An Observation of Packet Classification: Most Rules are at the Top

Chengjun Jia, Chenglong Li, Yifan Li, Xiaohe Hu, Jun Li
Tsinghua University, Beijing, China
{jcj18, liyifan18}@mails.tsinghua.edu.cn, {lichenglong, hxhe, junl}@tsinghua.edu.cn

***Abstract*—Multi-field packet classification plays a critical role in a modern computer network. Many algorithms are designed, but the analysis of the rule sets is insufficient. In this paper, we analyze the rule sets from the aspect of rule overlapping with the tools from graph theory and network verification. The correctness of some designs is confirmed, such as the partition is necessary. We also propose a patch, PCG, based on our observation that most rules are at the top. PCG can apply to various algorithms to further improve the classification performance. We evaluate PCG on a state-of-art algorithm, CutSplit, and find that PCG introduces about 20% throughput improvement.**

***Index Terms*—Packet Classification, Graph Theory, Measurement**

I. INTRODUCTION

Multi-field packet classification is a classical problem in the research field of computer networking and the fundamental component for the Firewall, Intrusion Detection System (IDS), Gateway, and other network devices. Much research has focused on how to design better algorithms with faster packet classification speed and faster rule update speed. However, the analysis for the rule set characteristics is insufficient, which is the key to the discrimination between packet classification and other classification problems, such as database lookup and point location in computational geometry.

Previous systematic analysis for rules of packet classification appears in the Classbench [20] and Classbench-ng [14], which aim at the rule set synthesis. This analysis mainly focuses on the distribution of one field in the rules, such as prefix length distribution and skew distribution, but the relationship among rules is neglected, which has significant impact on the effectiveness of the classification algorithms. For example, the overlap among rules lead to rule replication when we build a decision tree for the multi-field packet classification. Previous works observe the overlap and order relationship among rules, such as SAX-PAC [8], TCAM-update [7], but they focus on the efficient utilization of TCAM, rather than the inspiration for common algorithms.

In this paper, we analyze the rule set of packet classification from the aspect of overlap. The method is inspired by the tool PSA [23] from network verification. By the analysis, we observe that 1) the interaction among rules is not very complex. The rules can be divided into 10-20 groups and in each group, there is no overlapping among rules. 2) the decision from only one field is quite not enough. Ignoring a few fields could lead to a complexity explosion. These observations confirm that some design choices in previous algorithms are reasonable, such as reducing groups, selecting discrete bits from different fields. Based on the observations, we also propose a patch *PCG* for algorithms that make use of multiple groups. From the experiment, the classification performance is improved by 20% using PCG-augmented CutSplit compared with the original CutSplit.

The paper is organized as follows. §II gives detailed and normalized introduction to the packet classification problem. The related works are narrated in §III. §IV shows our observations by analyzing the rule sets. A patch to improve the algorithms which need to group original rule sets, is proposed and evaluated in §V. Possible future directions are discussed in §VI. §VII concludes the paper.

This work is supported by National Natural Science Foundation (No. 61872212) and Industry-university-research Innovation Fund for Chinese Universities (No. 2021FNA04002).

II. BACKGROUND

Packet classification is to classify network traffic according to the specified packet headers and the predefined rule set. A rule set R consists of the **ordered** rules, $r_1 < r_2 < \ldots < r_n$, and each rule is described by a **cartesian product** of d fields, i.e.

$$r_i = F_1^i \times F_2^i \times \ldots \times F_d^i \tag{1}$$

where F_j^i represents a finite set on the packet j-th header field. F_j^i can be expressed by prefix, range, or exact value, which are all continuous on the integer domain; and it means that one rule r_i can be viewed as a hypercube in the field of $\mathbf{Z}^d$. When a packet with the header $(v_1, v_2, \ldots, v_d)$, is to be classified, the classification engine would find the matched rule with the highest order. As mathematicians say, all matched rules are the set

$$R' = \{r_k | r_k \in R; v_j \in F_j^k, \forall j = 1, 2, \ldots, d\}$$

and the final result is

$$\hat{r} : \forall r_k \in R' \text{ and } r_k \neq \hat{r}, \hat{r} < r_k$$

There is a simple rule set for IPv4 packets in Table I. For example, the rule R_1 wants to match the packets whose source IP belongs to the subnet 175.77.88.155/32, destination IP belongs to 119.106.158.230/32, source port ranges from 0 to 65535, destination port is 6888 and protocol is exactly TCP. If a packet p, with the header (95.105.142.3, 193.4.164.231, 100, 80, TCP), wants to be classified on the rule set, the result

TABLE I: An example of packet classification rule set.

Source IP	Destination IP	Source Port	Destination Port	Protocol	Rule id
175.77.88.155/32	119.106.158.230/32	0:65535	6888	0x06/0xFF (TCP)	R_1
95.105.143.33/32	144.209.187.155/32	0:65535	27400	0x06/0xFF (TCP)	R_2
95.105.142.0/23	193.4.164.231/32	0:65535	0:65535	0x06/0xFF (TCP)	R_3
95.105.143.51/32	204.13.220.0/22	0:65535	0:65535	0x01/0xFF (ICMP)	R_4
95.105.143.6/32	192.206.76.132/32	0:65535	0:65535	0x00/0x00	R_5
0.0.0.0/0	0.0.0.0/0	0:65535	0:65535	0x01/0xFF (ICMP)	R_6
0.0.0.0/0	0.0.0.0/0	0:65535	0:65535	0x00/0x00	R_7

would be R_3 due to the obvious facts: 1) p does not match R_1 or R_2; 2) p matches R_3.

III. Related Work

Packet classification problem has been studied for more than two decades [6]; and there are different research directions including hardware acceleration [3] [4] and system design with cache [15]. In this section, we only focus on the algorithmic works which run on the CPU.

The main idea for the algorithm design is to gradually shrink the rule set (reducing the number of of potential match rules) until there are one or few rules so that the cost of linear search is acceptable. We summarize five main kinds of operations during the procedure and most algorithms can be viewed as the composition of different operations:

1) **Cut**. Cut operation selects several bits from the packet headers as the key to distinguish the rules. For example, HiCuts [5] selects consistent bits from one field, while HyperCuts [18] can select multiple consistent bits from different fields. Further, BitCuts [13] chooses several discrete bits to shrink the rule set.
2) **Hash**. Hash operation can be viewed as a variant *Cut*. The difference is the density of the rules. If N bits are used as the key but the rules only cover a few cases, much less than 2^N, *Cut* would lead to much waste of memory, and *Hash* is a better choice. The most typical approach is the TSS [19] algorithm in OvS. *Hash* is quite powerful for the exact matching, but NeuvoMatch [16] leverages the neural network, which can be viewed as another kind of hash scheme, to perform the range match, opening up a new research direction.
3) **Split**. Split operation shrinks the rule set by comparing the value of a certain field with a specific value. The process is similar to the classic binary search, benefiting for the range matching, and HyperSplit [3] makes use of it to design a better solution.
4) **Partition**. Partition does not directly shrink the rule set, but divides the original rules into K groups and respectively search in each of them. After that, the results from each group are unionized for further processing. It is critical how to group the rules. EffiCuts [21] consider the size of the selected fields in a rule, partitioning rules according to the large range or small range; and RVH [17] considers the aggregation of prefix mask length.
5) **Decomposition**. Decomposition shrinks the rules with different methods and then merges the results. Partition and Decomposition are both 'divide and conquer', but partition uses union while decomposition uses intersection. The most typical design with the method is BitVector [10] for FPGA.

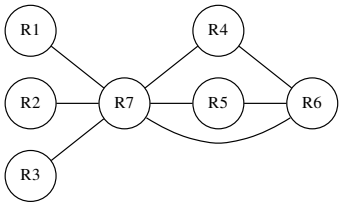


Fig. 1: The mapped graph for rules in Table I on all fields.

Many algorithms combine the methods above to create a new method. For example, ByteCuts [2] and CutSplit [11] combine cut, split and partition operations; NeuroCuts [12] takes advantage of reinforcement learning to synthesize cut and partition.

IV. Observation

In order to reveal the characteristics of the rule sets on packet classification, we use classbench [20] to generate three types of rule sets: ACL (Access Control List), FW (Firewall), and IPC (IP Chain), which are the most typical benchmarks for packet classification. We have three observations here.

A. The overlapping among rules is not complex.

Now that each rule is a hypercube, there is a relationship among rules: whether the intersection is empty. We can view a rule as a node and there is an edge among two nodes if there is overlap between them. Thus, a rule set would be mapped to a graph that uncovers the relationship among rules.

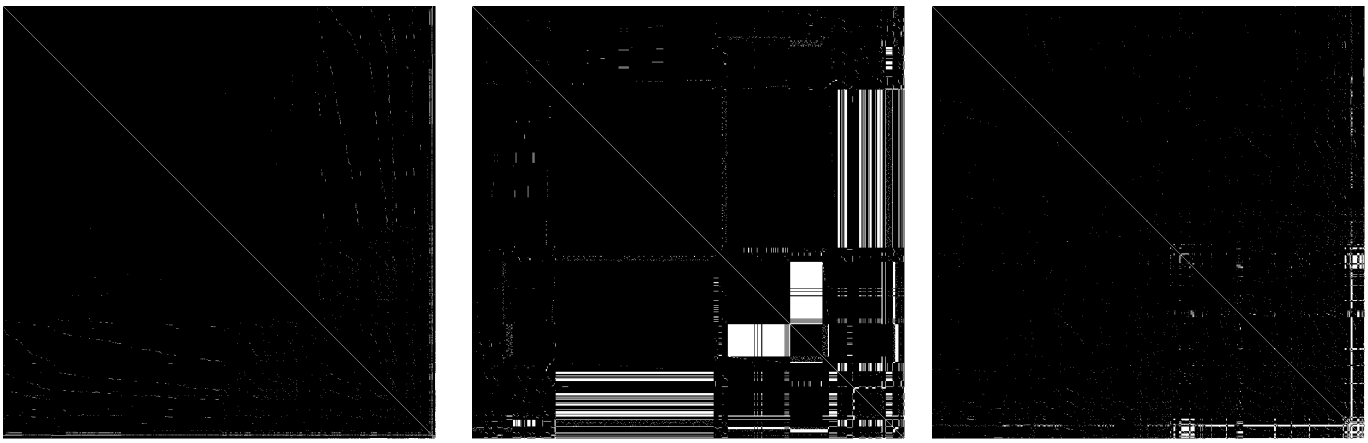
From the viewpoint of mathematics, we can calculate the intersection of two rules, i.e. with the definition of Eq.1,

$$r_i \bigcap r_j = (F_1^i \bigcap F_1^j) \times (F_2^i \bigcap F_2^j) \times \ldots \times (F_d^i \bigcap F_d^j) \quad (2)$$

and the edge $E(i,j)$ in the mapped graph G is defined as

$$E(i,j) = \begin{cases} 1, r_i \bigcap r_j \neq \emptyset \\ 0, r_i \bigcap r_j = \emptyset \end{cases} \quad (3)$$

For example, for the rules R_5 and R_6 in Table I, we could get $R_5 \bigcap R_6$ is not an empty set and thus $E(5,6) = 1$. For example, a packet (95.105.143.6, 192.206.76.132, 1, 1, ICMP)



(a) ACL_1k (b) FW_1k (c) IPC_1k

Fig. 2: The adjacent matrix for mapped graphs of different rule sets. The upper left corner is the origin. If the point at location (i, j) is white, there is overlapping between R_i and R_j.

belongs to R_5 and R_6. After calculating the overlap of any two rules, we have a graph in Fig. 1 for the rules of Table I.

For the three kinds of rule sets from classbench, we get three pictures of the adjacent matrix in Fig. 2. To see the difference clearly, we only show the results with 1k rules and the picture is similar for more rules. It is obvious that FW rule set is the most complex rule set and then IPC, ACL. To be more specific, we summarize some statistics about these graphs in Table II. The FW rule set has the highest density, which is about 10%; while ACL has the lowest, which is less than 1%. FW and IPC rule sets are not sparse[1] while ACL keeps the ratio between edge and node constant (around 2.3). The diversity of the rule sets is a little surprising.

TABLE II: Some characteristics of the mapped graphs

Rule Set	#node	#edge	#edge/#node	density(%)	#color
ACL_1k	917	2,168	2.36	0.95	8
FW_1k	792	22,984	29.02	7.84	12
IPC_1k	938	5,046	5.38	1.58	12
ACL_10k	9603	22,031	2.29	0.09	11
FW_10k	9311	4,170,923	447.96	9.67	8
IPC_10k	9037	198,194	21.93	0.53	18

Although some of the graphs have high density, especially for FW, the graph-coloring results show a high concentration for them. Graph coloring is to dye the nodes in a graph, ensuring that the two nodes of any edge have different colors. Graph coloring is a typical NP-Complete problem and we solve it by the function *greedy_color* of the python library networkx [25], with the strategy of connected_sequential [9]. Even for the FW rule sets, 12 colors are enough, which means that we can divide the rules into 12 groups and any two rules within the same group have no interaction. The characteristic can also be caught on from the Fig. 2: the highlighted area of FW is mainly composed of some separate rectangles. What's

[1]Sparsity means that the number of edges is around the number of nodes.

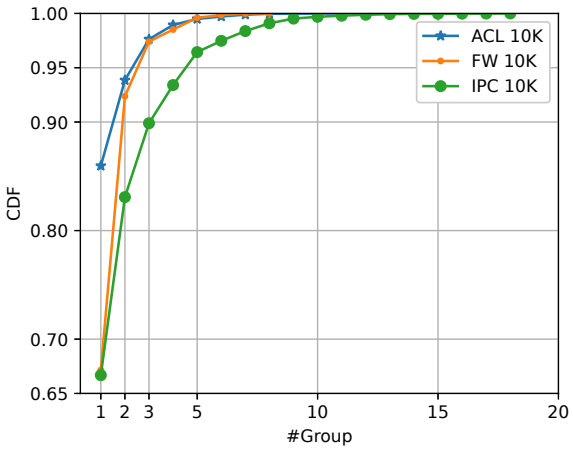


Fig. 3: The cumulative probability .

better, only 3 groups could cover more than 90% rules as depicted in Fig. 3. There are 914 rules left for IPC_10k, 242 for FW_10k, and 228 for ACL_10k when the three largest groups are removed from the rule sets, respectively.

These results inspire us:

1) Partition is a good method for some rule sets. The density of FW is quite large, which inspires us that it is much hard to take all rules in one group into consideration.
2) The partition with too many groups is not necessary. For the most complex rule set, twenty colors are enough for dyeing which means that a packet matches twenty rules at most. Naive TSS [19] could have up to 100 groups, most of which are not necessary and the merging of groups is very reasonable, e.g. TulpeMerge [1]. And TCAM, which takes all rules into consideration at the same time, has much redundancy to evolve into a better hardware design.

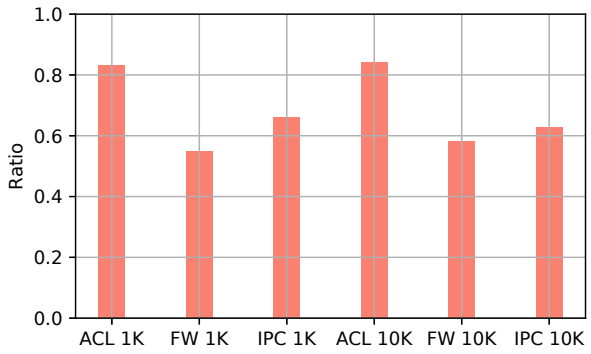


Fig. 4: Ratio of top rules.

3) The long tail after the partition is a critical problem for future algorithm design. For ACL_10k, the left ˜10% rules need 15 colors to dye, which means that the relationship among these rules is complicated. It is valuable to focus on the classification of these rules.

B. Most rules are at the top

Another observation which we could get from Fig. 2 is that the bright area concentrates in the lower region. It means that the most overlap happens between the rules with much lower priority.

We define a new property of one rule: *the top property*, which means that there is no overlap between the rule and other rules with higher priority. In mathematics, a rule r_m is top, meaning

$$\forall i = 1, 2, \ldots, m-1; E(i, m) = 0$$

we are curious about how many rules are top, so we calculate the ratio of top rules for different rule sets in Fig. 4. For any rule set, the ratio of top rules is larger than 50%. For the ACL rules, the ratio is even larger than 80%. It is noted that the top property is different from the independence, discovered by SAX-PAC [8]. A *top* rule can be placed as the first rule in the rule set, without change of semantics, but *independence* means that the rule can be at any location in the rule set.

A reasonable explanation for the observation is that the operators tend to assign more general rules, which means larger hypercube, with lower priority. If the general rules are with higher priority, it is much possible that a lower priority rule is fully covered by the higher priority rules and never take effect.

C. The overlapping after projection is complex.

The analysis in §IV-A suggests that partition simplifies packet classification a lot, but it contradicts the fact that replication still occurs after partition [21]. We find that there could still be overlapping on the projection of one field even if there is no overlapping between two rules. For example, R_3 and R_4 in Table I have no overlapping now that R3 is for TCP flows and R4 for ICMP, but they have overlapping on the source IP field because 95.105.143.51/32(R_4) is a subset of 95.105.142.0/23(R_3). What's worse, some algorithms tend to use only one field to distinguish rules. Here we define a new intersection of two rules on the projection of several fields $\phi = \{f_1, f_2, ..., f_k\}$ and

$$r_i \overset{\phi}{\bigcap} r_j = (F^i_{f_1} \bigcap F^j_{f_1}) \times (F^i_{f_2} \bigcap F^j_{f_2}) \times \ldots \times (F^i_{f_k} \bigcap F^j_{f_k})$$

thus we could get another graph $G^\phi$with

$$E^\phi(i,j) = \begin{cases} 1, r_i \overset{\phi}{\bigcap} r_j \neq \emptyset \\ 0, r_i \overset{\phi}{\bigcap} r_j = \emptyset \end{cases} \tag{4}$$

It is obvious that $E^\phi(i,j) = 1$ in Eq. 4 if $E(i,j) = 1$ in Eq. 3 and thus G is a *spanning subgraph* of G^ϕ.

Let us take Table I as an example. After projecting all rules into only the source IP field, we have a graph in Fig. 5. Obviously, Fig. 5 is much more complex and Fig. 1 is a subgraph of Fig. 5.

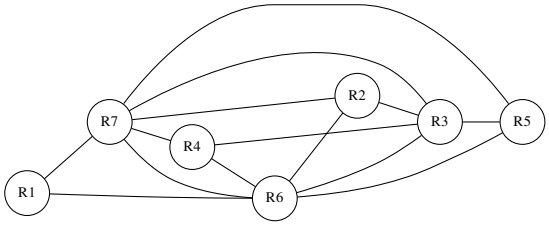
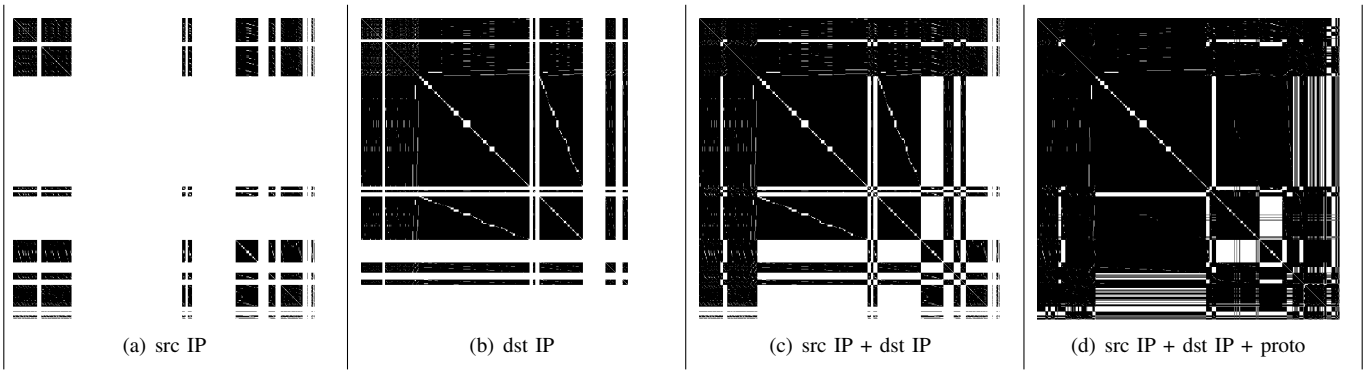


Fig. 5: The mapped graph for rules in Table I on src IP.

We choose FW_1k rule set as an example to show the difference on different projections. As depicted in Fig. 6, the projection on the source IP may lead to many overlapping which is understandable because the operators tend to write arbitrary source IP to block all traffic from some attackers for the firewall.

However, it is surprising that even when we take source IP, destination IP, and protocol fields into consideration, the projected graph is still a little complex. Because Fig. 6 (d) does not reveal it clearly, we summarize another table in Table III. Although the density for the projected graph on src&dst&proto (11.91%) is close to the original graph(9.67%), the number of dyeing colors is much large, 4.5x more than the original graph. After looking into the FW rules, we find that there are some rules with arbitrary source IP, destination IP, and source port, but exact destination port and exact protocol. These rules are understandable because the operator wants to block particular application service for all equipment, such as SSH (most with 22 port above TCP) and HTTP(s) (most with 80/443 port above TCP). It is a pity that quite a few existing algorithms before [11], [13], [21] focus on the source IP and destination IP field to partition rules, ignoring the importance of the destination port.

The main lesson from the observation above is that we need to take multiple fields into consideration at the same time instead of only one field.



(a) src IP (b) dst IP (c) src IP + dst IP (d) src IP + dst IP + proto

Fig. 6: The adjacent matrix for projected graph of different fields combination in FW_1k with the same explanation of Fig. 2.

TABLE III: Some characteristics of the mapped projected graphs in FW_1k.

Projection	#edge	#edge/#node	density(%)	#color
source IP	36,714,421	3943.12	84.75	5675
src IP & dst IP	13,478,571	1447.60	31.14	77
src & dst & proto	5,144,798	552.55	11.91	44
all fields	4,170,923	447.96	9.67	8

V. PCG DESIGN AND EVALUATION

Based on the observation that most rules are at the top, we propose a new patch *PCG* for current algorithms which use the method of partition. As depicted in Algorithm-1 line 12-23, in the process of looking for the matching rule of a packet in the groups one by one, the search ends once the matched rule is at the *top*. Because most rules are at the top, the patch could improve the throughput effectively. It can be viewed as a variant priority sorting of PSTSS [15], but it has many rules with the same highest priority to skip following groups.

This section will evaluate PCG from the improvement on packet classification and the cost of PCG. All experiments are conducted on a machine with Intel(R) Core(TM) i7-4790 CPU@3.6GHz and 32G DDR3 1600MHz DRAM. The operating system is Ubuntu 20.04 LTS.

A. Throughput improvement

We choose a state-of-art algorithm, CutSplit as a benchmark. The C++ source code is from the author and we compile it with gcc v9.3.0 and the parameter -O0 (no optimization). As depicted in Fig. 7, PCG improve the throughput of CutSplit in all rule sets and the increase ratio is 21%, 23%, 23%, 14%, 17% and 9% respectively. Specifically, there are 3 groups in CutSplit and ∼10% traffic leaves after the search in the first group, ∼30% leaves in the second. We expect the improvement would be higher for the algorithm with more groups such as TupleMerge or RVH.

B. Calculation for top rules

There is no free lunch for the throughput improvement. It comes with the cost of calculation of top rule. We show a simple traverse in Algorithm-1 line 1-10 and the time complexity is $O(n^2)$. The method for intersection calculation of two rules is from PSA [23], with directly computing the intersection of each field. BDD can be used for acceleration because it can get the union of traversed rules rapidly and remove the inner loop. We leave this to future work and just employ Python 3.8 to calculate the top rules here.

As depicted in Fig. 8, the calculation time is about 1s for 1k rules, 100s for 10k rules. Because the program is on python now, we expect it to be much faster with C. Nonetheless, the calculation time is not very critical for PCG because packet classification and the calculation of top rules can be detached. We can initialize every rule with *isTop* False and

Algorithm 1 PCG

Input: rule set R: $r_1 < r_2 < \ldots < r_n$;
Input: partition groups $S_1, S_2, \ldots, S_k$
1: // mark the top rules
2: **for** i=1, i++, $i \leq n$ **do**
3: $r_i.isTop = True$
4: **for** j=1, j++, $j < i$ **do**
5: **if** $r_i \bigcap r_j \neq \emptyset$ **then**
6: $r_i.isTop = False$
7: break
8: **end if**
9: **end for**
10: **end for**
11: // packet classification
12: initialize potential results $R = \emptyset$
13: **for** i=1, i++, $i \leq k$ **do**
14: search in S_i to get matched rule r
15: **if** r exists **then**
16: **if** $r.isTop = True$ **then**
17: return the rule r
18: **else**
19: $R = R \bigcup r$
20: **end if**
21: **end if**
22: **end for**
23: linear search in R to get the result

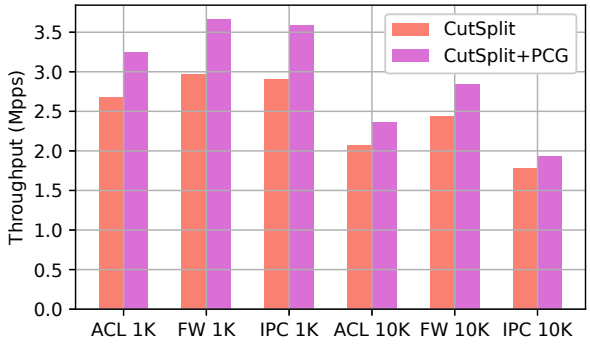


Fig. 7: Throughput improvement by PCG.

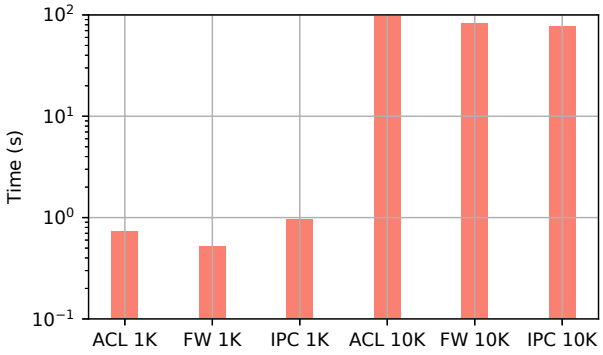


Fig. 8: Calculation time for the top rules.

the correctness of classification is kept. The calculation can be lazy and only involve the modified rules with the time consumption $O(n)$ for rule update.

VI. Future work

Here we give some of our thoughts about the following directions for future work..

1) The paper initiates a new angle to analyze the rule sets and more tools can be utilized, such as the GraphBLAS [24] from MIT, the formalization tools in the programming language area.
2) The method could work for more kinds of rules, such as IPv6 forwarding rules, data center rules. We would analyze more rule sets in the future.
3) There is no silver bullet for packet classification algorithms, but we can have the optimal solution for a specific scene. We would use all five kinds of actions in §III and use mathematical optimization instead of intuitive heuristic methods for the construction of data structure, even taking the cost of cache miss into consideration.

VII. Conclusion

In this paper, we study some properties of packet classification rule sets. We find that partition is necessary and it is better to take multiple fields into consideration when the rule set is shrunken, confirming the correctness of some designs of previous algorithms. We also propose PCG, a patch feasible for many algorithms that leverage partition, and demonstrate the improvement of classification performance using PCG.

Acknowledgment

The authors want to thanks to George Varghese for his excellent book *Network Algorithmics* [22].

References

[1] J. Daly, V. Bruschi, L. Linguaglossa, S. Pontarelli, D. Rossi, J. Tollet, E. Torng, and A. Yourtchenko. Tuplemerge: Fast software packet processing for online packet classification. *IEEE/ACM ToN*, 2019.
[2] J. Daly and E. Torng. Bytecuts: Fast packet classification by interior bit extraction. In *INFOCOM*. IEEE, 2018.
[3] J. Fong, X. Wang, Y. Qi, J. Li, and W. Jiang. ParaSplit: A scalable architecture on FPGA for terabit packet classification. In *HOTI*. IEEE, 2012.
[4] T. Ganegedara, W. Jiang, and V. K. Prasanna. A Scalable and Modular Architecture for High-Performance Packet Classification. *IEEE Transactions on Parallel and Distributed Systems*, 2014.
[5] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *IEEE Hot Interconnects*, 1999.
[6] P. Gupta and N. McKeown. Classifying packets with hierarchical intelligent cuttings. *IEEE Micro*, 20(1):34–41, 2000.
[7] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie. Partial order theory for fast tcam updates. *IEEE/ACM Transactions on Networking*, 26(1):217–230, 2018.
[8] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster. Sax-pac (scalable and expressive packet classification). *ACM SIGCOMM Computer Communication Review*, 44(4):15–26, 2014.
[9] A. Kosowski and K. Manuszewski. Classical coloring of graphs. 2008.
[10] C. Li, T. Li, J. Li, Z. Shi, and B. Wang. Enabling Packet Classification with Low Update Latency for SDN Switch on FPGA. *Sustainability*, 12(8):1–16, 2020.
[11] W. Li, X. Li, H. Li, and G. Xie. CutSplit: a decision-tree combining cutting and splitting for scalable packet classification. In *INFOCOM*. IEEE, 2018.
[12] E. Liang, H. Zhu, X. Jin, and I. Stoica. Neural packet classification. In *SIGCOMM*. ACM, 2019.
[13] Z. Liu, S. Sun, H. Zhu, J. Gao, and J. Li. BitCuts: A fast packet classification algorithm using bit-level cutting. *Computer Communications*, 2017.
[14] J. Matoušek, G. Antichi, A. Lučanskỳ, A. W. Moore, and J. Kořenek. Classbench-ng: Recasting classbench after a decade of network evolution. In *ANCS*, 2017.
[15] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, Oakland, CA, May 2015. USENIX Association.
[16] A. Rashelbach, O. Rottenstreich, and M. Silberstein. A computational approach to packet classification. In *SIGCOMM*, 2020.
[17] T. Shen, G. Xie, X. Wang, Z. Li, X. Zhang, P. Zhang, and D. Zhang. Rvh: Range-vector hash for fast online packet classification. *Techical Report of ICT*, 2018.
[18] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM*, 2003.
[19] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *ACM SIGCOMM*, 1999.
[20] D. E. Taylor and J. S. Turner. Classbench: A packet classification benchmark. *IEEE/ACM transactions on networking*, 2007.
[21] B. Vamanan, G. Voskuilen, and T. Vijaykumar. EffiCuts: Optimizing packet classification for memory and throughput. *SIGCOMM*, 2010.
[22] G. Varghese. *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005.
[23] X. Wang, W. Shi, Y. Xiang, and J. Li. Efficient network security policy enforcement with policy space analysis. *IEEE/ACM Transactions on Networking*, 24(5):2926–2938, 2015.
[24] Website. GraphBLAS: blocks for graph algorithms in the language of linear algebra. https://graphblas.org/. Accessed: 2021-12-16.
[25] Website. NetworkX: Network Analysis in Python. https://networkx.org/. Accessed: 2021-12-16.