FROD: An Efficient Framework for Optimizing Decision Trees in Packet Classification

Longlong Zhu[1], Jiashuo Yu[1], Jiayi Cai[1], Jinfeng Pan[1], Zhigao Li[1],
Zhengyan Zhou[2], Dong Zhang[1], Chunming Wu[2]
[1]College of Computer and Big Data, Fuzhou University, Fujian, China
[2]College of Computer Science and Technology, Zhejiang University, Zhejiang, China

2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS) | 978-1-6654-6824-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/IWQoS54832.2022.9812915

***Abstract*—To perform efficient packet classification, decision tree-based methods conduct decision trees via hand-tuned heuristics. Then the performance testing and optimization are executed to ensure an excellent searching speed and space overhead. Specifically, when the performance is below expectation, existing solutions attempt to optimize the algorithms, such as conducting more sophisticated heuristics. However, reconstruction or adjustment for algorithms produces an intolerable time overhead due to the long optimization period, caused by uncertain performance benefits and high pre-processing time. In this paper, we propose FROD, an efficient framework for optimizing the decision trees directly in packet classification. FROD raises a meticulous evaluation to accurately appraise decision trees constructed by different heuristics. It then seeks out the bottleneck components via a lightweight heuristic. After that, FROD searches the optimal division for inferior components considering structural constraints and characteristics of traffic distribution. Evaluation on ClassBench shows that FROD benefits existing decision tree-based solutions in classification time by 41% and memory footprint by 19% on average, and reduces classification time by up to 64%.**

I. INTRODUCTION

Packet classification is the problem of finding a matching rule for each incoming packet while ensuring an excellent temporal/spatial complexity. As the cornerstone of advanced forwarding, packet classification is of paramount importance for traffic engineering, firewalls, access control, and Quality of Service (QoS)[1–4]. Varied solutions have been proposed for packet classification, ranging from hardware-based[5–10] to software-based[1–4, 11–18]. To perform efficient packet classification, decision tree-based methods execute: 1) Considering characteristics of specific rulesets and traffic distributions to conduct effective heuristics[1–3, 11, 16–19]; 2) Conducting decision trees for rulesets based on heuristics; 3) Testing the decision trees using packets sampled from target network; and 4) Optimizing algorithms and reconstructing decision trees when classification performance is below expectation, otherwise launching.

Specifically, when the performance of classification speed or memory footprint is below expectation, existing approaches attempt to optimize the decision tree-based algorithms using more sophisticated heuristics[2, 3, 11, 19]. Then decision trees are rebuilt for testing performance again. Nonetheless, reconstruction or adjustment of algorithms provides an intolerable time overhead for the following reasons: 1) The benefits of classification performance may be tiny when the traffic distributions or characteristics of rulesets exceeds hand-tuned models; 2) There is a high pre-processing time to rebuild decision trees based on more complex methods; 3) Combining 1) and 2), the period for debugging and optimizing algorithms is lengthy. Consequently, a more efficient solution probably is seeking out the bottleneck components that affect the performance of searching speed and memory footprint and then making local adjustments for the decision trees instead of algorithms.

In this paper, we propose FROD, an efficient framework for optimizing decision trees directly in packet classification, which has high optimization efficiency and guaranteed positive performance benefits. However, it is not trivial to design such a framework due to the diversity of decision trees, the interaction inside trees, and the need to preserve a low pre-processing time. In response, we design FROD to address the above challenges. Specifically, FROD first evaluates given decision trees objectively considering characteristics of structure and traffic flow. In this step, FROD evaluates components from the following aspects to ensure high accuracy and strong universality: rule ratio between parent-child, inclination degree, depth, matching counts, number of rules, and rule priority. Thereafter, based on a lightweight heuristic, FROD seeks out the bottleneck components while decreasing the impact of interaction between nodes. After that, FROD transforms the optimization problem into an Integer Linear Programming (ILP) problem and searches the optimal division for each bottleneck component with the respect to structural constraints and traffic characteristics. Finally, FROD generates the partially reconstructed decision trees and check out whether to terminate.

In summary, we make the following contributions.

- We show the significance of optimizing conducted decision trees(§II). We identify the design challenges and propose FROD to tackle these challenges(§III).
- We design an efficient evaluation mechanism for decision trees conducted by different algorithms(§IV) and present a lightweight heuristic to seek out the inferior components that need to be optimized(§IV).
- We present an optimization framework that leverages structural characteristics and traffic distribution to tune-up

Dong Zhang is the corresponding author.

TABLE I: Characteristics of Decision Trees Built via Heterogeneous heuristics.

Multiple Algorithms	ACL			FW			IPC		
	Depth	*Nodes*	*Byte Per Rule*	*Depth*	*Nodes*	*Byte Per Rule*	*Depth*	*Nodes*	*Byte Per Rule*
HiCuts[1]	11	2512	221	28	193503	33881	17	3450	332
HyperCuts[11]	19	1795	144	28	28890	4806	25	2925	228
HyperSplit[19]	10	129	10	13	431	68	12	247	26
EffiCuts[2]	22	8257	69	26	3959	50	27	5984	71
CutSplit[3]	17	142	13	18	105	11	16	120	12

The size of all the rulesets used is 1K. And rule replication factor (i.e., byte per rule)[3]: #stored rules / rule set size

Src IP	Dst IP	TCP/UDP Src Port	TCP/UDP Dst Port	Protocol	Action
206.159.213.125/32	101.152.182.8/30	1024 : 65535	*	*	drop
15.25.70.8/30	*	*	1599	UDP	forward
*	18.152.125.32/30	65535	65535	UDP	enqueue
206.159.213.125/32	*	*	80	*	forward
*	*	*	*	*	drop

Fig. 1: A packet classification example. Real-world rulesets can have more than 100K rules.

given decision trees for packet classification(§V).

- We conduct extensive experiments and show that FROD benefits existing decision tree-based solutions in classification time by 41% and memory footprint by 19% on average, and reduces classification time and memory overhead by up to 64% and 59%(§VI).

II. BACKGROUND AND MOTIVATION

In this section, the background of packet classification is reviewed firstly. After that, we motivate our approach and emphasize the importance of optimizing conducted decision trees for packet classification.

A. Background

It is invariable the goal to improve efficiency of network transmission. As the bottleneck of advanced forwarding, packet classification is the crucial foundation for firewalls, access control, traffic engineering, security, network measurement, Quality of Service(QoS), and so on[1–4]. Figure 1 illustrates an example of packet classification. It is committed to matching packets and rules based on multiple header fields. In the past two decades, varied solutions have been proposed for packet classification, ranging from hardware-based[5–10] to software-based[1–4, 11–18].

The decision tree-based method is a pivotal category of packet classifications due to its high lookup speed. To perform efficient packet classification, decision tree-based methods firstly excavate characteristics of typical rulesets or traffic distribution to discover general regulars. And then sophisticated heuristics[1–3, 11, 18, 19] are proposed utilizing the regulars. Next, decision trees are constructed for rulesets based on the heuristics. After that, performance testing is executed using packets sampled from the target network. When the performance is below expectation, existing solutions focus on optimizing the decision tree-based algorithms and rebuilding decision trees for testing performance again.

Despite its abundance, the decision tree-based methods can not invariably build decision trees with an excellent performance of searching speed and space overhead due to the diversity of traffic distribution, the specific requirements of network applications, and the inherent defect of hand-tuned heuristics. As Table I shown, the characteristics of decision trees, constructed by existing solutions, are greatly varied under different rulesets. HiCuts[1] builds a decision tree with less depth than EffiCuts[2] under ACL rulesets while the opposite is true under FW rulesets. Similarly, HyperCuts[11], HyperSplit[19], and CutSplit[3] have their strengths and weaknesses in our detailed experiments. Besides, the theoretical bounds illustrate that it is infeasible to develop a single algorithm which performs well in all cases[17]. Therefore, optimization is a significant stage for decision tree-based methods to reach a high performance in packet classification.

B. Motivation

When the performance of searching speed or space overhead is below expectation, most existing techniques attempt to optimize the decision tree-based algorithms and rebuild decision trees, such as designing more sophisticated heuristics [2, 3, 11, 19].

However, reconstruction or adjustment for algorithms provides an intolerable time overhead due to a lengthy optimization cycle. Specifically, there is a high pre-processing time to rebuild decision trees based on more complex methods, which increases the time overhead of each optimization stage. For another, if the traffic distributions or the characteristics of rulesets exceeds hand-tuned models, the performance benefits of classification speed and memory footprint may be tiny. Therefore, there may be repeated optimization stages in optimizing decision tree-based algorithms. To summarize, optimization for algorithms(i.e., reconstruction or adjustment) makes a lengthy time.

Consequently, due to the high optimization efficiency and the guaranteed positive performance gain, a most efficient solution probably is seeking out the bottleneck sub-branches that affect the performance of searching speed and memory footprint and then making local adjustments for the decision tree structure at a low time overhead. In response, an evaluation is performed on given decision trees built by existing approaches. Then we seek out and tune up inferior sub-branches. After that, the partially reconstructed decision trees are generated. As far as we know, none of the state-of-art approaches is absorbed in optimizing a given decision tree constructed by varied heuristics to content the performance demand of classification speed and memory overhead.

III. DESIGN OVERVIEW OF FROD

In this section, we discuss the design challenges for optimizing decision trees for packet classification. Then we illustrate the corresponding design overview of FROD, a framework for optimizing decision trees in packet classification.

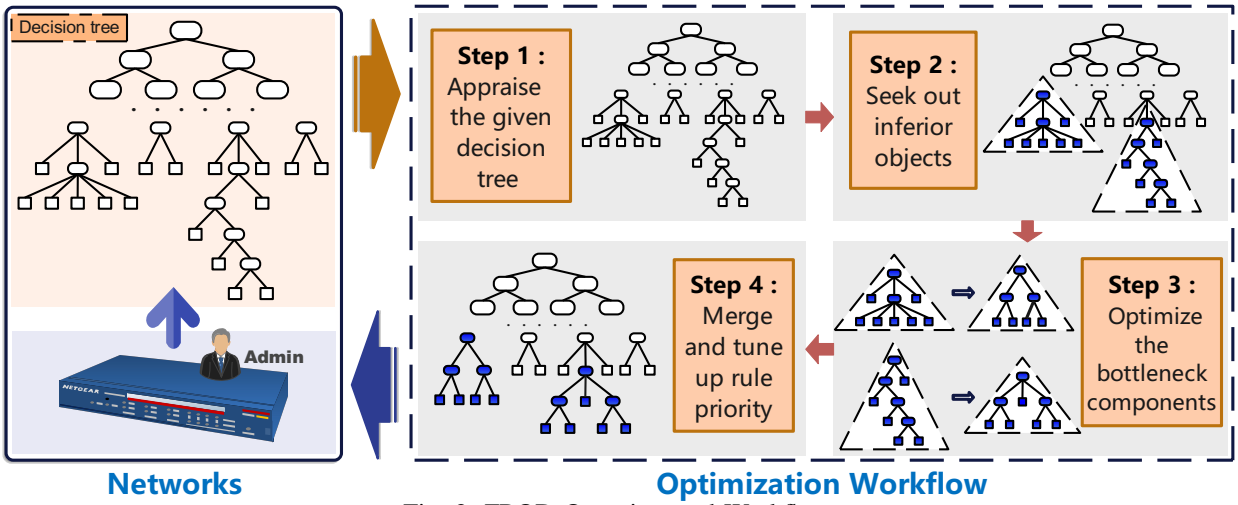


Fig. 2: FROD Overview and Workflow.

A. Challenges

We face three design challenges in tuning up decision trees for packet classification.

Diversity of decision trees. We need to precisely determine the components that should be adjusted for adequately heterogeneous decision trees. However, decision trees built by different algorithms provide varied structural characteristics, ranging from depth to rule ratio between parent-child.

Interaction inside decision trees. Adjustment of decision trees may cause potential problems. For one thing, adjustment for some nodes may affect the evaluation value of other parts, especially in parent-child. For another, since decision tree-based approaches achieve a high lookup speed and a bounded worst speed via constructing finite decisions, the processing logic of the decision tree should not be violated. Specifically, the matching rules may be wrong after executing a series of decisions because the rules are moved to a non-owned subspace.

Low pre-processing time. We need to achieve a low pre-processing time in optimization. Endlessly repeated assessments lead to an unacceptable pre-processing time. If the standard is too loose, it may adjust most sub-branches of a decision tree leading to an intolerable pre-processing time. Conversely, too strict and harsh methods can hardly maintain a considerable benefit of performance. Besides, for the requirements of launching speedily for network administrators, it is indispensable to reach a low pre-processing time.

B. Design Overview

As shown in Figure 2, we mirror the decision tree on the testing/running devices. Notice that, we maintain normal operations of original functions in the equipment. Then we measure sub-branches of the decision tree and calculate evaluation values, based on which we seek out the bottleneck components that should be adjusted. Thereafter, we tune up these objects and output superior sub-branches at a low time overhead. Finally, we merge these emerging sub-branches into the original decision tree and adjust rule priority based on current traffic statistics to improve average classification speed.

Step 1: Appraising the given decision tree. To seek out bottleneck components that affect the performance of time/memory, we make quantitative evaluations on the original decision tree(§IV). Moreover, to make the evaluation mechanism applicable under sufficiently different decision trees in §III-A, we adopt a rich diversity of evaluation dimensions(§IV-A).

Step 2: Seeking out the inferior objects. To decrease the impact of interaction between nodes in §III-A, we seek out the inferior objects both considering evaluation value and interaction numerical impact(§IV-B). To satisfy the gap between low finding time and high precision in generating a sequence of adjustment candidates, we identify the sub-branches that need to be adjusted most through a lightweight heuristic(§IV-B).

Step 3: Optimizing the bottleneck components. To prevent damaging the internal logic of decision trees, we do not directly move the sub-branches but transform the optimization problem into an Integer Linear Programming (ILP) problem(§V). Then we can obtain new division methods for sub-regions defined in the previous step. To avoid a high pre-processing time emphasized in §III-A, we combine ILP approaches and existing heuristic to reduce the time of a single optimization(§V). Namely, implementing ILP-based division in root and heuristics in the following conduction.

Step 4: Merging and tuning up rule priority. The classification speed depends on both leaf nodes searching in decision tree and linear matching inside leaf nodes. After merging new sub-branches, we tune up rule priority to improve average searching speed(§V-B). Besides, to avoid excessively repeated adjustments, we set the termination conditions combined with the current network environment to control its tightness(§V-B).

IV. EVALUATION MECHANISM

In this section, we describe (1) how FROD finely evaluates heterogeneous decision trees in §III-A via diverse and abundant perspectives, and (2) how FROD searches the bottleneck components considering interaction inside decision trees in §III-A.

A. Metrics Selection for Evaluating Mechanism.

Previous solutions primarily concentrate on measuring the whole decision tree through some performance metrics, which is coarse to seek out the inferior sub-branches. In response, FROD measures decision trees objectively via two granularity, including internal and leaf nodes, all with some metrics. For inter nodes, FROD calculates the ratio of rules between parent-child, inclination degree, and rule redundancy. For leaf nodes, FROD counts depth, matching times, number of rules inside leaf nodes, and the relationship between rule priority and matching rate. Details are as follows.

Internal Nodes. FROD takes metrics of inclination degree and rule ratio between parent-child for internal nodes.

Rule ratio between parent-child. For the parent v, we define the rule ratio as following sum of rules number inside child nodes u_1, u_2,..., u_i,..., u_n.

$$Ratio_v = \sum_{i=1}^{n} \frac{u_i.num}{v.num} \times 100\% \tag{1}$$

where $u_i.num$ and $v.num$ respectively represents rule number of child u_i and parent v. For classification time, a balanced structure of decision trees potentially means a preferable lookup speed because of fewer depth[20]. However, the child nodes with excessive rules, caused by an inaccurate division in the parent, will leads to the deformity of decision trees. Thereafter, there is a poor average and worst classification speed. For memory footprint, the rule replication in child nodes, resulting in excessive memory consumption, is expected to be minimal since the resources of deployment devices is invariably scarce. Furthermore, rule replication is also the culprit of insufficient scalability for decision tree-based packet classification[3]. Therefore, we take the rule ratio between parent-child as an indispensable indicator.

Inclination degree. Inclination degree directly represents the deformity of a decision tree, which causes high worst-case search time. We take the following variance of rules number inside children u_1, u_2,..., u_i,..., u_n as inclination degree attached to parent v.

$$\sigma_v^2 = \sum_{i=1}^{n} \frac{(u_i.num - \overline{u.num})^2}{n} \tag{2}$$

where $\overline{u.num}$ indicates the average number of rules inside child nodes. Take notice that the mean value here is the ideal division instead of the actual one, namely variance σ_v^2 = 0. In our experimental observation, the distribution of variance with tree depth is similar to the Poisson distribution in most cases, which conforms to the characteristics of decision tree division. Consequently, points that are vastly beyond Poisson distribution are probably possessed improper divisions.

Leaf Nodes. FROD takes the following metrics for leaf nodes.

Depth vs. matching counts. Classification time which is closely related to depth is a significant element in packet classification. In addition to the structural metrics, combining the traffic distribution, such as the count value of packets corresponding to each rule, is also an indispensable mensuration[21, 22]. In our observation, the matching counts of leaf nodes represents the characteristics of approximate Gaussian distribution. Consequently, the strategy used to hit the maladaptive parts can be set as assigning a dangerous value for sub-branches that exceed σ-based threshold.

The number of rules. Rule matching includes two processes: searching leaf nodes and linear search in leaf nodes. Hence, the number of rules inside leaf nodes is a non-negligible consideration. Furthermore, resemble the influence of binth[1] on memory footprint, rule number in leaf nodes reflects the memory overhead to a certain extent as well.

Rule priority vs. matching counts. Since linear search is based on rule priority, there are rules with low priority and high matching rates, resulting in low average classification speed. Therefore, we take it as a significant performance metric for leaf nodes, especially for changing traffic flows. Ideally, the rules with higher priority are matched more frequently in a leaf node. As a result, the worse evaluation is given to the leaf nodes that deviate from the ideal situation more.

B. Approach for Determining Sub-branches.

Computing evaluation value of each node. FROD aims to have an insight comprehend with each component of the decision tree to seek out sub-branches that should be tuned up. Based on rule ratio between parent-child, inclination degree, depth, matching counts, number of rules, and rule priority in §IV-A, FROD obtains abundant but heterogeneous evaluation values of each node, leading to a scattered and chaotic evaluation. Hence, FROD transforms it into a single evaluation value using the weighted sum method[23]. FROD defines the evaluation value of node i as a weighted sum:

$$E_i = \sum_{i=1}^{n} c_j \times E_i^j \tag{3}$$

where E_i^j is the evaluation value of node i on metric j and c_j is the impact factor of metric j. To determine a numerical number of c_j of each metric, we measure each c_j using the control variable method.

Moreover, to ensure the fairness between evaluation values given by each metric, we normalize the scoring mechanism of each measurement before calculating the weighted sum:

$$E_i^j = \frac{E_i^j - \overline{E^j}}{\sigma} \tag{4}$$

where E_i^j, E^j, and σ represents the practical evaluation value, the mean evaluation value, and the standard deviation of evaluation value given by metric j.

Selecting sub-branches to be adjusted. After measuring all nodes inside a given decision tree, it is foremost to select sub-branches which need to be adjusted. However, it is not ordinary to determine such nodes at a low overhead on account of there are oceans of nodes and adjustment may affect the evaluation value of other parts. In this regard, FROD proposes a heuristic to address the problem. Given a decision tree,

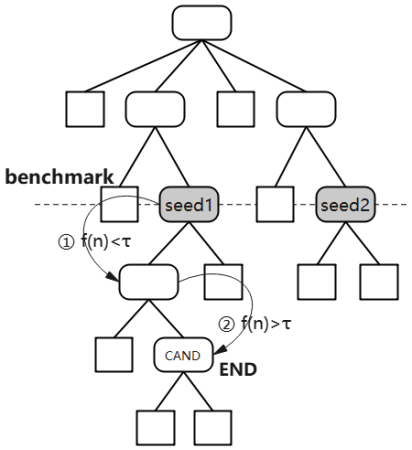


Fig. 3: An example of searching inferior nodes.

numbers of metrics that include individual metrics, and interactive metrics as input, FROD can easily receive an evaluation value that consists of individual stationary parts and interactive metabolic parts for each node. Then to excavate sub-branches which should be adjusted, FROD generates initiate searching adhering to the following heuristic:

$$f(n) = g(n) + \frac{s(n) + a(n)}{a(n)} \tag{5}$$

where $f(n)$ is a valuation of adjacent nodes, $g(n)$ is the cost from the current node jump to the next node in a decision tree, $s(n)$ is the value of individual metrics, and $a(n)$ is the value of interactive metrics. Notice that, $g(n)$ is calculated based on the actual time cost of the deployed system. Since the top-down characteristics of decision trees, there is a more severe consumption for $g(n)$ to trace back to the parent node than downward to a child node. Therefore, the probability of downward search to a child is larger than that of upward search to parent, which can minimize the complexity of following adjustment and then satisfy our requirements of low pre-processing time.

Moreover, to avoid searching a large number of nodes and to find candidate nodes faster, we both two factors: 1) Minimizing the maximum sum of interactive evaluation values throughout whole searching processes; 2) Maximize the sum of evaluation values with ultimate adjustment candidate nodes due to that the larger its value, the more it should be optimized. $\frac{s(n)+a(n)}{a(n)}$ is designed to make the searching being biased towards the direction with larger inherent evaluation value and smaller interactive evaluation value, which is tally with our original intention to reduce the impact between nodes.

In the initialization stage, FROD determines a benchmark number of branches to be searched according to the mean depth and median depth, based on which FROD generates limited seeds. And then FROD put them into different branches. After that, FROD sets the following termination conditions: 1) The f(n) of the current node is greater than the threshold τ; 2) Search until encountering root nodes, leaf nodes, and nodes that have been marked as candidate nodes; 3) Reach the maximum number of search steps, which is initial as the average depth; 4) After generating ordered candidates that satisfy the numerical requirements, FROD suspends the search of the heuristic algorithm.

TABLE II: Notation of symbols.

Symbol	Description
$\mathbb{M}$	Benchmark matrix
R	Rule entire
$R(i)$	Sub-ruleset contained in node i
r	Root node
u	Current node
v	Parent of u
K	Number of fields
D	Local division times
BLF	A parameter for decision tree balance objective
P_R	Predicted following heuristic-based division times
H_R	Maximum limit height of final decision trees
F_h	Function to calculate node height limit
D_i	Depth limit of current subbranches
C_i	Number of cutting inside node i
OR_i	Rule redundancy ratio of node i
$N(R_i)$	Number of rules inside $R(i)$
$Spfac$	A parameter weighs memory footprint and C_i

As shown in Figure 3, FROD determines the number of seeds based on the mean and median of depth and then randomly puts them into different branches. Take seed1 as an example. Based on the heuristic, seed1 search and jump to the next node. Evaluation value $s(n) + a(n)$ of the node is fewer than threshold τ. Therefore, FROD continues to search and then seeks out the next node. Since evaluation value $s(n) + a(n)$ is beyond than the threshold τ, FROD sets current node as a candidate. After that, FROD terminates the searching process for seed1 due to meeting the first termination conditions.

V. OPTIMIZATION FRAMEWORK

In this section, we illustrate the details of how FROD tunes up decision trees by transforming it into an Integer Linear Programming (ILP) problem. Table II summarizes the notations.

A. Input pre-processing

In many network architectures, such as Software Defined Network (SDN), there are up to a dozen of fields for packet classification. And it will be more complex with the development of new network applications, which means a huge field selection overhead for adjustment. However, there are potentially a few critical fields in practice. Since many approaches[1–3, 11, 24, 25] focus on fields selection which is not our key point in this paper, we use the existing dimensionality reduction techniques directly.

After that, FROD generates an initial partition, called benchmark matrix $\mathbb{M}$. The division in benchmark matrix is approximately irregular rule redundancy due to a heuristic combined with cutting and splitting. Therefore, FROD takes the benchmark matrix as the initial input to avoid high adjustment time caused by random input. Furthermore, with the help of benchmark matrix, we can express the constraints about rule replication more flexibly in the optimization model. The specific process is as follows:

As Figure 4 shown, FROD implements cutting, splitting, or no-action for each field. Please note that FROD can only cut

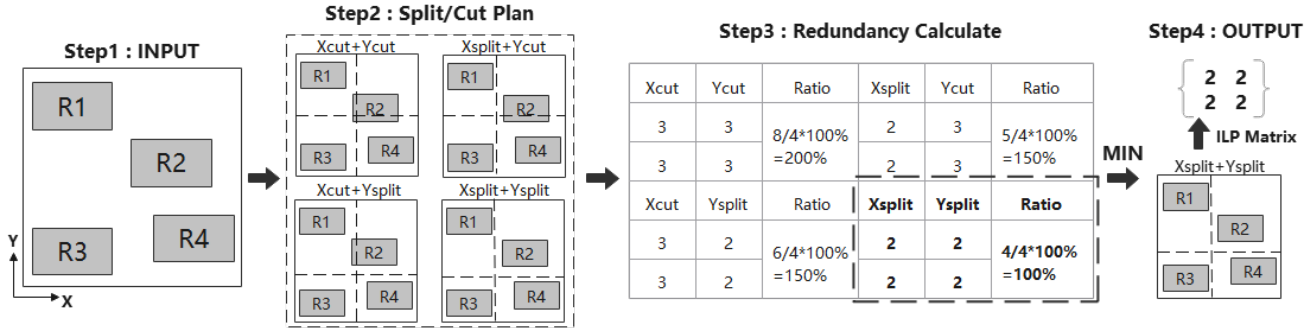


Fig. 4: An example of generating a benchmark matrix.

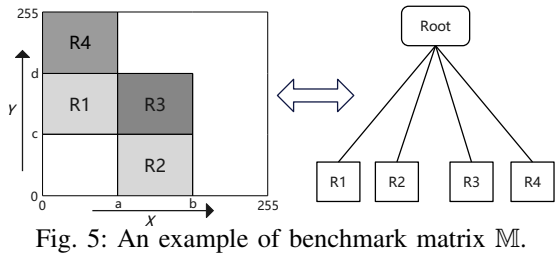


Fig. 5: An example of benchmark matrix $\mathbb{M}$.

or split with dichotomy at most here. Then by combining the results of different partitions, such as "cutting-X + splitting-Y", FROD hits a division with the minimal rules in children, i.e. minimizing rule replication. After that, the benchmark matrix is generated.

B. Optimization model

In this part, we introduce the optimization model of FROD in detail.

Problem Statement. FROD takes a benchmark matrix $\mathbb{M}$ of K columns and D rows as input, where K and D presents the number of fields and division times on each field respectively. And elements $\{x_i\}$ inside the matrix are number of rules contained in corresponding subspace. Similarly, the output is also a K-column matrix, based on which we can obtain a new division method. Figure 5 shows an example of $\mathbb{M}$.

Optimization object. FROD aims to seek out the optimal partition for each inferior component. The balance factor is used to reflect the advantages and disadvantages of a partition. An unbalanced decision tree leads to an increase in the worst search time and average search time. Therefore, a small balance factor represents the effective division. Based on the balanced factor, FROD takes the following quantitative optimization objectives.

$$\min BLF \tag{6}$$

$$BLF = \frac{\sum_{i=1}^{n} D_{u_i} - \bar{D}_u}{n-1} \tag{7}$$

Optimization Constraints. We elaborate on the following critical constraints:

(1) Tree height. The height of trees limits the worst classification time. The sum of local division time D in a matrix and predicted following heuristic-based division times P_R must not exceed the limit depth D_u of the current sub-branch:

$$D + P_R \leq D_u \tag{8}$$

$$D_u = min(F_h(R(u)), D_v - 1) \tag{9}$$

(2) Number of cuttings. The number of cuttings is directly proportional to memory footprint while too few cuttings cause overall tree depth, so cutting number should be in the trade-off between memory overhead and tree depth. Therefore:

$$C_i \leq F_c(D_i, Spfac) \tag{10}$$

(3) Rule redundancy. Rule replication is closely related to effective divisions. If it is too high during partition, the efficiency of current division is too low. To ensure an effective separation and reduce redundancy in child nodes, the number of rules for each child must satisfy:

$$E_f \geq OR_v = \frac{\sum_{i=1}^{n} N(R_{u_i})}{N(R_v)} \geq 1 \tag{11}$$

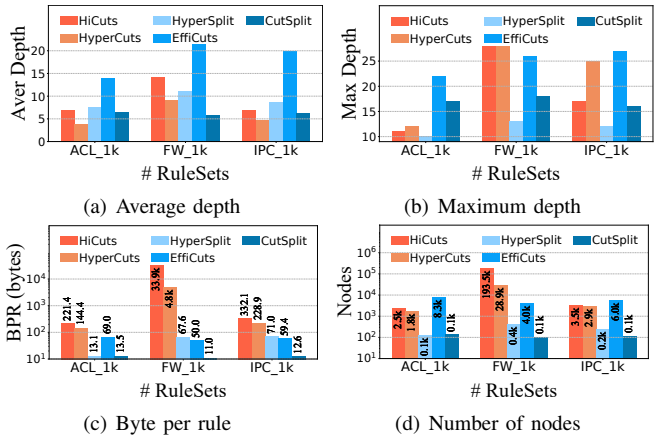
(4) Node dependencies. A new division of u must respect the parent-child node dependencies generated by decision tree logic. Specifically, except normal partition, FROD prohibits changing the sub-rulesets contained by each node across branches from the root to each leaf node, i.e.

$$E_f \geq OR_u = \frac{\sum_{i=1}^{n} N(R_{v_i})}{N(R_u)} \geq 1 \tag{12}$$

Notice that, we transform the constraints into corresponding forms for the matrix in practice. For example, we transform the constraints on tree height, node depth, and the number of cuts into constraints on the rows of a matrix, or the height of subbranches divided by the current matrix. Furthermore, we can define the deviation relative to the benchmark to constrain the rule redundancy of the output due to the benchmark matrix being approximately non-redundant.

Rule priority. After merging new subbranches, FROD reorders rule priority to improve average searching speed. To trade-off pre-processing time and improvement of average classification speed, FROD adjusts rule priority when simultaneously satisfying: 1) Rules with high matching rates have a low sequence in linear search. 2) Rules are not completely covered by rules with higher priority. 3) The overlap between rules involved in the adjustment is easy to cut.

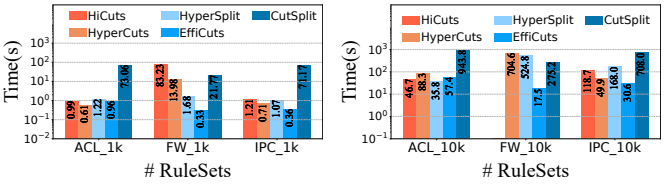
Termination Conditions. If the standard is too loose, endlessly repeated adjustments cause an unacceptable pre-processing time and memory footprint. Conversely, too strict and harsh methods will miss some nodes which should be



(a) Average depth (b) Maximum depth

(c) Byte per rule (d) Number of nodes

Fig. 6: Characteristics of decision trees.



(a) Ruleset with 1K rukes (b) Ruleset with 10K rules

Fig. 7: Reconstruction time of some decision tree-based approaches. We omit one entry for HiCuts that did not complete after more than 12 hours.

adjusted. Therefore, FROD sets the termination conditions combined with the current traffic flows to control the tightness for the request of low pre-processing time in §III-A. Specifically, FROD decides whether to stop by contrasting the average classification time of decision trees before and after optimization under-sampling packets in the current network.

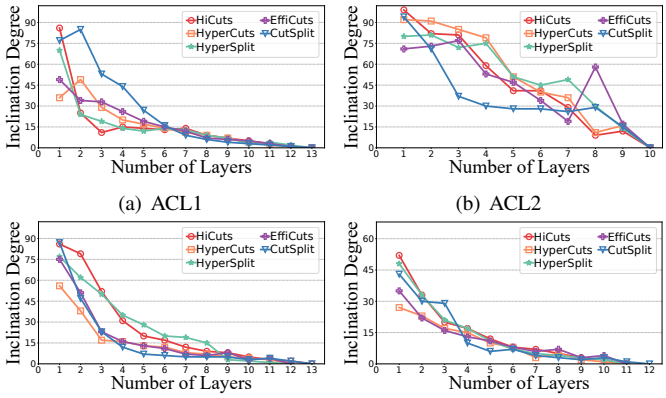
VI. EXPERIMENTAL RESULTS

In this section, we first validate the problem we proposed. After that, we present the performance results of FROD with other representative decision-tree techniques. Based on key metrics of packet classification, we evaluate FROD from memory access, memory consumption, and pre-processing time respectively.

A. Experimental Methodology

We compare FROD with the following algorithms: HiCuts[1], HyperCuts[11], HyperSplit[19], EffiCuts[2], and CutSplit[3]. Convenient for a fair comparison, we have made some modifications to the open-source code of the other four algorithms. Their performances are not affected by our modification justifiable.

Rule Sets. We use Classbench[26] to generate partial rule sets in our experiments. ClassBench[26] is a standard benchmark extensively used for evaluating packet classification algorithms[2–4, 14]. There are three types of rule sets: Access Control List (ACL), Firewall(FW), and IP Chain (IPC). We created rule sets of sizes 1K, and 10K, all with 5-fields rules: source and destination IP, source and destination port, and protocol. Moreover, the corresponding traffic sets are also generated by Classbench.



(a) ACL1 (b) ACL2

(c) FW (d) IPC

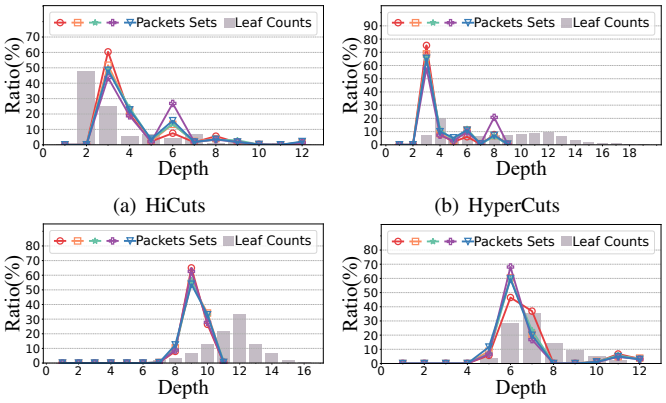
Fig. 8: Inclination degree distribution.

Machine Environment. We ran all the experiments on Intel(R) Xeon(R) Gold 6278C CPU@2.60GHz with 2 cores, 32KB L1d-, 32KB L1i-, 1MB L2-, and 35.8MB L3- caches. The operating system is Ubuntu 20.04.2 LTS (Linux kernel 5.4.0). To reduce the CPU jitter error, we take the average results by running each evaluation ten times.

B. Problem Verification

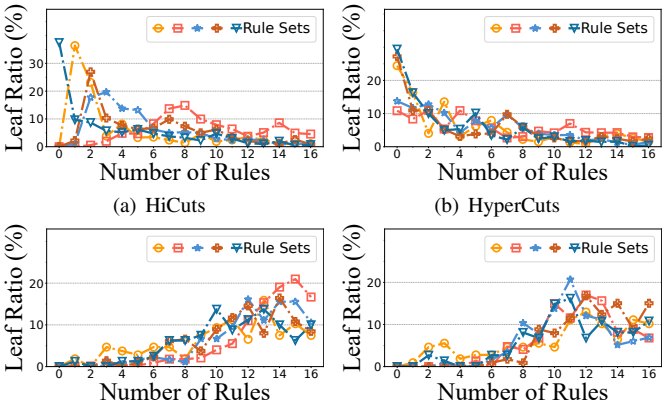
(Exp#1) Performance differences. Take HiCuts[1], HyperCuts[11], HyperSplit[19], EffiCuts[2], and Cutsplit[3] as example, we verify the discrepancy of decision trees generated by these algorithms under different rulesets. Figure 6 illustrate that decision trees constructed based on heuristic algorithms have multifarious structural features, including average depth, maximum depth, rule redundancy (i.e., byte per rule), and the number of nodes. For instance, HiCuts builds a decision tree with fewer average depth, maximum depth, and rule redundancy under ACL, while the opposite is true in the ruleset of FW. HyperCuts, HyperSplit, EffiCuts, and Cutsplit also have similar phenomena that the algorithm is more suitable for specific rulesets. Moreover, no one can conduct the optimal decision tree under all the rulesets of ACL, FW, and IPC while it is more common to conduct a non-optimal one. As a summary, our observation emphasizes the necessity of further adjustment and optimization for decision trees constructed by existing heuristic algorithms, which verifies the significance of problem solved.

(Exp#2) Reconstruction Time. Some algorithms adapt to the changing rulesets and traffic flows by reconstructing decision trees. However, the reconstruction time is unacceptable in practice. Figure 7 shows the performance in terms of reconstruction time for different decision tree-based algorithms. It is clear that the reconstruction time fluctuates in the range of seconds to more than ten minutes and has been increased by one order of magnitude at least. Notice that, we omit one entry for HiCuts that did not complete after even more than 12 hours. Besides, the scale will be tens of thousands in practice instead of one thousand in our experiment. Moreover, there are up to a dozen fields in a specific network, which makes clear that the reconstruction time will be more tremendous and



(a) HiCuts (b) HyperCuts (c) HyperSplit (d) CutSplit

Fig. 9: Matching ratio of different algorithms under 5 data sets.



(a) HiCuts (b) HyperCuts (c) HyperSplit (d) CutSplit

Fig. 10: Number of rule inside leaf nodes under ACL.

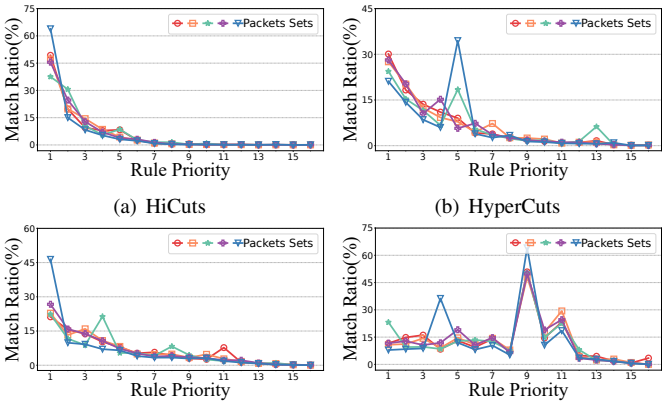
intolerable. Rebuilding is not invariably satisfactory because of the time overhead.

C. Evaluation Mechanism

(Exp#3) Characters of metrics. We elaborate salient features for some metrics in §IV-A based on experimental results.

Inclination degree. As shown in Figure 8, the distribution of inclination degree is similar to the Poisson distribution in most cases, which conforms to the characteristics of decision trees with top-down division. As shown in Figure 8(c), the curve with faster decline is what we expect for it means that the divisions of subsequent branches achieve a more balanced state faster. Conversely, curves with low curvature or fluctuating up and down are probably possessed improper divisions. Notice that, rule ratio between parent-child presents a similar character and identical strategy is applied.

Depth vs. matching counts. Figure 9 illustrates the characters of matching times under IPC. Figure 9(a,b) shows that the matching rate of shallow leaf nodes is higher than that of deep leaf nodes, which indicates a superior average search speed, while the matching in Figure 9(c,d) focuses on deep nodes, indicating the need of structural adjustment and optimization to reduce the depth of highly matched leaf nodes. Moreover, the fluctuation in Figure 9(a,b,d) also warn that we need to tune up nodes with both a high matching rate and biggish depth to



(a) HiCuts (b) HyperCuts (c) HyperSplit (d) CutSplit

Fig. 11: Relationship between rule priority and matching ratio under FW.

improve the average search speed. Due to space constraints, we omit EffiCuts in some experiments while EffiCuts performs the similar results.

The number of rules. We initialize the binth as 16, which means the number of rules inside leaf nodes is in the range of 1 to 16. Figure 10(a,d) shows the number of rules inside leaf nodes. For sparse leaf nodes, the number of rules is small, or the rule priority and matching rate are both low, then nodes should be merged into the upper layer or other nodes in the same layer. For dense leaf nodes, the rule priority is low but the matching rate is high. It is difficult to adjust the rule priority directly because of rule overlap, so nodes should be split.

Relationship between priority and rate. As shown in Figure 11(a), most of the matches in HiCuts focus on rules with high priority while the matching ratio curves of Hypercuts and Hypersplit fluctuate, indicating that major matches occur in rules with low priority. In the fifth data set, the matching rate curve of Hypercuts fluctuates greatly, namely rules with the fifth priority match more than the first. Therefore, it is urgent to adjust the priority to improve the average search speed under the current traffic distribution (i.e., the fifth packet set).

D. Classification Performance

(Exp#4) Classification time. In Figure 12, we conduct extensive experiments to compare the optimized decision trees generated via FROD against the original decision trees built by HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit in the ClassBench classifiers. As expected, FROD provides a 23%, 54%, 40%, 56%, and 34% median improvement over HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit respectively in average classification time (memory access) thanks to FROD reconstructs the bottleneck sub-branches. Besides, FROD reduces classification time by up to 64%. We accumulate the number of memory access (i.e., node visits) in a rule matching process to finely represent the classification time.

(Exp#5) Memory footprint. In Figure 13, we show the BPR (Bytes Per Rule) which is related to rule replication. FROD performs a 25%, 17%, 5%, 41%, and 6% median decrease over HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit separately. Moreover, FROD reduces memory foot-

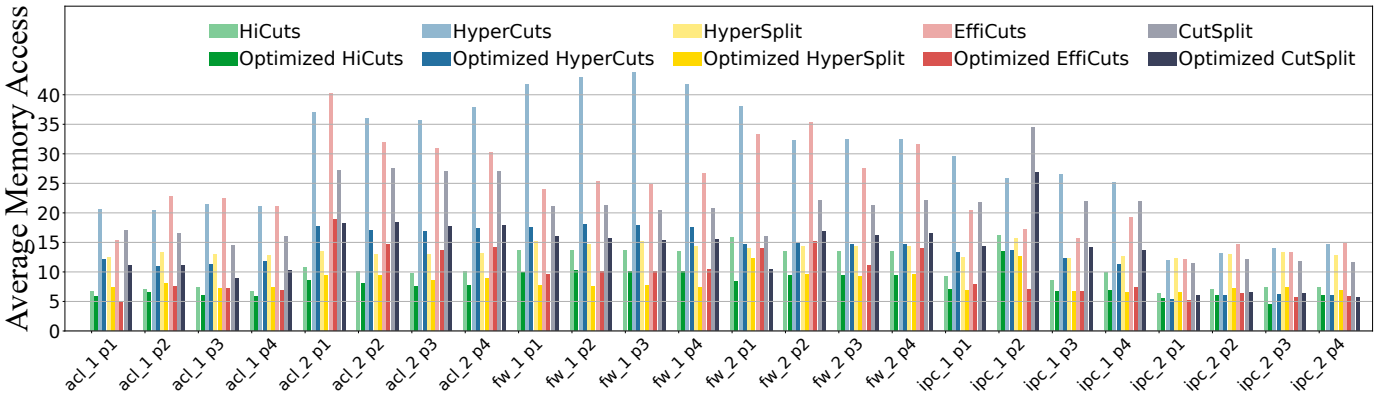


Fig. 12: Optimized classification time (memory access) for HiCuts, HyperCuts, HyperSplit, and CutSplit.

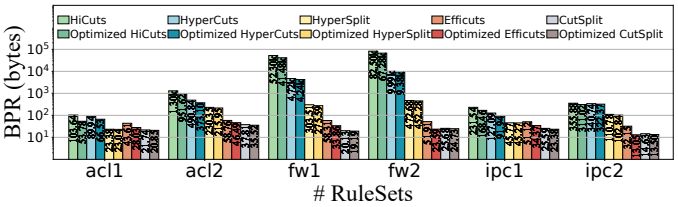
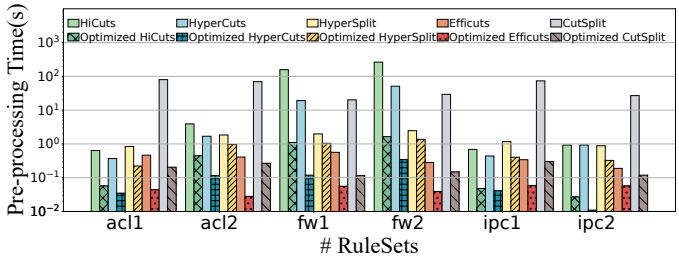
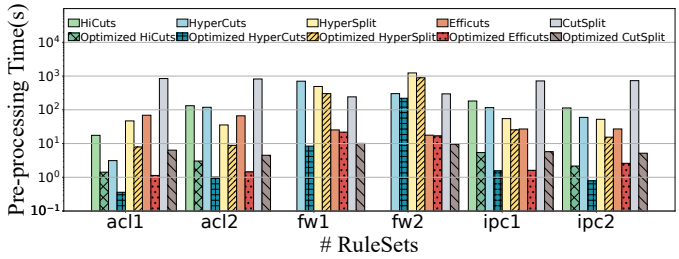


Fig. 13: Optimized memory footprint (bytes per rule) for HiCuts, HyperCuts, HyperSplit, and CutSplit.



(a) Ruleset with 1K rukes



(b) Ruleset with 10K rules

Fig. 14: Pre-processing time. We omit two entries for HiCuts that did not complete after more than 12 hours.

print by up to 59%. Notice that, we focus more on time-related evaluation metrics in Exp#4, Exp#5 and Exp#6, such as depth, matching counts, and rule priority. In practice, FROD can set the weights of evaluation metrics flexibly to fit different network environments and optimization goals.

(Exp#6) Pre-processing time. Pre-processing time of different solutions are shown in Figure 14. In Figure 14(a), compared to reconstructing decision trees, FROD provides a less pre-processing time than HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit. FROD performs a 95%, 95%, 57%, 85%, and 99% decrease of pre-processing time compared to HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit respectively. Moreover, Figure 14(b) shows that FROD keeps the advantage of pre-processing time as the size of rulesets increases.

VII. RELATED WORK

A. Structural optimization based on heuristics

Some solutions[1, 2, 11, 19, 20] attempt to conduct the optimal structure of decision trees using manually heuristics. HiCuts[1] performs an equal-size-cutting at each internal node locally based on four intelligent heuristics while it leads to the imbalance of decision trees when the density of rulesets is uneven. To address the non-uniform distribution of rule scope, Balanced HiCuts[20] adds a heuristic to specify cutting points. And HyperCuts[11] allows more than one cutting on multiple dimensions to reduce the depth and occupied memory of decision trees. However, they perform a huge memory footprint caused by rule replication, especially for the large set of rules. HyperSplit[19] conducts decision trees based on a heuristic of equal-density-splitting to guarantee worst-case classification time and reduce memory footprint. Similarly, EffiCuts[2] reduces rules replication and imbalance cutting via separable tree, selective tree merging, equi-dense cuts, and node co-location. Nevertheless, with the increase of dimension and size of rulesets, the overhead of EffiCuts might increase without an upper limitation.

B. Applicability optimization based on traffic statistics

Some algorithms[21, 22, 27–32] consider traffic characteristics to optimize packet classification, such as taking into account the relative filter usage to adapt to the input packet distribution[21], exploiting the characteristics of rulesets and traffic statistics to construct an efficient decision tree[22], utilizing traffic characteristics to maximize early rejection of unwanted flows[28], using statistical search tree to minimize the average packet matching time[28], coupling flow characteristics with analysis of the policy to accommodate varying traffic statistics while maintaining a high throughput[29], and combining traffic behavior and matching statistics to optimize the ordering of firewall rules dynamically[32].

C. Scalability optimization for packet classification

Some existing solutions[3, 4, 33, 34] exploit various design trade-offs to provide high search rates, power and space

efficiency, fast incremental updates, and the ability to scale to large numbers of filters. DCFL[33] leverages the structure of rulesets and the capabilities of hardware technology to provide a high search speed and ability to support large rule sets. CutSpilt[3] combines equal-sized cutting and equal-densed splitting to deploy a scalable decision tree with low update cost for packet classification. Similarly, based on the cutting and splitting, NeuroCuts[4] introduces deep reinforcement learning techniques to packet classification problem to satisfy the hard trade-off between computation and complexity. Hsieh et al present a scalable many-field packet classification algorithm[34]. By using a few scalable effective bits (EB) to divide a large ruleset, it reaches economic memory usage at the offline stage and quickly filter out the majority rules.

VIII. CONCLUSION

We propose FROD, a framework for optimizing decision trees in packet classification. It offers a meticulous evaluation for components of a given decision tree and then seeks out the inferior sub-branches accurately. After that, FROD searches the optimal adjustment with the respect to structural constraints and traffic characteristics. Extensive experiments show that FROD benefits for existing decision tree-based approaches in both classification speed and memory footprint. In the future, we plan to integrate programmable network processors to improve the performance gain and availability in practice.

ACKNOWLEDGMENT

This work is supported by the National Key R&D Program of China (2018YFB1800601), the Key R&D Program of Zhejiang Province (2022C01085, 2021C01036), and the Science and Technology Development Funds of China(2021ZY1025).

REFERENCES

[1] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," *In Hot Interconnects*, 1999.

[2] B. Vamanan, G. Voskuilen, and N. T. Vijaykumar, "Efficuts: optimizing packet classification for memory and throughput," *SIGCOMM*, pp. 207–218, 2010.

[3] W. Li, X. Li, H. Li, and G. Xie, "Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification," *IEEE INFOCOM*, pp. 2645–2653, 2018.

[4] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 256–269, 2019.

[5] X. A. Liu, R. C. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in tcams," *INFOCOM*, pp. 111–115, 2008.

[6] H. Che, Z. Wang, K. Zheng, and B. Liu, "Dres: Dynamic range encoding scheme for tcam coprocessors," *IEEE Transactions on Computers*, vol. 57, no. 7, pp. 902–915, 2008.

[7] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, 2010.

[8] B. Vamanan and T. N. Vijaykumar, "Treecam: Decoupling updates and lookups in packet classification," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. Association for Computing Machinery, 2011.

[9] Y. Ma and S. Banerjee, "A smart pre-classifier to reduce power consumption of tcams for multi-dimensional packet classification," *SIGCOMM*, pp. 335.0–346, 2012.

[10] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Sax-pac (scalable and expressive packet classification)," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. Association for Computing Machinery, 2014, p. 1526.

[11] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," *SIGCOMM*, pp. 213–224, 2003.

[12] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *SIGCOMM*, pp. 135–146, 1999.

[13] B. P. et al., "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 117–130.

[14] 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 Transactions on Networking*, pp. 1417–1431, 2019.

[15] A. Feldmann and S. Muthukrishnan, "Tradeoffs for packet classification," *INFOCOM*, pp. 1193–1202, 2000.

[16] S. Yingchareonthawornchai, J. Daly, X. A. Liu, and E. Torng, "A sorted-partitioning approach to fast and scalable dynamic packet classification," *IEEE/ACM Transactions on Networking*, pp. 1907–1920, 2018.

[17] W. Li, T. Yang, O. Rottenstreich, X. Li, G. Xie, H. Li, B. Vamanan, D. Li, and H. Lin, "Tuple space assisted packet classification with high performance on both search and update," *IEEE Journal on Selected Areas in Communications*, pp. 1555–1569, 2020.

[18] X. Dong, M. Qian, and R. Jiang, "Packet classification based on the decision tree with information entropy," *The Journal of Supercomputing*, pp. 1–15, 2020.

[19] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," *INFOCOM*, pp. 648–656, 2009.

[20] H. Abdoli, "Balanced hicuts: an optimized packet classification algorithm," *ICCOMP'09 Proceedings of the WSEAES 13th international conference on Computers*, pp. 406–411, 2009.

[21] Y. C. T. Woo, "A modular approach to packet classification: Algorithms and results," *INFOCOM*, pp. 1213–1222, 2000.

[22] Y. Qi and J. Li, "Dynamic cuttings: Packet classification with network traffic statistics," *the 3rd Trusted Internet Workshop (TIW)*, 2004.

[23] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, p. 853862, 2010.

[24] C. Li, X. Zeng, L. Song, and Y. Jiang, "A fast, smart packet classification algorithm based on decomposition," *JOURNAL OF CONTROL SCIENCE AND ENGINEERING*, 2020.

[25] P. Gupta and N. McKeown, "Packet classification on multiple fields," *SIGCOMM*, pp. 147–160, 1999.

[26] E. D. Taylor and S. J. Turner, "Classbench: a packet classification benchmark," *IEEE/ACM Trans. Netw.*, pp. 499–511, 2007.

[27] D. Morikawa and M. Iwata, "Fast packet classification on a data-driven network processor," Tech. Rep. 84, 2003.

[28] H. Hamed, A. El-Atawy, and E. Al-Shaer, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE Journal on Selected Areas in Communications*, pp. 1817–1830, 2006.

[29] A. El-atawy, T. Samak, E. Al-shaer, and H. Li, "Using online traffic statistical matching for optimizing packet filtering performance," *INFOCOM 2007, VOLS 1-5*, pp. 866–+, 2007.

[30] L. Li and X. Lu, "A novel algorithm for packet classification based on network traffic," in *Information Computing And Automation: (In 3 Volumes)*. World Scientific, 2008, pp. 737–740.

[31] B. Xu, G. Zhou, Y. Xue, and J. Li, "Ahsm: Adaptive packet filtering with network traffic statistics," *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 258–263, 2008.

[32] Z. Trabelsi, H. E. Sayed, and S. Zeidan, "Firewall packet matching optimization using network traffic behavior and packet matching statistics," in *Third International Conference on Communications and Networking*, 2012, pp. 1–7.

[33] E. D. Taylor and S. J. Turner, "Scalable packet classification using distributed crossproducting of field labels," *IEEE Infocom 2005: The Conference on Computer Communications, Vols 1-4, Proceedings*, pp. 269–280, 2005.

[34] C.-L. Hsieh, N. Weng, and W. Wei, "Scalable many-field packet classification for traffic steering in sdn switches," *IEEE Transactions on Network and Service Management*, pp. 348–361, 2019.