RVH: Range-Vector Hash for Fast Online Packet Classification

Tong Shen, Gaogang Xie, Xin Wang, Zhenyu Li, Xinyi Zhang, Penghao Zhang, Dafang Zhang

Abstract—Packet classification according to multi-field ruleset is a key component for many network applications. Emerging software defined networking and cloud computing need to update the rulesets frequently for flexible policy configuration. Their success depends on the availability of the new generation of classifiers that can support both fast ruleset updating and high-speed packet classification. However, existing packet classification approaches focus either on high-speed packet classification or fast rule update, but no known scheme meets both requirements. In this paper, we propose Range-vector Hash (RVH) to effectively accelerate the packet classification with a hash-based algorithm while ensuring the fast rule update. RVH is built on our key observation that the number of distinct combinations of each field prefix lengths is not evenly distributed. To reduce the number of hash tables for fast classification, we introduce a novel concept range-vector with each specified the length range of each field prefix of the projected rules. RVH can overcome the major obstacle that hinders hash-based packet classification by balancing the number of hash tables and the probability of hash collision. Experimental results demonstrate that RVH can achieve the classification speed up to 15.7 times and the update speed up to 2.3 times that of the state-of-the-art algorithms on average, while only consuming 44% less memory.

I. INTRODUCTION

Packet classification, where each incoming packet is matched against a multi-field ruleset in classifier, is one of the essential operations applied in switches, routers and other network appliances to support security [1], quality of service (QoS) [2], [3] and advanced functions [4], [5]. For example, to protect network resources from being abused, five-field firewall rules are often added to a switch to determine if packets should be forwarded or dropped. Generally, in traditional applications, rules are kept relatively static, thus fast classification can be achieved with the algorithms running over well-designed data structure of classifiers built offline. In the past, the main goal of classifier design is to perform high-speed packet processing, such as content detecting, load balancing, packet filtering, and forwarding, within reasonable memory footprint. As the rule update is infrequent at that time, the classifier can be built offline.

The emergence of software defined networking (SDN) [6]–[8] creates enormous opportunities for network innovation to support new features and value-added functions. This includes the support of traffic engineering [9], network function virtualization (NFV) [10], [11] and high-performance cloud computing [12], [13]. However, these new functions rely on the capability of dynamic update of rules in the classifier [14], [15], besides the basic fast packet classification. On the one hand, these new applications should respond to a large amount of requests from different users instantly, so that the classifier rules have to be updated frequently to meet different requirements. On the other hand, the regular migration of network functions and the alteration of network topology and policy will update the classifier rules accordingly. This requires packet classification to be performed online with the support of fast dynamic rule update, and which is a definitely mandatory requirement for current and future classifiers.

Although packet classification is very important and has drawn a lot of research attention, existing algorithms often cannot meet the above two requirements at the same time. Decision-tree based algorithms, such as HyperCuts (HyC) [15], EffiCuts (EC) [17] and SmartSplit (SS) [18] can achieve high-speed classification but not fast rule update. Hash-based algorithms, such as Tuple Space Search (TSS) [19] implemented in Open vSwitch (OVS) [20], can achieve fast rule updating but not high-speed classification. PartitionSort (PS) [14] and TupleMerge (TM) [21] can improve classification speed at the cost of updating time. Achieving fast packet classification and rule update simultaneously is one of the fundamental challenges that need to be addressed to meet the emerging needs of advanced network management and efficient cloud computing.

To support both high-speed packet classification and fast rule updates, we propose an online packet classification algorithm Range-vector Hash (RVH), which achieves ultra-performance rule update by using efficient hash function, and trades off between the number of hash tables and the probability of hash conflicts to minimize the packet classification time. Specifically, to reduce the classification time, we introduce a novel concept range-vector. To reduce the number of hash tables thus the classification time, we partition the ruleset into a series of sub-rulesets which are covered by range-vectors based on their distribution of each field prefix length. Each range-vector maintains a hash table, and the classification rules are mapped to different range-vectors (hash tables) with a proper design to bring down hash conflicts. In this way, updating a rule requires only one hash operation on average. Our main contributions are summarized as follows:

- We propose a novel concept range-vector to efficiently partition the rule space for both fast packet classification and rule update. RVH can largely reduce the hash operations for packet classification while only requiring one hash operation for a rule update.
- We propose the use of base-vector to guide the construction of hash keys in the presence of field prefixes with heterogenous lengths in the same hash table.
- We propose an analytical model to effectively capture the cost for packet classification, which can serve as a base to
minimize the classification time with the proper selection of the number of hash tables to construct.

- We propose a partitioning policy which effectively reduces the overlap of rules thus additional overhead (e.g., hash conflict) during the classification.

The packet classification module with mega-flow and micro-flow in OVS is originally implemented with TSS [19], and has been implemented with RVH, PS [14] and TM [21] in our experiments. We evaluate the performance of RVH with extensive experiments on a production operated cloud platform. Our results show that the packet classification performance of RVH is 15.7x, 14.1x, and 1.6x on average that of TSS, PS and TM respectively. The update speed of RVH is 3.9x that of PS, 1.7x that of TM and 1.1x that of TSS on average. Concerning the memory footprint, RVH occupies the space only 38% that of PS, 56% that of TSS and nearly the same as TM on average. In general, the performance of RVH outperforms all existing similar hash-based packet classification algorithms.

The remainder of this paper is organized as follows. In Section II we survey related work and summarize their differences from RVH. We describe the range-vector space and the algorithms in Section III. We present and verify the performance model of packet classification for RVH in Section IV. In Section V we describe the distribution of the prefix length on each field and propose the partition policy for range-vectors. We evaluate the performance of RVH in Section VI and discuss the ways to further improve RVH’s performance in Section VII. Finally, our work is summarized in Section VIII.

II. RELATED WORK

Existing packet classification schemes fall into three categories: the hardware based, the dimensionality reduction based, and the space partitioning based.

Hardware Based: Ternary Content Addressable Memory (T-CAM) [22] is the de facto standard chip in practice for high-speed packet classification. T-CAM utilizes a native hardware parallel search to achieve low deterministic lookup time. However, T-CAM suffers from limited memory size, slow rule update and power-hungry. Other hardware platforms proposed for packet classification include Graphics Processing Unit (GPU) [26] and Field Programmable Gate Array (FPGA) [3]. And hardware instruction [28], specified chip and programming language [29] have also been proposed for packet classification. Their memory size, flexibility and cost are the barriers that prevent these solutions from widespread usage.

Dimensionality Reduction Based: Cross-producing [30] and RFC [31] split multi-dimensional rules into several single-dimensional ones to first match each individually, and then merge the temporary results of individual match. The update is slow due to each single-dimensional table should be updated for one rule update. Furthermore, the final merging process will become the performance bottleneck when the ruleset is large.

Space Partitioning Based: In these approaches, the whole ruleset is divided into several sub-spaces, with a small set of rules fallen into each sub-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 sub-ruleset in the corresponding sub-space. This type of approaches further falls into two main subcategories: the decision tree based and the hash based approaches.

The key idea of decision tree based approaches such as HiCuts (HiC) [24] and HyperCuts (HyC) [16] is to partition the search space recursively into several regions until the number of rules in each region is below a certain threshold. Due to the efficiency of the decision tree, these approaches can achieve high-speed packet classification. However, one of the shortcomings is large memory consumption because of rule replication, as some rules may need to be copied into multiple partitions. Slow and complicated rule updating is another drawback of these approaches. While EffiCuts (EC) [17] and SmartSplit (SS) [18] adopt the different policies of rule space partition to reduce the rule replication and the number of memory access based on rule distribution, they still fail to support fast update.

Existing hash based approaches, on the other hand, can achieve fast rule update, but not high-speed classification.

Tuple Space Search (TSS) [19], which is implemented in OVS, partitions rules into different subsets based on tuples. A tuple is formed with a list of numbers, which represent the prefix lengths corresponding to the fields in the ruleset. Each subset is represented as a hash table to accelerate rule update, and packet classification is achieved with linear memory consumption. In order to classify a packet, there is a need to search all the hash tables, and the classification time will increase linearly with the number of hash tables. This will impair the classification performance seriously when dealing with large rulesets.

Pruned Tuple Space Search (PTSS) [19] improves the performance by filtering out a number of unmatched tuples using tries. Although the number of tuples can be indeed reduced in PTSS, merging the results is time consuming and the update operation is still complicated.

PartitionSort (PS) [14] combines the benefits of both TSS and decision trees. Rather than partitioning rules based on tuples, PS partitions rules into sortable rulesets and stored them through balanced search trees. As a result, PS can classify packet faster than TSS with the reduction of hash calculations, but at the cost of more time to process sortable ruleset. In other words, compared with TSS, PS is faster at classifying packets at the cost of slowing rule updating.

TupleMerge (TM) [21] improves the classification of TSS by reducing the number of resulting tables. TM defines compatibility between rules and tuples so that the rules with similar but not identical tuples can be placed in the same table. However, this approach may cause table overlaps. As a result, a rule may not be mapped to a definite table, which significantly compromises the performance of rule update. Moreover, its number of hash tables will increase over time, which also severely degrades its classification performance. For the performance improvement, all hash tables have to be reconstructed when the number of tables exceeds a certain
TABLE I
A SAMPLE CLASSIFIER

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

TABLE II
RANGE-VECTORS FOR THE SAMPLE CLASSIFIER

| Hash Table # | Range-vector | 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              |

Advantages of RVH: The goal of our proposed RVH is to simultaneously support high-speed packet classification and fast rule update to meet the urgent requirement of emerging applications such as SDN and cloud computing. RVH builds on the key observation that the number of distinct combinations of field prefix lengths is not evenly distributed. Following this observation, RVH introduces the concept of range-vector to balance the cost of the key hashing and the match verification upon the existence of hash conflict. No matter how the rules are updated, the number of hash tables will not increase beyond the original partition. Therefore, RVH can achieve high-speed classification and fast update at the same time.

III. RANGE-VECTOR HASH

Our goal in this work is to propose a new system model that can concurrently support high-performance packet classification and fast rule update to meet the emerging need of flexibly reconfiguring network functions for value-added services.

In this section, we first introduce the concept of range-vector along with the procedures for the construction of hash tables based on range-vectors, then describe the processes of packet classification and rule update based on RVH, and finally compare the time and space complexity between RVH and other three reference schemes.

A. Range-vector Hash Table

We introduce the concept of Range-vector Hash Table using an example classifier shown in Table I. The classifier contains ten rules, each is formed with four fields: source address (SA) and destination address (DA) each with the maximum length 5 bits, a priority (Pri) field, and an action (Act) field. For packet classification, we take the numeric fields as inputs to find the actions to take. In this example, for each incoming packet, we use SA and DA to search for the matching rule. We first introduce length-vector. Each rule can be mapped to a length-vector, which is a vector with each element representing the length of the corresponding field in the rule. As rule #0 has a 3-bit SA and a 5-bit DA, its length-vector is (3, 5). The length-vectors of all rules in this example classifier can be summarized with a range-vector (0, 6), (0, 6)). A range-vector is a vector with each element representing a length range of the corresponding field in the ruleset. To facilitate fast rule matching, a range-vector can be further divided into several disjoint finer-grained range-vectors, each containing a set of length-vectors. For instance, the range-vector (0, 6), (0, 6)) can be divided into four disjoint fine-grained range-vectors ((0, 3), (0, 4)), (0, 3), (4, 6)), (3, 6), (0, 4)) and (3, 6), (4, 6)). The length-vector of rule #0, (3, 5), will be mapped to the range-vector (3, 6), (4, 6)). Obviously, the first length 3 falls into the range (3, 6), and the second length 5 falls into the range (4, 6). In order to improve the speed of classification, in this work we divide the whole ruleset into several subsets based on the range-vectors. We apply a hash table to quickly index rules mapped to each range-vector. The rules in this example can be indexed through four hash tables based on four range-vectors in Table II, where each table is associated with a range-vector.

In Figure 1(a) we plot the length-vectors corresponding to rules in Table I over two dimensions, where each point represents a length-vector. Each range-vector is represented with a grey rectangle in Figure 1(b) and corresponds to a hash table, which stores a set of rules. In order to look for a rule with a hash function, we need a hash key. A natural way is to construct the key with the concatenation of field bits used for the rule matching. However, different rules may correspond to different lengths of field prefixes (e.g., SA and DA in Table I), while we need hash keys to have the same length. To address this issue, we further introduce the concept of base-vector to regulate the lengths of field prefixes used in the concatenation as the hash keys. More specifically, for each range element in the range-vector, we select its lower bound as a corresponding element of the base-vector to restrict the length for concatenation. As each hash table is associated with one range-vector, it is also associated with one base-vector.

Table III shows a sample for hash table #0 (ignoring the empty entries) built according to the range-vector
Although packet classification is very efficient by using the hash function, the existence of multiple rules with different priority values and the rule overlapping can compromise the performance. To further improve classification efficiency, we propose the exploration of priority sorting:

1) **Hash table priority sorting.** We set the priority of a hash table as the highest priority of the rules contained in the table, and sort hash tables from high to low according to their priority. The search of range-vector hash tables can terminate once we find a matched rule whose priority is not less than the priority of the next hash table. When two hash tables have the same priority, we sort them according to their base-vector’s modulus from large to small. Hash tables with larger modulus may have higher average priority, because whose rules have longer field lengths.

2) **Overlapped rule priority sorting.** A hash value may correspond to multiple rules due to hash collision or rule overlapping. To reduce the time needed to further determine which rule is matched, we sort the overlapped rules from high to low according to their priority during rule inserting. The verification can stop once a matched rule is found without the need of checking all the overlapped rules.

We list the pseudocode of packet classification using RVH in Algorithm 1.

We take the classifier in Table III as an example to classify a packet whose SA and DA fields are (111111, 100000). With the hash table priority sorting, the order of the hash table list is: hash table #0 with the priority of 4, hash table #2 with the priority of 2, hash table #3 with the priority of 0, and finally hash table #1 with the priority of 1. So we search for the matching rule from hash table #0. With the base-vector (3, 4), we form the hash key “1111000” and apply it to check the hash table #0. The hashing returns two rules, #2 and #3. Since rule #2 has a higher priority (as shown in Table I), we verify it first. We find that rule #2 matches the packet and its priority is larger than that of the next hash table. So, the search operation terminates, and rule #2 is returned as the matched one.

### Algorithm 2 Rule Insertion with RVH

**Input:** *u*, a rule to insert;  
**Input:** *L*, a list of hash tables with priority sorting;  
**Output:** *ret*, a boolean indicating the result of insertion;  

1. \( ret \leftarrow 0; \)  
2. \( t \leftarrow [\text{map}(u)]; \)  
3. if \( t[\text{hash}(u)] \leftarrow t[\text{hash}(u)] + u \) then  
4. \( \text{pri_sort}(t[\text{hash}(u)]); \)  
5. \( \text{pri_update}(t); \)  
6. \( \text{pri_sort}(L); \)  
7. \( ret \leftarrow 1; \)  
8. end if return *ret*;

### Algorithm 1 Packet Classification using RVH

**Input:** *p*, a data packet;  
**Input:** *L*, a list of hash tables with priority sorting;  
**Output:** *ret*, the highest-priority matched rule;  

1. \( ret \leftarrow \text{null}; \)  
2. for \( t \) in *L* do  
3. if \( \text{pri}(ret) \geq \text{pri}(t) \) then  
4. break;  
5. end if  
6. for \( r \) in \( t[\text{hash}(p)] \) do  
7. if \( r \) matches \( p \) then  
8. if \( \text{pri}(r) > \text{pri}(ret) \) then  
9. \( ret \leftarrow r; \)  
10. end if  
11. end if  
12. end for  
13. end for return *ret*;

(\([3, 6], [4, 6]\)), and the rules #0 ~ #5 are mapped to the hash table #0. The sample hash table #0 has the base-vector (3, 4), based on which the rule #0 forms its hash key “1001101” by concatenating the first 3 bits and the first 4 bits of its SA (“100+”) and DA (“11010”) prefixes. The key is hashed to the example address with the offset “0x90”. Other rules are stored at different hashed addresses using the same method.

As the use of the base-vector omits some bit information and provides a wild card for matching, two different rules may correspond to the same hash key. For example, in Table III both rules #2 and #3 have the same hash key “1111000”. We call this phenomenon the rule overlap. When searching the range-vector hash table, rule overlapping requires a further verification to find the exact matching rule.

### B. Packet Classification

When receiving a packet, the classifier needs to search each hash table to find the best matching rule. A packet may be matched with several rules, and we apply the priority to determine the final rule for action. Initially we set a dummy rule with the priority of 0. As a basic scheme, all existing range-vector hash tables are sequentially searched, and the matched rule with the highest priority is recorded. After searching through all tables, the classification module either returns the matched rule with the highest priority or reports that no rules are matched.

Although packet classification is very efficient by using the hash function, the existence of multiple rules with different priority values and the rule overlapping can compromise the performance. To further improve classification efficiency, we propose the exploration of priority sorting:

| Hash Value | Hash Key | Rules |
|------------|---------|-------|
| 0x12       | 1011001 | 1     |
| 0x34       | 0100110 | 4     |
| 0x56       | 0010100 | 5     |
| 0x78       | 1111000 | 2, 3  |
| 0x90       | 1001101 | 0     |

### Table III

* A Sample for Hash Table #0
TSS. Our experiments in Section VI confirm that RVH has
approach that has the update performance comparable with
highest packet classification performance, and is also the only
shown in Algorithms 2 and 3.

The pseudocode for the rule update operation, including rule insertion and deletion, is reordered according to their priority. The pseudocode for the algorithms is as follows:

Algorithm 3 Rule Deletion with RVH.

Input: $u$, a rule to delete;
Input: $L$, a list of hash tables with priority sorting;
Output: $ret$, a boolean indicating the result of deletion;

1: $ret \leftarrow 0$;
2: $t \leftarrow L[map(u)]$;  # $t$, current hash table
3: if $\{hash(u)\} \leftarrow \{hash(u)\} - u$ then
4: $pri\_update(t)$
5: $pri\_sort(L)$
6: $ret \leftarrow 1$;
7: end if
8: return $ret$;

The superior performance to meet the requirement for a modern classifier.

IV. PERFORMANCE MODEL

In order to optimize RVH with high classification and update performance, we need to determine the number of range-vectors thus the number of hash tables to build, and the policy to cover the whole ruleset with range-vectors. As a guide, we propose a performance model to analyze the bottlenecks and factors that impact the performance. We verify the accuracy of our model with experimental results.

A. Overview of RVH

In RVH, we have introduced the concept of range-vector. A set of disjoint range-vectors partition the matching space (i.e., the whole ruleset), with each range-vector associated with a hash table for the quick match of a packet against rules contained in that hash table.

In general, classifying a packet using RVH consists of three procedures: 1) searching all range-vector hash tables, and for each table a simple hash operation is performed to determine if there are rules matching the header of the incoming packet; 2) verifying the exact rules matched in a hash table if there exist several ones stored in the same hashed address to avoid the false positive caused by hash conflict or rule overlap; and 3) determining the rule with the highest priority among the verified ones.

When updating a rule, the hash table where the rule should be contained is determined first. Inserting a rule into a range-vector hash table takes only two steps: 1) mapping the rule to the corresponding range-vector (hash table), or creating one if the hash table does not exist; and 2) storing the rule at the proper entry of the hash table determined by the hash operation. The process of deleting a rule from range-vector hash table is similar to the inserting procedure.

B. Analytical Performance of RVH

Based on the logic and process of RVH approach described above, we establish its performance model for packet classification to formally explore the performance bottleneck of RVH. Table V lists the notations used in our analysis.

The total time $T$ taken to classify an incoming packet with a classifier consists of three parts: the time for hashing, for
match verification and for priority selection. The time for hashing is decided by the number of hash tables (range-vectors) and the cost of each hash function. The match verification should be undertaken for each hash table matched with multiple rules to avoid the false positive from hash collision or rule overlap. The amount of match verification for an incoming packet equals to the number of matched entries. Given the specific refined hash function, the probability of matching an entry in a hash table is proportional to the utilization ratio of each hash table. The utilization ratio is defined as the number of entries divided by the size of the hash table.

The rules with the same hash value have to be further checked to eliminate the ones with false positive, and the rule with the highest priority is selected when multiple ones are matched. Therefore, the rule overlap will increase time for match verification. The average time spent for rule verification is the time cost for a rule verification weighted by the average overlap ratio, which is defined as the number of rules in the hash table divided by the size of the hash table.

Revisiting the hit ratio and the overlap ratio in (1) with (2) and (3), the time cost to classify a packet using RVH can be converted to

\[
T = \sum_{i=1}^{m} h_i + \sum_{i=1}^{m} c_i r_i o_i + \bar{q} \tag{4}
\]

which can be simplified to

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

The number of entries in the i-th hash table (\(e_i\)) and the size of the i-th hash table (\(s_i\)) are related to the design of the hash. Besides, the number of hash tables (\(m\)) depends on the distribution of the prefix length in each dimension. Given a specific hardware platform, the average time for hashing \(\bar{h}\), for verification \(\bar{c}\) and for priority comparing \(\bar{q}\) are often invariant. Therefore, based on Eq. (5) we can conclude safely that both the number of hash tables and the overlap ratio in individual tables determine the classification performance of RVH: the fewer the number of hash tables produced by the ruleset and the lower overlap ratio, the higher the performance.

**Table VI**

| Rulesets | \(m\) | Saturation | \(T\) (\(\mu s\)) | Error (%) |
|----------|------|------------|-----------------|==========|
| ACL1     | 703  | 0.71       | 45.23           | 4.31     |
| ACL2     | 702  | 0.71       | 45.16           | 5.28     |
| FW1      | 181  | 0.73       | 11.66           | 6.90     |
| FW2      | 179  | 0.74       | 11.54           | 6.67     |
| IPC1     | 71   | 0.80       | 4.59            | 11.64    |
| IPC2     | 71   | 0.82       | 4.60            | 10.07    |
| Cloud1   | 226  | 0.72       | 14.55           | 6.45     |
| Cloud2   | 251  | 0.77       | 16.22           | 8.37     |

![Fig. 2. Comparison of the theoretical and experimental results for packet classification with RVH. The (theoretically) estimated performance is generated by our analytical model (Eq. 5), while the experimental results are obtained using the RVH implementation in OVS.](image)

C. Verification of the Performance Model

We verify the above performance analysis via experiments. To this end, we use the RVH core module implemented in OVS without any flow cache. The detailed experiment setup and the rulesets can be found in Section VI.

Firstly, we tested one billion times of hashing, match verification and priority comparison, and obtained the average time of hashing a packet (\(\bar{h}\)), the average time of verifying a match (\(\bar{c}\)), as well as the average time of comparing priority (\(\bar{q}\)), which are 61.0ns, 4.7ns and 0.9ns respectively. We then determined the utilization ratio and the overlap ratio of each hash table. The average overlapping ratio equals to \((1/m) \cdot \sum_{i=1}^{m} n_i/s_i\) and is denoted as saturation in the result Table VI. Based on these parameters, we can estimate the time of classifying a packet based on Eq. (5).

Secondly, we run experiments with the RVH implementation in OVS without flow cache over the rulesets in Table VII to obtain the classification performance in practice. Figure 2 shows the quantile time used to classify a packet over 10 runs (the boxes and whiskers), along with the theoretically estimated time (diamond points). The average relative error between estimated time and average actual time is also listed in Table VI. We can visually see that the estimated time taken to classify a packet is very close to the actual values, though the actual value is slightly higher than the estimated one. This is because that additional calculations and CPU cache replacement (which our model does not capture) may take extra time, which is indeed not negligible.

In summary, we can safely make the following conclusions from the above analysis: 1) our performance model captures the major components of RVH, and provides reasonably precise estimation for its performance; 2) the key to significantly
improve RVH’s classification performance is to reduce the number of hash tables (i.e., \( m \)) while maintaining a low overlap ratio.

For the same reason, the main idea of both PS and TM is to reduce the number of hash tables to improve their packet classification performance. However, PS combines the decision tree, which makes the updates more difficult and results in additional overhead. While for TM, a rule may be mapped into different tables, depending on the table utilization. In addition, as time goes on, the number of tables increases, which will trigger an overall table reconstruction. These processes cause a dramatic drop in the update speed of TM. With the proper determination of the number of hash tables to use based on our performance model, RVH can achieve a high classification speed while maintaining the fast update property. We will discuss how to partition the whole ruleset into range-vectors in the next section.

V. RANGE-VECTOR PARTITION POLICY

From our performance model and experimental studies in Section IV, the classification speed is highly impacted by the number of hash tables. In this section, we first introduce our key observations on the features of ruleset, and then propose the distribution-based policy to partition the packet matching space for hashing in RVH. Finally, we study the stability of prefix-length distribution in each dimension.

A. Tradeoff for Partition

From our performance model, the time taken for packet classification depends on both the number of hash tables (range-vectors) and the rule overlap ratio in individual tables. These two factors are conflicting: the more the number of hash tables, the lower the chance of rule overlapping in each table. At one extreme, if we put all rules into one hash table and all rules are also mapped to the same hash entry, classifying a packet requires the search of the rule list linearly. At the other extreme, if we let each range-vector contain only one length-vector, then the classification approach is essentially TSS.

The classification time will reach the minimum if we can find the ideal balance between the two factors. In practice, it is hard to represent the relationship of the two with a close-form equation. We resort to experiments to show their interaction. We use the rulesets in Table VII to classify packets based on the source address (SA) and destination address (DA). We partition the 2-dimensional overall rules evenly into a series of range-vectors, with each dimension split into a number of segments. We report the time for hashing (i.e., \( mh \) in Eq. 5), the time for match verification (i.e., \( c \sum_{i=1}^{m} n_i/s_i \) in Eq. 5), and the total time cost (i.e., \( T \) in Eq. 5) in Figure 3, where \( x \)-axis represents the number of segments in each dimension. Note that \( \bar{h}, \bar{c} \) and \( \bar{q} \) are taken from Section IV, where \( \bar{h} = 61.0 \text{ns} \), \( \bar{c} = 4.7 \text{ns} \) and \( \bar{q} = 0.9 \text{ns} \).

As expected, for each ruleset, there is a “sweet point” of the number of segments to split in each dimension, where the total time cost reaches the minimum. This “sweet point” is ruleset dependent, but falls into the range between 3 and 5. With this range of segments in one dimension, the number of two-dimensional regions corresponding to the number of range-vectors (hash tables) varies from 9 to 25. A value smaller than the “sweet point” yields an extraordinarily high time cost for match verification, which dominates the total time cost. Beyond this point, the total time for verification grows slowly, but the time for hashing increases quadratically with the number of segments.

B. Distribution-based Partition

The range-vector formed with the overall lengths of prefixes to match is partitioned into multiple range-vectors with two features: 1) all range-vectors together covers the overall lengths of prefixes to match, that any rule can be mapped to one of range-vectors; and 2) range-vectors do not intersect with each other, so that a rule can be mapped to only one range-vector.

We follow the principles below for the division of range-vectors:

1. The number of range-vectors should be as small as possible. The number of hash calculations for packet classification is proportional to the number of range-vectors. Our experimental results show (see Figure 3) that keeping this number within the ranges [9, 25] can achieve fast classification.
2. The number of overlapped rules in hash tables should be kept as few as possible to reduce the time taken for verifying the match (see Figure 3). As the number of range-vectors decreases, each hash table will contain more rules, which raises the chance of rule overlap.
3. The field lengths of rules that are mapped to a range-vector should be as close as possible to that specified by
the corresponding base-vector. In other words, for each field of a rule that is used for matching, its length should be as close to the lower bound of the corresponding range as possible. Otherwise, as we use the cut-short prefix to form the hash key, it will increase the likelihood of rule overlap.

To find the best construction of range-vectors that meets the above principles, we may apply mathematical tools such as those based on linear programming. However, these tools often take a long time to run and are not applicable for online operation. In this paper, we propose a simple yet effective construction method.

We first examine the distribution of prefix length combinations, taking SA and DA of the rulesets in Table VII as examples. From the results in Figure 4, the distribution is not uniform in the 2-dimensional space, but clustered in several ranges. This clustering behavior can be observed from all four rulesets we examined. This is the result of two facts. First, IP addresses are often assigned in blocks with several typical prefix lengths (e.g., 16-bit prefix, 24-bit prefix). Second, network functions are configured to work on IP blocks. For instance, firewall policies may block all source IP addresses from some blocks, which forms the left-most cluster in Figure 4. We construct range-vectors following the clustering behavior. As the number of clusters is limited, there will be only several range-vectors, which effectively reduces the number of hash tables. In addition, we can align the range-vectors to allow the rules to be close to the lower bounds of range-vectors. Finally, more rules are likely to fall into a range whose duration is large. Setting the range based on the cluster size helps to reduce the range-size thus the chance of rule overlapping.

In our construction policy, we partition the overall length range for prefixes along each dimension separately. Taking SA and DA fields as an example, we partition the range-vector first along the SA dimension, then along the DA dimension. To partition the range-vector along a specific dimension, we first compute the empirical cumulative distribution function (CDF) of the prefix of rules along this dimension (see Figure 5), and then take the following steps:

1) **Locate partition point:** find the prefix length that most rules are projected to. To this end, we compute the derivative of CDF at each possible prefix length, and then pick out the ones whose derivatives are greater than the average derivative value calculated as the slope of the line connecting the beginning and end points. These values correspond to the prefix lengths that many rules have. In the example of Figure 5(a), these values are 12, 14, 15, 16, 17, 23, 24, 25, 26, 30, 31, and 32.

2) **Combine length:** Combine adjacent lengths obtained from the first step as a small range, and add the smallest length as a range to incorporate all possible prefix lengths. In this example, we get [12, 12], [14, 17], [23, 26], [30, 32], and we also add the smallest length range [0, 0] as considering the rules whose prefix length is less than 12.

3) **Merge ranges:** Merge two adjacent ranges, if the gap between them does not exceed D while the size of the merged range is less than S to constrain the probability of rule overlapping. In our example, we set D = 2 to ensure the two ranges to be close as possible and S = 8 to ensure the number of overlaps among rules is small. We introduce more criteria for the setup of these parameters in Section VII. With these parameters, we merge [12, 12] and [14, 17] to form a new range [12, 17], and get [0, 0], [12, 17], [23, 26], and [30, 32].

4) **Align range:** Align all ranges backwards to ensure the
whole range space is included for the rule matching. We get [0, 11], [12, 22], [23, 29], and [30, 32] in our example. These ranges are the partitions along this dimension.

Partition along other dimensions follows the same procedure. Finally, the Cartesian products of every dimension’s ranges generate the range-vectors.

C. Stability of Rule Distribution

In general, for a classifier, its prefix length distribution for each field is stable [33], which barely changes as the rule updates. We have tracked the changes of the rule distribution in two real ruleset Cloud1 and Cloud2 (Table VII) during a period of three months that they served for the public network. The result tells us that, although rules have undergone a lot of updates, their prefix length distribution for each field has not changed.

Therefore, for a specific ruleset, we only need to perform the range-vector partition during the initialization period. Of course, if the prefix length distribution of any dimension changes, the partition will be dynamically adjusted. In this case, only a few corresponding hash tables need to be rehashed, rather than reconstructing the whole classifier.

VI. EXPERIMENTAL RESULTS

In this section, we perform extensive experiments to compare the performance of RVH with three state-of-the-art packet classification approaches, TSS [19], PS [14], and TM [21]. We evaluate the performance of packet classification, performance of rule updating, and the memory footprint.

A. Platform Implementation

Open vSwitch (OVS) [20] is a software implementation of a distributed virtual multi-layer switch. OVS can support multiple protocols (such as OpenFlow [34]) and standards used in computer networks. As a widely used software-based switch, OVS is designed with mega-flow table and micro-flow table to cache rules that were hit recently. This caching mechanism effectively improves the processing speed of packets. To comprehensively evaluate the performance of different classification schemes, we implemented RVH, TSS, PS and TM to generate the ‘big’ flow table in OVS without changing the caching mechanism. When a packet arrives, it is first matched against rules in the micro-flow table. If it is a miss, the packet is further matched against rules in the mega-flow table, and finally the ‘big’ flow table if it misses again in the match of mega-flow table.

B. Experimental Setup

All the experiments were run on a Sugon I620-G20 server with an Intel Xeon CPU E5-2630 v3 @ 2.40GHz, 32 cores, and 128GB DDR3 memory. Each core is integrated with a 64KB 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.

To evaluate the performance of packet classification algorithms, we used 8 different rulesets, which fall into four types (ACL, FW, IPC and Cloud) as shown in Table VII. The first six rulesets are generated by ClassBench [33]. ClassBench generates packet traces and rules following the distribution of real rulesets. The last two rulesets were collected from two operating OpenStack cloud nodes of a major ISP.

In the experiments, we take SA and DA as the fields to match, because other fields are very sparse. We have shown the distribution of rules in four types of rulesets in Figure 4. The distribution is not uniform, but clustered in some ranges, especially for IPC and FW rulesets.

C. Updating

We first examine the performance for updating rulesets, because supporting fast rule update is mandatory in SDN and cloud networks. For instance, OpenFlow rules are often dynamically changed by controllers in SDN networks. For this reason, TSS, which was proposed about 20 years ago for fast update, is used by OVS.

We consider both insertion and deletion operations. Each ruleset was first divided into five subsets evenly, where four

Due to confidential agreement, we are not allowed to reveal the name of the company.
were used as the original ruleset to initialize the classifier, and the remaining one was inserted into the classifier thereafter. We use million updates per second (Mups) to measure the updating speed. As shown in Figure 6(a), RVH outperforms other three approaches greatly. The updating speed of RVH is 3.9× that of PS on average, 1.7× that of TM and 1.1× that of TSS on average. While the time complexity of TSS is the same as that of RVH (see Table IV), RVH achieves a slightly higher performance. This is because RVH reduces the number of hash tables, which causes the cache hit rate to be higher than that of TSS. The low updating performance of PS is due to its use of a tree structure, which degrades the performance. Besides, PS’s updating performance is not stable across rulesets, because the decision tree is dependent on the properties of rulesets. TM fails to achieve a high updating performance as RVH or TSS because of the ruleset overlapping and classifier reconstruction.

We then evaluate the impact of classifier size (i.e., the number of rules in the classifier) on the updating performance in Figure 6(b). In this set of experiments, we used the cloud ruleset and formed it into four parts with various sizes: 0.1K, 1K, 10K and 100K. As expected, the updating speed decreases with the increase of classifier size. Nevertheless, the speed of RVH decreases much slower than the other approaches.

D. Packet Classification without Updating

Next, we examine the packet classification performance when there are no updating operations in Figure 7, where we use million lookups per second (Mlps) as the unit.

We can observe that the packet classification performance of RVH is 15.7× that of TSS, 14.1× that of PS and 1.6× that of TM on average. By using range-vectors instead of tuples in TSS, RVH significantly reduces the number of hash tables, leading to superior performance. PS was proposed to improve the packet classification performance of TSS, but as we can see, for some rulesets (e.g., IPC), its performance is even lower than TSS. This again shows that the performance of decision tree (used by PS) depends on the properties of rulesets. TM achieves a lower performance than RVH, because it maintains a slightly larger number of hash tables and also suffers from the issue of rule overlapping.

While varying the size of classifier (see Figure 7(b)), we observe that RVH outperforms others when the sizes of rulesets exceed 1K rules. Below this value, PS has a better performance. Nevertheless, the performance of PS degrades sharply when increasing the size.

E. Packet Classification with Updating

We then evaluate the impact of classifier updating on packet classification performance using the 8 rulesets in Figure 8 where we vary the update speed. We make the following interesting observations. First, RVH achieves the highest performance regardless of rulesets and the updating speed. Second, very frequent updates will degrade the packet classification performance, especially when the update speed exceeds 10Kups. Third, when the update speed reaches 1000kups, except for RVH, the other three approaches (particularly PS) almost cannot classify packets. Fourth, PS’s performance is heavily dependent on ruleset properties. For instance, PS achieves the performance comparable to TM for cloud rulesets, while it has the lowest performance for IPC rulesets.

F. Memory Footprint

Finally, we examine the memory footprint of different approaches in Figure 9(a). We again see that RVH outperforms others in terms of memory footprint as well. Indeed, RVH and TM are very similar in terms of memory footprint. The memory footprint of RVH is only 38% and 56% of that used by PS and TSS respectively. Recall that PS uses decision tree structures to filter tuples. As a decision tree takes a lot of memory, the footprint of PS is significantly higher than those of other two approaches. Cuckoo hash used in RVH and TSS can greatly improve the utilization of hash tables.

From Figure 9(b), we can see that as the classifier becomes larger, the memory footprint of each algorithm also increases. Nevertheless, the memory footprint of RVH and TM are always less than those of TSS and PS.

VII. DISCUSSION

In this section, we discuss some potential steps to take to further improve the performance of RVH in our future work. Our algorithm partitions the overall prefix length into a few fine-grained range-vectors based on the distribution of rules. As a result, classifying a packet only requires a few hashes, and the performance is significantly improved. This sounds like an ad hoc, but in fact RVH can be applied to any classifier and ruleset.

To further improve the classification speed, we also introduce two procedures, range-vector priority sorting and overlapped rule priority sorting. However, it takes extra time...
to select the rule for match when multiple rules are found to be related to a hash key. To reduce the time for rule verification, in Figure 10 we show statistics on the average number of overlapped rules for each entry based on the sizes of prefix-length ranges. We take SA and DA as example fields to match in this study. The number of overlapped rules for each entry grows rapidly as the size of range increases. We found that as long as the range size is less than 8, the average number of overlapped rules for each entry is less than 1.0. Thus in our partition policy (Section V-B), we would ensure that the size of each range is less than 8.

In the future, we will extend RVH to run in multi-core systems. There are two modes in parallel, Run to Complete (RTC) and Software Pipeline (SPL). We intend to combine these two modes. For packet classification, we intend to run an independent thread for each core while sharing the ruleset for each process. For rule update, we will separate the two operations, locating and updating, and put them into a pipeline. The multi-thread RVH just needs to lock the corresponding rules not the whole classifier, when updating rules. We expect these procedures can significantly increase the speeds for both packet classification and rule update.

VIII. CONCLUSION

In this paper, we propose RVH, a packet classification approach that supports both fast rule update and fast packet classification. RVH is built on the key observation of biased distribution of prefix lengths over the multi-dimensional space. Specially, RVH divides the whole range-length of prefixes into
a few range-vectors to group rules into a small number of hash tables, which significantly increases the packet classification speed while ensuring fast rule update. We provide details on the design of RVH and introduce additional procedures to take for further speeding up the classification. We have evaluated the performance of RVH using both the rulesets generated by ClassBench and the rulesets in production cloud networks. The results consistently show the superior performance of RVH in both rule update and packet classification, compared with the state-of-the-art approaches.

ACKNOWLEDGMENT

This work is supported by the Major State Basic Research Development Program of China (973 Program) under Grant No. 2012CB315805 and the National Natural Science Foundation of China under Grant Nos. 61472130 and 61473123.

REFERENCES

[1] M. Suh, S. H. Park, B. Lee, and S. Yang, “Building firewall over the software-defined network controller,” in Advanced Communication Technology (ICACT), 2014 16th International Conference on. IEEE, 2014, pp. 744–748.

[2] C. Lenzen and R. Wattenhofer, “Tight bounds for parallel randomized load balancing,” Distributed Computing, vol. 29, no. 2, pp. 127–142, 2016.

[3] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song, “Flowqos: Qos for the rest of us,” in Proceedings of the third workshop on HotSDN. ACM, 2014, pp. 207–208.

[4] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, “Nvf: state of the art, challenges, and implementation in next generation mobile networks (vepc),” IEEE Network, vol. 28, no. 6, pp. 18–26, 2014.

[5] J. Grimes and D. McGuinness, “Mobile telecommunications billing routing system and method,” Jan. 7 2004, uS Patent App. 10/541,908.

[6] L. Csikor, “Dataplane specialization for high-performance openflow software switching,” in Conference on ACM SIGCOMM, 2016, pp. 539–552.

[7] A. Sivaraman, C. Kim, C. Kim, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, S. Licking, and S. Licking, “Packet transactions: High-level programming for line-rate switches,” in Conference on ACM SIGCOMM, 2016, pp. 15–28.

[8] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, E. Chen, and E. Chen, “Clicknp: Highly flexible and high performance network processing with reconfigurable hardware,” in Conference on ACM SIGCOMM, 2016, pp. 1–14.

[9] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu et al., “B4: Experience with a globally-deployed software defined wan,” in ACM SIGCOMM Computer Communication Review, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[10] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, “Nfp: Enabling network function parallelism in nfv,” in Conference on ACM SIGCOMM, 2017, pp. 43–56.

[11] J. Rexford, J. Rexford, J. Rexford, J. Rexford, and J. Rexford, “Dynamic service chaining with dyso,” in Conference on ACM SIGCOMM, 2017, pp. 57–70.

[12] D. Firestone, “VFP: A virtual switch platform for host SDN in the public cloud,” in 14th USENIX Symposium on NSDI, 2017, pp. 315–328.

[13] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, “Embark: Securely outsourcing middleboxes to the cloud,” in 13th USENIX Symposium on NSDI, 2016, pp. 255–273.

[14] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, “A sorted partitioning approach to high-speed and fast-update openflow classification,” in Network Protocols (ICNP), 2016 IEEE 24th International Conference on. IEEE, 2016, pp. 1–10.

[15] M. Kuznial, P. Peresini, and D. Kostic, “What you need to know about sdn flow tables,” Cham, pp. 247–259, 2015.

[16] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet classification using multidimensional cutting,” in Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications. ACM, 2003, pp. 213–224.