International Journal of Engineering Trends and Technology
ISSN: 2231-5381/ https://doi.org/10.14445/22315381/IJETT-V74I11P131

Volume 74 Issue 11, 480-499, November 2026
© 2026 Seventh Sense Research Group®

Original Article

A Data-Driven Partition-Hash Tree for Efficient Multi-Field Packet Classification

Sangita J. Wamane[1], Balasaheb S. Agarkar[2], Sachin V. Chaudhari[3]

[1,2,3]*Department of Electronics and Telecommunication Engineering, Sanjivani College of Engineering, Kopargaon, India.*

[1]*Corresponding Author: wamanesangita6@gmail.com*

Received: 13 May 2026 Revised: 10 September 2026 Accepted: 30 September 2026 Published: 25 November 2026

Abstract - *Packet classification has become one of the key operations in today's networking systems, which includes routers, firewalls, and SDN. However, all currently existing algorithms such as the tuple-space search and tuple-tree algorithm make decisions based on predetermined heuristics to partition rules, which limits their capacity to adapt. This work presents the idea of a new Adaptive Partition-Hash Tree (APHT) technique for machine learning-based packet classification. Specifically, it utilizes a Random Forest Regressor to predict the best parameters for building classification trees based on statistical features of the set of rules. Such features include the number of rules, distribution of prefix lengths, entropy, density, and distribution of tuples. By doing this, the algorithm adapts its decisions based on data, making it possible to dynamically select such features as split dimension, splitting threshold and the leaf size. The efficiency of the suggested technique has been tested on ClassBench datasets (ACL, Firewall, IPC), which have shown that there was considerable improvement of the algorithm in question. Namely, it increased throughput by 10-15%, improved lookup time by 12-18% and reduced memory consumption by 5-8%.*

Keywords - *Packet classification, TupleTree, Machine Learning, Random Forest, SDN, ClassBench, Adaptive algorithm.*

1. Introduction

The need for faster packet processing algorithms has become a necessity because of the tremendous advancements in networking technology and the constantly increasing amount of Internet traffic [1, 2]. Packet classification is a critical function in networking that finds many useful applications in firewalls, Intrusion Detection Systems (IDS), Virtual Private Networks (VPNs), and Software-Defined Networking (SDN) [3, 4]. This involves comparing information contained in a packet's header to a certain rule set for determining how the packet should be processed [1]. The emergence of high-speed data center networks, cloud computing, and the implementation of SDN on a grand scale has made it necessary for packet classifiers to perform their function at extremely fast speeds with minimal memory usage [5].

Current packet classification methods can be broadly classified into hardware-assisted and software-assisted types [1, 4, 6]. While the hardware-based approaches, especially the one based on TCAMs, have the advantage of performing lookups in constant time, they are expensive, require high energy, and lack scalability [6-9]. Software-based approaches like decision tree-based methods (HiCuts, HyperCuts, HyperSplit) and hashing-based approaches (Tuple Space Search, Tuple Tree), however, offer flexibility and lower costs [2, 10-14]. But there are some problems associated with these approaches as well. The problem with the current approaches is that almost all the current algorithms use static heuristic algorithms for rule partitions that make them rigid with regard to different and dynamic rule sets [15, 16].

This limitation shows another area where further research is needed, and that is in the area of adaptability in packet classification algorithms. Contemporary networks require intelligent systems that will be capable of adapting to changes in the rules or in the nature of the traffic itself. In this situation, the role of machine learning becomes particularly useful, because it uses data for decision making [17]. However, despite its benefits, ML in packet classification is still poorly researched [17, 18].

Despite the significant advances that have been made in the parcel class through the selection tree-based hash-based algorithms, the majority of the approaches in use currently use predetermined inference rules in determining the partition size, the cutting criteria and tree shape. These rigid approaches reduce the adaptability of the algorithms due to the rule-based properties used being enforced across the communities. More adaptability is offered by recent knowledge acquisition-based methods, which apply reinforcement learning to the process of building trees using learned knowledge, but at the expense of higher overhead, computational complexity, and optimization time. A lightweight record-driven method for packet

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

classification that may automatically grow into many rule blocks is therefore required. For tackling these problems, in this paper, a novel Adaptive Partition-Hash Tree (APHT) is proposed. APHT is a machine learning-driven packet classification scheme that employs a Random Forest Regressor to learn the optimal parameter settings of the tree based on the characteristics of the rule set. This can include rules per table, rule prefix lengths' distribution, entropy, density of rules, and tuple distribution of the rule sets [19]. By employing a learning approach, APHT adjusts its partitioning and hashing schemes to suit the data at hand, thereby providing an adaptive mechanism that yields a hybrid classification algorithm [13].

Unlike heuristic-based traditional algorithms used for packet classification, the APHT algorithm uses machine learning along with the hybrid partition-hash tree approach. Here, the optimal values of construction parameters such as split dimension, partition threshold, and leaf size are predicted using a random forest regression technique, based on the statistics of the rules. This adaptivity allows for dynamic tree construction, prevents unnecessary memory accesses, avoids latency, and enhances scalability with different ClassBench benchmarks without any computation cost involved with reinforcement learning techniques.

In packet classification, packets are matched against rules by following the tree learned from the rule set [1, 12]. The effectiveness of the proposed approach is tested on ClassBench datasets [20], such as ACL, Firewall, and IPv4/IPv6 Classless Inter-Domain Routing (IPC), with different rule sizes. The results show that APHT outperforms conventional TupleTree-based methods in terms of throughput (10-15% improvement), lookup time (12-18% improvement), and memory usage (5-8% improvement).

This paper's primary contributions can be listed as follows:

- An innovative Adaptive Partition-Hash Tree (APHT) structure that integrates decision-tree partition and hash-based search techniques for efficient multi-field packet classification.
- An adaptive prediction model based on Random Forest that computes best tree structure parameters automatically by analyzing rule-set properties.
- A compact adaptive tree construction algorithm that enhances throughput and lowers lookup delay and memory utilization.
- Comprehensive performance study conducted on ClassBench ACL, Firewall, and IPC rule sets varying from 1K to 100K rules in size.
- Performance comparison highlighting superiority of APHT approach in terms of higher throughput, less lookup delay and memory usage than current state-of-the-art packet classification algorithms.

The novelty of this paper lies in the design of a new APHT model, the use of machine learning to dynamically predict parameters, the design of a hybrid partition-hash tree structure, and the experiments.

The paper is structured as follows: Section 2 surveys the literature; Section 3 outlines the system design; Section 4 explains the methodology; Section 5 explains the algorithm; Section 6 presents the experimental results; and Section 7 concludes the paper and outlines potential future work.

2. Background and Related Work

Packet classification is a well-studied problem in the field of networking due to its application in routing, firewall filtering, and Software-Defined Networking (SDN) [21]. The rise of high network traffic because of cloud computing, the Internet of Things, and the secure transfer of data has made this process more challenging [4]. Thus, a number of approaches have been proposed to improve the efficiency, scalability, and memory consumption of classification. These include decision tree, hashing, and machine learning techniques [2, 12, 17, 22, 23].

Due to the advancements in cloud computing, SDN, NFV, edge computing, and 5G/6G technology, the difficulty involved in packet classification has increased tremendously. The modern packet classifiers are supposed to achieve all of the following: high throughput, fast lookup time, low memory consumption, and fast update time despite scalability problems in the rule set. This has led to the development of many optimization methods that make use of decision trees, hashing, and machine learning.

2.1. Background

Packet Classification entails designing a classifier that uses a collection of rules to perform its tasks [11]. Every rule consists of a collection of matching fields, a priority, and a defined action. Packets are classified according to the rules using matching fields such as Source IP address (SIP), Destination IP address (DIP), Source Port number (SP), Destination Port number (DP), and protocol type; the combination is called the five-tuple [2, 12, 24].

Each rule defines a region within a multi-dimensional space, while Packet Classification can be viewed as locating the highest priority rule among the rules that match the incoming packet. The rules have the property that a high-priority rule does not cover a low-priority rule. Most benchmarks, such as ClassBench, have this property [20].

As an example, Figure 1 shows an adapted rule-set with 2 fields from a packet-classification application [1]. Geometrically, packet classification can be viewed as a multi-dimensional search problem where each rule is a hyperrectangle, and each packet is a point [12].

Table 1. A two-field example classification rule set

Rule #	Priority	Field X	Field Y	Action
R_1	6	000*	0***	Forward
R_2	5	001*	0***	Forward
R_3	4	00**	****	Enqueue
R_4	3	01**	010*	Forward
R_5	2	01**	011*	Forward
R_6	1	01**	****	Enqueue
R_7	0	****	****	Drop

2.2. Decision Tree-based Solutions

The packet classification is implemented using decision tree algorithms by space decomposition that involves recursive partitioning of the rule space using the process of creating nodes in a tree structure. In this regard, the space to be decomposed is referred to as the root node, and a compressed set of rules to compare against is in the leaf nodes. Several partitioning mechanisms can be employed, such as in the case of HiCuts [10], which uses equal-size cutting and leads to deeply structured trees.

With respect to the use of multidimensional cutting, the HyperCuts technique [12] does not lead to deep trees but requires a lot of replication. HyperSplit [26] combines boundary-based splitting to achieve short trees at the expense of higher memory requirements, while EffiCuts [25] uses distribution-based partitioning to eliminate the need for replication. There are also techniques such as CutSplit [27], which combine cutting and splitting, as well as bit-level-based algorithms such as BitCut [28, 29] and MBitTree [30] that make use of selected bits.

Recently, researchers have tried to optimize decision tree construction by decreasing the tree depth, minimizing redundancy of rules, and decreasing memory access. The MultiSplit technique [42] employs an equivalent priority grouping scheme, adaptive splitting, and intra-level binary search for creating shallow decision trees with minimal memory access. From the experimental results, it is observed that the MultiSplit technique has better lookup performance and uses less memory than traditional decision tree techniques. Like earlier techniques, MultiSplit also requires the use of predefined heuristics for rule partitioning. Recently developed decision tree based algorithms.

Further work has focused on further optimizing tree-based rule separation and reducing memory requirements. For example, MultiSplit [42] incorporates similar-priority grouping combined with multiple splitting and intra-layer binary search to minimize unnecessary look-up operations and memory accesses, even though the tree construction is still guided by some preset algorithmic approach, rather than learnt from the features of the rule set.

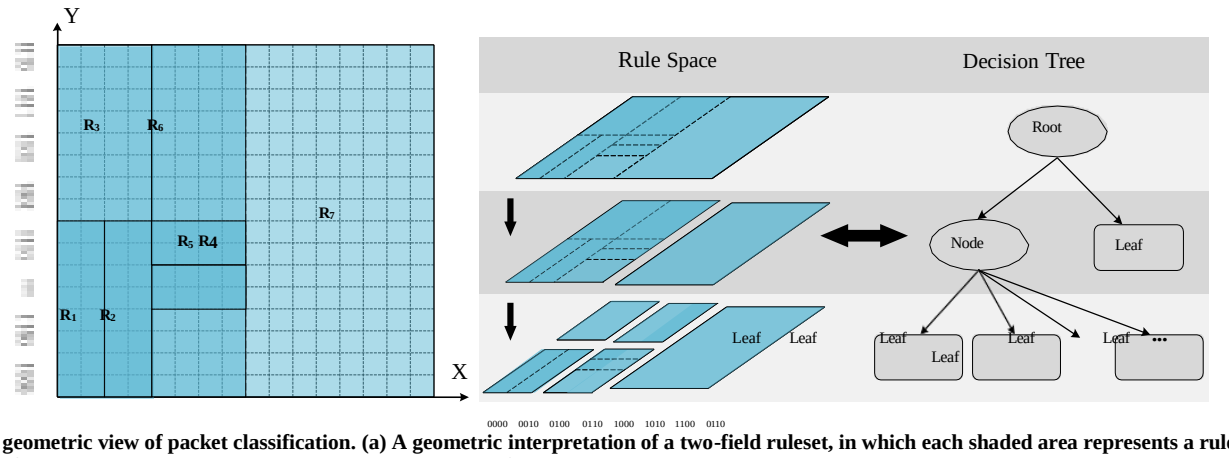


Fig. 1 A geometric view of packet classification. (a) A geometric interpretation of a two-field ruleset, in which each shaded area represents a rule from the classifier space, and (b) The decision tree approach to classification, showing how the rule space is recursively partitioned into smaller sub-spaces for effective packet classification [2, 6].

2.3. Hash-based Solutions

In hash-based packet classification algorithms, the main goal is to classify packets quickly and efficiently in terms of space utilization by organizing rules in tuples and classifying them using hashing. Tuple Space Search (TSS) [2, 31] divides rules according to prefix lengths, and then tuples are assigned to hash tables. Although the method allows fast updates and linear storage, several searches in hash tables can result in high delays. This limitation is overcome in TupleMerge [16] through merging tuples in order to reduce the number of searches required. However, hash search and large tuples may negatively affect performance. The DynamicTuple [2]

algorithm employs dynamic programming to generate tuples that can be classified efficiently depending on the distribution of rules. Priority-based pruning operations are performed to reduce searches. However, scaling may become a problem as a consequence of hash collision and memory access problems [1, 12].

Recently, a new generation of tuple-based approaches have incorporated learning for assist with the problem of scalability, namely learning-based partitioning. The learning-based partitioning technique, named Learning Tuple, utilizes SARSA-based reinforcement learning for recursive partitioning of the tuple-space, employing a strategy that depends on rule overlap and the data of hash collisions [43].

The implementation shows positive aspects regarding classification and updating capabilities; however, the integration of learning algorithms can also produce a more complex set-up for the algorithm, and it requires training and more optimizations as well.

Packet classification approaches based on hashes have been further enhanced through the use of adaptive tuple arrangements, hybrid hashing, and cross-producting schemes. They help eliminate unnecessary access to the hash table and scale well for rule set sizes that are larger. However, they are still sensitive to manual tuple selection and static partitioning strategies.

2.4. Machine Learning-based Approaches

The development of machine learning techniques has facilitated the data-driven optimization of packet classification. Specifically, the NeuroCuts [32] approach leverages deep reinforcement learning to optimize decision tree construction and yields better performance in packet classification than some of the existing heuristics; nevertheless, this technique is characterized by the requirement of extensive training and computational efforts.

The NuevoMatch [33] scheme uses the learned index structure according to the RQ-RMI approach to compress the rules and increase the cache efficiency. Using model inference instead of multiple memory accesses, NuevoMatch achieves better throughput and memory scalability in comparison with several advanced solutions.

The HybridTSS [34] approach exploits the combination of recursive tuple-space search and reinforcement learning to optimize tuple partitioning while ensuring efficient updating of rules. In spite of the fact that HybridTSS improves the lookup performance, this approach is still associated with computational complexity due to reinforcement learning.Besides the packet classification data structures, deep learning algorithms such as CNN, LSTM, GNN, and Transformer have been recently studied to classify the network traffic and to analyze the encrypted traffic [35, 36]. These approaches have shown a high prediction accuracy in the traffic classification process, but they have mainly been used in traffic analysis than in packet classification data structures. The application of these machine learning techniques yields excellent results in terms of traffic classification accuracy but is mainly oriented to traffic analysis rather than data.

Even with these improvements, there has been no substantial study in lightweight supervised learning approaches for adaptive packet classification. The algorithms that currently exist are either dependent on static heuristics or complex reinforcement learning.

It is for this reason that the Adaptive Partition-Hash Tree (APHT) approach was developed using the Random Forest regression algorithm to estimate optimal parameters for partitioning from the rule set attributes.

2.5. Research Gap

In past years, the efficiency of packet classifiers has been improved through decision tree partitioning [3, 6], tuple-based hashing [8] and learning based approaches [41]. In general, decision tree and hash based approaches use off-the-shelf heuristics for selecting partition and hash function parameters.

As a result, the performance could differ under the variation of rule set sizes, prefix distributions, rule densities, and tuple characteristics. The learning-based method has stronger adaptability over it, nevertheless, the prior state-of-the-art machine learning approaches are largely focusing on either reinforcement learning or learned index structures [43, 44].

Those reinforcement-learning-based methods can enhance classifier construction and adaptation; however, training and fine-tuning classifiers are computationally cost added due to the iterative computations [43, 44]. The learned-index mechanism can minimize memory accesses via inference of a model, yet employ specific data indexing structures and training procedures.

There is still a lack of a lightweight supervised learning method to determine the construction parameter set for a hybrid partition hash classifier dynamically on the basis of rule-set statistics [33]. With this motivation, we develop APHT, an architecture including a Random Forest Regressor together with a partition hash tree.

The model will estimate candidate configurations with rule set statistics and choose the configuration with lowest predicted cost. This will guarantee selection of parameter set in an adaptive and automated fashion, while move computations off to offline trainings.

3. Machine Learning-Guided Adaptive Partition-Hash Tree (APHT)

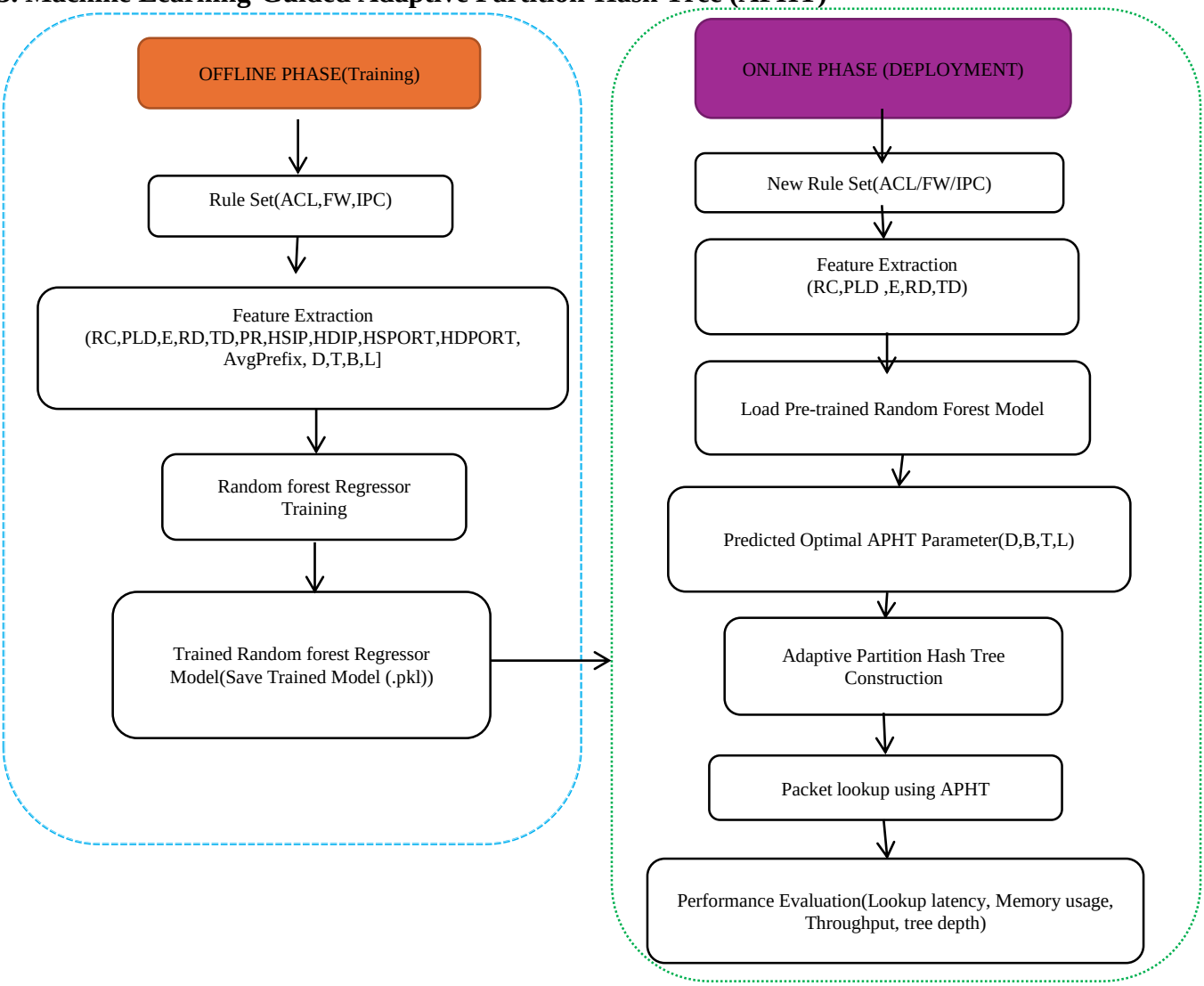


Fig. 2 APHT architecture for packet classification

The APHT framework guided by Machine Learning (APHT-ML) involves two key stages: (i) Offline Phase (Training) and (ii) Online Phase (Deployment). The goal of the Offline phase is to prepare the training dataset and learn the correlations between the characteristics of rule-set, construction parameters of APHT, and the performance outcome of the APHT. Later, in the Online Phase, the trained Random Forest regression model is used to decide an appropriate construction parameter for the APHT in the test time for a new rule-set.

The proposed method aim to automate the selection of (D, T, B, L) to build an APHT that offers lookup latency, memory footprint, number of memory accesses, and depth and achieves high classification accuracy in terms of packet lookups. Figure 2 illustrates the complete workflow of the proposed ML-guided APHT framework.

3.1. Offline Phase (Training)

The offline phase is carried out prior to deployment to create training samples and training a Random Forest Regressor (300 trees) to intelligently select the most suited APHT configuration.

3.1.1. ClassBench Rule Sets

The proposed method use ClassBench benchmark rule sets, which include ACL, FW, and IPC. The representation of each rule is based on 5 packet fields: source IP, destination IP, source port, destination port and protocol.

Let $R = r1, r2, \ldots, rN$

Represent the input rule set, where N denotes the total number of packet classification rules.

3.1.2. Feature Extraction of Rules Set

To characterize the statistics, the features were extracted of each rule set, which has 11 features.

F=[RC,PLD,E,RD,TD,PR,HSIP,HDIP,HSPORT,HDPORT, AvgPrefix]

Where RC denotes the Rule count, PLD is the Prefix length distribution, E is the entropy, RD is the Rule Density, TD is the Tuple distribution, and PR represents the Port range ratio. The rest are the field-wise entropy and the average Prefix length.

3.1.3. APHT Configuration Candidates

The next stage generates different candidate configurations for constructing the APHT.

Four APHT parameters are considered:

C= (D, T, B, L)

Where D stands for the split dimension, T for partition threshold, B for hash-bucket size and L for max leaf size, Therefore 4* 4* 4* 3 = 192 different combinations (configurations) for each rule set.

3.1.4. Generating Training Data

The 11 features extracted from the rule set and the different APHT parameters for generating 15 dimensions vector, which will be the input for the Random Forest regressor.

X=[RC,PLD,E,RD,TD,PR,HSIP,HDIP,HSPORT ,HDPORT, AvgPrefix, D,T,B,L]

Thus, for each rule set, 192 training data samples will be generated.

3.1.5. Evaluating APHT Performances

For each candidate combination, the proposed method evaluates the lookup time, memory usage, memory accesses and the depth of the tree of the created APHT. The results obtained have been represented by a performance score:

S=w1L+w2M+w3MA+w4D

Where w1,w2,w3,w4$ are weight coefficients. A performance score S indicates a better-performing APHT at it is smaller.

3.1.6. Random Forest Training

The obtained training samples composed of the 15-dimensional vector (X) and the correlated score (S) are then used to train the RFR that learns the relation:

S^=f(F,D,T,B,L)

The RFR is composed of 300 decision trees trained with feature selection, hyperparameter tuning and cross-validation.

3.2. Online Phase (Deployment)

When a new ACL, Firewall or IPC rule set is supplied (an event that signals the beginning of the online phase), the goal is to automatically choose the right APHT configuration by using the pre-trained 300-tree Random Forest Regressor (RFR). Specifically, the same 11 statistical features which were used during training for classifying rule sets are calculated for the new rule set. Combined with the 192 candidate APHT configurations (Cj= (Dj,Tj,Bj,Lj)), this gives 192 input vectors:

X= [F1,F2,…,…….F11,D,T,B,L]

The RFR, after having been trained previously, returns the predicted performance for each candidate APHT configuration.

S^j=RF(Xj)

And the optimal APHT configuration ((D^,T^,B^,L^)) is chosen

$$(D*, T*, B*, L*) = \arg(D, T, B, L)\min RF(Fnew, D, T, B, L)$$

The proposed framework is thereby able to find the best configuration without actually constructing and testing all candidates during runtime. The RFR model quickly predicts the performance and picks the one with the minimal predicted cost.

3.2.1. APHT Building and Rules Classification

APHT is constructed adaptively based on the parameters estimated from the data (D, T, B, L). The split dimension decides which field is chosen to partition; partition threshold defines how the partitioning process proceeds recursively; hash bucket size sets the arrangement of hash nodes; while maximum leaf size sets the stop condition of the tree construction process.

Partition node (P-node): This type of node partitions rules in the rule space based on the selected splitting dimension, hence narrowing down the search space [12].

A partition node is generated if $|R|>T$

Hash node (H-node): This is a node in which rules are hashed in buckets for fast searching purposes [2].

A hash node is generated if $L<|R|\leq T$

Creation of the tree is performed recursively through estimation of the parameters based on machine learning principles and creation of a balanced tree without duplication of rules. When the number of rules in a certain node drops below the estimated threshold value, then a leaf node is created.

A leaf node is generated if $|R| \leq L$

During the classification process, packets traverse the entire APHT tree starting from the root node. Traversal of nodes is dependent on the schemes utilized in the creation of the tree through partitioning and hashing. Traversing nodes is less resource-intensive in terms of accessing nodes as well as memory access. Eventually, at the leaf node, some rules will be accessed using linear search method. The hybrid and adaptive nature of the APHT has made its implementation efficient in terms of classification delay and throughput.

3.2.2. Performance Evaluation

The performance of the proposed APHT framework is evaluated with different sizes of ClassBench ACL, FW, and IPC rule sets. The performance is reported in terms of lookup latency, classification throughput, and memory usage, memory accesses per lookup, tree depth, and classification accuracy. The results show that utilizing machine learning for APHT parameter estimation makes it efficient for various types of rule sets in terms of tree construction efficiency, memory footprint and packet classification performance.

4. Algorithm Design

Algorithm 1. APHT-based Packet Classification Framework

Adaptive Partition-Hash Tree (APHT) architecture that makes use of feature-based optimization, along with hybrid data structure generation, is introduced to perform effective multi-field packet classification. The entire process involves four different steps: feature extraction, learning-based parameter generation, adaptive tree generation, and packet classification.

Given a rule set

$R = \{r1, r2, \ldots, rN\}$

Each rule is represented as a five-tuple:

$ri = (SIP, DIP, SP, DP, PR)$

Where SIP, DIP, SP, DP, and PR denote source IP, destination IP, source port, destination port, and protocol, respectively

A. Feature Extraction

The rule set is transformed into a feature vector:

$$X = [F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11]$$

Where

$F1 = |R| = N$ (Rule Count)

$F2 = \{l1, l2, \ldots, lN\}$ (Prefix Length distribution)

$F3 = -\sum p(xi) \log 2\, p(xi)$ (entropy)

$F4 = N/\text{Search Space}$ (Rule density)

F5=distribution of rule tuples

F6= Port Range Ratio (PR)

F7= SIP Entropy

F8= DIP Entropy

F9= SPORT Entropy

F10= DPORT Entropy

F11= Average Source/Destination Prefix

These features collectively provide a compact representation of the rule set and serve as input to the machine learning model.

B. Random Forest-Based Performance Prediction

The 15-dimensional candidate vectors are supplied to a pre-trained Random Forest Regressor containing 300 decision trees**.**The Random Forest predicts the expected performance score of each candidate configuration.

$S_j = M_\{RF\}(X_j)$

Where ($M_\{RF\}$) denotes the trained Random Forest model.

For a Random Forest containing (m=300) trees, the predicted score is calculated as

The Random Forest prediction is computed as

$$\hat{Sj} = 1/300 \sum_{t=1}^{300} \text{Ti(Xj)}$$

Where:

S^j = Predicted performance score for the jth APHT candidate configuration

300 = Number of decision trees in the Random Forest

Ti (Xj) = Performance score predicted by the ith decision tree for candidate input Xj

Xj = 15-dimensional input vector for the jth candidate configuration

The Random Forest therefore predicts the performance score**,** rather than directly predicting (D,T,B,L).

C. Adaptive Tree Construction

The APHT is constructed recursively as:

T=BuildAPHT(R,D,T,B,L)

Where the tree consists of partition nodes (P-nodes), hash nodes (H-nodes), and leaf nodes.

D. Packet Classification

Each packet is represented as:

$$P = (PSIP, PDIP, PSP, PDP, PPR)$$

Traversal rules:

Partition node:

$node \leftarrow fd(P)$

Hash node:

$node \leftarrow h(Pd) \bmod k$

Final Matching Rule

$$r^* = \arg \max_{ri \in Rleaf} priority(ri)$$

Input: Rule set R={r1,r2,…,rN}
Output: APHT tree T and classified packets

1. Load rule set R
2. Parse each rule ri into (SrcIP, DstIP, SrcPort, DstPort, Protocol, Priority, Action)
3. Extract rule-set statistical features
   F1 ← RuleCount(R)
   F2 ← PrefixLengthDistribution(R)
   F3 ← Entropy(R)
   F4 ← RuleDensity(R)
   F5 ← TupleDistribution(R)
   F6 ← PortRangeRatio(R)
   F7 ← SIP Entropy
   F8 ← DIP Entropy
   F9 ← SPORT Entropy
   F10 ← DPORT Entropy
   F11 ← Average Source/Destination Prefix
4. Construct feature vector
   X ← [F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11]
5. Generate 192 candidate configurations:
   C= {C1,C2,…,C192}
6. Construct the 15-dimensional input vector:
   Xj=[F1,F2,…,F11,Dj,Tj,Bj,Lj]
7. Predict the performance score using the trained Random Forest: S^j=MRF(Xj)
8. Select the candidate configuration having the minimum predicted performance score:
   C∗=1≤j≤192argminS^j
9. Predict APHT construction parameters
   [D, T, B, L] ← M(X)
   where
   D = Split Dimension
   T = Partition Threshold
   B = Hash Bucket Size
   L = Maximum Leaf Size
10. Construct the APHT
    T ← BuildAPHT(R, D, T, B, L)
    ▷ Refer to Algorithm 2
11. for each incoming packet P do
12. node ← Root(T)
13. while node is not a Leaf Node do
14. if node is a Partition Node then
15. node ← CorrespondingChild(P[D])
16. else if node is a Hash Node then
17. bucket ← hash(P[D]) mod B
18. node ← Child(bucket)
19. end if
20. end while
21. Search rules stored in the leaf node
22. Select the highest-priority matching rule
23. Apply the corresponding forwarding action
24. end for
25. return T

Algorithm 2: APHT Tree Construction

The APHT tree is constructed recursively by combining partitioning and hashing techniques, which depend on machine learning-based parameters. Instead of using the heuristic-based approaches employed in standard decision trees, APHT uses parameters predicted by machine learning, such as the split dimension d, partitioning threshold θ, and the number of leaves L.

The main goal is to construct a tree that has low memory accesses and classification latency, and that can scale to accommodate a huge number of rules.

A. Leaf Node Condition

The construction process begins with the complete rule set:

$$R = \{r1, r2, \dots, rN\}$$

The node will be a leaf node if:

$$| R | \leq L$$

The rule set will be stored at the leaf node:

$$Nleaf = R$$

This will stop the further breakdown, and the tree will be a finite level.

B. Partition Node (P-node)

If the rule set has more than the partition threshold number of rules:

$$|R| > T$$

Then a Partition Node (P-node) will be created.
The rule set will be split into disjoint subsets:

$$\bigcup_{i=1}^{k} Ri \quad \text{Ri} \cap \text{Rj} = \emptyset (\text{i} \neq \text{j})$$

Each subset will be treated as follows:

$$\langle r \in R \mid rd \in Si \rangle$$

d: dimension for subset split
Si: partitioned subspace
The subsets will be further processed recursively.

C. Hash Node (H-node)

If the size of the rule set is less than the partition threshold and larger than the leaf size:

$$R| \leq T$$

Then a Hash Node (H-node) will be created.
The rule set will be partitioned into buckets: $b = (rd) \bmod k$
The rule set will become:

$$R = \cup_{b=0}^{k-1} Bb$$

Each bucket will be recursively processed.

D. Node Selection Strategy

$$\text{Node Type} = \{ \text{P-node}, |R|>T; \text{H-node}, |R| \leq T$$

This hybrid approach will allow that:
Effective rule set splitting Fast search via hashing in medium-sized sets

E. Recursive Construction

T=BuildAPHT(R,D,T,B,L)

The resulting tree consists of:

Partition nodes

Hash nodes

Leaf nodes

F. Complexity

$$\text{Lookup Complexity} = O(\text{tree depth} + \text{leaf search})$$

Since a leaf node consists of a few rules, the searching time is trivial.

Algorithm 2: BuildAPHT

Inputs: R (set of rules), d (dimension on which splitting will be performed), T (threshold), B(Bucket size),L (leaf size)

Outputs: APHT node

1: BuildAPHT(R, D, T,B, L)
2: If |R| ≤ L, then
3: Create node N_leaf
4: Set N_leaf's payload to R
5: return N_leaf
6: end if
7: If |R| > T, then
8: Create partition node
9: Create node P
10: Split R into subsets {R1, R2, …, Rk}
11: for all i from 1 to k do
12: child_i ← BuildAPHT(Ri,D,T,B,L)
13: Attach child_i to P
14: end for
15: return P
16: else
17: ▷ Create hash node
18: Create node H
19: for all rules r ∈ R do
20: b ← h(r_d) mod k
21: Add r to bucket B_b
22: end for
23: for all buckets B_i do
24: child_i ← BuildAPHT(Bi,D,T,B, L)
25: Attach child_i to H
26: end for
27: return H
28: end if
29: end function

4.1. Packet Traversal and Classification

The incoming packet undergoes a traversal operation in the APHT, which starts from the root node and involves making decisions based on the node types. The traversal through the P-node proceeds by traversing the split dimension ddd to move to the relevant child node according to the value of the corresponding field.

On the other hand, for H-nodes, the traversal is accomplished by hashing the particular field of the packet to determine the relevant node. This operation recurs until a leaf node is reached. In the latter, the following set of rules is evaluated by a linear search operation to select the most prioritized rule:

$$r^* = \arg \max_{ri \in Rleaf} priority(ri)$$

The above approach considerably shrinks the size of search space and number of memory accesses, thus increasing classification efficiency.

4.2. Complexity Analysis and Comparison

Complexity analysis of the proposed Adaptive Partition-Hash Tree (APHT) largely depends on the tree construction and rule distribution used.

4.2.1. Worst-Case Complexity

In the worst-case scenario, where a highly imbalanced tree is formed due to overlapping rules or poor partitioning, the classification process could deteriorate into an O(N) problem, as shown below.

$$O(N)$$

Where N is the total number of rules.

4.2.2. Average-Case Complexity

Due to machine learning-guided parameter selection, APHT normally results in a well-balanced tree with logarithmic depth.

Given that there can be at most L number of rules in each leaf node, the resulting lookup time complexity is given by:

$$O(\log N + L)$$

where:

log N: Tree traversing cost

L: Linear search cost inside the leaf node.

5. Methodology and Experimental Setup

5.1. Simulation Environment

The performance evaluation of the proposed Adaptive Partition Hash Tree (APHT) is carried out using the ns-3 network simulator, which allows a true packet-level emulation of a network for experimentation purposes.

The implementation of the APHT Packet Classifier is carried out using the C++ language, which has been incorporated into the ns-3 simulator [38-40].

The simulation was carried out using a computer running on an Intel(R) Core(TM) i5-10505 CPU @ 3.20GHz processor and 16GB RAM. The system runs on the Ubuntu 20.04 operating system.

All implementations done in C++ have been compiled using the GCC v9.4.0 compiler with C++11 support. All simulations have been performed under single-threaded mode.

5.2. Machine Learning Model Configuration

The machine learning portion of the proposed APHT uses the Python-based Scikit-learn library with a Random Forest Regressor of 300 decision trees. Eleven statistical rule-set features (rule count, prefix length distribution, entropy, rule density, tuple distribution, port-range ratio, source/destination IP and port entropy, and avg. Prefix length) along with four parameters for APHT: Split Dimension (D), Partition Threshold (T), Hash Bucket Size (B) and Maximum leaf size (L). These are concatenated into a 15-dimensional feature vector where each of the 192 possible configurations represents one input point.

A value Sj = MRF(Xj) is predicted, where M is the Random Forest Regressor for each configuration xj. The minimum predicted score gives the optimal configuration. C = argminj=1..192Sj is chosen, for which the combination of features [D, T, B, L] are returned, the APHT configuration is created with these parameters, adapting itself without needing manual heuristic selection of the parameters.

5.3. Rule Set and Traffic Generation

In this experiment, the datasets are created by using the ClassBench tool [20] that generates synthetic rule sets, which are designed to be highly realistic in terms of representing the actual network policies used in practice. The rule sets are classified into three different types, including Access Control List (ACL), Firewall (FW), and IP Chain (IPC). Each type of rule set comes in three different sizes: 1K (Small-scale), 10K (Medium-scale), and 100K (Large-scale). Thus, there are nine different datasets used in this research: ACL_1K, ACL_10K, ACL_100K, FW_1K, FW_10K, FW_100K, IPC_1K, IPC_10K, IPC_100K. The traffic trace files are created for each of the aforementioned datasets using the ClassBench trace generator. Specifically, the number of packets is set to ten times the number of rules in each rule set, e.g., ACL_10K contains 100,000 packets, and IPC_100K includes 1,000,000 packets.

5.4. Comparative Algorithms

The suggested APHT approach is compared to some well-known packet classification algorithms such as HiCuts [10] , HyperCuts [12], HyperSplit [26], TupleMerge [16], CutSplit [27], CutTSS [41], and TupleTree [13]. Each of the above-mentioned algorithms was implemented in its default configuration, as described in the original papers on these approaches.

5.5. Performance Metrics

All algorithms are evaluated based on four fundamental metrics, namely, throughput (measured in millions of packets per second, Mpps), lookup delay (measured in microseconds, µs), memory utilization (measured in kilobytes, KB), and classification delay, which is the total processing time taken by the algorithm for the whole packet set.

6. Results & Discussion

The proposed Machine Learning-Guided Adaptive Partition-Hash Tree (APHT) has been evaluated on nine ClassBench rule sets belonging to ACL, FW, and IPC classes on three different scales: 1K, 10K, and 100K rules. For each rule set, the set of features was extracted and presented to a Random Forest model whose task was to predict the performance of several APHT configurations.

The best predicted performance configuration was then chosen to build the APHT, and the classifier was evaluated by measuring its throughput, lookup latency, memory consumption, construction time, tree depth, number of nodes and average memory accesses.

6.1. Rule-Set Feature Analysis.

A summary of statistical properties collected from the 9 rule sets within ClassBench is shown in Table 2 (Appendix). These properties are extracted on various aspects of the rule sets, ranging from sizes to IP/port entropy and from prefix/rule density to tuple/port-range properties. Huge differences can be found both between the ACL/FW/IPC rule sets and among rule sets of different sizes. As an example, the entropy/prefix statistics differ depending on the type/size of a rule set, which clearly tells there are large differences in structure of rule spaces. This serves as a motivation to use an adaptive APHT instead of one single configuration.

The statistic properties presented are used to represent the input to an ML-based configuration selection process. Using machine learning models like Random Forest, the feature vector may be utilized to characterize a certain rule set and determine the link between the attributes of the rule set and other possible APHT parameters like split dimension, partition threshold, bucket size, and leaf size.

6.2. Random Forest Prediction Performance

The Random Forest regression model was built on 1,436 training data points and tested on the remaining 359 data points. This is reflected by 80:20 train/test split of data points used. The model uses the 15-dimensional feature vector and 300 decision trees for predicting the performance scores for the candidate APHT configurations.

MAE of 0.002093, RMSE of 0.005757, and R2 = 0.998827(coefficient of determination), which implies that the model accounts for about 99.88% of the variance in the performance score in the assessed test data. Mean prediction error of -0.000297 and maximum absolute error 0.070884, which implies that the predicted values conform very closely to the observed values.

Table 3. Random Forest regression performance

Metric	Value
Total Samples	1795
Training Samples	1436
Testing Samples	359
Number of Features	15
Random Forest Trees	300
MAE	0.002093
RMSE	0.005757
(R^2)	0.998827
Mean Error	0.000297
Maximum Absolute Error	0.070884

6.3. Actual versus Predicted Performance Score Values

Figure 3 compares the observed performance score values with the predicted performance score values. There is remarkable agreement between model estimates and observed performance score values, as depicted by the relative alignment of observed against predicted score values to the diagonal where y=x (ideal case for predicting score values) of which only a relatively few samples diverge significantly. The density of samples clustering around the diagonal is expected given the MAE, RMSE, and R-squared.

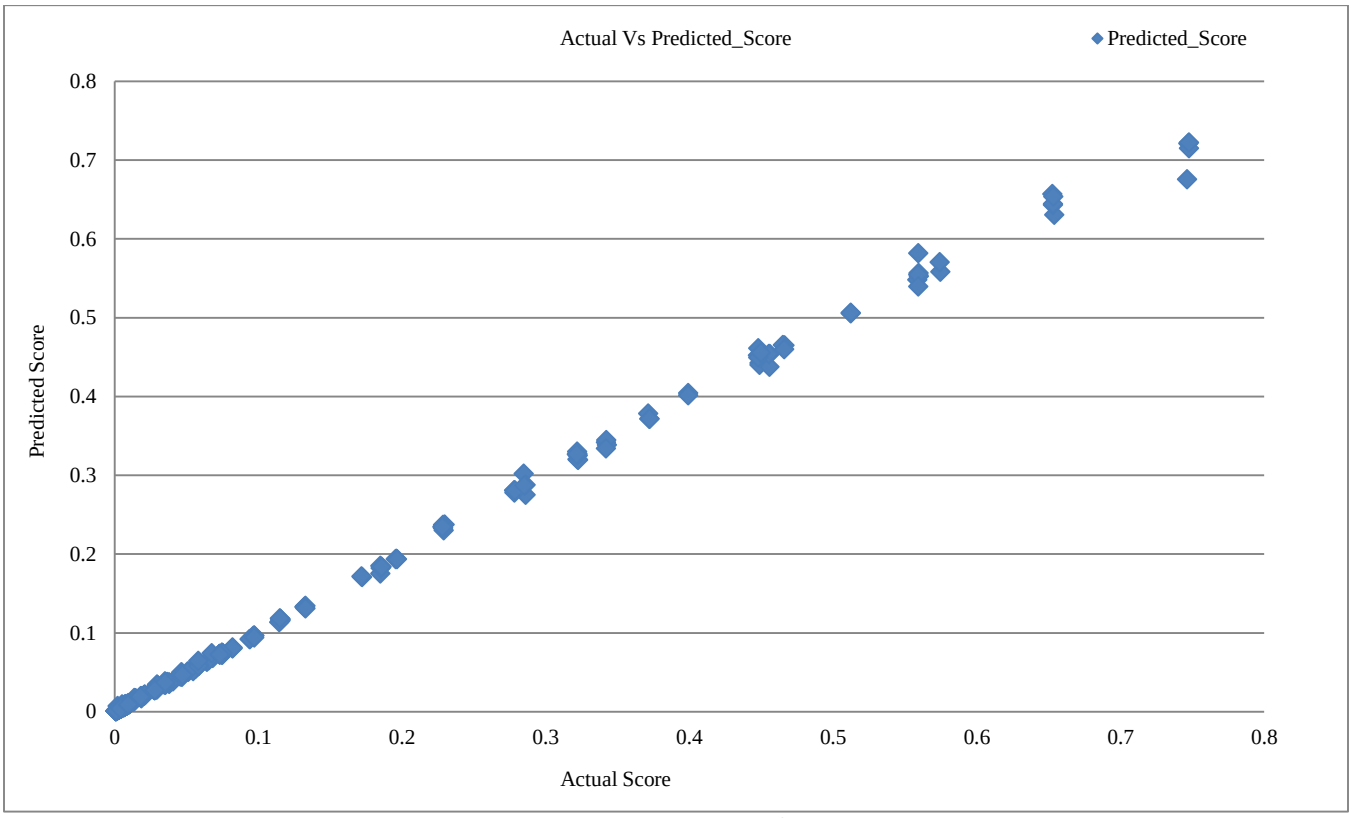


Fig. 3 Actual and predicted APHT performance scores

6.4. Analysis of Prediction Error

Figure 4 shows the distribution of prediction error of the Random Forest model. It appears that the majority of prediction errors cluster around the origin, suggesting a good agreement of predicted APHT performance scores against the true performance scores. The symmetry around the origin, approximately, suggest no strong bias toward over- or underestimation.

The majority of the prediction error falls in a narrow range, with only a few occurrences of relatively larger error. The highest absolute error is 0.070884, with very few occurrences, as the maximum error is associated to only few candidate configurations.

The MAE is quite low (0.002093) with the predicted error performance scores, indicating a very low mean prediction error level. The clustering at the origin is well related to the high (R^2) (0.998827) and small RMSE (0.005757) as provided in Table 3.

This suggests that Random Forest model generates a stable and accurate prediction all over most of the tested SPHT config space and can therefore predict the performance score and rank the candidates for a certain rule-set structure.

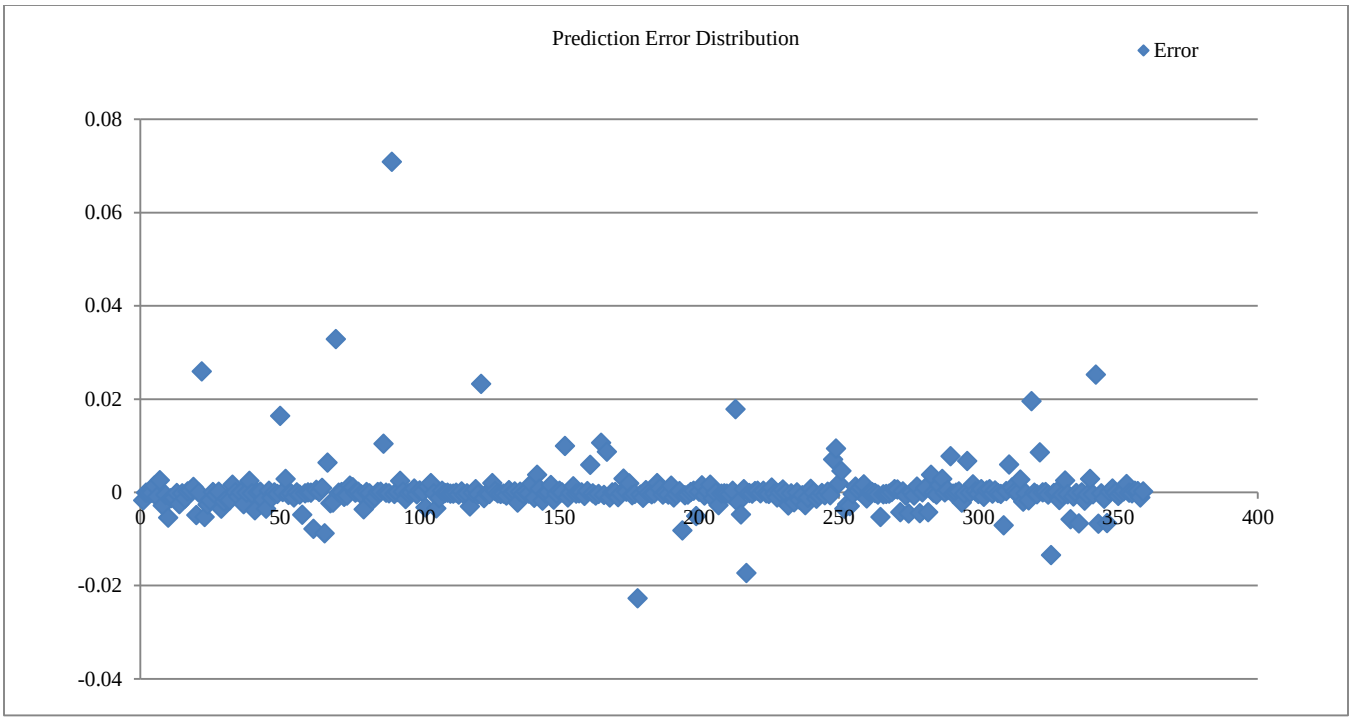


Fig. 4 Prediction error distribution of the Random Forest model

6.5. APHT Parameter Optimization Evaluation

A set of APHT configurations, chosen among those tested experiments for each of the nine considered ClassBench rule sets, is reported in Table 4. The chosen values of split dimension D, partition threshold T, hash bucket size B and maximum leaf size L vary between datasets, thus shows that the proposed framework is not sensitive to one configuration, but can work without a prefix of tree parameters on a particular set of rules. For example: whereas ACL 1K has configuration of (D=3), (T=8), (B=128), and (L=16), ACL 100K was set up with (D=0), (T=32), (B=128), and (L=4).

Table 4. Selected APHT configurations for ClassBench rule sets

Dataset	Rules	D	T	B	L
ACL-1K	942	3	8	128	16
ACL-10K	9,774	3	8	128	4
ACL-100K	99,319	0	32	128	4
FW-1K	857	0	8	128	4
FW-10K	9,379	2	32	128	16
FW-100K	88,081	0	32	128	16
IPC-1K	974	1	24	128	4
IPC-10K	9,518	3	32	64	4
IPC-100K	99,325	0	24	128	4

Table 5. Performance of the proposed APHT on ClassBench rule sets

Dataset	Build (s)	Thr. (Mpps)	Lat. (µs)	Mem. (KB)	Depth	Nodes	MA
ACL-1K	0.0002	9.849	0.1015	55.16	4	19	2
ACL-10K	0.0026	8.601	0.1163	402.59	5	30	3
ACL-100K	0.0584	7.588	0.1318	3886.32	5	25	4
FW-1K	0.0002	9.778	0.1023	48.59	5	15	3
FW-10K	0.0024	9.338	0.1071	555.78	5	30	3
FW-100K	0.0284	9.732	0.1028	5422.93	5	127	3
IPC-1K	0.0003	9.735	0.1027	47.43	5	22	3
IPC-10K	0.0021	9.683	0.1033	517.43	5	91	3
IPC-100K	0.0462	6.348	0.1575	4289.80	5	38	4

6.5.1. Throughput Analysis

APHT throughput for the nine ClassBench rule sets is given in Figure 5. The method keeps the throughput of proposed method is above 6 Mpps in all the datasets, ranging from 6.348 Mpps to 9.849 Mpps. With a rule set size of 1K, APHT attains 9.849 Mpps, 9.778 Mpps, and 9.735 Mpps for ACL, FW, and IPC, respectively. For rule set size of 10K, they are 8.601 Mpps, 9.338 Mpps, and 9.683 Mpps, respectively, and for 100K rule sets, 7.588 Mpps, 9.732 Mpps, and 6.348 Mpps for ACL100K, FW100K and IPC100K, respectively. As seen from Figure 5 while throughput declines with increases in the size of certain rule sets, this trend varies depending on

the traffic policy. Importantly, throughput on the FW100K dataset is still 9.732 Mpps, demonstrating the significant impact of rule-set structure on classification speed apart from its size. The average throughout all rule sets is about 8.96 Mpps. Based on these results, the adaptive partitioning and hash-based lookup system can effectively support up to about 100K rule sets without significantly sacrificing packet processing performance.

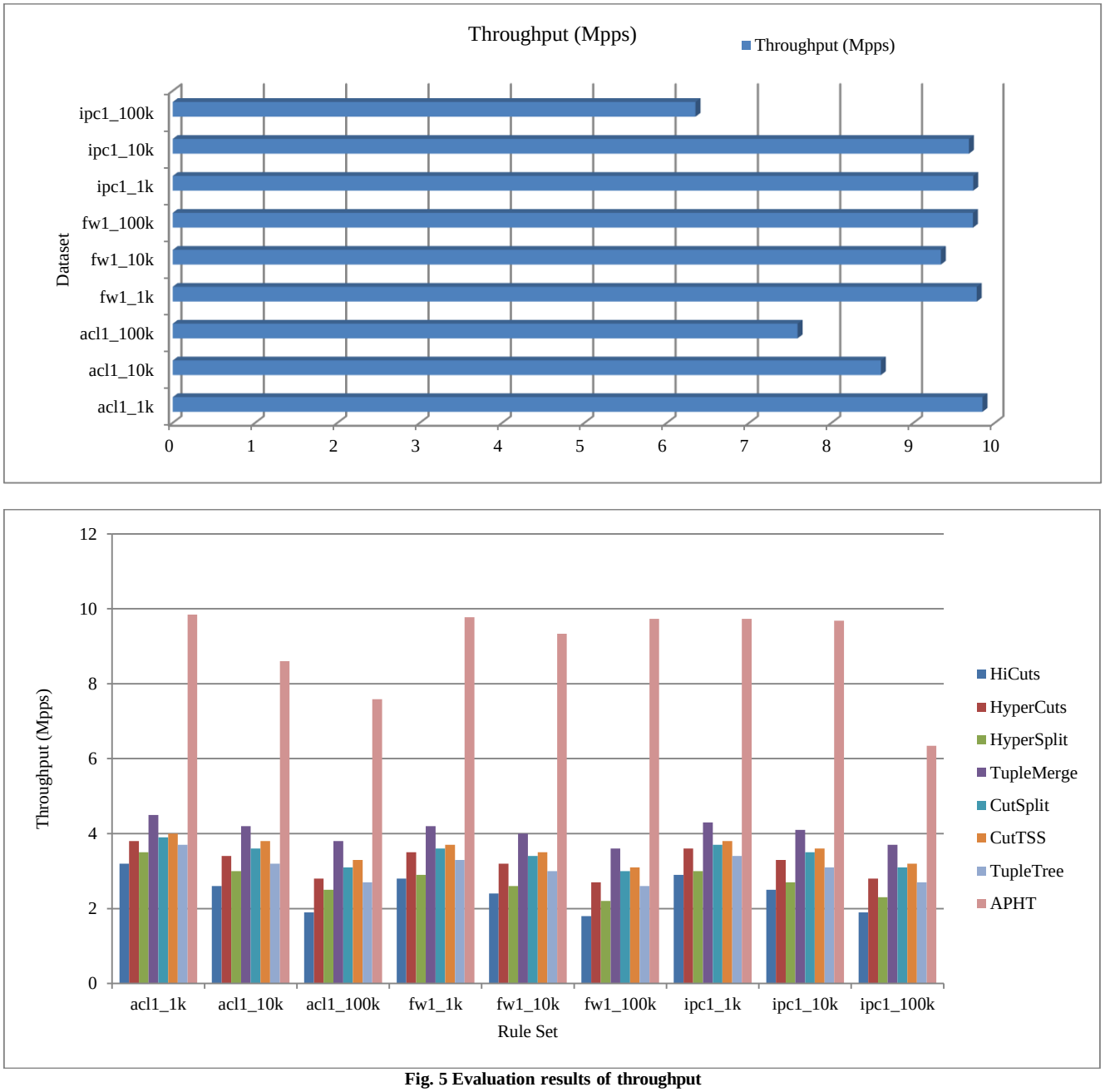


Fig. 5 Evaluation results of throughput

6.5.2. Latency Analysis

Figure 6 shows the lookup latency of APHT on the nine ClassBench rule sets. APHT consistently shows a low lookup latency on all the datasets (range: 0.101535s - 0.157521s). For the 1K rule sets, the proposed method achieve a lookup latency of 0.101535s for ACL1K, 0.102266s for FW1K, and 0.102718s for IPC_1K. For the 10K rule sets, the corresponding lookups take 0.116265s for ACL10K, 0.107088s for FW10K, and 0.103276s for IPC10K. Finally, for the 100K rule sets, APHT achieved a lookup latency of 0.131780s for ACL100K, 0.102752s for FW100K, and 0.157521s for IPC100K. Note that despite the significant increase in rule set sizes, the lookup latency never exceeds 0.16s for any of the rule sets. This relative constant lookup

latency can be attributed to the dynamic nature of APHT in terms of using partitioning (adaptively selecting splitting dimension D, threshold T, and bucket size B) and hash-based indexing (adaptively selecting leaf size L).

This greatly reduces the traversal cost and number of candidate rules that are explored. Table 5 further proves the low lookup cost in terms of the average memory-access count per packet. For the nine datasets, APHT only incurs 2-4 memory access per lookup, and the minimum cost 2 is for ACL_1K, while 3 to 4 is for other 8 datasets. Future work should involve making a qualitative comparison with decision trees such as HiCuts [5], HyperSplit [4], CutSplit [3] and hash-based schemes like TupleTree [6], TupleMerge [4], CutTSS [9] to quantitatively express the latency advantage. Decision tree-based schemes might suffer from more traversal overhead when the trees become deeper, whereas hash-based schemes might incur additional costs for exploring multiple candidate matches if the distributions of tuples and rules are uneven. APHT mitigates both issues through hybrid design. In conclusion, experimental results show that APHT performs reasonably well for rule sets from around 1K to 100K in terms of low, scalable lookup latency with less than 0.16 seconds and 2-4 memory access per packet.

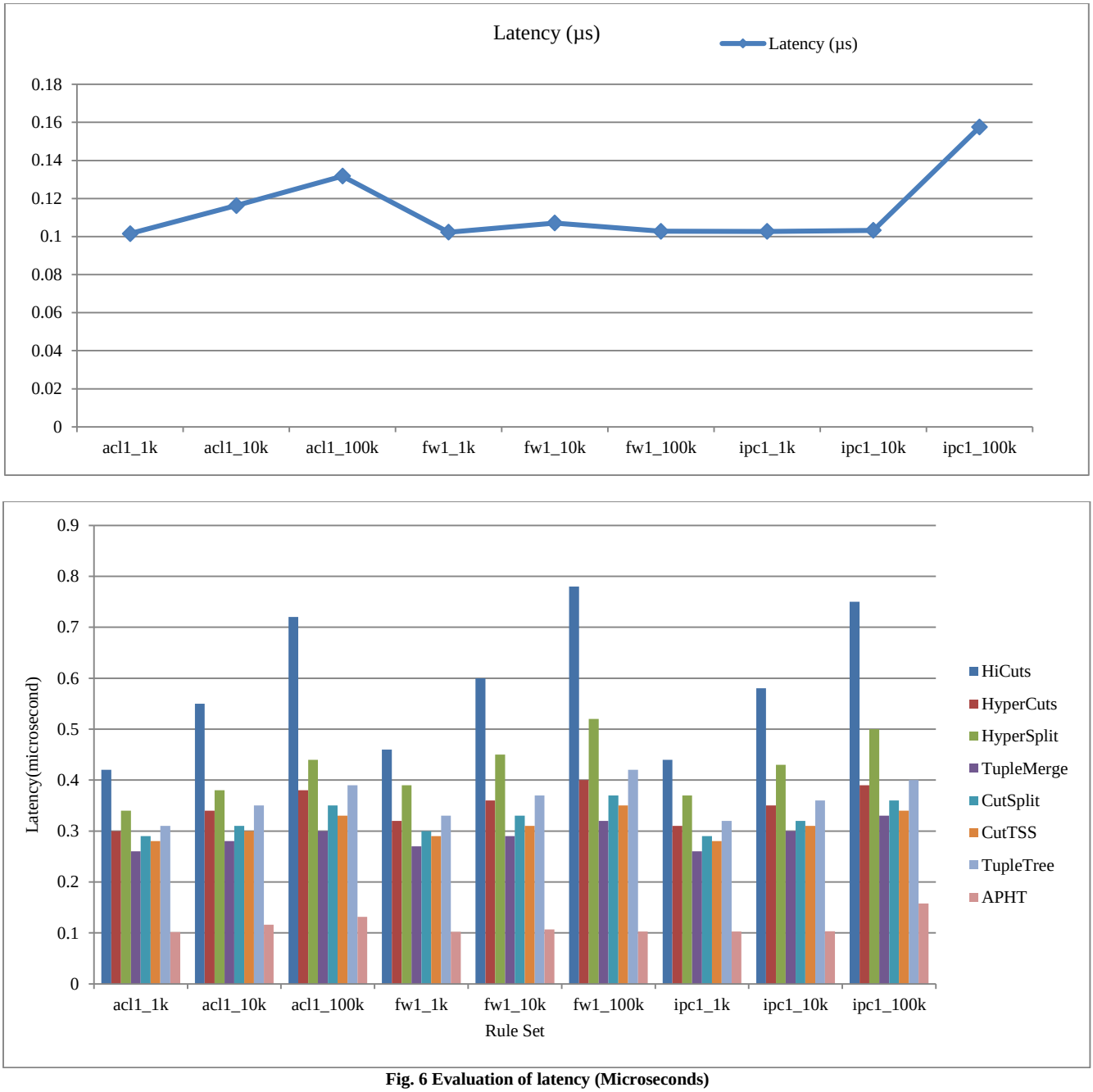


Fig. 6 Evaluation of latency (Microseconds)

6.5.3. Memory Consumption

The memory consumption is remarkably smaller in APHT because of its hybrid partition-hash method and machine learning optimization, where fewer rules are repeated, and fewer nodes are expanded unnecessarily. In terms of small data sets, APHT needs 55.15 KB (ACL_1K), 48.59 KB (FW_1K), and 47.43KB (IPC_1K). For big data sets, APHT is still efficient; consuming only 10087 KB for ACL_100K, while HyperCuts require 32000 KB [12], and HyperSplit requires 29000 KB [26]. Therefore, on average, APHT saves about 5-8% memory usage compared to TupleTree [13] and significantly more compared to decision tree-based methods, demonstrating improved scalability.

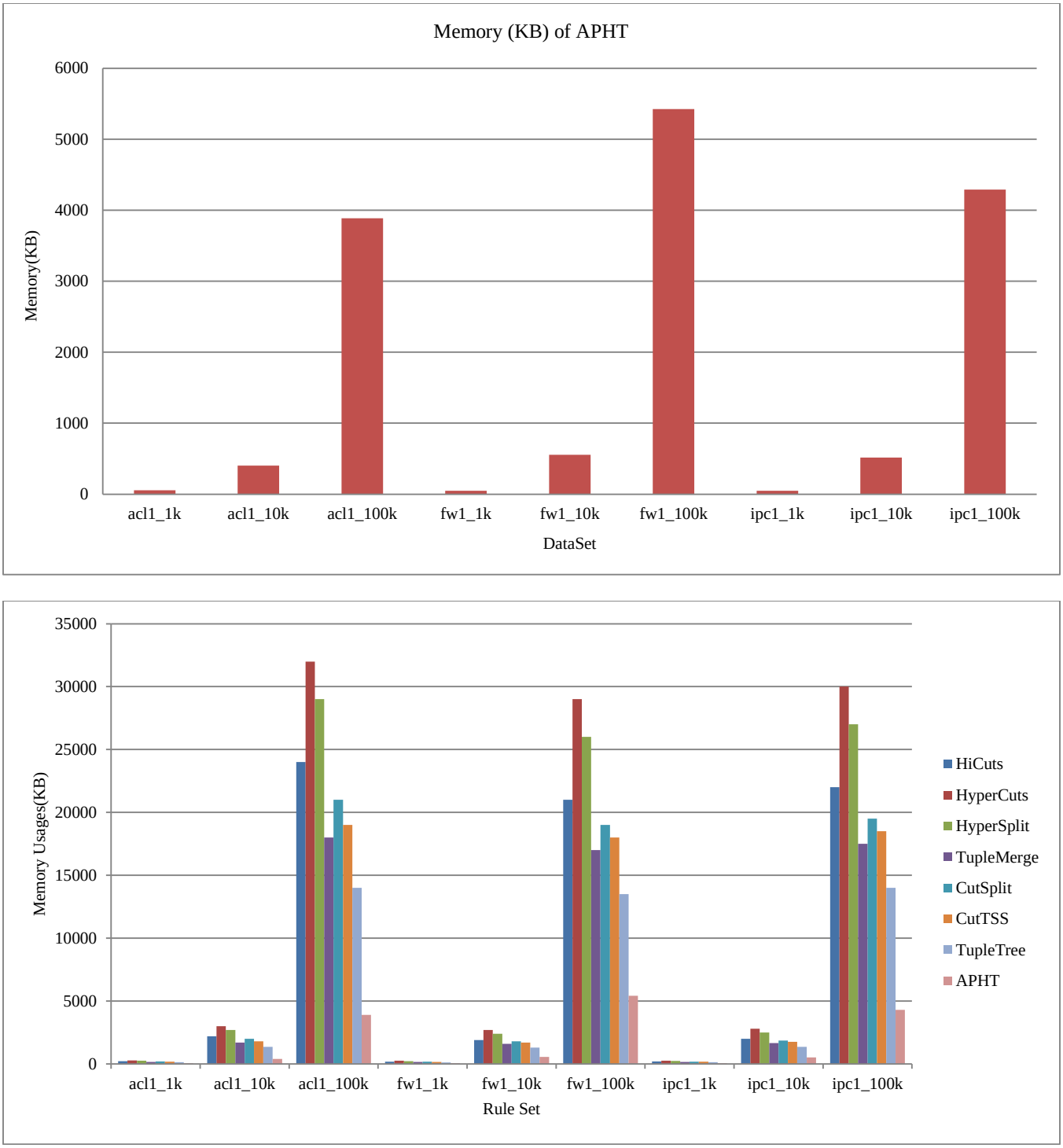
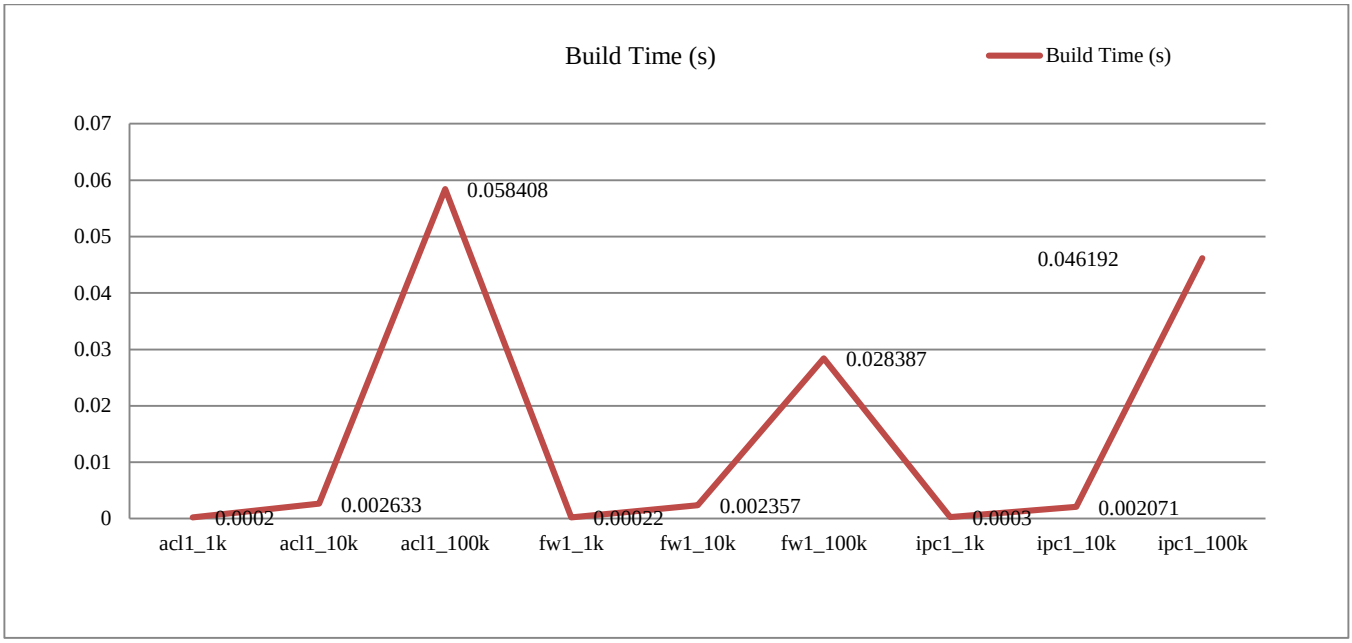


Fig. 7 Evaluation of memory usages

6.5.4. Build Time

Figure 8 presents the APHT tree construction time for different ClassBench rule sets. The construction time is directly proportional to the rule set size and complexity since large data sets will have higher levels of recursion and hash bucket formation. Even for rule sets of 1K size, APHT only builds a tree under 0.0003 seconds, while it takes less than 0.003 seconds build time for rule-sets of 10K. For instance, it takes only 0.002633 s, 0.002357 s and 0.002071 s to construct a tree by APHT for the ACL10K, FW10K and IPC_10K rule-sets, respectively. When the rule-sets become 100K size, the construction time goes up to 0.058408 s for ACL100K, 0.028387 s for FW100K and 0.046192 s for IPC100K, respectively. Thus, the ACL100K ruleset requires the longest time to build a tree, but the FW_100K ruleset needs the shortest building time, and this suggests that rule-set structure impacts on the building cost, in addition to the number of rules it holds. Nonetheless, APHT can still provide a low pre-processing cost, as the highest build time is still under 0.06 second even if the rule-set size is up to 100K. The results show that the APHT has a well-built adaptive partition-hash table, as its tree structure controlled for different rule-set sizes.



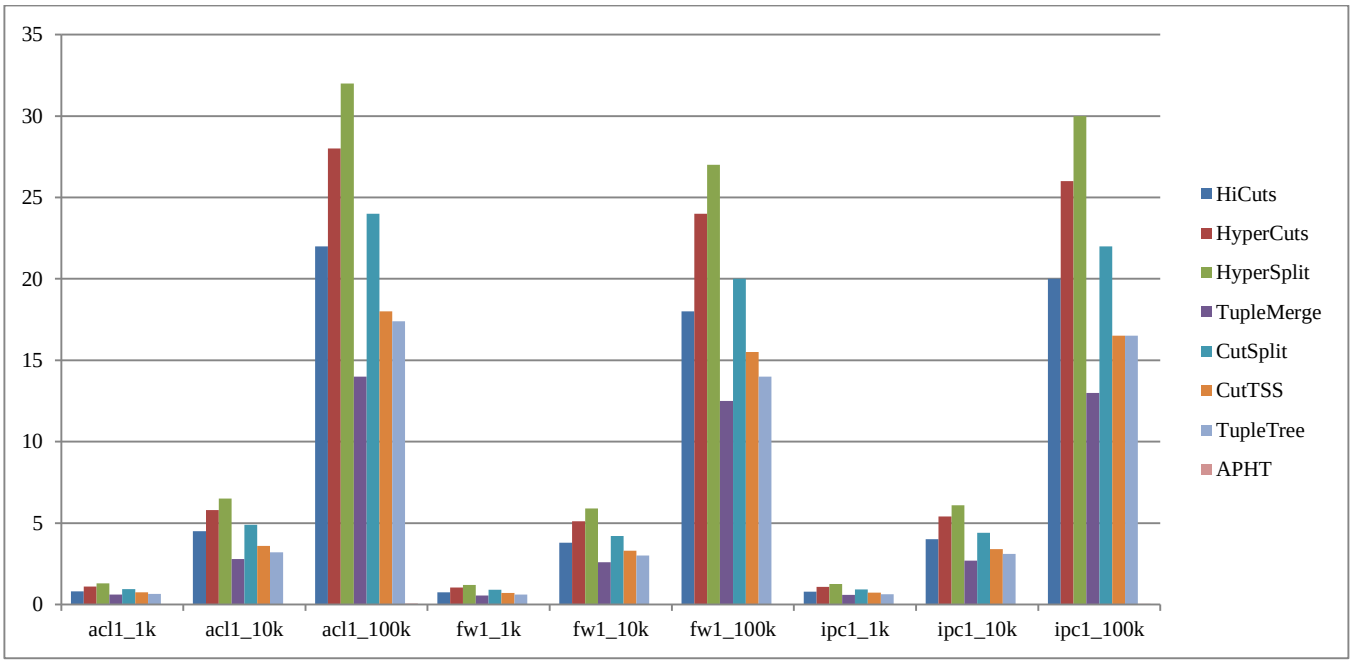


Fig. 8 Evaluation of build time

6.5.5. Tree Depth and Memory Access Analysis

The proposed APHT can keep a well-controlled tree depth of 4-5 levels for all nine ClassBench datasets even for the rule sets having up to almost 100K rules. For example, the tree depth ofACL1K is 4. The tree depth is 5 for ACL100K andIPC_100K. Table 5. Average memory accesses per lookup the average memory access times per lookup of APHT are as low as 2-4 times for all datasets. Especially forACL_1K, the number of memory accesses per lookup is as small as 2. Most other datasets achieve an average memory access times per lookup that fall into 3-4. A shallower tree height, low memory access times per lookup and thus, fewer tree traversals enable low lookup time and high throughput of APHT. Table 4 and 5.

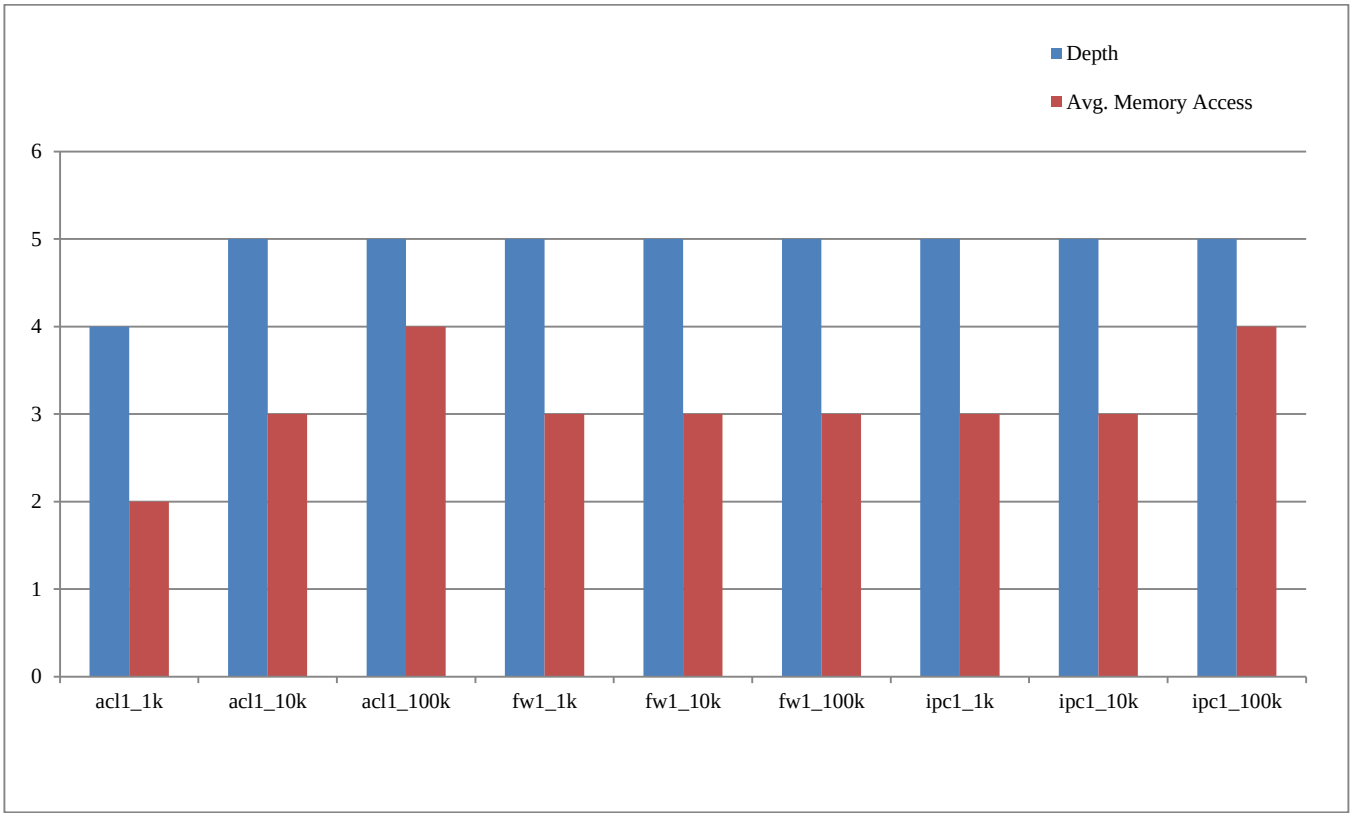


Fig. 9 Tree depth and memory access analysis

6.6. Performance Advantages of APHT over Existing Methods

The effectiveness of the proposed Adaptive Partition-Hash Tree (APHT) can mainly be attributed to two factors; the adaptive algorithm used to predict the optimal values of tree construction parameters and the hybrid partition-hash tree structure. Whereas conventional packet classifiers depend on predetermined heuristic guidelines for determining tree partitioning dimensions, cutting thresholds and tree structure, APHT uses a Random Forest regression to determine the optimal parameters based on rule-set features including the number of rules, prefix-length distribution, entropy, rule density and tuple distribution.

The hybrid partition-hash tree structure improves lookup time by using both recursion in space partitioning and hash function for lookup in the appropriate node. Through adaptive partitioning, the lookup space becomes smaller, whereas hashing operations are performed in that particular leaf node. Thus, there is a significant reduction in memory access, tree traversal and hash collision which make the process of lookups more efficient. As such, APHT has better packet classification times compared to the existing decision tree-based techniques.

As compared with heuristic partitioning algorithms including decision trees like HiCuts, HyperCuts, HyperSplit, EffiCuts, and CutSplit, the proposed solution does not rely on manual design of heuristic partitioning rules. Unlike heuristic partitioning algorithms, which use static parameters in partitioning rules for all sets of packet classification rules, the proposed approach automatically tunes the split dimension, partition threshold, hash bucket size, and leaf size based on the distribution of packet classification rules. As a result, adaptive tuning leads to balanced tree structure, minimal unnecessary rule replication, and better scalability across heterogeneous ClassBench rule sets.

In contrast with reinforcement learning-based techniques like NeuroCuts and HybridTSS, the proposed packet

classification algorithm relies on lightweight supervised learning approach built on top of Random Forest regression. The prediction model is trained offline and used once during tree construction to make an inference on the required parameters. Hence, unlike reinforcement learning, no iterative search for the best policy, its optimization, and convergence are needed.

Therefore, APHT provides much less computational overhead while retaining good classification performance. Thus, the proposed solution is better suitable for deployment in SDN environment. The experimental results shown in Sections 6.1-6.5 clearly show that the above architectural properties lead to better performance in packet classification. On the ClassBench ACL, Firewall, and IPC rule sets having from 1K to 100K rules, the proposed algorithm provides significantly better throughput, latency, and memory consumption in comparison with other packet classification techniques. Such an improvement is achieved due to adaptive parameter prediction, balanced tree construction, fewer memory accesses, efficient hash lookup, and reduced computational complexity.

7. Conclusion & Future Work

As illustrated above, it is evident that the use of lightweight machine learning algorithms together with hybrid data structures can lead to improved packet classification performance while incurring little or no computational costs. The proposed Adaptive Partition-Hash Tree (APHT) system provides a way of optimizing the classification process by utilizing machine learning algorithms in order to predict the most suitable parameters in each case. Evaluation of the system based on ClassBench [20] packet classification benchmarks reveals a relative throughput improvement of 10% to 15%, a reduction of up to 18% in lookup latency, and a decline in memory requirements of up to 5% when compared to TupleTree classification schemes. In conclusion, from the study above, it can be seen that machine learning techniques can be quite useful in improving the efficiency of packet classification processes in networking environments such as SDNs and data centers. For future research, it would be appropriate to expand the current work through the use of advanced techniques such as deep learning for better prediction, incorporation of online learning capability, and hardware acceleration using FPGA and GPUs.

References

[1] P. Gupta, and Nick McKeown, "Algorithms for Packet Classification," *IEEE Network*, vol. 15, no. 2, pp. 24-32, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[2] V. Srinivasan, S. Suri, and G. Varghese, "Packet Classification using Tuple Space Search," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 135-146, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[3] Athanasios Liatifis et al., "Advancing SDN from OpenFlow to P4: A Survey," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1-37, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[4] David E. Taylor, "Survey and Taxonomy of Packet Classification Techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[5] 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. [CrossRef] [Google Scholar] [Publisher Link]

[6] Yaxuan Qi, Bo Xu, and Jun Li, "Performance Evaluation and Improvement of Algorithmic Approaches for Packet Classification," *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, Papeete, Tahiti, pp. 7-7, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[7] Banit Agrawal, and Timothy Sherwood, "Modeling TCAM Power for Next Generation Network Devices," *2006 IEEE International Symposium on Performance Analysis of Systems and Software*, Austin, TX, USA, pp. 120-129, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[8] Kai Zheng et al., "TCAM-based Distributed Parallel Packet Classification Algorithm with Range-Matching Solution," *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Miami, FL, vol. 1, pp. 293-303, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[9] E. Spitznagel, David E. Taylor, and Jonathan S. Turner, "Packet Classification using Extended Tcams," *11th IEEE International Conference on Network Protocols*, Atlanta, GA, USA, pp. 120-131, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[10] Pankaj Gupta, and Nick McKeown, "Packet Classification using Hierarchical Intelligent Cuttings," pp. 213-224, 2003. [Google Scholar]

[11] Pankaj Gupta, and Nick McKeown, "Packet Classification on Multiple Fields," *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 147-160, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[12] Sumeet Singh et al., "Packet Classification using Multidimensional Cutting," *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer*, pp. 213-224, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[13] Jincheng Zhong et al., "TupleTree: A High-Performance Packet Classification Algorithm Supporting Fast Rule-Set Updates," *IEEE/ACM Transactions on Networking*, vol. 31, no. 5, pp. 2027-2041, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[14] T.S. Urmila, and R. Balasubramanian, "Decision Tree based Network Packet Classification Algorithms," *Proceedings of the UGC Sponsored National Conference on Advanced Networking and Applications*, 2015. [Google Scholar]

[15] Cheng-Liang Hsieh, and Ning Weng, "Many-Field Packet Classification for Software-Defined Networking Switches," *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pp. 13-24, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[16] James Daly et al., "TupleMerge: Fast Software Packet Processing for Online Packet Classification," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1417-1431, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[17] S.S. Srithick et al., "Machine Learning based Network Packet Classification," *Journal of Ubiquitous Computing and Communication Technologies*, vol. 5, no. 4, pp. 352-365, 2023. [CrossRef] [Publisher Link]

[18] Yu Liu, "A Survey of Machine Learning based Packet Classification," *Symposium on Computational Intelligence for Security and Defence Applications*, 2009. [Google Scholar]

[19] Veeramani Sonai, and Indira Bharathi, "Packet Classification using Improved Random Forest Algorithm," *Proceedings of the International Conference on Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication*, Springer, Cham, pp. 157-168, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[20] David E. Taylor, and Jonathan S. Turner, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 499-511, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[21] Shuhui Chen et al., "CMT: An Efficient Algorithm for Scalable Packet Classification," *The Computer Journal*, vol. 64, no. 6, pp. 941-959, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] Sorrachai Yingchareonthawornchai et al., "A Sorted-Partitioning Approach to Fast and Scalable Dynamic Packet Classification," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1907-1920, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[23] Jeffrey Fong et al., "ParaSplit: A Scalable Architecture on FPGA for Terabit Packet Classification," *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*, Santa Clara, CA, USA, pp. 1-8, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[24] Midde Adiseshaiah, and M. Sailaja, "Packet Classification Algorithms and Methods: A Critical Review," *International Journal of Emerging Technologies*, vol. 11, no. 1, pp. 454-461, 2020. [Publisher Link]

[25] Balajee Vamanan, Gwendolyn Voskuilen, and T.N. Vijaykumar, "EffiCuts: Optimizing Packet Classification for Memory and Throughput," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 207-218, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[26] Yaxuan Qi et al., "Packet Classification Algorithms: From Theory to Practice," *IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, pp. 648-656, 2009. [CrossRef] [Google Scholar] [Publisher Link]

[27] Wenjun Li et al., "CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification," *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, Honolulu, HI, USA, pp. 2645-2653, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[28] Zhi Liu et al., "BitTuts: A Fast Packet Classification Algorithm using Bit-Level Cutting," *Computer Communications*, vol. 109, pp. 38-52, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[29] V. Anand Prem Kumar, Vidya Thiyagarajan, and N. Ramasubramanian, "A Survey of Packet Classification Tools and Techniques," *2015 International Conference on Computing Communication Control and Automation*, Pune, India, pp. 103-107, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[30] Jing Tan, GaoFeng Lv, and GuanJie Qiao, "MBitTree: A Fast and Scalable Packet Classification for Software Switches," *2021 IEEE Symposium on High-Performance Interconnects*, Santa Clara, CA, USA, pp. 60-67, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[31] HyunYong Lee, and BhumCheol Lee, "Approaches for Improving Tuple Space Search-based Table Lookup," *2015 International Conference on Information and Communication Technology Convergence*, Jeju, Korea (South), pp. 99-104, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[32] Rishi Shah et al., "NeuroCut: A Neural Approach for Robust Graph Partitioning," *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2584-2595, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[33] Alon Rashelbach, Ori Rottenstreich, and Mark Silberstein, "A Computational Approach to Packet Classification," *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 542-556, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[34] Yuxi Liu et al., "HybridTSS: A Recursive Scheme Combining Coarse- and Fine-Grained Tuples for Packet Classification," *Proceedings of the 6th Asia-Pacific Workshop on Networking*, pp. 43-49, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[35] Jinsong Zhang et al., "A CNN-LSTM-GRU Hybrid Model for Spatiotemporal Highway Traffic Flow Prediction," *Systems*, vol. 13, no. 9, pp. 1-21, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[36] Xinyi Zhang et al., "Fast Online Packet Classification with Convolutional Neural Network," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2765-2778, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[37] Tong Yang et al., "Fast OpenFlow Table Lookup with Fast Update," *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, Honolulu, HI, USA, pp. 2636-2644, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[38] Ashok Kumar, and Arun Kumar, "Network Simulators: Perspective View," *International Journal of Advance Research in Science and Engineering*, vol. 6, no. 8, pp. 1742-1748, 2017. [Online]. Available: https://www.ijarse.com/images/fullpdf/1503896114_IETE_C_3029IJARSE.pdf

[39] Avneet Kaur Saluja, Sweta A. Dargad, and Krupali Mistry, "A Detailed Analogy of Network Simulators—NS1, NS2, NS3 and NS4," *International Journal on Future Revolution in Computer Science and Communication Engineering*, vol. 3, no. 12, pp. 291-295, 2017. [Google Scholar]

[40] Saba Siraj, Ajay Kumar Gupta, and Rinku-Badgujar, "Network Simulation Tools Survey," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 4, pp. 199-206, 2012. [Google Scholar] [Publisher Link]

[41] Wenjun Li et al., "Tuple Space Assisted Packet Classification with High Performance on both Search and Update," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1555-1569, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[42] Chenshuo Tan, and Zhuo Li, "MultiSplit: An Efficient Algorithm for Packet Classification with Equivalent Priority," *Electronics*, vol. 13, no. 15, pp. 1-21, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[43] Zhuo Li et al., "LearningTuple: A Packet Classification Scheme with High Classification and High Update," *Computer Networks*, vol. 254, pp. 1-11, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[44] Zhi-Xing Ou et al., "Improving Hierarchical Tree-based Packet Classification by Reinforcement Learning," *2024 Fifteenth International Conference on Ubiquitous and Future Networks*, Budapest, Hungary, pp. 336-341, 2024. [CrossRef] [Google Scholar] [Publisher Link]

Appendix

Table 2. Statistical features extracted from classbench rule sets

Dataset	SrcEntropy	DstEntropy	SrcPortEntropy	DstPortEntropy	AvgSrcPrefix	AvgDstPrefix	PrefixDistribution	Density	TupleDistribution
acl1_1k_rules	5.3414	6.74234	0	4.95675	29.758	28.4597	29.1088	2.63866	0.083864
acl1_10k_rules	11.3125	5.41712	0	4.9929	30.1003	28.7502	29.4252	2.1133	0.01504
acl1_100k_rules	16.5848	6.97915	0	4.96526	30.1985	28.7583	29.4784	0.987983	0.001681
fw1_1k_rules	3.89338	5.40378	1.39165	4.0656	11.0688	20.6873	15.8781	2.80065	0.102684
fw1_10k_rules	5.73158	10.1344	1.48091	4.05582	10.7719	22.1225	16.4472	0.930272	0.013861
fw1_100k_rules	6.1275	13.0499	1.39609	4.10625	10.3762	23.386	16.8811	1.02199	0.001567
ipc1_1k_rules	6.8984	7.93991	1.44062	2.83132	22.3819	24.7023	23.5421	1.37765	0.193018
ipc1_10k_rules	9.0736	10.2757	1.47741	2.8825	22.3357	24.2676	23.3016	2.24959	0.031414
ipc1_100k_rules	14.9038	15.515	1.43519	2.87437	22.3501	24.3865	23.3683	0.562978	0.003504