MultilayerTuple: A General, Scalable and High-performance Packet Classification Algorithm for Software Defined Network System

Chunyang Zhang * ‡, Gaogang Xie†‡
*Institute of Computing Technology, Chinese Academy of Sciences, China
†Computer Network Information Center, Chinese Academy of Sciences, China
‡University of Chinese Academy of Sciences
zhangchunyang@ict.ac.cn, xie@cnic.cn

***Abstract*—Packet classification is one of the core components in Software Defined Network (SDN) systems, e.g., Open vSwitch. However, the current packet classification algorithm Tuple Space Search (TSS), which is implemented in SDN systems, has low lookup speed and can be attacked. Although some algorithms that support incremental updates are proposed to improve the lookup speed, e.g., TupleMerge and PartitionSort, but are not general and scalable to apply in SDN systems and replace TSS. In this paper, we propose a general, scalable, and high-performance packet classification algorithm MultilayerTuple. MultilayerTuple reduces the number of tuples by splitting the prefix lengths of rules into ranges in each layer, then creates the next layer to replace the long rule chain recursively. The experimental results demonstrate that compared to TSS, TupleMerge, and PartitionSort, MultilayerTuple achieves 21.8x, 2.1x, 2.2x lookup speed and 2.3x, 12.3x, 8.5x update speed. Furthermore, we have implemented MultilayerTuple in the OpenFlow table and MegaFlow cache of Open vSwitch, and it achieves 16.0x and 10.2x lookup speed than TSS. Especially when TSE attack happens, MultilayerTuple can effectively defend against it.**

***Index Terms*—packet classification, Software Defined Network, Open vSwitch**

I. Introduction

Software Defined Network (SDN) [1] systems, e.g., Open vSwitch [2], are the core components for network functions. Compared to hardware devices, SDN systems are flexible and scalable for complex applications, and convenient to implement new technologies. However, packet classification is the bottleneck of performance all the time. The current packet classification algorithm TSS [3] in these systems not only performs low lookup speed but also can be attacked. Therefore, it is important to implement a general, scalable, and high-performance packet classification algorithm to replace TSS in these systems.

Although a lot of algorithms have been proposed to improve the lookup speed for packet classification, none is general and scalable to replace TSS in SDN systems. The decision-tree-based methods such as SmartSplit [4], ByteCuts [5] and NeuroCuts [6] focus on high lookup speed, but can not support incremental updates. Recently, TupleMerge [7], [8] and PartitionSort [9] attempt to speed up the lookup with incremental updates. However, TupleMerge only reduces the prefix lengths of the source and destination IP addresses by the authors' experience, and PartitionSort requires determining the order of fields at first and then implementing the complex tree structure. Therefore, both TupleMerge and PartitionSort are not general and scalable to replace TSS. Furthermore, attackers can generate specific rules and packets to attack TSS easily. To improve the performance and defend against the attack, we propose a general, scalable, and high-performance packet classification algorithm *MultilayerTuple*, which can replace TSS in systems.

The packet classification mainly involves rule matching, where a rule is generally described with d fields $f_1, f_2, ..., f_d$. Each field f_i in a rule corresponds to a range $[l_i, r_i]$. In a typical packet classification problem, the rule has five fields, containing the source IP address, the destination IP address, the source port, the destination port, and the protocol. To perform classification, the d-field packet $(p_1, p_2, ..., p_d)$ from a packet header is first parsed. A packet is considered to match a rule if it matches all fields of the rule. If a packet can match multiple rules, the one with the highest priority will be selected.

To achieve high lookup speed, MultilayerTuple divides rules into fewer tuples with prefix lengths in ranges. For 32-bits IP address, MultilayerTuple splits the prefix lengths of rules into three ranges 32-32, 16-31, and 0-15. In each tuple, the prefix of a rule is reduced to the shortest prefix length in range, and the reduced prefix is used as the key in the hash table. Although the number of tuples is reduced, rules with the same reduced prefix will be stored in a rule chain. To replace the long rule chain, MultilayerTuple creates the next layer structure and further splits the prefix lengths into two ranges, e.g., splits 16-31 into 24-31 and 16-23. With fewer tuple and rule accesses, MultilayerTuple has a higher lookup speed than TSS, PartitionSort, and TupleMerge. Furthermore, MultilayerTuple can also perform higher speed updates.

With generality for different field lengths and scalability for multiple fields, MultilayerTuple can replace TSS in the OpenFlow table and MegaFlow cache of Open vSwitch. Furthermore, with few rules and packets, TSE attack [10]

ISBN 978-3-903176-39-3

generates a large number of tuples to slow down the lookup speed in the MegaFlow cache of OVS. However, by implementing MultilayerTuple in Megaflow cache, the tuple access number for a packet is restricted within 129. Therefore, MultilayerTuple can defend against TSE attack effectively.

Our experimental results demonstrate that MultilayerTuple achieves higher performance than previous methods that support incremental updates. Compared to TSS, TupleMerge, and PartitionSort, MultilayerTuple achieves 21.8x, 2.1x, 2.2x lookup speed and 2.3x, 12.3x, 8.5x update speed. Furthermore, we implement MultilayerTuple in the OpenFlow table and MegaFlow cache of OVS, it not only performs 16.0x and 10.2x lookup speed but also effectively defends against TSE attack.

II. Related Work

Existing representative packet classification algorithms can be divided into three categories. The first category is the hardware-based methods where the lookup is performed through additional hardware, but is expensive and inflexible. The second category is the decision-tree-based methods that only focus on high lookup speed, but are hard to perform incremental updates. The algorithms in the third category support incremental updates, including TSS, TupleMerge, and PartitionSort, but have low lookup speeds and only TSS can apply in SDN systems. Our proposed MultilayerTuple falls into the third category. It not only achieves high lookup speed but also has generality and scalability to apply in systems.

The hardware-based methods: Ternary content addressable memory (TCAM) is a hardware device. It is mainly used to quickly find ACLs, routers, and other entries. TCAM has been used for packet classification [11]–[14] to improve the lookup speed, but its memory size is too small to store a large number of rules. In addition, TCAM can not support incremental updates. Some FPGA methods [15], [16] and GPU methods [17]–[19] have the same problems. Furthermore, hardware methods increase the costs of hardware and energy. Therefore, these algorithms are difficult to apply in the SDN environment where rules need to be flexibly updated.

The decision-tree-based methods: Some decision-tree-based methods such as HiCuts [20], HyperCuts [21], HyperSplit [22], Efficuts [23], SmartSplit [4], BitCuts [24], CutSplit [25], ByteCuts [5], and NeuroCuts [6] use one or more decision trees to build data structure. Each node in the decision tree represents a range of five fields and stores the rules that overlap with the node ranges. The node splits the range into multiple intervals to form children nodes and allocates rules to the children nodes. The rule that overlaps with multiple nodes will be copied multiple times, which could lead to the explosion of memory consumption in decision-tree-based methods. The state-of-art decision-tree-based method, ByteCuts, uses byte extraction to split rules, which achieves the highest lookup speed and small memory cost. However, the splitting scheme of these methods depends on the rule set at the building time. If the rules are constantly updated, the previous splitting scheme could be inappropriate, which can further affect the lookup speed and the memory cost. Therefore, these decision-tree-based methods are not suitable for incremental updates.

TSS: Tuple Space Search (TSS) [3] divides rules into multiple tuples according to the combination of prefix lengths. The lookup speed of TSS is quite slow because the lookup operation requires traversing a large number of tuples to look for the rule with the highest priority. However, TSS has a high update speed, the prefix lengths of rule can be applied to quickly find the corresponding tuple. Furthermore, TSS has generality for different field lengths and scalability for multiple fields, thus TSS is implemented in SDN systems currently.

TupleMerge: TupleMerge [7], [8] improves upon TSS by reducing the prefix lengths of rule, and the prefix lengths of a tuple are decided by the first inserted rule. Each tuple can store the rules with longer prefix lengths and restricts the rule-chain length within 10. If a rule can not be inserted into any current tuple, it will build a new tuple. TupleMerge reduces the number of tuples for higher lookup speed. However, its update operation requires traversing the tuples rather than quickly finding a tuple, thus its update speed is slower than TSS. Furthermore, TupleMerge only reduces the source and destination IP addresses by the authors' experience, which is difficult to apply in multiple fields with different lengths.

PartitionSort: PartitionSort [9] classifies a packet based on binary search tree. It splits the rules into multiple parts, each containing sortable rules and the sorting orders of five fields in these parts can be different. Each part exploits a binary search tree to store sortable rules, and the complexity of lookup and update operation in a tree is $O\left(logN\right)$, where N is the number of rules in this tree. With fewer trees, its lookup speed is faster than TSS. However, its update speed is slower than TSS with the need for traversing binary search trees. Furthermore, PartitionSort requires determining the order of fields at first and then implementing the complex tree structure. Therefore, PartitionSort is hard to apply in SDN systems.

III. Multilayer Tuple Search

The classification algorithms that support updates usually split rules into multiple parts, and each part that uses a hash table is called a tuple. The lookup process requires traversing tuples to look for the matching rule with the highest priority. Tuple Space Search (TSS) creates tuples according to the prefix lengths of rules, thus has a large number of tuples and low lookup speed. Therefore, we first split prefix lengths into three ranges to reduce the number of tuples. Second, we create the next layer for the long rule chain with the same reduced prefix. Third, to demonstrate its generality and scalability, we extend MultilayerTuple to multiple fields with different prefix lengths. At last, we explain the lookup process of MultilayerTuple.

A. Prefix Length Ranges

Existing algorithms divide rules into different tuples according to the prefix lengths. Each tuple in Tuple Space Search (TSS) corresponds to the different prefix lengths. For the classic five fields classification problem (32-bits source/destination

TABLE I
THE RULE SET

Rule	IP address	Prefix length	Priority
R1	11111111	8	8
R2	11110000	8	7
R3	0000111*	7	6
R4	000010*	6	5
R5	000001*	6	4
R6	00000*	5	3
R7	1111*	4	2
R8	1*	1	1

IP addresses, 16-bits source/destination ports, and 8-bits protocol with full or exact matching), TSS has up to $33*33*17*17*2 \approx 629K$ tuples. Because the lookup process requires traversing a large number of tuples, the lookup speed of TSS is quite slow.

Based on TSS, TupleMerge combines tuples with different prefix lengths by omitting bits. However, the scheme of TupleMerge is defective. First, the reduced prefix length of IP address is according to the authors' experience without any proof. When the new distribution of rules appears, the performance of TupleMerge may not be efficient. Second, if TupleMerge forms a long rule chain with the same reduced prefix, it will create a new tuple by the split scheme, which increases the number of tuples. Third, the update process requires traversing tuples to find the first tuple that can be inserted. It not only reduces the update speed but also disorders the rules in tuples. As a result, the lookup speed of TupleMerge is limited with many tuples and disordered rules. Even though the lookup speed of TupleMerge is faster than TSS, there are problems with its scheme to reduce the prefix lengths of rules.

Assuming the length of IP address is 8 bits, Table. I shows the rules that contain prefix IP addresses and priorities. The rule set contains 6 different prefix lengths {8, 7, 6, 5, 4, 1}, thus TSS creates 6 tuples to store these rules. To reduce the number of tuples for higher lookup speed, MultilayerTuple splits prefix lengths into three ranges. As shown in Fig. 1, for 8-bits IP, the prefix lengths are split into three ranges 8-8, 4-7, and 0-3. Therefore, the rules with prefix length 8-8, 4-7, and 0-3 will be stored in Tuple1, Tuple2, and Tuple3 respectively, and the number of tuples in MultilayerTuple is up to three. In each tuple, the prefixes of rules are reduced to the shortest prefix length, and the reduced prefixes are used as the key in the hash table. The rules with the same reduced prefix are stored in a Group. For example, because the IP address of R3 is 0000111* and the shortest prefix length of Tuple2 is 4, the reduced prefix of R3 is 0000*. And the reduced prefixes of R4, R5, and R6 are also 0000*, thus R3, R4, R5, and R6 are stored as a rule chain in Group3.

The lookup process for a packet in Fig. 1 contains three steps. First, the packet traverses three tuples. Second, we use the packet to find the matching group in the hash table of each tuple. Third, we traverse the rule chain to find the matching rule in the group.

Furthermore, the tuples, groups, and rules are sorted by their

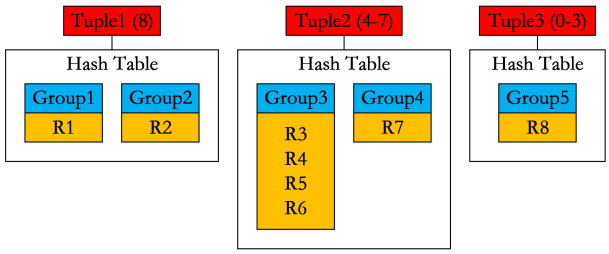


Fig. 1. The first layer structure of MultilayerTuple.

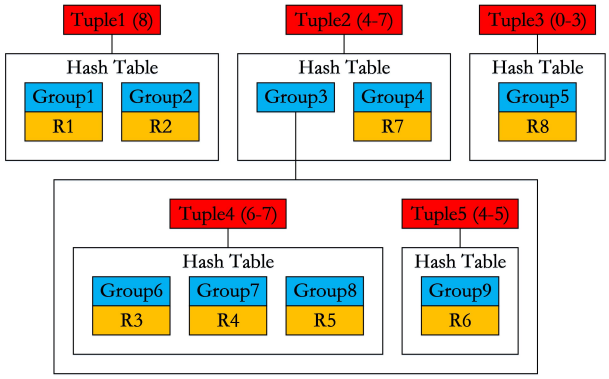


Fig. 2. The multilayer structure of MultilayerTuple.

priorities. The priority of a rule is shown in Table. I, and the priority of a tuple or a group depends on the inside rule with the highest priority. If the priority of the current matching rule is higher than or equal to the current group or rule, we can skip the rest and continue the lookup process in the next tuple. And if it is higher than or equal to the current tuple, we can stop the lookup process and return the current matching rule.

B. The Next Layer

The reduced prefixes lead to fewer tuples on the one hand but form long rule chains on the other hand. For example, in Fig. 1, R3, R4, R5, and R6 have the same reduced prefix and are stored as a rule chain in Group3. If a packet lookups in Group3, it requires traversing the long rule chain to find the matching rule. However, the long rule chain will slow down the lookup speed.

To further improve the lookup speed, we create the next layer to replace the long rule chain. As shown in Fig. 2, Group3 contains a next layer structure. In the first layer, Group3 stores the rules with prefix lengths in range 4-7. In the next layer, we split the prefix lengths into two ranges 6-7 and 4-5. Therefore, Tuple4 and Tuple5 contain the rules with prefix lengths 6-7 and 4-5 respectively. With a two-layers structure, all rules in Table. I can be stored in different groups. If the rules chain is still too long in the second layer, we will continue to create the third layer structure. The tuple in the third layer only contains one prefix length by splitting the prefix lengths 6-7 or 4-5 into two ranges.

Because of the next layer, the lookup process for the third step is changed. If a group contains a rule chain, we traverse the rule chain to find the matching rule. Otherwise, it contains

TABLE II
THE NUMBER OF TUPLE ACCESS

Rule	TSS	MultilayerTuple
R1	1	1
R2	1	1
R3	2	3
R4	3	3
R5	3	3
R6	4	4
R7	5	2
R8	6	3
maximal	6	4
average	3.125	2.5

a next layer structure, we continue to lookup in the next layer recursively. Furthermore, the priority of the current matching rule can also be used to skip tuples, groups, and rules in the next layer.

The numbers of tuple access for packets that matching R1-R8 are shown in Table. II, and the rules can be divided into three types. First, by reducing the number of tuples to three in the first layer, the tuple access number of matching R7 is reduced from 5 to 2, and that of matching R8 is reduced from 6 to 3. It not only reduces the average access number but also reduces the maximal access number. Second, no matter we use TSS or MultilayerTuple, the rules with accurate prefixes like R1 and R2 are stored in the first tuple, thus the tuple access numbers for matching R1 and R2 are both 1. Third, the tuple access numbers of R4, R5, and R6 are not changed, and that of R3 even higher than TSS because of the cost that we use multiple layers. The packets that matching these rules require traversing Tuple1 and Tuple2 in the first layer to find Group3, then find the matching rule in Tuple4 or Tuple5 in the second layer. However, the cost is worth it when TSS has a large number of tuples. As a result, MultilayerTuple has less average and maximal access numbers of tuples than TSS simultaneously.

For IP address with 32 bits and multiple fields, the number of tuples for TSS will explode. Even for TupleMerge, there are still too many tuples. In contrast, MultilayerTuple traverses less number of tuples to find the matching rule in each layer. Even if the packet is required to lookup in a few layers, the average and maximal access numbers of tuples are less than TSS and TupleMerge. Furthermore, in the following layers, the search space is much smaller and smaller recursively, which is beneficial for MultilayerTuple to find the matching rule more quickly.

C. Generality and Scalability

MultilayerTuple is designed for Software Defined Network (SDN) systems like Open vSwitch (OVS). Although many packet classification algorithms with higher lookup speeds were proposed, OVS still implements the slowest algorithm TSS. The most benefits of TSS are its generality for different field lengths and scalability for multiple fields. OpenFlow supports a lot of fields and the lengths of these fields are different. For example, the length of IPv6 is 128 bits and the length of protocol is only 8 bits. Furthermore, a rule may contain a few fields and the rest fields are ignored. TupleMerge only improves lookup speed for source and destination IP addresses by the authors' experience. PartitionSort requires determining the order of fields at first and then implementing the complex tree structure. In contrast, TSS only considers the prefix lengths of existing fields in rules, thus works for multiple fields with different lengths. Even though the previous methods have higher lookup speeds than TSS, they are unlikely to apply in OVS. Maintaining the generality for different field lengths and the scalability for multiple fields, MutilayerTuple achieves higher lookup speed and can apply in OVS to replace TSS.

Generality: MultilayerTuple can handle the fields with different lengths. For a field contains w bits, we split the rules with different prefix lengths into three ranges $[w, w]$, $[w/2, w-1]$, $[0, w/2-1]$ in the first layer. The first kind of rules with prefix lengths w is matching for accurate packets, and these rules usually occupy a large part of the rule set. Therefore, we keep the tuple with prefix length w without changes. The second kind of rules has prefix lengths in range $[w/2, w-1]$. With the first $w/2$ bits, we can quickly reduce the search space within one tuple, then through the rule chain or the next layer to find the matching rule. The third kind of rules has shorter prefix lengths in range $[0, w/2-1]$, which means this field is unimportant for these rules to classify packets, thus we ignore this field in the first layer and consider it in the next layer if necessary. The rules with the same reduced prefix will be stored in a rule chain. If the rule chain is too long, we create the next layer to replace it. For a rule chain that contains rules with prefix length in range $[l, r]$, we split it into two ranges $[l, (r-l)/2]$ and $[(r-l)/2+1, r]$ in the next layer. The range can be split recursively until it just contains one prefix length. When the length of a rule chain is longer than k, we create the next layer. And if it is reduced to $k/2$, we restore this next layer to a rule chain. The suitable choice for k varies for different rule sets, we recommend $k = 20$ for the five fields classification problem with source and destination IP addresses, source and destination ports, and the protocol.

Scalability: MultilayerTuple can handle the rules with multiple fields. If rules contain multiple fields, TSS creates tuples according to the combinations of prefix lengths. Similarly, MultilayerTuple creates tuples according to the combinations of reduced prefix lengths. Therefore, MultilayerTuple has the same scalability as TSS for handling multiple fields. Assuming the rule set contains d fields and the length of each field is w, TSS has up to w^d tuples with different combinations of prefix lengths. Because the lookup process requires traversing tuples to find the matching rule, TSS has the slowest lookup complexity $O(w^d)$. For MultilayerTuple, each field has up to three prefix lengths in the first layer and two in the next layers. Thus the numbers of tuples are at most 3^d in the first layer and 2^d in the next layers. For the fields with w bits, MultilayerTuple has at most $log(w)$ layers. Furthermore, because the reduced prefix has the false positive, a packet may lookup in p next layers. However, our experimental results show that the expectation

of p is quite small. Therefore, the lookup complexity for MultilayerTuple is reduced to $O(3^d+(p+log(w))*2^d)$, which is much smaller than TSS.

D. The Lookup Process

The pseudocode for the lookup process in MultilayerTuple is shown in Algorithm 1. Lines 1-27 traverse tuples in this layer to find the matching rule with the highest priority. Lines 3-5 prevent from searching tuples with lower priority than the current matching rule. Line 6 calculates the reduced prefix of packet by the mask of tuple. Line 7 finds the matching group in the hash table of tuple. If there is no matching group or the priority of matching group is lower than the current matching rule, we continue the lookup process in the next tuple in lines 8-10. If the group contains a rule chain, the packet traverses the rule chain to find the matching rule in lines 11-23. Otherwise, if the group contains the next layer, the packet requires searching in the next layer recursively in lines 24-26. Finally, we return the matching rule with the highest priority in line 28.

Algorithm 1: MultilayerTuple Lookup

Input: the layer stucture $layer$, the current matching rule r, the packet $packet$;
Output: the matching rule r;

```
1  for i = 0; i < layer.tuples_num; i++ do
2      tuple = layer.tuples[i];
3      if r.pri ≥ tuple.pri then
4          break;
5      end
6      reduced_prefix = packet&tuple.mask;
7      group = tuple.hashtable.Find(reduced_prefix);
8      if group == NULL or r.pri ≥ group.pri then
9          continue;
10     end
11     if group.rule_chain! = NULL then
12         rule = group.rule_chain;
13         while True do
14             if Match(rule, packet) == True then
15                 r = rule;
16                 break;
17             end
18             rule = rule.next;
19             if rule == NULL or r.pri ≥ rule.pri
                 then
20                 break;
21             end
22         end
23     end
24     else if group.next_layer! = NULL then
25         r = Lookup(group.next_layer, r, packet);
26     end
27 end
28 return r;
```

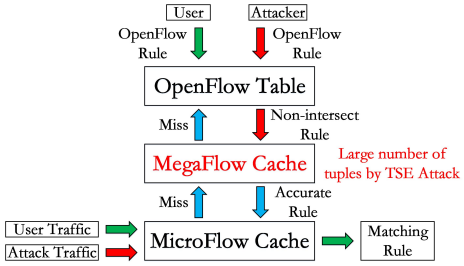


Fig. 3. The lookup stucture of Open vSwitch (OVS).

IV. Implementation in Open vSwitch

MultilayerTuple is designed for Software Defined Network (SDN) systems like Open vSwitch (OVS). To prove its generality, scalability, and high performance, we have implemented MultilayerTuple in the OpenFlow table and MegaFlow cache of OVS. Besides the improvement of performance, experimental results show that MultilayerTuple can defend against TSE attack effectively, which is more important for systems. We will first illustrate the lookup structure of OVS and how TSE attack happens. Then, we implement MultilayerTuple in MegaFlow cache to defend against TSE attack.

A. TSE Attack in OVS

As shown in Fig. 3, OVS has three steps to classify packets. A packet will lookup in the MicroFlow cache, MegaFlow cache, and OpenFlow table in order until it matches a rule. (1) OpenFlow table stores the original rules that users upload. Because OVS only process the rules with prefixes, the rule with ranges will be split into multiple rules with prefixes. (2) Considering that TSS has too many tuples and low lookup speed in OpenFlow table, OVS designs the MegaFlow cache. MegaFlow cache also implements TSS to perform packet classification. The difference between MegaFlow cache and OpenFlow table is that rules are non-intersect in MegaFlow cache. It means a packet can match at most one rule in MegaFlow cache, then stop the lookup process and return this rule immediately. If a packet can not match a rule in MegaFlow cache, it will lookup in OpenFlow table, then the matching rule will be transformed into a non-intersect rule to insert into MegaFlow cache. If a rule can not be matched in TTL seconds, it will be deleted. With fewer non-intersect rules, MegaFlow cache improves the lookup speed. (3) Because a flow contains multiple packets with the same header, OVS designs MicroFlow cache with a hash table to find the matching rule for these packets. If a packet can not match a rule in MicroFlow cache, it will lookup in MegaFlow cache, then insert the accurate rule into MicroFlow cache.

TSE attacks OVS by generating a large number of tuples in MegaFlow cache. Assuming the length of IP address is 6 bits. The attacker first uploads two rules R1 and R2 in Table. III, then transmits the packets with specific IP addresses to generate rules with different prefix lengths in Table. IV. For example, the IP address of p9 is 00000000, and p9 matches R1. But OVS can not insert R1 into MegaFlow

TABLE III
RULES IN OPENFLOW TABLE

Rule	Prefix	Action
R1	*	deny
R2	11111111	allow

TABLE IV
ATTACK PACKETS AND RULES IN MEGAFLOW CACHE

Packet	IP of packet	Rule	IP of rule	Action
p1	11111111	r1	11111111	allow
p2	11111110	r2	11111110	deny
p3	11111100	r3	1111110*	deny
p4	11111000	r4	111110*	deny
p5	11110000	r5	11110*	deny
p6	11100000	r6	1110*	deny
p7	11000000	r7	110*	deny
p8	10000000	r8	10*	deny
p9	00000000	r9	0*	deny

cache because R1 and R2 are intersected. Therefore, OVS generates a non-intersect rule r9 with prefix 0*, which has the shortest prefix length. By transmitting p1-p9, the attacker generates 9 rules with 8 different prefix lengths in MegaFlow cache. In OVS, MegaFlow cache can contain rules with 32-bits source/destination IP addresses and 16-bits source/destination ports. The number of tuples with different combinations of prefix lengths can up to $32{*}32{*}16{*}16 \approx 262K$. With too many tuples, the lookup speed of OVS is greatly reduced. It also affects the performance of other users within the same Open vSwitch. Furthermore, TSE can attack OVS with few rules and packets, which is difficult to detect and defend against. If we remove MegaFlow cache from OVS, the performance will be also affected because the lookup speed of OpenFlow is too slow. Therefore, OVS has no effective methods to defend against TSE attack until now.

B. MultilayerTuple Defends Against TSE Attack

As shown in Fig. 4, we build the data structure of MultilayerTuple to store rules r1-r9. The tuples are sorted by their prefix lengths from long to short, which is different from MultilayerTuple in OpenFlow table. If a packet lookups in MutilayerTuple, the matching rule can only be stored in the first matching group in each layer. Therefore, with each layer accessed at most once, MultilayerTuple only accesses few tuples to defend against TSE attack.

For example, if p4 with IP address 11111000 lookups and r4 exits in Fig. 4, p4 may lookup in three types of tuples. First, the prefix length of matching rule l is shorter than the range of a tuple $[L, R]$, e.g., the prefix length of r4 is 6, which is shorter than the range of Tuple1 $[8, 8]$. For each field, we constructs a trie with all prefixes of rules in OpenFlow table, then generates the shortest prefix that non-intersect with other rules. Therefore, the first l bits of p4 are different from all rules, and p4 matches no groups in Tuple1. Second, the prefix length of matching rule l is in the range of a tuple $[L, R]$. In this situation, the packet finds the tuple that may contain the matching rule. Third, the prefix length of matching rule l is longer than the range of a tuple $[L, R]$. In Fig. 4, p4 can match Group5 in Tuple3. However, the tuples are sorted by their prefix lengths, p4 will find r4 in Tuple2 at first, then skip the lookup process in Tuple3. **Therefore, if the matching rule exists in MultilayerTuple, it is stored in the first matching group in each layer. In other words, no matter whether the matching rule exists, only the first matching group may contain it in each layer.** For the lookup process, we can only check in the first matching group in each layer and skip the rest, thus each layer can be accessed at most once.

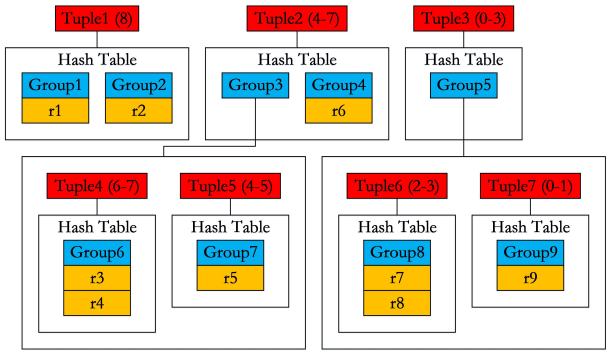


Fig. 4. The data structure of MultilayerTuple in TSE attack.

For multiple fields, e.g., 32-bits source/destination IP addresses and 16-bits source/destination ports, tuples are sorted by the sum of prefix lengths. The first layer contains up to $3^4 = 81$ tuples and each next layer contains up to $2^4 = 16$ tuples. Because MultilayerTuple constructs 4 layers at most, the tuple access number in the lookup process is within $81 + (4 - 1) * 16 = 129$, which is much smaller than 262K tuples in TSS and can effectively defend against TSE attack.

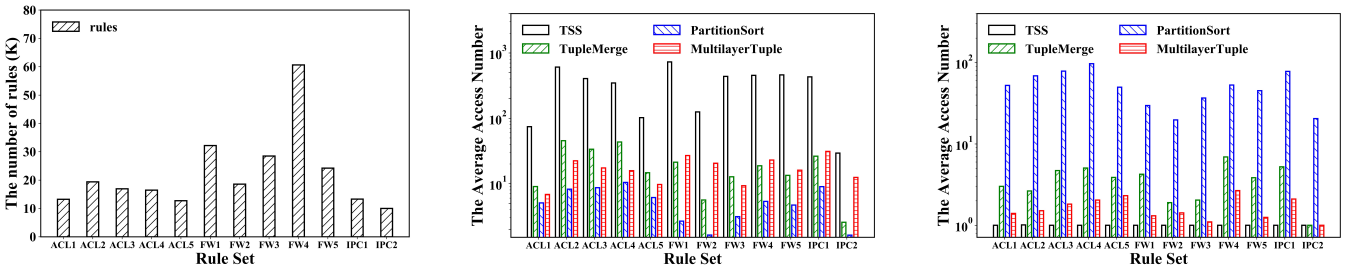
V. EXPERIMENTAL RESULTS

We compare MultilayerTuple with the methods that support incremental updates. These methods contain Tuple Space Search (TSS), TupleMerge, and PartitionSort. TSS is general and scalable for multiple fields with different prefix lengths, and it is implemented in Open vSwitch until now. Furthermore, TSS has the current highest update speed. TupleMerge and PartitionSort have higher lookup speeds, but lower update speeds. TupleMerge only reduces source and destination IP addresses by the authors' experience without any proof. PartitionSort requires determining the order of fields at first and then implementing the complex tree structure. Therefore, both TupleMerge and PartitionSort are hard to replace TSS in OVS.

Experiments are carried on a computer with Intel(R) Core(TM) i7-8700 CPU@3.20GHz, 64KB L1, 256KB L2, 12MB L3 cache respectively, and 8GB of DRAM. The operating system is Ubuntu 16.04.

A. Rule Set and Packet Set

ClassBech is a packet classification benchmark to imitate different rule sets and packet sets in real environments. It contains twelve types of rules: five access control lists (ACL), five firewalls (FW) and two IP chains (IPC). For packet



(a) The number of rules. (b) The average access number of tuples. (c) The average access number of rules.

Fig. 5. The factors of methods.

sets, ClassBench selects a rule randomly at each time, then generates multiple packets to match it. Furthermore, Some parameters can control the distribution of rules and the number of packets to match a rule. We use the sample parameters in ClassBench to generate twelve types of rules and packets. For each type, we generate it with three different sizes 10K, 50K, and 100K.

The source and destination of IP addresses and the protocol are in form of prefix, but the source and destination ports are ranges. Considering that OVS only processes the rules with prefixes, the rule with ranges will be split into multiple rules with prefixes.

B. Methods to Compare

The following methods are implemented.

Tuple Space Search: We implement TSS that forms tuples according to the prefix lengths of five fields. For experiments in OVS, we use the original TSS in OpenFlow table and MegaFlow cache.

TupleMerge: TupleMerge is published on GitHub[1]. TupleMerge only reduces the prefix lengths of source and destination IP addresses by the authors' experience.

PartitionSort: We use the PartitionSort from GitHub[2]. PartitionSort splits rules into multiple parts and implements the binary search tree to store the sortable rules in each part.

MultiplayerTuple: We implement MultiplayerTuple as described in Section III and publish it on Github[3]. MultilayerTuple splits the prefix length of each field into three ranges in the first layer. If the rule chain is longer than 20, MultilayerTuple creates the next layer recursively and further splits the prefix length of each field into two ranges. We also implement MultialyerTuple in the OpenFlow table of OVS. For MegaFlow cache of OVS, MultilayerTuple sorts the tuples according to the sum of prefix lengths as described in Section IV.

C. Evaluation Metrics:

We mainly use three metrics to measure the performance of each algorithm: the lookup speed, the update speed, and the memory cost. For each rule set, the packet set contains 100K packets to perform the lookup operation and get the average lookup throughput in 100 rounds for this packet set. The update operation includes the insertion and deletion, and we use 25% rules in each rule set to perform the insert and delete operations. The memory cost is the memory space required by each algorithm to build its data structure.

[1]https://github.com/drjdaly/tuplemerge
[2]https://github.com/sorrachai/PartitonSort
[3]https://github.com/zcy-ict/MultilayerTuple

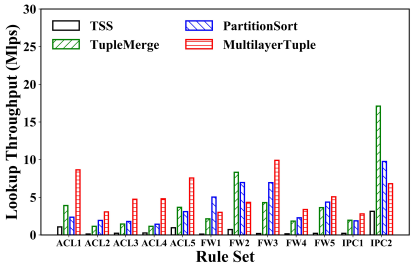
D. The Performance of Methods

We compare the performance of MultilayerTuple with the methods that support incremental updates, including TSS, TupleMerge, and PartitionSort. First, the rule with ranges is split into multiple rules with prefixes, and we observe the expansion ratios of 10K rules for different types. Then we calculate the access numbers of tuples and rules for each method. Finally, we compare the lookup speeds, update speeds, and memory costs among these methods.

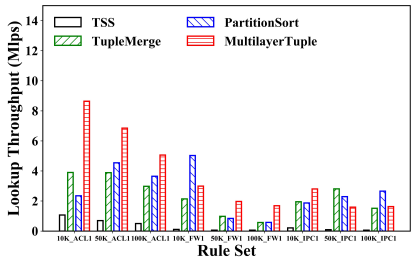
The number of rules: Considering that OVS only processes the rules with prefixes, we first split the rule with ranges into multiple rules with prefixes. As shown in Fig. 5(a), the expansion ratios of twelve 10K rule sets are different. Each rule in IPC2 can be expressed by prefixes, thus the number of rules in IPC2 does not change. However, a lot of rules in FW4 have ranges in source port or destination port. The 10K rule set of FW4 can expand to 61K. On average, the 10K rule set will expand to 22K.

The average access number of tuples: The average access numbers of tuples for different methods and rule sets are shown in Fig. 5(b). The lookup process of TSS requires traversing a large number of tuples, thus the tuple access number for TSS is much larger than other methods. TupleMerge only creates tuples according to the prefix lengths of source and destination IP addresses, and accurate source and destination ports, it ignores to deal with ranges. PartitionSort uses the complex tree structure to store sortable rules with specific fields. Even though TupleMerge and PartitionSort have fewer tuple access numbers, both two methods are unlikely to be implemented in OVS. With generality and scalability for multiple fields with different lengths, the tuple access number for MultilayerTuple is only 9% that of TSS, which greatly improves the lookup speed.

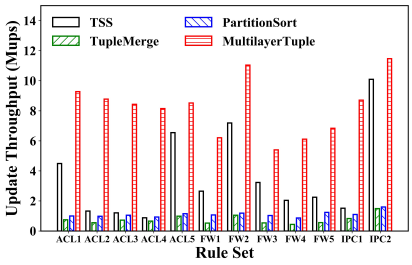
The average access number of rules: As shown in Fig. 5(c), the average access number of rules for MultilayerTuple is 1.6x, 0.5x, and 0.04x that of TSS, TupleMerge, and PartititonSort. TSS creates tuples according to the combinations of all prefix lengths. At the cost of more tuple accesses,



(a) The lookup throughput in 10K rule sets.



(b) The lookup throughput in different sizes of rule sets.



(c) The update throughput.

Fig. 6. The performance of methods.

the rule access number for TSS is smaller than other methods. On the contrary, to reduce the tuple accesses, TupleMerge and PartitionSort have more rule accesses. TupleMerge has too many rules with the same reduced prefix, the long rule chains increase its average rule access number. PartitionSort performs the binary search in each tree structure with more rule accesses. Compared to TupleMerge and PartitionSort, MultilayerTuple has fewer rule accesses. If a rule chain in MultilayerTuple is too long, MultilayerTuple will create the next layer to replace it.

The lookup throughput: We compare the lookup throughput among TSS, TupleMerge, PartitionSort, and MultilayerTuple. The sizes of rule sets are 10K in Fig. 6(a), and sizes vary in 10K, 50K, and 100K in Fig. 6(b). TSS has a large number of tuple accesses, thus the lookup speed of TSS is the slowest. TupleMerge and PartitionSort have fewer tuple accesses at the cost of more rule accesses. Both two methods can perform higher lookup speeds than TSS, especially for FW2 and IPC2. However, the lookup speed of MultilayerTuple is higher than TupleMerge and PartionSort in most rule sets. Compared to TSS, TupleMerge, and PartitionSort, MultilayerTuple achieves 21.8x, 2.1x, and 2.2x lookup speed. Furthermore, the lookup speed of MultilayerTuple is more stable in different rule types and sizes.

The update throughput: The update throughput is shown in Fig. 6(c). For the update process of a rule, TSS can quickly find the corresponding tuple according to the combination of prefix lengths, then updates in the hash table. However, TSS maintains a large number of tuples with their priorities ordered from high to low, which slows down its update speed. TupleMerge and PartitionSort traverse tuples to find the first tuple in which the rule can be inserted, thus their update speeds are much slower than TSS. MultilayerTuple can not only quickly find the corresponding tuple like TSS but also has fewer tuples to maintain their order in each layer. Therefore, the update speed of MultilayerTuple is 2.3x, 12.3x, and 8.5x that of TSS, TupleMerge, and PartitionSort.

The memory cost: The memory cost is shown in Fig. 7. PartitionSort implements the complex tree structure to store sortable rules, thus its memory cost is much higher than other methods. MultilayerTuple can achieve the small linear memory cost as TSS and TupleMerge.

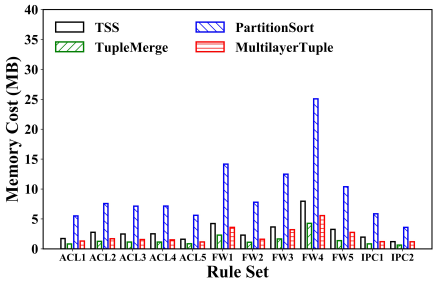


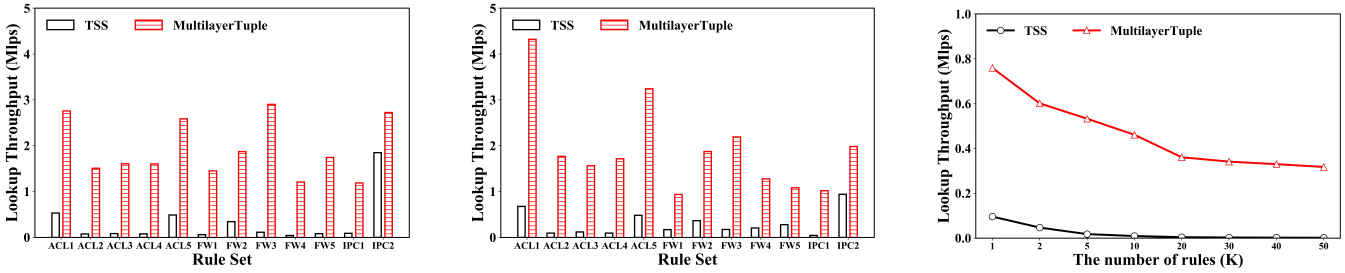
Fig. 7. The memory cost of methods.

E. The Performance in OVS

OpenFlow table: As shown in Fig. 8(a), we have implemented MultilayerTuple in the OpenFlow table of OVS. Because the system code is more complex, the performance of TSS is much slower than Fig. 6(a). Compared to PartitionSort and TupleMerge, only MultilayerTuple has the generality and scalability to replace TSS in OpenFlow table. By implementing MultilayerTuple, the lookup speed in OpenFlow table is 16.0x higher than before.

MegaFlow cache: As shown in Fig. 8(b), we have implemented MultilayerTuple in the MegaFlow cache of OVS as described in Section IV. Because the rules in MegaFlow cache are non-intersect, we generate them according to the packet sets. The TSS in MegaFlow cache still has too many tuples that influence its lookup speed. MultilayerTuple constructs multilayer data structure to reduce the number of tuples, and the packet only requires searching in the first matching group in each layer. Therefore, the lookup speed of MultilayerTuple in MegaFlow cache is 10.2x that of TSS.

MegaFlow cache in TSE attack: As described in Section IV, the MegaFlow cache in OVS can be easily attacked by a few rules and packets, and generates a large number of tuples with different combinations of prefix lengths. Fig. 8(c) shows the lookup speed of MegaFlow cache with different numbers of rules. When TSE attack generates 1-50K rules, the lookup speed of TSS is only 0.096-0.002 Mlps, which influences the performance of OVS seriously. However, by implementing MultilayerTuple, MegaFlow cache still has 0.76-0.32 Mlps lookup speed, which is 8-190x higher than TSS. Therefore, MultilayerTuple can defend against TSE attack effectively.



(a) The lookup throughput of OpenFlow table.

(b) The lookup throughput of MegaFlow cache.

(c) The lookup throughput of MegaFlow cache in TSE attack.

Fig. 8. The performance of OpenFlow table and MegaFlow cache in OVS.

VI. Conclusion

We mainly propose a general, scalable, and high-performance algorithm MultilayerTuple for packet classification problem. MultilayerTuple can handle rules with multiple fields and different prefix lengths. To reduce the number of tuples, MultilayerTuple splits the prefix lengths into three ranges for each field. If there is a long rule chain with the same reduced prefix, MultilayerTuple creates the next layer to replace it recursively and further splits the prefix lengths into two ranges. The experimental results demonstrate that compared to TSS, TupleMerge, and PartitionSort, MultilayerTuple achieves 21.8x, 2.1x, 2.2x classification speed and 2.3x, 12.3x, 8.5x update speed. Furthermore, we have implemented MultilayerTuple in the OpenFlow table and MegaFlow cache of OVS, and MultilayerTuple achieves 16.0x and 10.2x lookup speed respectively. Especially when TSE attack happens, MultilayerTuple can effectively defend against it.

VII. Acknowledgement

This work is supported in part by National Key R&D Program of China (Grant No. 2019YFB1802800), and the NSFC with Grant NO. 61725206. Corresponding author: Gaogang Xie.

References

[1] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about sdn flow tables," in *International Conference on Passive and Active Network Measurement*. Springer, 2015, pp. 347–359.

[2] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.

[3] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 135–146.

[4] P. He, G. Xie, K. Salamatian, and L. Mathy, "Meta-algorithms for software-based packet classification," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 308–319.

[5] J. Daly and E. Torng, "Bytecuts: Fast packet classification by interior bit extraction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 2654–2662.

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

[7] J. Daly and E. Torng, "Tuplemerge: Building online packet classifiers by omitting bits," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–10.

[8] J. Daly, V. Bruschi, L. Linguaglossa, S. Pontarelli, D. Rossi, J. Tollet, E. Torng, and A. Yourtchenko, "Tuplemerge: Fast software packet processing for online packet classification," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1417–1431, 2019.

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

[10] L. Csikor, D. M. Divakaran, M. S. Kang, A. Kőrösi, B. Sonkoly, D. Haja, D. P. Pezaros, S. Schmid, and G. Rétvári, "Tuple space explosion: a denial-of-service attack against a software packet classifier," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT, 2019, pp. 292–304.

[11] D. Pao, Y. K. Li, and P. Zhou, "Efficient packet classification using tcams," *Computer Networks*, vol. 50, no. 18, pp. 3523–3535, 2006.

[12] A. Bremler-Barr and D. Hendler, "Space-efficient tcam-based classification using gray coding," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 18–30, 2010.

[13] A. X. Liu, C. R. Meiners, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 490–500, 2010.

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

[15] Y.-K. Chang and H.-C. Chen, "Fast packet classification using recursive endpoint-cutting and bucket compression on fpga," *The Computer Journal*, vol. 62, no. 2, pp. 198–214, 2018.

[16] Y.-K. Chiu, S.-J. Ruan, C.-A. Shen, and C.-C. Hung, "The design and implementation of a latency-aware packet classification for openflow protocol based on fpga," in *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*. ACM, 2018, pp. 64–69.

[17] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: A gpu-accelerated software router," *Sigcomm Comput.commun.rev*, vol. 41, no. 4, pp. 195–206, 2011.

[18] K. Kang and Y. S. Deng, "Scalable packet classification via gpu metaprogramming," pp. 1–4, 2011.

[19] M. Varvello, R. Laufer, F. Zhang, and T. Lakshman, "Multilayer packet classification with graphics processing units," *IEEE/ACM Transactions on Networking (TON)*, vol. 24, no. 5, pp. 2728–2741, 2016.

[20] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," in *Hot Interconnects VII*, vol. 40, 1999.

[21] V. George and J. WANG, "Packet classification using multidimensional cuts," in *Proceedings of SIGCOMM*, 2003.

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

[23] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "Efficuts: optimizing packet classification for memory and throughput," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 207–218, 2011.

[24] Z. Liu, X. Wang, B. Yang, and J. Li, "Bitcuts: Towards fast packet classification for order-independent rules," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 339–340.

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