Fast Online Packet Classification With Convolutional Neural Network

Xinyi Zhang, Gaogang Xie, *Senior Member, IEEE*, *Member, ACM*, Xin Wang, *Senior Member, IEEE*, *Member, ACM*, Penghao Zhang, Yanbiao Li, and Kavé Salamatian, *Member, IEEE*, *ACM*

***Abstract*— Packet classification is a critical component in network appliances. Software Defined Networking and cloud computing update the rulesets frequently for flexible policy configuration. Tuple Space Search (TSS), implemented in Open vSwitch (OVS), achieves fast rule updating at the sacrifice of the classification rate. In TSS, each tuple is managed by a hash table and classifying a packet needs to go through all hash tables. Merging tuples can reduce the number of hash tables, but inevitably increases the hash conflicts that may even worsen the classification performance in some cases. No existing algorithm meets the need of both fast packet classification and online rule updating. In this paper, we propose Convolutional Neural Network (CNN)-based Range Partition (CRP) to achieve fast packet classification and online update simultaneously. CRP exploits CNN-based image recognition to quickly partition tuples into range spaces upon the change of ruleset distribution, which reduces hash operations while avoiding rule overlapping caused by hashing many rules to the same location of the hash table. Experimental results demonstrate that CRP achieves $3.2\times$ classification speed and $4.2\times$ update speed on average compared with state-of-the-art algorithms. We also implement CRP in OVS. The throughput of CRP-OVS is $10\times$ that of native OVS.**

***Index Terms*— Packet classification, Software Defined Networking (SDN), Open vSwitch (OVS).**

I. Introduction

PACKET classification is one of the most critical operations in switches, routers, firewalls, load balancers and other network appliances to support security [1], QoS [2], [3] and other advanced functions [4]. Packet classification aims at categorizing packets into "flows". All packets belonging to the same flow obey pre-defined rules and are processed in a similar action. Generally, the rulesets in these appliances are relatively static, and a well-designed data structure could achieve high-speed packet matching and forwarding. However, with the emergence of Software Defined Networking (SDN) [5], enormous opportunities for network innovation have been created to support new features, including traffic engineering [6], Network Function Virtualization (NFV) [7], [8] and high-performance cloud computing [9]. Unlike traditional packet classification methods based on the static ruleset, the controller program in SDN dynamically updates the ruleset by inserting and deleting rules [10]–[12]. Thus, we need a packet classification method to support both fast packet classification and quick online rule update.

As one of the primary functions in network appliances, packet classification has received enormous attention [13], [14]. Decision tree approaches [15]–[18] classify packets quickly but update rules slowly. In contrast, hash-based [12], [19]–[21] methods guarantee fast updates but fail to support high-speed packet classification. Hash-based methods generally follow the basic strategy of Tuple Space Search (TSS) [19], where rules are divided into a number of tuples, and each tuple is managed through a hash table. Although TSS supports fast updates, it needs to search all the hash tables one by one to classify a packet, which makes the classification slow. Some improved hash-based algorithms, such as Pruned Tuple Space Search [19], TupleMerge [12], promote the classification speed at the cost of update performance. To accelerate the classification speed by reducing the number of hash tables, Range-Vector Hash [21] partitions the ruleset into coarser groups according to the distribution of the number of rules on the combination of IP prefix lengths. However, improper partition could lead to an unbalanced distribution of rules in different groups. Some rules are hashed to the same key by shortening their prefix lengths, which leads to a high *rule overlapping* ratio. In this case, although the number of hash tables is reduced, the classification speed slows down. Besides, if we partition the ruleset with both the source IP and destination IP addresses, the total number of partition methods is $(C_n^0 + C_n^1 + C_n^2 + \ldots + C_n^n)^2$, where n is equal to 32 for IPv4 addresses and 128 for IPv6 addresses. Thus, finding an optimal solution via exhaustive search is time-consuming. Moreover, as the distribution of ruleset varies over time, a fixed partition method that cannot fits all scenarios will even reduce the classification performance.

In summary, existing methods cannot achieve high-speed packet classification and fast rule update simultaneously.

Manuscript received January 17, 2020; revised September 12, 2020 and March 31, 2021; accepted July 19, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Rao. Date of publication August 9, 2021; date of current version December 17, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1802800 and in part by the National Science Fund for Distinguished Young Scholars under Grant 61725206. *(Corresponding author: Gaogang Xie.)*

Xinyi Zhang and Penghao Zhang are with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing 100190, China, and also with the School of Computer Science and Technology, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: zhangxinyi@ict.ac.cn; zhangpenghao@ict.ac.cn).

Gaogang Xie and Yanbiao Li are with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), Beijing 100190, China, and also with the School of Computer Science and Technology, University of Chinese Academy of Sciences (UCAS), Beijing 100049, China (e-mail: xie@cnic.cn; lybmath@cnic.cn).

Xin Wang is with the Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, NY 11794 USA (e-mail: x.wang@stonybrook.edu).

Kavé Salamatian is with the LISTIC, Universite Savoie Mont-Blanc, 73000 Chambéry, France (e-mail: kave.salamatian@univ-smb.fr).

Digital Object Identifier 10.1109/TNET.2021.3100114

To solve this problem, we propose a novel method called Convolutional Neural Network (CNN)-based Range Partition (CRP) in this paper. Rather than traversing all possible ways to find an optimal partition method to construct hash tables, CRP transforms the traversal process into the problem of image recognition. More specifically, each rule locates a pixel by the pair of lengths of its source and destination IP prefixes; the pixel's value is determined by the number of rules sharing the same prefix length-pair. In this way, the rule distribution of a given ruleset upon the combination of IP prefixes can depict an image. Accordingly, CRP uses CNN [22], [23] to abstract the characteristics of the image defined by a ruleset, and label it with an optimal partition method that can reduce the number of hash tables while keeping the number of collisions low. Furthermore, once detecting the distribution variation, CRP can update the partition method and reconstruct the hash tables in milliseconds.

Although CRP has the potential to overcome the limitations of traditional packet classification methods by using deep learning technique, it faces three major technical challenges:

1) *Transforming diversified rulesets into images and categorizing them.* In order to exploit image recognition to categorize rulesets, we need a method to abstract the features of rulesets into images. In addition, there is a need to deal with a large number of ruleset patterns. We propose an Image Model to transform diverse rulesets into images and put similar distributions into a group based on their similarity.
2) *Labeling each category properly.* Our goal is to find the best method to partition the rulesets for fast packet classification and rule update. Therefore, different from traditional labels used for images classification problem, we propose to label each category of ruleset with a partition method. By analyzing the overhead of hash-based approaches, we propose a metric to guide the labeling process.
3) *Maintaining the system performance upon the change of ruleset.* To keep the high performance of a system, the partition method should adapt to the ruleset update in a practical environment. We apply a Monitor Module in CRP to timely detect the variation of the ruleset and perform repartition upon need.

CRP is an intelligent system that supports both high-speed packet classification and fast online updates. We have evaluated the performance of CRP using both synthetic and real rulesets and implemented CRP in Open vSwitch (OVS) [24]. To the best of our knowledge, CRP is the first effort that exploits deep learning to accelerate the hash-based packet classification which is critical for many advanced network operations.

II. Related Work

Existing packet classification schemes fall into two categories: hardware-based schemes and software-based schemes. The software-based schemes consist of dimensionality reduction and space partitioning.

Hardware-Based: *T-CAM* [25]–[27] is the de facto standard chip for high-speed packet classification. It applies hardware-based parallel search to achieve low deterministic lookup time, but suffers from problems of slow rule update. To reduce the update overhead in *T-CAM*, TreeCAM [28] decouples updates and lookups in packet classification by utilizing tree-based architectures. Nonetheless, the limited memory size and high power consumption still hinder the widespread use of *T-CAM* compared with cheaper, more energy-efficient and more scalable software-based schemes. Besides *T-CAM*, packet classification can also run on other hardware platforms, such as GPU [29] and FPGA [30]. These platforms require specific designs of hardware instructions, chips and programming language. The limited flexibility and high cost are the barriers that prevent these solutions from widespread usage.

Dimensionality Reduction: *Cross-producting* [31] and *RFC* [32] first split multi-dimensional rules into several single-dimensional ones to match individually, and then merge the results. When the ruleset is large, the merging procedure becomes sophisticated. Furthermore, the rule update is slow, because the rule tables corresponding to every dimension need to be updated for one rule update.

Space Partitioning: The main idea of the space partitioning approach is to sub-divide the rule space. Instead of matching an incoming packet against the overall ruleset, the classification procedure is divided into two steps: determining the sub-space to search and matching the packet against the small ruleset in the corresponding sub-space. This approach further falls into two main subcategories: decision tree approaches and hash-based approaches.

The core of decision tree approaches such as *HiCuts* [15] and *HyperCuts* [16] is to partition the search space recursively into several regions until the rule number in each region is below a certain threshold. The efficiency of decision trees allows for high-speed packet classification, but the tree updating is slow. In addition, some rules may need to be copied into multiple partitions, which consumes a large amount of memory. *EffiCuts* [17] separates rules into several subsets with each implemented as a decision tree. *SmartSplit* [18] separates rules into at most four subsets to build balanced trees dynamically. Even though these methods can alleviate the rule replication, they still fail to support fast rule updates. *NeuroCuts* [33] is a machine learning-based method to classify packets with decision trees. By utilizing deep reinforcement learning, NeuroCuts provides significant improvements on classification time and memory footprint. However, this method still does not support online updates due to the inherent defect of the decision trees.

In contrast, hash-based approaches can update rules quickly but suffer from slow packet classification. As CRP is proposed based on the hash-based approach which can be used for online packet classification, we now describe them in more detail.

Tuple Space Search (TSS) [19] is a classical hash-based packet classification algorithm. Rules used for packet classification are defined over several fields in each ruleset. In TSS, a tuple is a combination of prefix lengths (or ranges) of fields. Each tuple can be identified by a K-tuple $(R[1], R[2], \ldots, R[K])$, where K is the number of fields selected to classify packets and $R[K]$ is a prefix length or a range, *e.g.*, [1024,2048] for port number. The range can be identified by the *RangeID* that is defined through the nesting level [19].

Rules with identical $R(i)$ over its i^{th} field for $i = 1, \ldots, K$ will be put into the same tuple. Moreover, each tuple is managed through a hash table. The example ruleset in Table I contains ten rules with four fields each: source address (SA), destination address (DA), priority (Pri) and an action (Act). For each incoming packet, we use SA and DA to search for

TABLE I
A SAMPLE RULESET

Rule #	SA	DA	Pri	Act
0	100*	11010	2	Fwd 0
1	101*	1001*	2	Fwd 1
2	11111	10000	3	Drop
3	111*	1000*	2	Fwd 4
4	0100*	0110*	2	Fwd 0
5	001*	01001	3	Fwd 2
6	00*	01001	2	Drop
7	01110	*	4	Drop
8	110*	1*	1	Fwd 1
9	*	*	0	Fwd 3

TABLE II
TUPLES FOR THE SAMPLE RULESET

Tuple #	Tuple	Rules Mapped to
0	(5,5)	2
1	(5,0)	7
2	(4,4)	4
3	(3,5)	0,5
4	(3,4)	1,3
5	(3.1)	8
6	(2,5)	6
7	(0,0)	9

the matching rule. The ruleset can be transformed into eight tuples, as shown in Table II.

The lookup in each hash table $(R[1], R[2], \ldots, R[K])$ consists of extracting $R(i)$ bits for $i = 1, \ldots, K$ from each relevant packet field, concatenating them and hashing the resulting bit set into a l bit value, which is used as the key to map the rule in the hash table. Consequently, each hash table contains 2^l entries, and each entry consists of a l bits hash key and a bucket that contains rules matching the key. When the hash collision happens, a bucket will contain several rules that are stored as a linked chain. For this reason, when finding a matched entry in a hash table, rules in the corresponding bucket need to be verified sequentially. To classify a packet, TSS requires the search of all hash tables linearly. At the last stage, matching rules from different tuples are compared, and the rule with the highest priority is returned as the final matching rule.

Rule update in TSS is straightforward. Rule insertion only needs hashing its key and inserting it to the corresponding hash table. Removing a rule consists of finding the rule in the hash table and removing it. The low update complexity in TSS benefits from the characteristics of the hash table structure. However, such structure also leads to low classification performance.

Several algorithms improve the classification speed of TSS at the cost of update performance. *Pruned Tuple Space Search* (PR-TSS) [19] reduces the number of hash tables by using tries to filter out some unmatched tuples, but the update operation of tries is time-consuming. *PartitionSort* (PS) [20] combines the benefits of both TSS and decision trees. Rather than partitioning rules based on tuples, rules are partitioned into sortable rulesets and stored through balanced search trees. With the reduction of hash calculations, PS can classify packets faster than TSS, but spends more time processing the sortable ruleset. *TupleMerge* (TM) [12] reduces the number of hash tables by shortening the prefix length of IP addresses. However, due to the use of a pre-defined merging method, TM may suffer from high classification time as a result of the large ratio of rule overlapping, thus hash collisions. When the number of collisions exceeds a certain threshold, TM splits a hash table into two, which will slow down the update speed. CutTSS [34] is a combination of decision tree and TSS-based scheme. On the one hand, CutTSS adopts cutting techniques to build decision trees at a coarse granularity, so that it can shrink the searching space of any packet to a few steps for fast lookup. On the other hand, for efficient rule updates, CutTSS utilizes TSS-based schemes at a fine granularity to organize the rules falling into each leaf node of a decision tree. Since the "best" shape of a decision tree highly relies on the characteristics of rule distribution, CutTSS still needs a refactor on decision trees over long-time rule update.

TABLE III
RANGE VECTORS FOR THE SAMPLE RULESET

Range Vector #	Combination	Rules Mapped To
0	([3,6),[4,6))	0,1,2,3,4,5
1	([3,6),[0,4))	7,8
2	([0,3),[4,6))	6
3	([0,3),[0,4))	9

Range-vector Hash (RVH) [21] reduces the number of hash tables by partitioning the ruleset into several disjoint subsets based on the rule distribution. It changes the combination of prefix lengths in TSS into a combination of prefix ranges, *i.e.,* $R[i]$ becomes a range that contains contiguous prefix lengths. And such combination is named as *range vector*. For the ruleset in Table I, RVH partitions it into four disjoint *range vectors* ([0,3),[0,4)), ([0,3),[4,6)), ([3,6),[0,4)) and ([3,6),[4,6)), as in Table III. The rule 0 is mapped to the range vector([3,6),[4,6)), which means the prefixes of its SA and DA fall into the range[3,6) and [4,6), respectively. Each *range vector* is managed by a hash table, with the lower bounds of the source and destination ranges to restrict the hash key formulation. For the range vector ([3,6),[4,6)), three bits source prefix along with four bits destination prefix are used together as a key.

In a *range vector*, some rules might have the same prefix value by the reduction of the prefix length and are stored in the same bucket, which results in collisions. To be distinguished from the hash collision that happens even without merging, such collision is called *rule overlapping*. For this reason, after finding the corresponding hash entry during packet classification, further verification is needed to find the rule that exactly matches the input packet. A packet may be matched with several rules in different hash tables, and the one with the highest priority will be selected as the final result. Although RVH reduces the number of hash tables and rule overlapping simultaneously through a heuristic algorithm based on subjective observations, it cannot quickly respond to rule updates to keep the performance of the system at a high level.

III. CRP OVERVIEW

To achieve both fast packet classification and online rule update based on hash-based approaches, we refer to the idea

of supervised learning. Supervised learning [35], [36] is a machine learning task of learning a function that maps an input to an output based on training examples. Each training example is a pair that consists of an input object and the desired output value (also called *label*). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples quickly.

According to the analysis of the real rulesets used for packet classification in [37], the rules usually have two major components: an IP address pair and an application specification. The IP address pair identifies the communication subnets by specifying a source IP address with a prefix length and a destination IP address with a prefix length. The application specification identifies a specific session by the transport protocol, source port number and destination port number. In the real ruleset, the port pair usually contains a wildcard in one field and an exact number in the other. Merging exact port numbers that are not continuous is not realistic. Besides, the prefix lengths of IP addresses vary from 0 to 32. The number of possible combinations of the source and destination IP prefix lengths is 1089 (33×33), which is much larger than that of the port pair. Reducing them properly to make some rules have the same prefix lengths can decrease the number of hash tables in TSS. Therefore, we reduce the prefix lengths of both the source and destination IP addresses to reduce the number of hash tables in CRP. To represent the rulesets' feature, we convert the distributions of the number of rules on the combination of the source and destination IP prefix lengths into matrices, and further transform them into images for image recognition in CNN.

In a large operation cloud platform, we can obtain all the rulesets belonging to different software switches at different times in the past. If we provide a partition method for every ruleset, the total number of methods needed could be intractable. As a result, we implement a method to measure the similarity between different rulesets and divide them into different classes. We define a class of rulesets with similar distributions as a *ruleset type*. All the rulesets belong to the same *ruleset type* share the same partition method, i.e., the same label. The more accurate definition of the *ruleset type* will be introduced in §IV-A. In each *ruleset type*, we select one ruleset as the *reference ruleset* π. Rather than providing a partition method for every ruleset, we utilize the offline computation resources to find the optimal partition method for π via exhaustive search and label every ruleset that belongs to this *ruleset type* with it.

Finally, transformed images and their labels serve as the training set in supervised learning. With the progress in pattern and image recognition [38]–[42] of CNN during the past few years, we propose CRP to achieve fast packet classification and online update simultaneously.

As shown in Figure 1, a CRP system has two parts: Online and Offline. The Online part contains three components: a CNN-based Range Partition Module to determine the partition scheme based on the rule distribution, a Classification and Forwarding Module to construct hash tables used for matching and forwarding packets, and a Monitor Module to watch for changes of ruleset patterns periodically for notifying repartition the ruleset when needed and detect a new *ruleset type*. In the Offline part, a Calculation Module is applied to determine the proper partition scheme for a *ruleset type*, and CNN Training Module is used to train the CNN model.

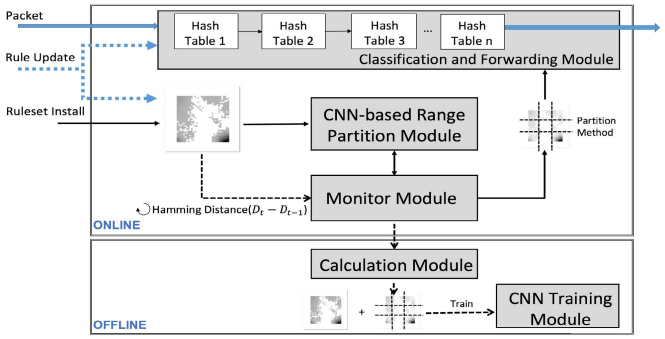


Fig. 1. CRP design overview.

There are usually multiple software switches in a large-scale network. Each software switch runs on an independent server and each is regarded as a separate Online part. In contrast, only one Offline part runs on a single server, which collects all rulesets that have appeared in each software switch at a specific frequency. In this paper, to introduce CRP more concisely and clearly, we only present the collaboration between one Online part and one Offline part.

In the initialization phase, the CNN Training Module trains a CNN model with images mapped from various *ruleset types* along with their pre-determined labels. In other words, CRP has prior knowledge of the rulesets before it is able to function. Then CRP applies the well-trained model in the CNN-based Range Partition Module. After that, Online and Offline parts cooperate in the following way. First of all, the Online System transforms the installed ruleset into an image and passes it to the CNN-based Range Partition Module. After checking whether the ruleset is a new type, the Monitor Module outputs the partition method, and the Classification and Forwarding Module constructs hash tables according to it. When the Monitor Module discovers that the rule distribution has changed into another type, it updates the partition method and repartitions the ruleset. If such distribution is a new type, the Monitor Module also passes it to the Calculation Module to calculate a proper partition method. Detailed information about the Monitor Module will be introduced in §IV-E.2. After obtaining the partition method, both the new distribution type and partition method are delivered to the CNN Training Module to update the training set and train a new model. Finally, CRP replaces the old online CNN model with the latest trained one.

In this way, the complicated procedure of calculating the optimal partition method is carried out in the Offline system, which has sufficient CPU and memory resources and is not as sensitive as the Online system to the processing latency. The Online system is able to categorize rulesets with the use of a well-trained neural network and construct hash tables quickly. With the coordination between the Online and Offline systems, CRP can detect a new distribution type of ruleset and track the network's most up-to-date information.

IV. CRP Design

The success of CRP relies on the efficient categorization of diversified rulesets into different types through CNN. In this section, we first show how to translate rulesets into images with the Image Model and how to label those images with the Calculation Module. Then we describe the packet classification and rule update procedure in the Classification and Forwarding

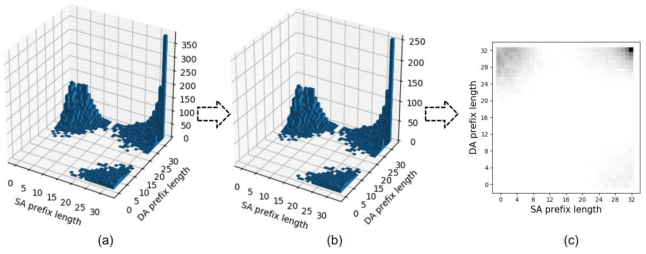


Fig. 2. Ruleset normalization and transformation: (a) Distribution of rules on the prefix lengths combination of SA and DA (b) Normalized distribution (c) Transformed 2D image.

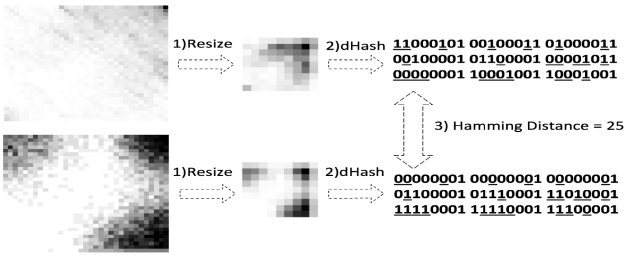


Fig. 3. Hamming distance calculation.

Module. Finally, we introduce the Monitor Module to make the partition method adapt to the ruleset update in practice.

A. Modeling Rulesets With Images

The fact that an image has correlations between nearby pixels is one of the principles of CNN for image recognition. For a hash table representing tuple (a, b), we define the hash tables $(a, b+1), (a, b-1), (a+1, b), (a-1, b)$ as its *neighbors*. When partitioning a ruleset, each hash table and its *neighbors* might be eventually placed into the same hash table. Such possibility is based on the number of *rule overlappings*, which largely depends on the number of rules in the corresponding hash tables. In other words, the selection of the partition method is highly related to the distribution of the combination of the source and destination IP prefix lengths.

Consequently, we set a matrix to show the combination of IP prefix lengths and the relationship among the neighboring hash tables simultaneously. One dimension of the matrix represents the length of the source IP prefix and the other represents the length of the destination IP prefix. For the IP address with prefix length varying from 0 to 32, we can obtain a matrix of size 33×33. In this matrix, the value of each element is the number of rules that have the corresponding prefix length combination in the ruleset, that is, the number of rules in the corresponding tuple in our example. CNN can recognize either grayscale images or color images, and the essence of the image is a matrix. Each pixel in the image has a value range based on the image type. In our case, it is sufficient to represent the distribution of rules on prefix length combinations by using a grayscale image. The color range of each pixel is from 0 to 255 and is represented with an 8-bit value. However, in each ruleset, the number of rules of each combination is not limited, so we need to normalize the value to make it fall into the same range as the pixel value. Figure 2 (a) is the original distribution of rules on prefix lengths combinations, and the height of each column is the number of rules falling into the corresponding combination. After the normalization, the distribution is converted into Figure 2 (b), which will be further transformed into a 2D image in Figure 2 (c).

To measure the *distribution similarity* among images, we utilize perceptual hashing [43] to produce each image's fingerprint. Different from cryptographic hashing schemes (MD5 or SHA-1) that rely on the avalanche effect of a small change in input value creating a drastic change in the hash value, the hash values generated by perceptual hashing are analogous if the input features are similar. By converting the comparison of two images into the comparison of two fingerprints, the complexity can be decreased. Besides, the coarse-grained fingerprint can make the image categorizing more easily. Perceptual hash contains three mainstream schemes: aHash (average Hash), pHash (perceptive hash) and dHash (difference hash). aHash focuses on average values, pHash evaluates frequency patterns and dHash tracks gradients. In CRP, each pixel in the image corresponds to a specific combination of prefix lengths, so the pixel value is highly related to its position on the image. Therefore, the perceptual hashing method in CRP needs to ensure that the original image's spatial information is maintained in the fingerprint. Besides, to track the rulesets' variation, the perceptual hashing method should also be calculated fast. Among the perceptual hashing algorithms above, the accuracy of aHash is low, and pHash utilizes the time-consuming discrete cosine transform (DCT). dHash contains spatial information between pixels, and it is also accurate and fast. Consequently, we select dHash to measure the similarity among images in our work.

We follow three steps to obtain the fingerprint with dHash: 1) Resizing the prefix distribution image to 9×9 by Nearest-neighbor Interpolation; 2) Comparing two neighboring pixels to translate nine pixels per row into eight values of difference; 3) Assigning binary bits to encode each difference, with a bit set to "1' if the pixel P[x] $<$ P[x + 1] and "0" otherwise, and taking the 72-bit (8×9) string as the fingerprint of the image. When the image does not change fundamentally but only shifts spatially, the relationship among the adjacent pixels will not change. Consequently, the value of dHash remains unchanged. After obtaining the fingerprints of images, we need to measure the similarity between them. Since the fingerprints have the equal lengths and each bit has meaning, we utilize *Hamming distance* to counts the number of bits that are not equal between two fingerprints. Such method can reflect the difference between the two fingerprints quickly and accurately. The above procedure is shown in Figure 3.

We further need to choose an appropriate threshold value to judge whether two images belong to the same type. If the Hamming distance is larger than the threshold, the two images are considered to belong to different types. The selection of the threshold value will be introduced in §IV-E. Based on the threshold, the definition of the *ruleset type* in §III could be more precise. *Ruleset type* is a set of rulesets whose paired Hamming distance is less than a pre-defined threshold.

B. Labeling Rulesets

To train the CNN model, we also need to label the images of the rulesets. As our purpose is to look for a proper partition method for fast rule matching and update, we label each category with a partition method selected. Even though the ruleset and partition method is the same, different input packets may experience different performance due to their various matching paths. Therefore, we need to compare the effectiveness of all possible partition methods using the average performance of

TABLE IV
SYMBOL AND DEFINITION

Sym.	Definition
h_i	Hash calculation time of the i-th hash table
c_i	Match verification time of the i-th hash table
r_i	Hit ratio in the i-th hash table
o_i	Average overlap ratio of the i-th hash table
e_i	Number of entries in the i-th hash table
s_i	Size of the i-th hash table
n_i	Number of rules in the i-th hash table
$\bar{h}$	Average hash calculation time
$\bar{c}$	Average comparison time for verification
$\bar{q}$	Average time to compare priority
m	Total number of the hash tables

each partition method. The Calculation Module in CRP will do the above procedure and select the best one as the label. Table IV lists the notations used in our analysis.

Classifying a packet with the ruleset partitioned into range spaces generally consists of three steps: 1) Searching all hash tables, with a hash operation to determine whether rules in a hash table match the packet; 2) Verifying the exact rules matched with the packet to avoid false positive caused by hash conflict or rule overlapping; 3) Choosing the rule with the highest priority, if multiple rules are matched. The time to classify a packet thus contains three components. The hashing time is related to the number of hash tables. The time for match verification depends on the number of hash conflicts or rule overlappings. Given a hash function, the probability of matching an entry in a hash table is proportional to the utilization ratio of each hash table, which is defined as the number of entries divided by the size of the hash table:

$$r_i = \frac{e_i}{s_i} \tag{1}$$

The rules with the same hash value have to be further checked to eliminate the false positive. After finding a matched entry in a hash table, the time for verification is the time taken to find the exact rule that is matched weighted by the average overlap ratio, defined as the number of rules in the hash table divided by the number of entries:

$$o_i = \frac{n_i}{e_i} \tag{2}$$

Consequently, the average time for packet classification is

$$T = \sum_{i=1}^{m} h_i + \sum_{i=1}^{m} c_i r_i o_i + \bar{q} \tag{3}$$

where m is the number of hash tables. Replacing the hit ratio and overlap ratio in (3) with (1) and (2), the time cost to classify a packet can be converted to

$$T = \sum_{i=1}^{m} h_i + \sum_{i=1}^{m} c_i \cdot \frac{e_i}{s_i} \cdot \frac{n_i}{e_i} + \bar{q} \tag{4}$$

which can be simplified to

$$T = \bar{h} \sum_{i=1}^{m} + \bar{c} \sum_{i=1}^{m} \frac{n_i}{s_i} + \bar{q} \tag{5}$$

By comparing the average packet classification time T of all partition methods, we choose the one that gives the lowest time. As traversing all partition schemes may take a long time, our Calculation Module will run in the Offline mode.

The rulesets being pre-determined partitioned are used as the data for training. We train the CNN model with both real rulesets and synthetic rulesets, which makes the model more general and reliable. The detail of the CNN model architecture and data set for training will be introduced in §V-B.

C. Packet Classification

The image of a ruleset can be quickly classified into a specific type by CNN, based on which the ruleset will be properly partitioned into a number of range spaces, with each managed by a hash table. When querying the ruleset to find the matched rule for guiding the next step action on the packet, it needs to search hash tables one by one. To speed up the packet classification, we further propose two strategies. First, we set the priority of each hash table to the highest priority of the rules it contains, and sort the hash tables in order of priority from high to low. By doing so, we can terminate the search operation once we find a matched rule whose priority is not less than the priority of the next hash table. Second, we sort the overlapped rules in each bucket with the priority from high to low during the rule insertion so we can stop the verification once we find a matched rule.

D. Rule Update

Rule update in CRP leads to the change of hash tables and the rule distribution on the combination of the IP prefix lengths. To insert a rule into the ruleset, we first identify which hash table the rule should be inserted, and then insert the rule into the correct location by hashing its key. The key is formed with the combination of fields defined by the ruleset, and the length of each field is determined by the low bounds of the corresponding range vector. If other rules exist at the mapped location, we reorder these overlapped rules on the basis of their priority. Deleting a rule is even simpler: locating the rule in the hash table and deleting it. If the hash table becomes empty after the deletion, we also delete the hash table. The value of the corresponding combination in the matrix also needs to be updated, which eventually alters the pixel value in the image.

E. Ruleset Repartition

In the SDN environment, frequent updates lead to the variation of the rule distribution, and it is hard to provide a partition method that fits all scenarios. To guarantee the performance of the system, the partition method should be updated once the ruleset is changed into another type. So it is critical to choose a proper threshold to determine whether or not the ruleset has changed to another type. Besides, we propose a Monitor Module in CRP to track the distribution variation and repartition the ruleset upon change.

1) Threshold Selection: In CRP, a small threshold for repartition will cause the frequent reconstruction of the hash tables and even decrease the classification performance. In contrast, a large threshold will neglect the distribution variation of the ruleset, which cannot guarantee the performance of the system. Therefore, selecting a proper threshold is critical. The threshold in CRP is determined by the range and variation of rulesets in a system. So we first calculate the Hamming

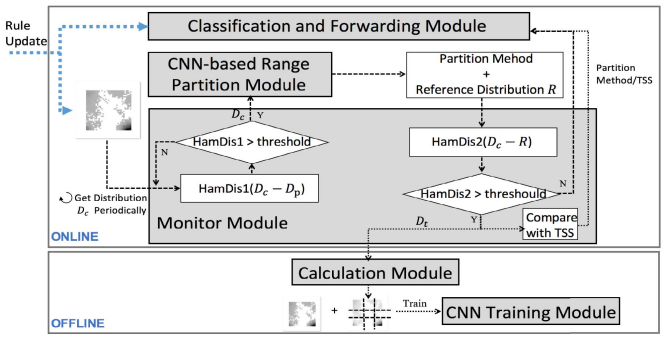


Fig. 4. Function of the monitor module.

distance between the *reference rulesets* of every two *ruleset types* already known in the system. Based on the result, we obtain the box plot of Hamming distances and select the distance at the First quartile (Q1) as the threshold. The threshold is also dynamic and can be adjusted according to the variation of the ruleset. More specifically, the box plot of the Hamming distances is also updated when a new *ruleset type* is added.

2) Monitor Module in CRP: The Monitor Module aims to monitor the variation of the ruleset distribution and to discover the new *ruleset type*, as shown in Figure 4.

Once the ruleset is updated, the distribution of rules on the prefix lengths will be changed. To discover this variation, the Monitor Module measures the Hamming distance between the current distribution D_c and the distribution D_p on the previous partition at regular intervals. If the Hamming distance is larger than the threshold we discussed in the previous section, we need to send the distribution D_c to the CNN-based Range Partition Module to look for a new label, thus a new partition method. Besides the new label, the CNN-based Range Partition Module also outputs a distribution R which is the *reference ruleset* in the corresponding *ruleset type*. If the Hamming distance between D_c and R is smaller than the threshold (which is the same as the threshold for repartition), it means the distribution has changed into another known type, and we need to repartition the ruleset with the method identified by the label. After the ruleset is repartitioned, the Monitor Module stores the current distribution as D_p for later use. Otherwise, if the Hamming distance between D_c and R is larger than the threshold, we consider D_c as a new *ruleset type* and deliver it to the Offline system. The Offline system uses it to retrain the CNN. However, the Monitor Module still needs to output a partition method. It compares the performance between the partition method identified by the label and TSS by utilizing Formula (5) in §IV-B, and sends the better scheme between them to the Classification and Forwarding Module to construct hash tables. In the initialization phase, the administrator might install a new ruleset that does not belong to any *ruleset types* in the training set. The Monitor Module also measures the Hamming distance between the distribution D of the installed ruleset and the distribution R generated by the CNN-based Range Partition Module. If the distance is larger than the threshold, it can be treated as a new *ruleset type*, and then be processed with the procedure above.

V. Experimental Results

We carry out extensive experiments to compare the performance of CRP with six state-of-the-art packet classification algorithms, including Tuple Space Search (TSS), Pruned Tuple Space Search (PR-TSS), PartitionSort (PS), TupleMerge (TM), Range-vector Hash (RVH) and CutTSS. We first introduce the training of the CNN model and verify the Calculation Module in CRP. After that, we evaluate packet classification performance without rule update, then the performance of rule update and the packet classification performance under update. Finally, we test the memory footprint and discuss the overhead and limitations of CRP.

TABLE V
Rulesets for Experiments

Ruleset	#Rules	#Hash Tables			Mem(MB)	#Cycles(M)
		TSS	RVH	CRP		
ACL_10K	9672	768	25	10	31.16	241.06
ACL_50K	50000	815	49	36	34.28	240.76
ACL_100K	100000	841	64	42	37.43	240.06
FW_10K	9375	474	25	20	31.15	240.58
FW_50K	50000	801	49	28	34.44	240.38
FW_100K	100000	839	63	48	37.42	241.53
IPC_10K	10000	526	22	18	30.96	245.39
IPC_50K	50000	740	46	24	34.34	242.19
IPC_100K	100000	786	63	35	37.42	243.38
Cloud1	1427	202	14	4	30.87	243.44
Cloud2	1660	190	14	4	30.86	243.38

A. Experimental Setup

In our experiment, the CNN model is implemented with Tensorflow [44] and trained on 2 NVIDIA Tesla K80 GPUs, and each of them has 4992 NVIDIA CUDA cores and 480 GB/s aggregate memory. All other experiments run on a server with an Intel Xeon CPU E5-2630 v3@2.40GHz, 32 cores and 128GB DDR3 memory. Each core is equipped with a 64 KB L1 data cache and a 256KB L2 cache. A 20MB L3 cache is shared among all cores. Ubuntu 16.04.1 with Linux kernel 4.10.0 is installed as the operating system.

We choose eleven different rulesets as illustrated in Table V. Nine of them are generated by ClassBench [37] with various sizes and two are from the large operational cloud platform of a major Internet company. The implementation of CRP is publicly available on https://github.com/crp2021/crp.git.

B. Training of CNN Model Offline

1) Training Dataset Generation and Threshold Selection: To train the CNN model, we utilize both the synthetic rulesets and real rulesets. We use ClassBench to generate diverse synthetic rulesets based on seed files. ClassBench is a suite of tools for benchmarking packet classification algorithms and devices. It can produce synthetic rulesets based on twelve seed files generated from real rulesets, including access control list (ACL), firewall (FW), and IP chain (IPC). According to the synthetic rulesets, ClassBench can also produce a sequence of packet headers to exercise it. After selecting a seed file, two parameters are used to determine the rule distribution on the combination of source and destination IP prefix lengths: Smoothness and Address Scope. Smoothness controls the combination types of prefix length to generate. A larger Smoothness means the ruleset will have more types of combinations. For a given Smoothness, Address Scope adjusts the likelihood of rules with different prefix lengths. If Address

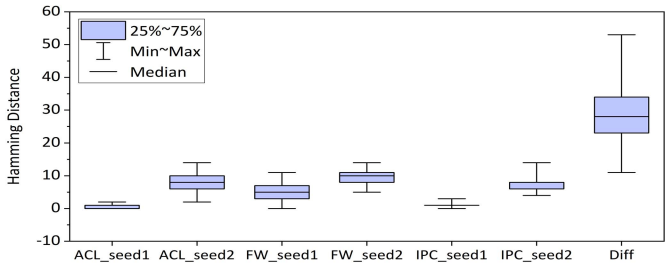


Fig. 5. Hamming distance between rulesets.

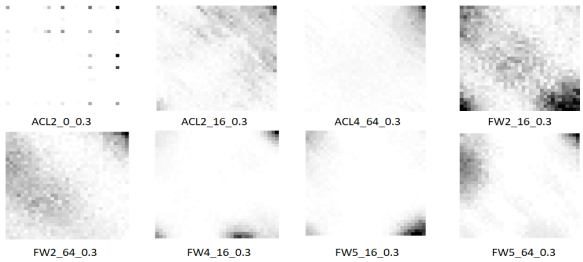


Fig. 6. Distributions of the number of rules on prefix length combinations generated by ClassBench.

Scope is large, more rules in the ruleset will contain longer prefix lengths.

To evaluate the impact of Address Scope and Smoothness on ruleset features, we choose six seed files (acl1, acl2, fw1, fw2, ipc1, ipc2) in the following tests. We first set the Smoothness to 0 to evaluate the impact of Address Scope. For each of the six seed files, we generate the rulesets by adjusting Address Scopes from 0 to 1 with an interval of 0.005 and setting the ruleset size to 10K. Then we calculate the Hamming distance between every two rulesets that generated by the same seed file. The six box-plots from left to right in Figure 5 represent the distances. Next, we measure the effect of Smoothness by fixing the Address Scope to 0. We set the Smoothness value from 0 to 64 with an interval of 16. We calculate the Hamming distance between every two rulesets generated by the same seed file but different Smoothness values. The Hamming distances of them are in the last box-plot. From Figure 5, we observe that the maximum Hamming distances on the left six box-plots are comparable to the minimum value in the last box-plot, which means the Smoothness parameter plays the central role in the rule distribution on the combination of IP prefix lengths.

To obtain the training dataset for the CNN model, we first set Address Scope to 0, and then adjust the other three parameters in ClassBench: seed file, Smoothness and size of rulesets to generate different *ruleset types*. The ruleset whose Address Scope is set to 0 is treated as the *reference ruleset* in its *ruleset type*. Based on 12 seed files, by varying Smoothness from 0 to 64 at interval 16, we can obtain 60 different *ruleset types* for each of the three ruleset sizes, namely 10K, 50K and 100K. Totally, we obtain 180 different synthetic *ruleset types*. Then an additional parameter, Address Scope, is varied to generate slightly different rulesets that belong to the same *ruleset type*. For each *ruleset type*, we get 500 slightly different rulesets by varying Address Scope from 0 to 1 with an interval 0.002. Besides these synthetic rulesets, we also have two real rulesets obtained from a cloud platform. These two are also considered as *reference rulesets* of two different *ruleset types*. Totally, we get 182 *ruleset types* with 182 *reference rulesets*. 180 of them are generated by ClassBench and 2 from real rulesets. After that, we calculate the Hamming distances between every two of the 182 *reference rulesets*. According to the threshold selection scheme introduced in §IV-E.1, we choose the Hamming distance at the First quartile (Q1) of the box plot as the threshold, which is 15 in our test.

For each of the generated *ruleset types*, we select 400 rulesets from the 500 rulesets that have been obtained, and the Hamming distances between the generated rulesets and their *reference rulesets* are less than 15. For *ruleset types* obtained from the cloud platform, we transform each real ruleset into 400 different rulesets by controlling the Hamming distance between the generated ruleset and the real one less than the threshold 15.

After obtaining 400 different rulesets for each *ruleset type*, we divide them into three subsets: 320 for the training set, 40 for the validation set and 40 for the test set. To reduce the time taken by the exhaustive search of the best partition method for labeling each ruleset type, our Calculation Module chooses 4 as the granularity to vary the prefix length during the search, and it takes 2∼3 minutes to find the partition method of each type. The Calculation Module can choose a smaller granularity that generates a more accurate partition method and takes more time to calculate. As CRP is a universal model, the selection of parameters depends on the actual application scenarios and the computation ability of the hardware.

2) CNN Model Selection: After obtaining the dataset for training, we construct the CNN model. A CNN model is formed by a stack of distinct layers that consist of an input and an output layer, as well as several convolutional layers, pooling layers and fully connected layers. Over the past couple of years, many CNN models for quickly and accurately classifying images have sprung up, such as AlexNet, VGG [40], ResNet [42], Inception [41], and DenseNet [45], etc. Inspired by these state-of-the-art CNN models, we consider both the sizes and features of images in CRP, and design four different CNN models. The size of the training set is 258MB. The model structures, the number of parameters, the size of each model and the accuracy are shown in Table VI. The training time of Model-1, Model-2 and Model-3 is less than 3 hours, while Model-4 spends almost 5 hours due to its complex structure. Since Model-3 achieves the highest accuracy on the test set, we select it as our CNN model in CRP. Model-3 consists of three convolutional layers, two sub-sampling layers, three fully connected layers and a softmax classifier. Convolutional layers in it use 3×3 convolutions with stride 1 and sub-sampling layers are 2×2 maximum pooling layers.

The well-trained CNN model is uploaded to the CNN-based Range Partition Module. CRP Online system transforms the ruleset into an image and categorizes it with the CNN model to obtain a partition method. The hash tables are constructed according to this partition method. We evaluate the time of the above procedure for each ruleset in Table V. The average time is 4.8ms, with a maximum of 7.0 ms and a minimum of 2.1 ms. Because of the rapid construction of hash tables, CRP can meet the requirement of online packet classification.

C. Verification of Calculation Module

To validate the Calculation Module's effectiveness in labeling the translated images with the partition methods selected, we do a series of tests. And we also choose 4 as the granularity

TABLE VI
INFORMATION OF THE CNN MODEL TRAINING

Model name	Model Description	Model size	Accuracy
Model-1	1conv+1pool+2conv+1fc	18.90MB	89.4%
Model-2	1conv+1pool+1conv+1pool+2fc	26.07MB	91.1%
Model-3	1conv+1pool+1conv+1pool+1conv+3fc	31.17MB	95.6%
Model-4	1conv+1pool+1conv+1pool+3conv+3fc	74.32MB	93.3%

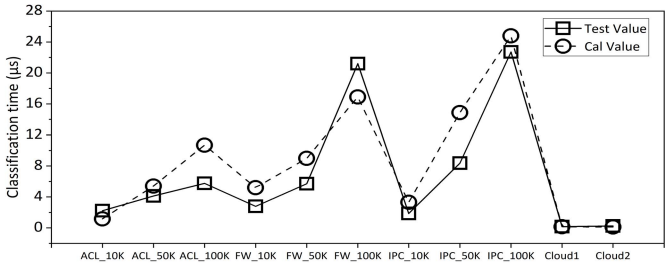
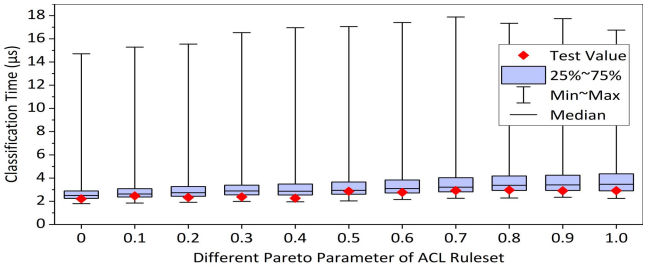
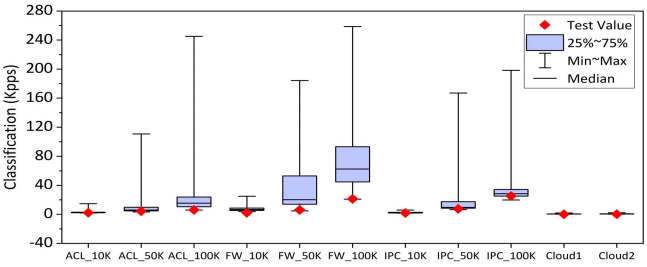


Fig. 7. Comparison between the calculation and experimental classification time.



(a) on different traffic patterns



(b) on different rulesets

Fig. 8. Classification time comparison between the method generated by the Calculation Module and all the partition methods.

to vary the prefix length during the exhaustive search in this part.

First, we verify the accuracy of our model in estimating the packet classification time. We find the average time of hashing a packet ($\bar{h}$), verifying a match ($\bar{c}$) and comparing priority ($\bar{q}$) in our test environment to be 20 ns, 5 ns and 0.9 ns, and substitute their values for those in the Equation (5). For each ruleset, we traverse all partition methods to find the one with the smallest time T for packet classification. We use the partition method found to divide the ruleset and construct the hash tables, which is further applied to evaluate the actual average time for classification. Then we compare the time T with the actual *average* classification time. As shown in Figure 7, the estimated time is close to the experimental one.

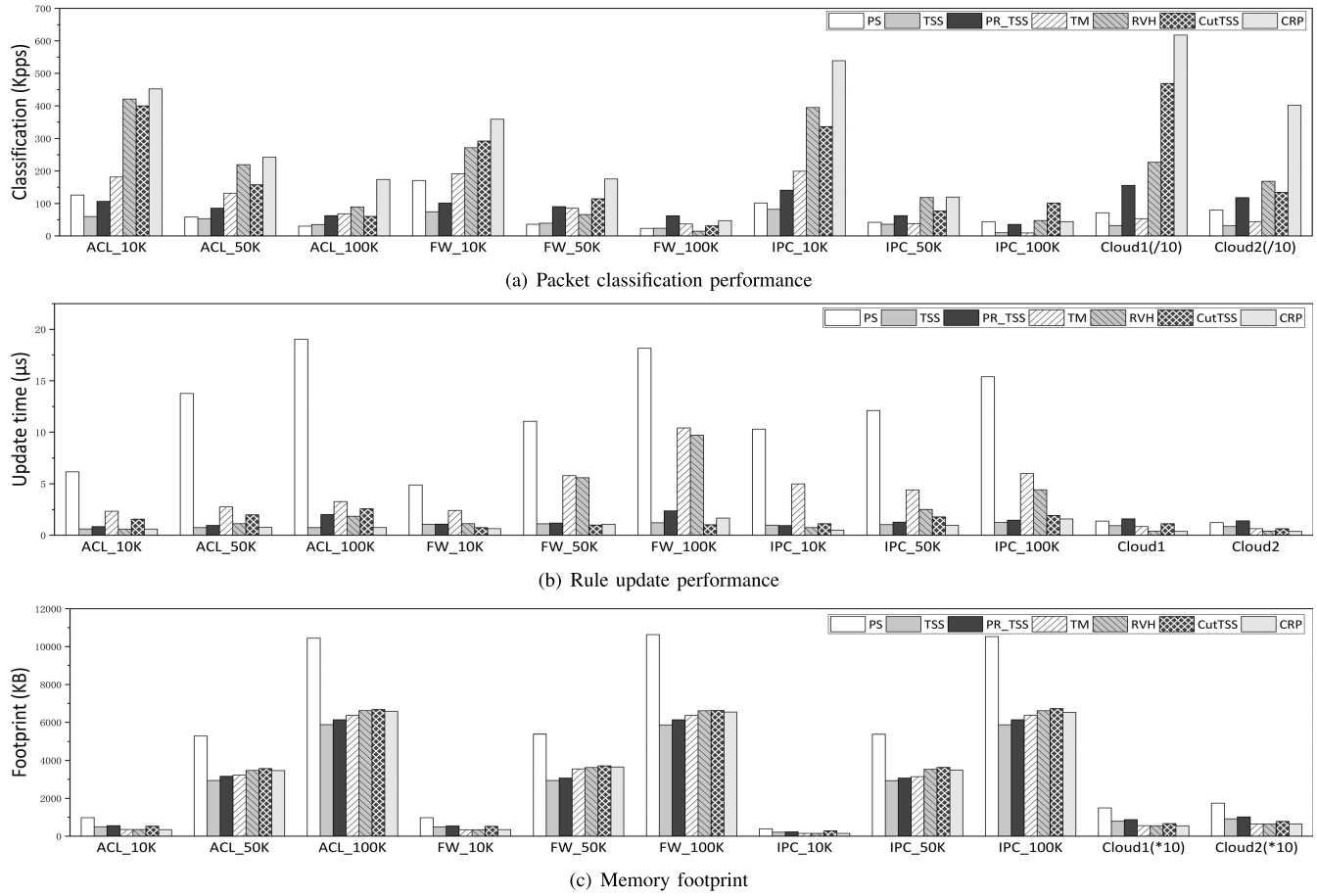
Second, to validate the partition method's stability in the face of different traffic patterns, we generate diverse traffic patterns by adjusting the Pareto parameter with the traffic generator in ClassBench. The ruleset used for this test is ACL_10K. In Figure 8(a), different box-plots correspond to different traffic patterns, and each box-plot represents the classification time of all partition methods of ACL_10K when the traffic pattern is fixed. The diamond point is the classification time using the partition method provided by the Calculation Module. It is evident that the partition method obtained by the Calculation Module can achieve fast classification speed faced with various traffic distributions.

Finally, we verify the efficiency of Calculation Module with rulesets in Table V. All the Pareto Parameters of the packet distributions in this test are 0. In Figure 8(b), each box-plot also represents the classification time of all partition methods when both the ruleset and traffic pattern are fixed, and the diamond point is the classification time taken by using the partition method of the Calculation Module. Apparently, the Calculation Module can generate accurate partition methods under distinct rulesets.

D. Packet Classification Without Update

Figure 9(a) and Figure 10 show the performance of packet classification when there are no update operations, where *kilo packets per second* (kpps) is used as the classification throughput unit. Throughput for Cloud1 and Cloud2 is divided by 10 to fit in the figures. We observe that the packet classification performance of CRP is $4.12\times$ that of PS, $6.52\times$ that of TSS, $1.86\times$ that of PR-TSS, $3.76\times$ that of TM, $1.85\times$ that of RVH and $1.65\times$ that of CutTSS on average. As for PS, the number of decision trees depends on the properties of rulesets, and the lookup speed will decrease when dealing with a large number of trees. Compared with TSS, CRP significantly reduces the number of hash tables to achieve faster classification. TM merges hash tables rigidly and suffers hash collision or rule overlapping in some cases, so the classification performance of TM is lower than CRP. Being able to provide a more reasonable partition method than RVH, CRP achieves a higher packet classification speed than RVH. Even though CutTSS utilizes decision trees to accelerate the packet classification, according to the results in our test, CRP can achieve better performance than CutTSS by reducing the number of hash tables. In addition, in Figure 9(a), the performance improvement of CRP compared with RVH in the Cloud rulesets is higher than that in the ClassBench rulesets. As shown in Table V, under the Cloud rulesets, the hash table number of RVH is more than $3\times$ that in CRP, whereas the hash table number of RVH is only around $2\times$ that in CRP under the ClassBench rulesets. As the cost of hash operation is $4\times$ of verifying a match, while maintaining the low hash collision ratio, fewer hash tables mean faster classification speed. Besides, from Figure 9(a), we can also see the impact of ruleset size on the classification performance. Larger rulesets result in lower classification throughput. Nonetheless, the classification throughput of CRP is still significantly higher than those of other methods in most cases.

We also compare CRP with the cases of partitioning the prefix length at a fixed length: 4, 8 and 16. Although shortening the prefix length reduces the number of hash tables, the probability of hash collision or rule overlapping is prone to increase. As shown in Figure 10, the packet classification performance of CRP outperforms the fixed partition method in every ruleset.



(a) Packet classification performance

(b) Rule update performance

(c) Memory footprint

Fig. 9. Comparison between CRP and other algorithms.

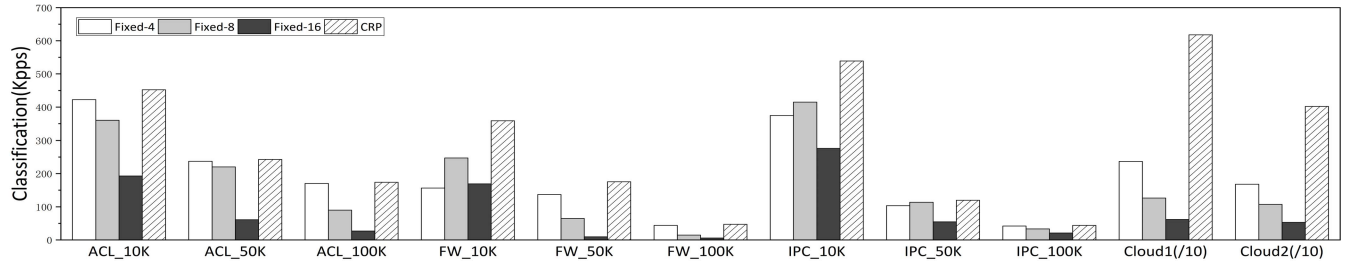


Fig. 10. Packet classification performance without update: comparison between CRP and fixed partitions.

E. Update Time

Update operations can be either insertion or deletion. To test the update time, we divide each ruleset evenly into five subsets, with four used to initialize the original ruleset and the last one used to perform the update. In Figure 9(b), the performance of CRP is close to TSS on miniature rulesets. When the size of the ruleset becomes larger, the number of collisions in CRP increases and update time rises, while the update time of TSS is relatively stable. Compared with other methods, the update speed of CRP is $12.55\times$ that of PS, $1.67\times$ that of PR-TSS, $1.43\times$ that of TM, $2.5\times$ of RVH and $3.26\times$ that of CutTSS on average.

The poor update performance of PS and PR-TSS is due to the utilization of the tree structures to accelerate the classification speed. TM fails to achieve fast updates due to the split-up of the hash table when the number of collisions exceeds a certain threshold. CRP has fewer hash collisions or a lower rule overlapping ratio than RVH so that it could achieve faster updates. As the update in CutTSS needs to locate the decision tree first and then update the rule in the corresponding decision tree, the nodes in decision trees need reconstruction when the characteristics of the ruleset change. For this reason, in most rulesets, the update speed of CRP is faster than that of CutTSS.

From Figure 9(b), we can also see the impact of ruleset size on the update performance. The update time of PS increases sharply when the size of the ruleset is growing. This happens because of its tree-based data structure with $\mathcal{O}(logN)$ update cost, where N is related to the size of the ruleset. Unlike PS, the update speed of the tree in PR-TSS is only related to the number of hash tables. The increase in the number of hash tables is much slower than the rise in the number of rules. For this reason, with the expansion of the ruleset size, the update time of PR-TSS grows more slowly than PS. In TM,

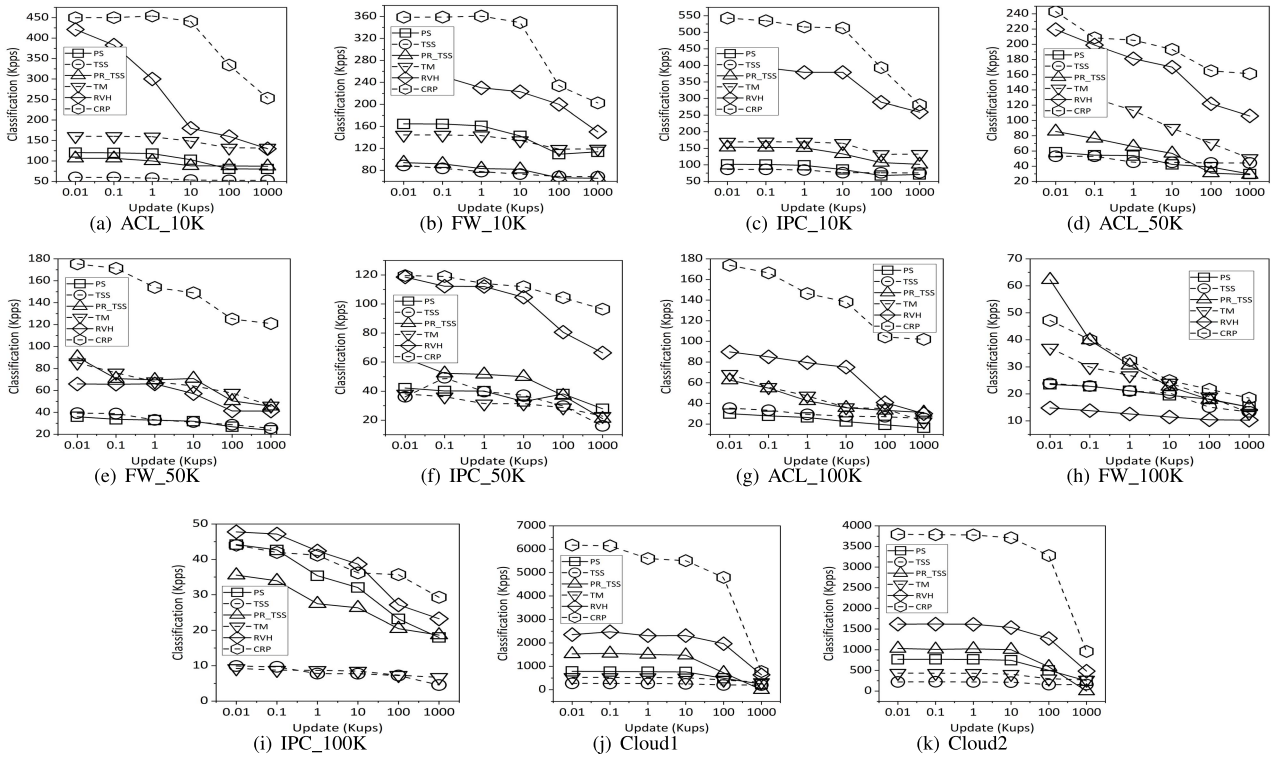


Fig. 11. Packet classification performance with different update speeds.

RVH and CRP, when the ruleset becomes larger, the update time increases due to the growing number of collisions. More specifically, the increasing number of collisions makes the TM split-up the hash tables frequently. When inserting rules in RVH and CRP, the rules in the same bucket will be sorted by their priorities. For this reason, more rules will take more time to compare priorities.

F. Packet Classification With Update

1) Update Without Ruleset Repartition: Assuming that update will not lead to repartition, we evaluate the packet classification performance with the rule update speed varying from 0.01Kups to 1000Kups *(Kilo update per second)*. We utilize two threads to simulate this scenario. One thread continuously classifies the packets, and the other updates the ruleset at a certain speed. As two threads compete for access to the ruleset, we use Mutex Lock for exclusive access. From Figure 11, we could draw the following conclusions. First, frequent updates lead to a decrease in the classification speed for all classification methods. Second, for most rulesets, CRP achieves the highest classification performance and update speed. In Figure 11(h) and Figure 11(i), CRP outperforms other algorithms when the rule update rate exceeds 10Kups.

2) Update With Ruleset Repartition: When the ruleset is updated, the distribution of the number of rules on the combination of IP prefix lengths changes and the partition method is no longer applicable for the updated ruleset. The Monitor Module tracks such changes and performs repartition when needed. As the repartition of the ruleset requires the reconstruction of the hash tables, we need to design a scheme to reduce the packet loss during the reconstruction process. When repartitioning the ruleset, we allocate a new piece of memory and store the new set of hash tables constructed by the new partition method on it. After finishing the reconstruction, We make the pointer to the old hash tables point to new hash tables and release the old hash tables' memory. During the changing of the pointer, the packets could not be processed. The classification throughput drops slightly and the packet loss rate is only 0.15% on average.

In Figure 12, to validate the necessity of the ruleset repartition, we simulate the update procedure by changing the ruleset sequentially from ACL_10K, FW_10K to IPC_10K. The update process removes 200 rules from the old ruleset and inserts 200 rules from the new ruleset every second. More specifically, 200 rules from ACL_10K are removed and 200 rules from FW_10K are inserted every second from time 1 to *re1*. From time *re1* to *re2*, 200 rules from FW_10K are removed and 200 rules from IPC_10K are inserted every second. As the partition method is fixed and no longer applicable for the updated ruleset, the classification throughput decreases during each period. When discovering that the ruleset distribution has changed into another type, CRP repartitions the ruleset and reconstructs the hash tables at time *re1* and *re2*. By doing this, CRP is able to adapt to the update of the ruleset and guarantee the classification speed with a low packet loss rate. When the ruleset is repartitioned at *re2*, there is a significant performance boost. This happens because the throughput of IPC_10K is larger than that of FW_10K when the hash tables are constructed with their corresponding optimal partition methods, as shown in Figure 9(a). Moreover, the performance result in Figure 12 also demonstrates the selection of the threshold in §IV-E.1 is proper.

G. Memory Footprint

In Figure 9(c), the memory footprint of Cloud1 and Cloud2 is multiplied by 10 to fit in the figures. The memory footprint of CRP is on average 51.8%, 85.1% that consumed by PS and CutTSS, respectively. The utilization of

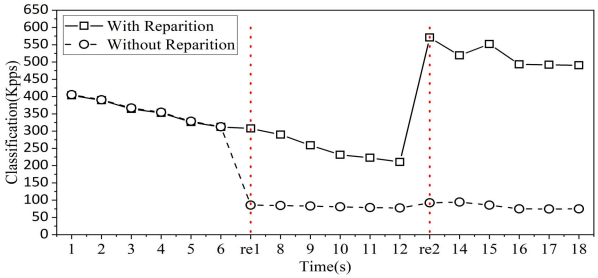


Fig. 12. Throughput comparison between system with repartition and system without repartition.

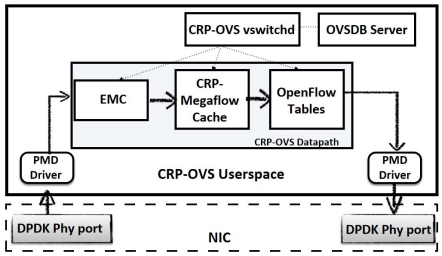


Fig. 13. Architecture of CRP-OVS.

decision trees in PS makes its memory much higher than other approaches. CutTSS utilizes decision trees to speed up the classification performance, so it consume more memory than CRP. As TM, RVH and CRP all reduce the number of hash tables of TSS, the subtle differences between their memory footprints are due to the differences in data structures even though the rules themselves occupy the same memory. As expected, the memory footprint increases sharply when the ruleset becomes larger.

H. Overhead and Limitations

In the Online part of CRP, Classification and Forwarding Module is the core module that realizes the packet classification function. However, the overhead of other parts that coordinate with it should also be considered. When a ruleset is installed into CRP, or when a ruleset needs repartition, the CNN-based Partition Module will classify the ruleset by the CNN model, and the Monitor Module will check whether it is a new type and output the partition method. More specifically, at the initialization phase, the overhead of memory footprint is only consumed by loading the well-trained CNN model into the Online part. When the ruleset needs repartition, the overhead of memory footprint consists of two parts: the memory consumed by loading the well-trained CNN model and the memory consumption of the new set of hash tables during the reconstruction process. Therefore, we regard the latter circumstance as the worst case and measure both the memory consumption and the number of CPU cycles in this case. The result is shown in Table V. After the construction of the hash tables, the CPU and memory overhead will be released.

In our experiment, the training a CNN model takes about 3 hours. In some extreme cases, such as the rule set distribution changes frequently, the system cannot react perfectly when a model needs to be replaced shorter than the training period. This phenomenon is a limitation of CRP. However, the system converges when more and more samples are included in the training set, and the probability that the CNN model needs to be retrained is getting smaller.

VI. CRP Implementation in Open vSwitch

Open vSwitch (OVS) [24] is the de facto implementation platform for multilayer distributed virtual switch needed by the major hypervisor. The datapath of DPDK-OVS consists of Exact Match Cache (EMC), Megaflow Cache and OpenFlow tables. EMC is used to match each field of the header exactly. Megaflow Cache contains a single flow lookup table to support the generic matching and OpenFlow tables keep all the rules installed by an SDN controller. Both Megaflow Cache and OpenFlow tables are implemented with TSS. In [24], the authors report the overall cache hit rate is 97.7% over a RackSpace deployment. In other words, most packets will hit the cache instead of being sent to the OpenFlow tables. Since CRP is impossible to be applied on EMC, we replace TSS in Megaflow Cache with CRP to speed up the classification. We call this design CRP-OVS.

A. CRP-OVS Architecture

As shown in Figure 13, CRP-OVS inherits the architecture of native OVS and contains three parts: EMC, CRP-Megaflow Cache and OpenFlow Table. When a packet enters the userspace datapath, it first checks against the EMC to find a matched rule. If not, the packet enters the CRP-Megaflow Cache. If the rule is still not found, the packet enters the OpenFlow slow path. After searching a series of OpenFlow tables, the packet generates a rule with a wildcard mask and an action list. And the system inserts them into the CRP-Megaflow Cache. Meanwhile, an exact match entry will also be inserted into the EMC. Therefore, the packets with the same header can avoid the overhead of traversing a slow path in the future.

The ruleset repartition in CRP-Megaflow Cache is similar to that in CRP. When the Monitor Module discovers the variation of distribution, it repartitions the CRP-Megaflow Cache. According to the partition scheme, the CRP-Megaflow Cache will be reconstructed into new hash tables. The reconstruction process is lightweight and will not influence the performance of the system.

B. Evaluation of CRP-OVS

In this section, we evaluate the throughput of native OVS and CRP-OVS in different situations. OVS version is 2.9.0 and DPDK version is 17.11.4. We use one server to generate 64-byte packets with Pktgen at a speed of 10 Gbps and the other to run the native OVS or CRP-OVS to forward packets. We select eight rulesets generated by ClassBench from §V. The port fields of rules in ClassBench are represented by ranges, e.g.,[0, 65535], while the port fields of rules in OVS are formed with a value and a mask, e.g.,0×0000/0xFFFF. Thus, to add the ClassBench rules into OVS, the port field values need to be transformed, which might lead to an explosion of the rule space. In our test, the ACL_10K used in §V contains 9672 rules, and the ruleset is expanded to 18072 rules by converting the port field.

Megaflow Cache in OVS uses two optimizations to speed up the packet classification. First, before querying the Megaflow Cache, the packets are pre-processed by EMC. Second, the rules in Megaflow Cache do not have priorities. When a packet matches a rule in the Megaflow Cache, the packet

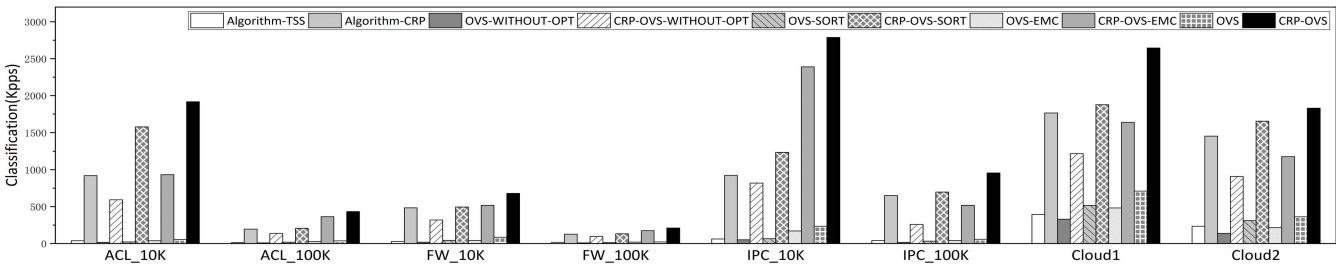


Fig. 14. Throughput comparison between OVS and CRP-OVS.

classification process stops and the matched rule is returned directly. For this reason, to make the packets match the rules as quickly as possible, Megaflow Cache sorts the hash tables in descending order periodically according to hit counts. In our test, we first disable the above two optimizations and obtain two classification performance results, i.e., OVS-WITHOUT-OPT and CRP-OVS-WITHOUT-OPT. Then we only enable hash table sorting and obtain OVS-SORT and CRP-OVS-SORT. After that, we disable hash table sorting and enable EMC to get OVS-EMC and CRP-OVS-EMC. Finally, both EMC and the hash table sorting are enabled, and we obtain OVS and CRP-OVS. The performance results are shown in Figure 14.

Additionally, to show the overhead of additional parts in OVS, we extract the source code of Megaflow Cache (Algorithm-TSS) and implement CRP (Algorithm-CRP) on it. We do not use the TSS and CRP implemented in §V because their implementation is inconsistent with that in Megaflow Cache. In §V, TSS and CRP need to be compared with other existing algorithms (PartitionSort, CutTSS) on ClassBench rulesets. The port field of rules in ClassBench is represented by ranges. To ensure the consistency of the comparison results, we cannot transform the rules and only consider the IP field when constructing hash tables. For OVS, both IP and port fields of rules in OVS are in the form of prefixes. Megaflow Cache can construct hash tables by considering both fields. Due to the selection of different fields when constructing hash tables, TSS (CRP) implemented in §V and Algorithm-TSS (Algorithm-CRP) produce different numbers of hash tables even with the same ruleset, and their classification performance results are not comparable.

In Figure 14, without optimization, the throughput of CRP-OVS-WITHOUT-OPT is improved by $14\times$ on average that of the OVS-WITHOUT-OPT, which is similar to the improvement of Algorithm-CRP over Algorithm-TSS. By eliminating the overhead of other parts in OVS, the performance of Algorithm-TSS (Algorithm CRP) is better than that of OVS-WITHOUT-OPT (CRP-OVS-WITHOUT-OPT) as expected. EMC and hash table sorting have different degrees of performance improvement under different rulesets. When both of them are enabled, the throughput of CRP-OVS is improved by $10\times$ on average that of the native OVS. By reducing the number of hash tables while ensuring low hash collision and rule overlapping ratio in Megaflow Cache, CRP-OVS successfully improves the throughput of native OVS under all rulesets experimented.

VII. Conclusion

In this paper, we propose a two-level system CRP to support both high-speed packet classification and fast online rule update. We have evaluated the performance of CRP using both synthetic rulesets and the real rulesets. CRP achieves $3.2\times$ classification speed and $4.2\times$ update speed on average compared with state-of-the-art algorithms. We also implement CRP over OVS, and the throughput is $10\times$ that of the native OVS. The remarkable results demonstrate the superior performance of CRP in practice. It successfully explores the use of deep learning technique to achieve both high-speed packet classification and online update.

References

[1] M. Suh, S. H. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in *Proc. 16th Int. Conf. Adv. Commun. Technol.*, Feb. 2014, pp. 744–748.
[2] C. Lenzen and R. Wattenhofer, "Tight bounds for parallel randomized load balancing," in *Proc. 43rd Annu. ACM Symp. Theory Comput. (STOC)*, 2011, pp. 11–20.
[3] M. S. Seddiki *et al.*, "FlowQoS: QoS for the rest of us," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 207–208.
[4] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the art, challenges, and implementation in next generation mobile networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, Nov./Dec. 2014.
[5] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[6] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.
[7] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.
[8] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.
[9] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *Proc. NSDI*, vol. 17, 2017, pp. 315–328.
[10] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 49–54.
[11] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2015, pp. 347–359.
[12] J. Daly *et al.*, "TupleMerge: Fast software packet processing for online packet classification," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1417–1431, Aug. 2019.
[13] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
[14] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.
[15] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Proc. Hot Interconnects VII*, Stanford, CA, USA, vol. 40, Aug. 1999.
[16] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2003, pp. 213–224.
[17] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "Efficuts: Optimizing packet classification for memory and throughput," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 207–218, Oct. 2011.
[18] P. He, G. Xie, K. Salamatian, and L. Mathy, "Meta-algorithms for software-based packet classification," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 308–319.

[19] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 135–146, Aug. 1999.

[20] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, "A sorted partitioning approach to high-speed and fast-update OpenFlow classification," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, Nov. 2016, pp. 1–10.

[21] T. Shen *et al.*, "RVH: Range-vector hash for fast online packet classification," ICT, Tech. Rep.,2018.

[22] Y. Le Cun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1995, pp. 255–258.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[24] B. Pfaff *et al.*, "The design and implementation of open vswitch," in *Proc. NSDI*, vol. 15, 2015, pp. 117–130.

[25] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 193–204, Oct. 2005.

[26] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "SAX-PAC (scalable and expressive packet classification)," in *Proc. SIGCOMM CCR*, Aug. 2014, vol. 44, no. 4, pp. 15–26.

[27] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, "Partial order theory for fast TCAM updates," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 217–230, Feb. 2018.

[28] B. Vamanan and T. N. Vijaykumar, "TreeCAM: Decoupling updates and lookups in packet classification," in *Proc. 7th Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2011, pp. 1–12.

[29] M. Varvello, R. Laufer, F. Zhang, and T. V. Lakshman, "Multilayer packet classification with graphics processing units," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2728–2741, Oct. 2016.

[30] W. Jiang and V. K. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.

[31] P.-C. Wang, "Scalable packet classification with controlled cross-producting," *Comput. Netw.*, vol. 53, no. 6, pp. 821–834, Apr. 2009.

[32] P. Gupta and N. McKeown, "Packet classification on multiple fields," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, pp. 147–160, Oct. 1999.

[33] E. Liang, H. Zhu, X. Jin, and I. Stoica, "Neural packet classification," 2019, *arXiv:1902.10319*. [Online]. Available: http://arxiv.org/abs/1902.10319

[34] W. Li *et al.*, "Tuple space assisted packet classification with high performance on both search and update," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1555–1569, Jul. 2020.

[35] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. London, U.K.: Pearson, 2016.

[36] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. Cambridge, MA, USA: MIT Press, 2018.

[37] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499–511, Jun. 2007.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[39] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 818–833.

[40] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[41] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

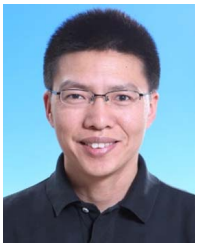
[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[43] X.-M. Niu and Y.-H. Jiao, "An overview of perceptual hashing," *Acta Electron. Sinica*, vol. 36, no. 7, pp. 1405–1411, 2008.

[44] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*. [Online]. Available: http://arxiv.org/abs/1603.04467

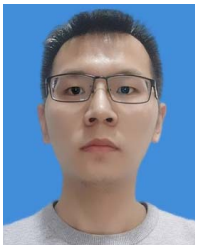
[45] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.

Xinyi Zhang received the B.S. degree in network engineering from Beijing University of Posts and Telecommunications in 2016. She is currently pursuing the Ph.D. degree with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). Her research interests include Internet architecture, software defined networking and packet classification.



Gaogang Xie (Senior Member, IEEE) received the Ph.D. degree in computer science from Hunan University in 2002. He is currently a Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). His research interests include Internet architecture, packet processing and forwarding, and Internet measurement. He is a member of the ACM.

Xin Wang (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Columbia University, USA. She is currently an Associate Professor with the Department of Electrical and Computer Engineering, the State University of New York at Stony Brook, USA. Her research interests include algorithm and protocol design in wireless networks and communications, mobile and distributed computing, and networked sensing and detection. She was a member of ACM in 2004. She received the NSF Career Award in 2005 and the ONR Challenge Award in 2010.



Penghao Zhang received the B.S. degree in computer science from Hunan University, Changsha, China, in 2017. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). His research interests include programing switch, packet classification, and SDN policy anomaly detection.

Yanbiao Li received the B.S. degree in mathematics and the Ph.D. degree in computer science from Hunan University in 2009 and 2016, respectively. He is currently an Associate Professor with the Computer Network Information Center (CNIC), Chinese Academy of Sciences (CAS), and the University of Chinese Academy of Sciences (UCAS). His research interests include networked systems, packet processing algorithms and routing security.

Kavé Salamatian (Member, IEEE) received the bachelor's degree in electronic engineering, the bachelor's degree in mathematics, the master's degree in telecommunications systems, the M.B.A. degree in business strategy, the D.E.A. degree in theoretical computer science, and the Ph.D. degree in computer science from the University of Paris Sud. He has been a Professor with the University of Savoie since September 2009. His main areas of expertise are Internet measurement and information theory in networks. He is a member of the ACM.