings of the-IEEE INFOCOM-2018*, pp. 2645–2653, Honolulu, HI, USA, April 2018.
- [3] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.
- [4] W. Li, D. Li, Y. Bai, W. Le, and H. Li, "Memory-efficient recursive scheme for multi-field packet classification," *IET Communications*, vol. 13, no. 9, pp. 1319–1325, 2019.
- [5] J. Van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 4, pp. 560–571, 2003.
- [6] O. Ahmed, S. Areibi, and D. Fayek, "PCIU: an efficient packet classification algorithm with an incremental update capability," in *Proceedings of the 2010 International Symposium on Performance Evaluation of Computer & Telecommunication Systems SPECTS'10*, pp. 81–88, Ottawa, Canada, July 2010.
- [7] W. Li and X. Li, "Hybridcuts: a scheme combining decomposition and cutting for packet classification," in *Proceedings of the-IEEE 21st Annual Symposium on High-Performance Interconnects, HOTI*, pp. 41–48, Santa Jose, CA, USA, August 2013.
- [8] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1668–1680, 2012.
- [9] M. Irfan, Z. Ullah, and R. C. C. Cheung, "Zi-CAM: a power and resource efficient binary content-addressable memory on FPGAs," *Electron*, vol. 8, no. 5, pp. 1–12, 2019.
- [10] C. Li, T. Li, J. Li, D. Li, H. Yang, and B. Wang, "Memory optimization for bit-vector-based packet classification on FPGA," *Electron*, vol. 8, no. 10, pp. 1–16, 2019.
- [11] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 193–204, 2005.
- [12] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "On finding an optimal TCAM encoding scheme for packet classification," in *Proceedings of the-IEEE INFOCOM*, pp. 2049–2057, Turin, Italy, April 2013.
- [13] S. Zhou, Y. R. Qu, and V. K. Prasanna, "Large-scale packet classification on FPGA," in *Proceedings of the 26th International Conference on Application Specific Systems (ASAP), Architectures and Processors*, pp. 226–233, Toronto, Canada, July 2015.
- [14] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 213–224, Karlsruhe, Germany, August 2003.
- [15] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," *Computer Communication Review*, vol. 33, no. 4, pp. 213–224, 2003.
- [16] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "Effcuts," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 207–218, 2010.
- [17] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: from theory to practice," in *Proceedings of the-IEEE INFOCOM*, pp. 648–656, Rio De Janeiro, Brazil, April 2009.
- [18] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang, "Parasplit: a scalable architecture on FPGA for terabit packet classification," in *Proceedings of the-2012 IEEE 20th Annual Symposium High-Performance Interconnects, HOTI*, pp. 1–8, Santa Clara, CA, USA, August 2012.
- [19] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 203–214, 1998.
- [20] U. Trivedi and M. L. Jangir, "An optimized RFC algorithm with incremental update," in *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI*, pp. 120–127, New Delhi, India, September 2014.
- [21] F. Baboescu and G. Varghese, "Scalable packet classification," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 2–14, 2005.
- [22] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 147–160, 1999.

- [23] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 191–202, 1998.
- [24] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 135–146, 1999.
- [25] J. Daly, V. Bruschi, L. Linguaglossa et al., "TupleMerge: fast software packet processing for online packet classification," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1417–1431, 2019.
- [26] D. E. Taylor and J. S. Turner, "Classbench: a packet classification benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499–511, 2007.