Accelerating Packet Classification with Counting Bloom Filters for Virtual OpenFlow Switching

Jinyuan Zhao[1,2], Zhigang Hu[1,*], Bing Xiong[3], Keqin Li[4]

[1] School of Software, Central South University, Changsha 410075, China
[2] School of Computer and Communication, Hunan Institute of Engineering, Xiangtan 411104, China
[3] School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China
[4] Department of Computer Science, State University of New York at New Paltz, New York 12561, USA

Abstract: The growing trend of network virtualization results in a widespread adoption of virtual switches in virtualized environments. However, virtual switching is confronted with great performance challenges regarding packet classification especially in OpenFlow-based software defined networks. This paper first takes an insight into packet classification in virtual OpenFlow switching, and points out that its performance bottleneck is dominated by flow table traversals of multiple failed mask probing for each arrived packet. Then we are motivated to propose an efficient packet classification algorithm based on counting bloom filters. In particular, counting bloom filters are applied to predict the failures of flow table lookups with great possibilities, and bypass flow table traversals for failed mask probing. Finally, our proposed packet classification algorithm is evaluated with real network traffic traces by experiments. The experimental results indicate that our proposed algorithm outperforms the classical one in Open vSwitch in terms of average search length, and contributes to promote virtual OpenFlow switching performance.

Keywords: OpenFlow; virtual switching; packet classification; flow table lookups; counting bloom filters

I. INTRODUCTION

Network virtualization decouples physical infrastructure and network services, and allows multiple heterogeneous network architectures to cohabit on a shared physical substrate. It is widely considered as a long-term solution to the gradual ossification problem faced by the existing Internet [1]. The growing trend of network virtualization results in virtual switches as the primary provider of network services for virtual machines in cloud environments [2]. In particular, virtual switches can provide virtual machines, their tenants, and administrators with logical network abstractions, services and tools identical to dedicated physical networks [3]. However, virtual switching has an inherent performance bottleneck regarding packet classification owing to its softwarization [4]. More seriously, the performance bottleneck becomes more prominent in OpenFlow-based software defined networks [5].

OpenFlow is a widespread standard communication interface protocol between the data and control planes in the software-defined networking paradigm [6]. It enables network controllers to remotely administrate packet forwarding rules in layer 3 switches, by dynamically adding, modifying and removing

Received: Apr. 26, 2018
Revised: Jun. 4, 2018
Editor: Jun Bi

entries of their flow tables. OpenFlow supports wildcards in the matching fields of flow entries, which facilitates flexible definitions of network flows, but gives rise to heavy packet classification overheads especially in virtual switching. Consequently, it is urgent to accelerate packet classification with respect to flow table lookups for virtual OpenFlow switching.

To date, much work has been done on improving packet classification performance for OpenFlow switching, and can be primarily classified into three types from the perspective of flow tables: applying caching techniques, optimizing data structure, and employing bloom filters. Packet classification achieves significant promotion by caching elephant flows [8][9][10] and forwarding decisions [11] in virtue of network traffic locality. Flow table lookups also have been accelerated in OpenFlow switches by optimizing applicable data structures, such as trie trees [12][13][14][15], decision trees [16][18][19][20], and hash tables[21][22][23]. Close to our work, Some researchers strive to explore probabilistic data structure typically bloom filters to improve packet classification performance in OpenFlow's context [24][25][26][27].

The above literatures greatly speed up packet classification through a variety of avenues, but pay little attention to heavy overheads of flow table lookups for failed mask probing in virtual OpenFlow switching. For the above motivations, this paper strives to accelerate the packet classification by avoiding flow table traversals for failed mask probing. In particular, the acceleration is investigated with the following methodology.

We first depict the conceptual framework of packet classification in virtual OpenFlow switching, and offer a profound insight into the packet classification performance bottleneck. Then, the performance are optimized by predicting the failures of flow table lookups for each packet. Finally, we give the pseudo-code implementation of our proposed packet classification algorithm, and evaluate its lookup performance in terms of average search length.

With the above methodology, we aim to achieve the following objectives as the main contributions of this paper: (a) pointing out that mask probing along with flow table lookups is a huge obstacle to packet classification in virtual OpenFlow switching; (b) applying a counting bloom filter to foretell the failures of flow table lookups for the majority of mask probing; (c) verifying the superiority of our proposed packet classification algorithm in terms of average search length by theoretical deduction and experiments.

The rest of this paper is organized as follows. In Section II, we introduce the related work. Section III employs counting bloom filters to promote packet classification performance in virtual OpenFlow switching. In Section IV, we give the pseudo-code implementation and algorithmic complexity of our proposed algorithm. Section V evaluates the lookup performance of the algorithm in terms of average search length with real network traffic traces. In Section VI, we conclude the paper.

II. Related Work

To support wildcarding, OpenFlow defines a mask in each flow entry to indicate all wildcards in the matching fields. On receiving a packet, the switch cannot determine its mask, and has to probe all masks by looking up flow tables with the probed mask and flow identifier. This results in heavy overheads of flow table lookups for an arrived packet. In the past few years, there have been many literatures to improve packet classification performance regarding flow table lookups mainly through three approaches: applying caching techniques, optimizing data structure, and employing bloom filters.

The caching techniques are extensively applied to accelerate flow table lookups especially for virtual OpenFlow switching. Pfaff et al. designed and implemented an open-source virtual switch, Open vSwitch, for networking in virtualized deployment environments. It first cached megaflows with wildcards and no priority in its kernel, enabling most packets to be processed in fast kernel datapaths [7]. Then it designed a microflow cache, mapping a single transport connection to the mask of its corresponding megaflow, to shorten the search length of a packet in the kernel [8]. Congdon et al. built a per-port cache of packet signature and flow key to predict the flow classification of incoming packets by exploiting the temporal locality in network traffic [9]. This can bypass the full lookup process for most packets and reduced packet forwarding latency. Lee et al. proposed a differential flow cache framework using a hash-based placement and localized LRU-based replacement mechanisms in software-defined data center networks in virtue of network traffic locality [10]. Shelly et al. considered how to cache forwarding decisions that depend on packet fields with high entropy, and arrived at algorithms that can efficiently compute near optimal flow cache entries spanning several transport connections [11].

A number of advancements have been achieved by optimizing applicable data structures for better packet classification in OpenFlow switches. Veeramani et al. proposed an efficient way based on existing y-fast trie-partitioning algorithm to reduce the index TCAM size with the search time complexity of O(loglog *n*) [12][13]. Shen et al. presented a trie-based approach to achieve efficient wildcard-aware search for flow recognition with reasonable cost of both memory and update in OpenFlow switches [14][15]. Decision trees, such as HyperCuts[16], EffiCuts[17][18], HyperSplit[19], and SmartSplit[20], can achieve high-speed packet classification but not fast updates especially in OpenFlow's context. The Tuple Space Search (TSS) algorithm with delicate hash table [21][22] achieves fast updates but not high-speed packet classification in virtual switches. By combining decision tree and tuple space search algorithm, Yingchareonthawornchai et al. proposed a hybrid approach PartitionSort with the novel notion of ruleset sortability, which results in far fewer partitions and achieves logarithmic classification and update time for each sortable ruleset partition [23].

Some researchers have attempted to explore probabilistic data structure typically bloom filters to enhance OpenFlow switching. McHale et al. utilized bloom filter to pre-classify network traffic by leveraging the locality of packet flows, which aims to decouple likely legitimate traffic from malicious traffic within SDN data plane [24]. Yuan et al. designed a masked parallel bloom filter with twice verification hash table to achieve full flexibility of multi-protocol query and lower false positive possibility for OpenFlow switches [25]. However, their design will lead to a great many hash collisions with the increasing number of table entries, and require a large-size extra CAM module to cache all collided entries. Lv et al. combined a counting bloom filter with dynamic linked lists to accelerate the exact matching in packet classification [26], but it does not apply to wildcard matching in OpenFlow. Varvello et al. introduced a novel algorithm called Bloom search that takes advantage of Bloom filters to avoid expensive table lookups [27]. The algorithm optimizes tuple search, but does not support the insertion and deletion of the table. This paper focuses on heavy overheads of flow table lookup failures for failed mask probing in virtual OpenFlow switching, and strives to predict the lookup failures by employ counting bloom filters for fast packet classification.

III. Packet Classification for Virtual Openflow Switching

This section sheds light on the performance bottleneck of virtual OpenFlow switching, and accelerates its packet classification by applying counting bloom filters to predict the failures of flow table lookups.

3.1 Virtual OpenFlow switching

Virtual OpenFlow switches usually run on general-purpose hardware, and store all flow rules as flow tables on SRAM or DRAM with address access mode. Since OpenFlow introduces wildcards into the matching fields of a flow rule, conventional lookup methods such as hash tables can no longer apply to flow tables. Generally, wildcards in the matching fields of a flow rule are identified by a mask with identical form of the matching fields. Currently prevalent virtual switches such as Open vSwitch employ a tuple space search classifier [8], which classifies a large number of flow rules into a small amount of tuple sets in terms of mask, and search for flow rules in each tuple set on packet arrivals.

Fig.1 illustrates the abstraction of virtual OpenFlow switching from the perspective of packet classification. As for an incoming packet p_i, an OpenFlow switch extracts its header fields to compute its flow identifier *fid*. Then, the packet probes each of mask array by looking up the flow table with the flow identifier and the probed mask. In particular, we compute the flow identifier *fid* and the probed mask $mask_i$ with AND operation to get a temporal masked key key_i, and take the masked key and the probed mask to match against each entry in the flow table. If the match succeeds, the switch will apply the action set in the matched entry to the packet. Otherwise, the packet proceeds to probe the next mask $mask_{i+1}$ identically as above, until a flow entry is matched or there is no more mask. If all masks are probed in failure, the packet is supposed to belong to a new flow, and should be delivered partly or wholly to the SDN controller for instructions.

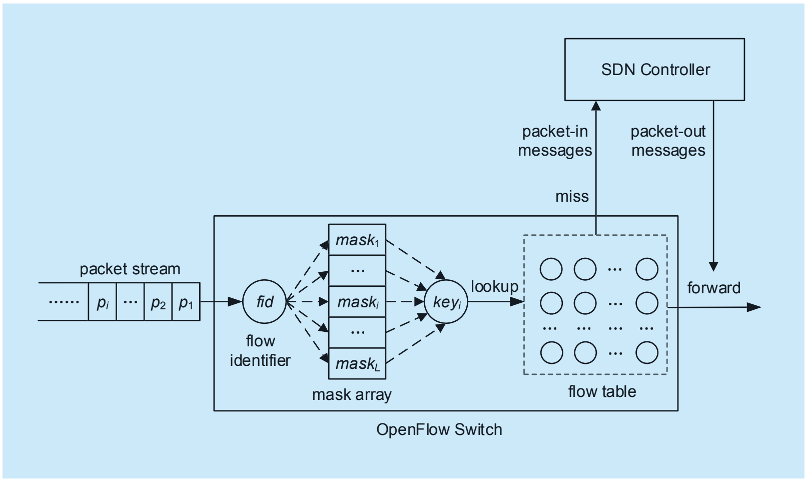

Fig. 1. *Virtual OpenFlow switching abstraction regarding packet classification.*

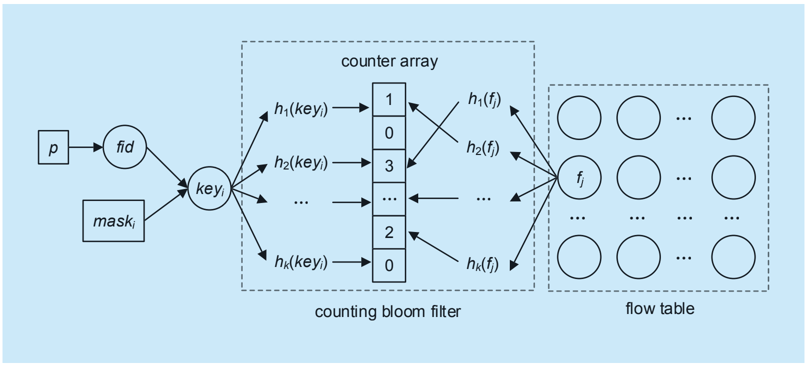

Fig. 2. *Packet classification applying a CBF in virtual OpenFlow switching.*

As seen from figure 1, the performance of packet classification in virtual OpenFlow switching greatly depends on the mask probing along with flow table lookups. Since OpenFlow support wildcarding, a flow is identified by not only its flow identifier but also a mask indicating the positions of wildcards in the flow identifier. As for an arrived packet, the switch cannot determine the corresponding mask, and has to probe all masks one by one until a successful probing is achieved. Since each failed mask probing needs to traverse flow tables, there will be heavy flow table lookup overheads for a packet. More seriously, the lookup overheads will be further expanded with the increasing size of the flow table 1n large-scale networks such as data centers. Thus it is necessary to reduce flow table lookup overheads of packet classification in virtual OpenFlow switching.

3.2 Packet classification applying counting bloom filters

As seen from the above virtual switching, heavy flow table lookup overheads in the packet classification are primarily dominated by failed mask probing with subsequent flow table traversals. If there is a way to predict the failures of mask probing without searching the flow table, the lookup overheads will be significantly decreased. Fortunately, this can be achieved by employing a probabilistic data structure, counting bloom filters (CBF) [28] [29], which can test whether an element is a member of a set and support insertion and deletion in the set. Fig.2 depicts packet classification applying a counting bloom filter in virtual OpenFlow switching.

As shown in figure 2, the CBF is composed of a counter array with m counters initially valuing 0, and k independent hash functions $h_1(.)$, $h_2(.)$, …, $h_k(.)$ each with range $\{0, 1, …, m\text{-}1\}$. When a flow entry with masked key f_j is inserted into or deleted from the flow table, the counters at positions $h_1(f_j)$, $h_2(f_j)$, …, $h_k(f_j)$ of the counter array are increased or decreased by 1 accordingly. As for an incoming packet p, we calculate its masked key key_i by masking its flow identifier *fid* with a mask $mask_i$, and query the CBF whether the masked key key_i is located in the flow table or not by checking the counters at positions $h_1(key_i)$, $h_2(key_i)$, …, $h_k(key_i)$ of the counter array. If any of the counters is zero, it is affirmed that the masked key key_i is not existent in the flow table, and there is no need to search the flow table for key_i. Otherwise, we cannot determine whether the masked key key_i stays in the flow table due to a certain probability of false positive. In such case, we still need to search the flow table for an exact match. Table 1 demonstrates the query, insertion and deletion of the CBF.

IV. Packet Classification Implementation and Analysis

This section describes the pseudo-code implementation of our proposed packet classification and analyzes its algorithmic performance in terms of average search length.

4.1 Pseudo-code implementation

For convenience, OPC-CBF is short for our proposed packet classification applying the counting bloom filter in virtual OpenFlow switching. The packet classification primarily is to classify all incoming packets in terms of flow tables in Algorithm 1. Note that the flow tables will be dynamically adjusted including insertions and deletions, according to OpenFlow specifications. In particular, an entry should be inserted into the flow table if the controller sets up a flow rule to the switch, and be deleted from the flow table if it has expired or the switch receives a message from the controller to remove it. These operations on the flow tables are implemented in Algorithm 2 to 4.

Algorithm 1 illustrates the pseudo-code implementation of the OPC-CBF packet classification. Upon receiving a packet p, we first extract key fields from its protocol headers, such as source MAC address (*src_mac*), destination MAC address (*dst_mac*), source IP address (*src_ip*), destination IP address (*dst_ip*), source port (*src_port*), destination port (*dst_port*), protocol (*ip_proto*), and etc (line 1). Then its flow identifier *f id* is computed with these key fields (line 2), and is used to match against the flow table with each probed mask.

Specifically, we calculate each mask $mask_i$ and flow identifier *fid* with AND operation to get a masked key key_i (line 3-4). After that, the masked key is utilized to query the CBF whether there is a corresponding entry in the flow table (line 5). If the CBF replies in the negative, it implies that there is definitely no corresponding entry in the flow table, and we need to continue to probe the next mask $mask_{i+1}$. Otherwise, we should look up the flow table for a match with the masked key and the mask (line 9). If a match is found, the actions

Alogrithm 1. Pseudo-code for the OPC-CBF packet classification.

```
PacketForward (Packet p)
1.  Parse the packet p to get its key fields, such as src_mac, dst_mac, src_ip, dst_
    ip, src_port, dst_port, ip_proto, and etc.;
2.  Compute the flow identifier fid with these key fields;
3.  for i ← 1, l do
4.      $key_i$ ← fid & $mask_i$;
5.      if (CBF_query($key_i$)), then
6.          h ← Hash($key_i$);
7.          f ← flow_table[h].next;
8.          while f ≠ NULL, do
9.              if f→key = $key_i$ && f→mask = $mask_i$, then
10.             Execute the actions set in the flow entry f to the packet p;
11.                 return 1;
12.             f ← f→next;
13.         end while
14.     end if
15. end for
16. Send a flow setup request containing the packet p in part or in whole to the
    SDN controller;
17. return 0;
```

set in the matched flow entry are applied to the packet p (lines 10). If there is no match for all masks, the switch will send a flow setup request containing the packet p in part or in whole to its SDN controller.

Alogrithm 2. Pseudo-code for the insertion of a flow entry.

```
Insert (Message m)
1.  Distinguish the type of the message m;
2.  if the message type is not flow-mod with the command ADD, then
3.      return 0;
4.  Create a flow entry f, and fill it with the message m, such as flow identifier fid,
    mask mask, actions, and etc.;
5.  key ← f→fid & f→mask;
6.  h ← Hash(key);
7.  f →next ← flow_table[h].next;
8.  f →prior ← & flow_table[h];
9.  f →next→prior ← f;
10.     f →prior→next ← f;
11.     CBF_insert(f);
12.     return 1;
```

Alogrithm 3. Pseudo-code for the deletion of a flow entry for a flow-mod message.

```
Delete (Message m)
1.  if the message m is not a flow-mod with the command DELETE, then
2.      return 0;
3.  Extract flow identifier fid and mask mask from the message m;
4.  h ← Hash(fid & mask);
5.  f ← flow_table[h].next;
6.  while f ≠ NULL, do
7.      if key =f→key && mask = f →mask, then
8.          f→next→prior ← f→prior;
9.          f→prior→next ← f→next;
10.         CBF_delete(f);
11.         free f;
12.         return 1;
13.     end if
14.     f ← f→next;
15. return 0;
```

Algorithm 2 shows the insertion of a flow entry into the flow table in a virtual OpenFlow switch. On receiving a message m from a SDN controller, the switch checks whether it is a flow-mod message with the command ADD. If the check succeeds, we create a new flow entry f in terms of the message m, and insert it into the flow table. Meanwhile, the CBF must be synchronously updated with the insertion in Table 1.

A flow entry will be deleted from the flow table in two cases: (1) the switch receives a flow-mod message with the command DELETE; (2) either of the two timers IDLE_TIME and HARD_TIME in the flow entry is expired. The pseudo-code implementations of the flow entry deletions for these two cases are respectively illustrated in Algorithm 3 and Algorithm 4.

Algorithm 3 summarizes the deletion of a flow entry in a virtual OpenFlow switch on the arrival of a flow-mod message from the controller. On receiving a message, the switch first checks whether it is a flow-mod message with the command DELETE. If the check succeeds, we extract the flow identifier and the mask from the message, and compute the masked key by operating them with AND. Then, we search the flow table with the masked key for an entry, and delete it. Finally, the CBF is accordingly updated.

Algorithm 4 demonstrates the deletion of a flow entry from the flow table if the entry is expired. As for a given flow entry, we first check whether its timeouts IDLE_TIME and HARD_TIME have elapsed. If one of them has run out, we directly delete the entry from the flow table. Meanwhile, the CBF is updated correspondingly.

Table I. *The operations of a counting bloom filter.*

The query of the CBF	The insertion of the CBF	The deletion of the CBF
CBF_query(FlowIdentifier *key*)	CBF_insert(flow **f*)	CBF_delete(flow **f*)
1. **for** $i \leftarrow 1, k$ **do**	1. **for** $i \leftarrow 1, k$ **do**	1. **for** $i \leftarrow 1, k$ **do**
2. $pos \leftarrow H_i(key)$;	2. $pos \leftarrow H_i(f \to key)$;	2. $pos \leftarrow H_i(f \to key)$;
3. **if** $CA[pos] == 0$ **then**	3. $CA[pos] \leftarrow CA[pos]+1$;	3. $CA[pos] \leftarrow CA[pos]-1$;
4. **return** 0;	4. **return** 1;	4. **return** 1;
5. **return** 1;		

4.2 Algorithmic performance analysis

We evaluate the lookup performance of our proposed OPC-CBF algorithm in terms of average search length, based on the investigation into the false positive probability of counting bloom filters.

4.2.1 Probability of false positive

The counting bloom filter in the above algorithm can provide a fast query whether a masked key is kept in a flow table or not. However, there is a certain probability of false positive. In particular, if the query gives a positive result, the masked key may not reside in the flow table actually. The false positive probability mainly depends on the number of the counters m, the number of hash functions k, and the amount of flow entries n in the flow table. With these parameters, the probability that a particular counter is still 0 is can be calculated in (1):

$$(1-\frac{1}{m})^{kn}. \tag{1}$$

Hence we achieve the probability of false positive in this situation as follows [30][31]:

$$p=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-kn/m}\right)^{k} \tag{2}$$

The right hand side of formula (2) is minimized for $k = ln2*m/n$, in which case it becomes $(1/2)^k = (0.6185)^{m/n}$. In fact k must be an integer and in practice we might choose a value less than optimal to reduce computational overhead. For examples, the probability of false positive p is about 2.15% when m/n=8, k=6, and is only 0.314% when m/n=12, k=8. Therefore, we can limit the probability of false positive within an acceptable level by carefully setting the parameters m, n and k.

4.2.2 Average search length

In production networks, an OpenFlow switch is generally pre-installed with flow rules proactively by its SDN controller. Thus we can assume that each packet will find a match in the flow table for simplicity. With the assumptions of the length of the mask array L and the load factor of the flow table α, we can further deduce the average search length of our proposed OPC-CBF algorithm based on the above probability of false positive.

Alogrithm 4. Pseudo-code for the deletion of a flow entry for a timeout.

```
Delete (flow $f$)
1. **if** neither of the timeouts IDLE_TIME and HARD_TIME in the flow $f$ is
   elapsed, **then**
2.     **return** 0;
3. $f \rightarrow next \rightarrow prior \leftarrow f \rightarrow prior$;
4. $f \rightarrow prior \rightarrow next \leftarrow f \rightarrow next$;
5. CBF_delete($f$);
6. **free** $f$;
7. **return** 1;
```

An incoming packet should probe a certain number of masks with subsequent flow table traversals before its right mask. Suppose the packet corresponds to the ith mask, the front i-1(1≤i≤L) masks will be probed in failure. As for any mask $mask_j$ (1≤j≤i-1), the counting bloom filter can directly predict its flow table lookup failure with the probability (1-p). If the prediction succeeds, the search length is considered as 1 for the query of the counting bloom filter. Otherwise, we need to continue to traverse the flow table with the search length α. In summary, it takes the average search length for probing the jth mask as follows.

$$ASL_{ij}=p(1+\alpha)+(1-p)=p\alpha+1. \tag{3}$$

As for the ith mask, the counting bloom filter will give a positive and we need to search the flow table for a match. Therefore, its average search length is the sum of 1 for the query of the counting bloom filter and (1+α/2) for the flow table lookup in (4).

$$ASL_{ii}=1+(1+\frac{\alpha}{2})=\frac{\alpha}{2}+2. \tag{4}$$

To sum up, it takes the average search length for a packet corresponding to the ith mask in (5). With the assumption of packet traffic uniformly corresponding to all masks, we can eventually achieve the average search length of our proposed packet classification algorithm OPC-CBF in (6).

$$ASL_i=\sum_{j=1}^{i-1} ASL_{ij}+ASL_{ii}=(p\alpha+1)(i-1)+\frac{\alpha}{2}+2.$$

(5)

$$ASL_{OPC-CBF} = \frac{1}{L}\sum_{i=1}^{L} ASL_i = \frac{1}{2}(p\alpha+1)(L-1)+\frac{\alpha}{2}+2. \quad (6)$$

As for currently popular virtual OpenFlow switches typically OvS [7][8], its packet classification need to look up flow tables for each probed mask, called OPC-OVS in this paper. Note that the caching technique in OvS is compatible with the prediction of flow table lookups via counting bloom filters. Then we analyze the algorithmic performance of packet classification in OvS without the optimization of caching techniques for a fair comparison. With the above assumptions, we can similarly calculate its average search length in (7).

$$ASL_{OPC-OVS} = \frac{1}{L}\sum_{i=1}^{L}[(i-1)\alpha + (1+\frac{\alpha}{2})] = \frac{1}{2}L\alpha + 1. \quad (7)$$

With the average search lengths in (6) and (7), we can compute the speedup ratio of our proposed OPC-CBF algorithm compared to the OPC-OVS one in (8).

$$Speedup = \frac{ASL_{OPC-OVS}}{ASL_{OPC-CBF}} = \frac{L\alpha+2}{(p\alpha+1)(L-1)+\alpha+4}. \quad (8)$$

As in (8), the probability of false positive p is only about 0.314% for 8 hash functions and the number of counters as 12 times as that of entries in the flow table. The load factor of the flow table α is generally expected to be no more than 10 by setting a biggish integer for flow table length. Thus $p\alpha$ in (8) is negligible compared to 1. Additionally, the length of the mask array L is 16 in the default settings of OvS. As a consequence, the formula of the speedup in (8) can be simplified as below.

$$Speedup \approx \frac{L\alpha+2}{L+\alpha+3} = \frac{16\alpha+2}{\alpha+19}. \quad (9)$$

As seen from (9), the speedup ratio mainly relies on the load factor of the flow table α. It can be easily inferred from (9) that our proposed algorithm will perform flow table lookups at shorter average search lengths than the OPC-OVS one if α>1.13. In fact, α is much greater than 1.13 especially in large-scale networks. For example, if α=2, 4, 8, the speedup ratio is respectively 1.62, 2.87, and 4.81. In conclusion, our proposed OPC-CBF algorithm is expected to outperform the OPC-OVS one.

V. EXPERIMENTS

This section evaluates the lookup performance of our proposed OPC-CBF algorithm in terms of average search length in virtue of real network traffic traces.

5.1 Experimental methodology

For convenient multiple evaluation and comparison, we implement different packet classification algorithms in C/C++ language and integrate them into a test program. Then the program is repeatedly executed to perform packet classification off-line on physical network traffic traces. In particular, it reads packets from each traffic trace in order of their capture time, parses them to get their key header fields and compute their flow identifiers. Subsequently, the program classifies each packet by searching flow tables with its flow identifier and probed mask.

For simplicity, the flow identifier in our experiments is defined as 5 tuples composed of protocol type, source IP, destination IP, source port, and destination port. The masks of these fields are set as follows: 0*xff* for protocol type, 0*x*8000 for both source port and destination port, and default subnet mask for both source IP and destination IP (0*xffffffff* for classes D and E addresses). Besides, we set 10*s* as the timeout of a flow entry.

5.2 Traffic traces

In our experiments, we select 3 traffic traces (CERNET20140509, CERNET20130122 and CERNET20120922) [32], collected from a 10Gps main channel in the CERNET (Jiangsu). These traces respectively last for 58*s*, 106*s*, 85*s*, and each of them contains 15,420,235 packets. With the above configurations, we can give the varying number of simultaneous flows in these traces in figure 3. As seen from figure 3, the numbers of simultaneous flows chiefly fall between 25*k* and 30*k* at stable states.

5.3 Probability of false positive

The probability of false positive is a key performance metrics of counting bloom filters. In our experiments, the counting bloom filter is configured in terms of m=nk/ln 2, where m, n and k respectively represent the numbers of counters, flow entries and hash functions. As seen from figure 3, the number of flow entries n is approximately 28k on average. Thus the number of counters m is set as 224k since we select 6 hash functions. With the above configurations, we perform our proposed packet classification algorithm on the 3 traffic traces, and count the number of false positive of the counting bloom filter and that of failed flow table lookups. Eventually, we dividing them to get the probability of false positive for each 10^5 packets in figure 4.

As shown in figure 4, the probabilities of false positive mainly fall between 2% and 5% for all traces, which indicates that the counting bloom filter can provide an accurate prediction of flow table lookups. In particular, the probabilities for the traces CERNET20140509, CERNET20130122 and CERNET20120922 respectively average 3.18%, 3.57%, 3.27%, which are slightly larger than the theoretical probability 2.15% concluded in Section 4.2. This is attributed to the fact that flow table lookups are predicted only by the masked key. In contrast, a packet matches a flow entry by not only the masked key but also the mask.

5.4 Average search length

Average search length is a classical metrics to measure the lookup performance of packet classification, and significantly depends on its flow table length (FTL). By setting different value of FTL, we read packets for each traffic trace, perform both packet classification algorithms OPC-CBF and OPC-OVS, and get their average search lengths for each 10^5 packets in figure 5.

As seen from figure 5, our proposed OPC-CBF algorithm performs flow table lookups at much smaller average search length than the OPC-OVS one for all traffic traces. Particularly, the average search length of the OPC-OVS algorithm goes up rapidly with the increasing flow table length, which keeps consistent with our theoretical derivation in (7). In contrast, our proposed OPC-CBF algorithm always

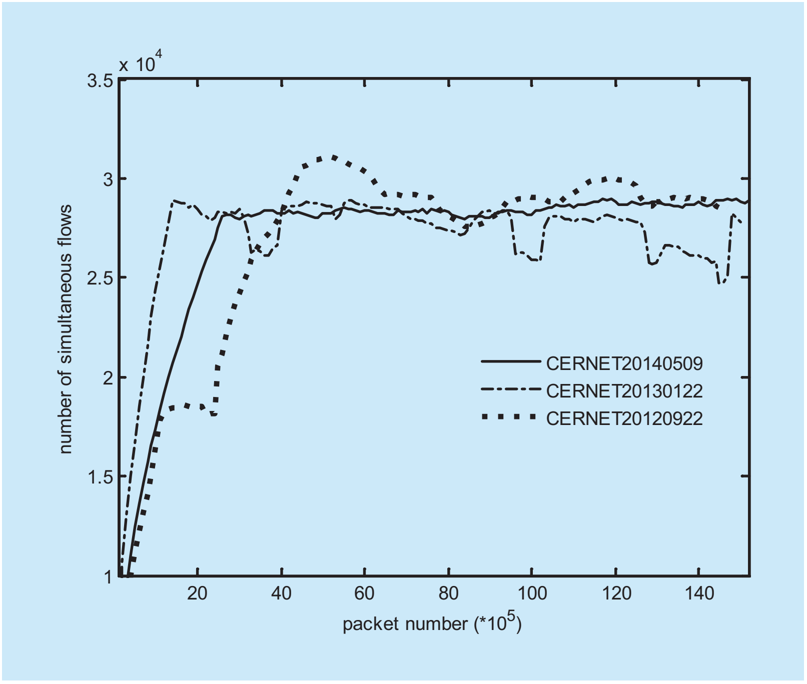

Fig. 3. *The number of simultaneous flows in traffic traces.*

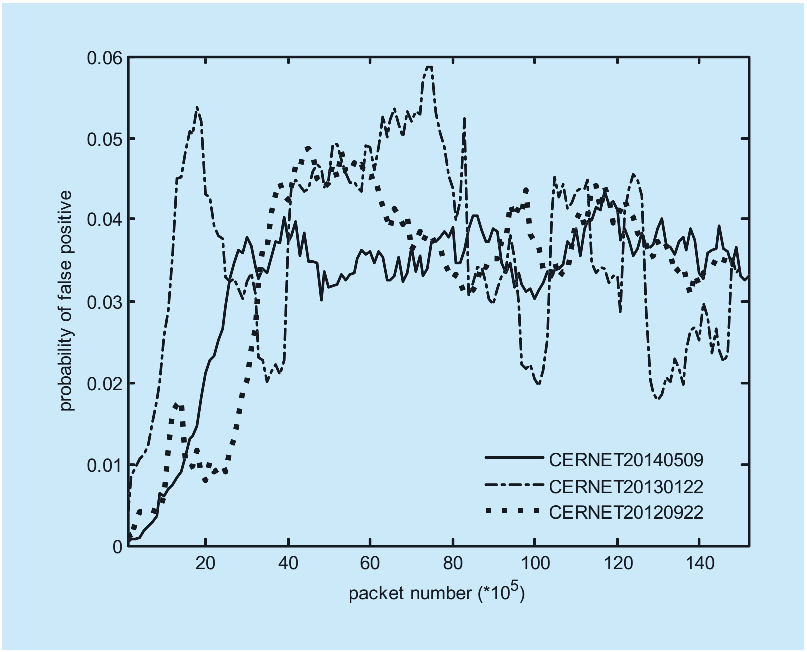

Fig. 4. *False positive probabilities of the counting bloom filter.*

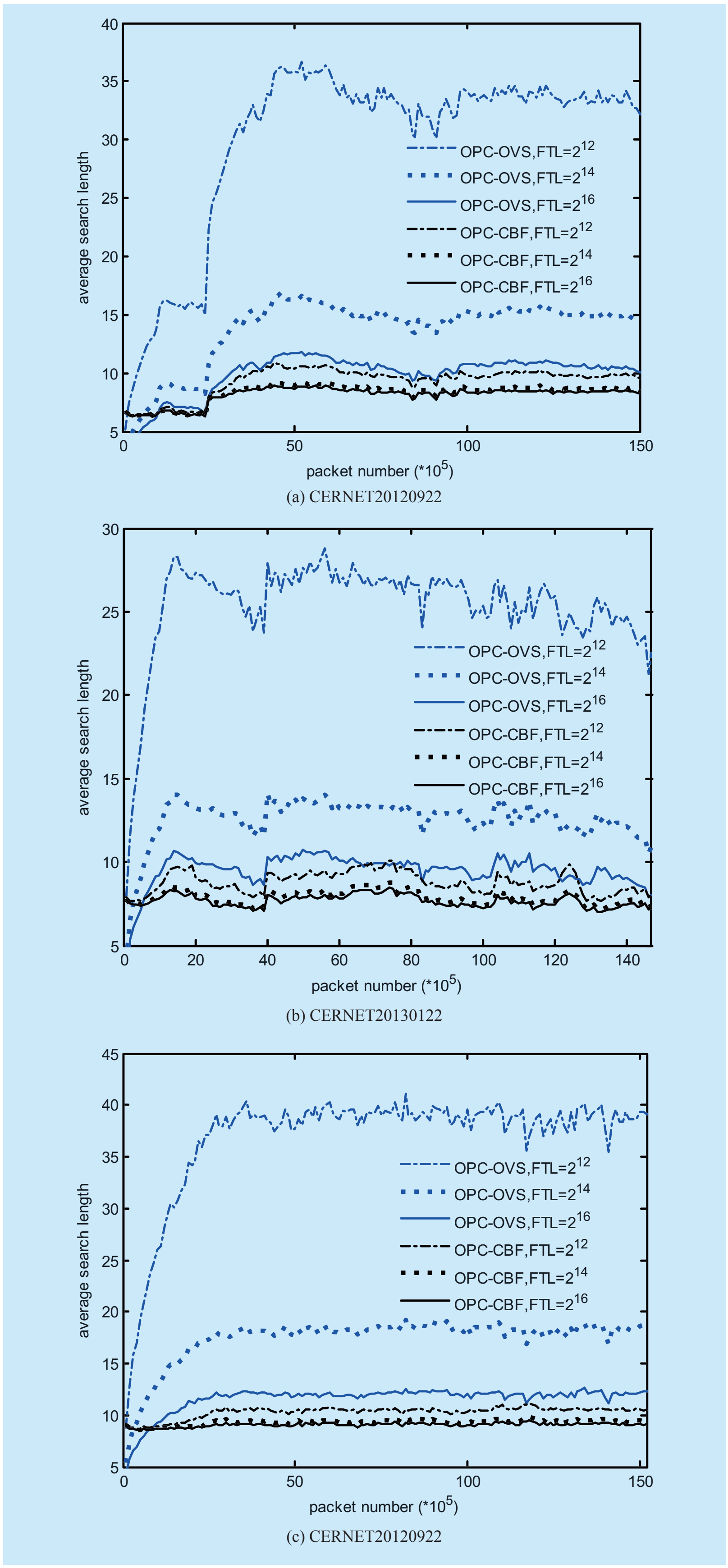

Fig. 5. *Average search length for traffic traces.*

takes steady average search length below 10 no matter how long the flow table length is. This phenomenon is attributed to the extremely low probability of false positive of the counting bloom filter shown in figure 4, which implies that almost all failed mask probing can be predicted without needing to traverse the flow table.

VI. Conclusion and Future Work

Packet classification is a primary fundamental component in virtual switching, and has a critical impact on packet forwarding performance especially in OpenFlow-based software defined networks. For this motivation, this paper presents an efficient packet classification algorithm OPC-CBF, which applies counting bloom filters to predict the failures of flow table lookups without traversing flow tables for most mask probing.

The experimental evaluations indicate that our proposed algorithm performs much better than the packet classification algorithm in currently prevalent virtual switches in terms of average search length. In particular, our proposed algorithm executes flow table lookups always at steady length below 10 for all traffic traces, whereas the average search length of the classical algorithm in Open vSwitch goes up rapidly with the increasing flow table length.

This paper has provided an effective way by employing counting bloom filters to accelerate packet classification in virtual OpenFlow switching. In our future work, more traffic traces from different network scenarios will be utilized to evaluate and validate our proposed algorithm. Meanwhile, we also plan to deploy it in popular open-source software switches including Open vSwitch. Furthermore, applications of our proposed algorithm to other flow-based network systems are also within our future work plan.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (61272148, 61572525, 61502056, and 61602525), Hunan Provincial Natural Science Foundation of China (2015JJ3010), and Scientific Research Fund of Hunan Provincial Education Department (15B009, 14C0285).

References

[1] Chowdhury N M M K, Boutaba R, "Network virtualization: State of the art and research challenges"[J], I*EEE Communications Magazine,* 2009, 47(7): pp20-26.

[2] Firestone D, "VFP: A virtual switch platform for host SDN in the public cloud"[C], 1*4th USENIX Symposium on Networked Systems Design and Implementation* (NSDI), Boston, USA, 2017: pp315-328.

[3] Yi B, Wang X, Li K, et al, "A comprehensive survey of network function virtualization"[J], *Computer Networks*, 2018, 133: pp212-262.

[4] Emmerich P, Raumer D, Wohlfart F, et al, "Performance characteristics of virtual switching"[C], *3rd IEEE International Conference on Cloud Net-working* (CloudNet), Luxembourg, Luxembourg, 2014, 25(2): pp120-125.

[5] Bianco A, Birke R, Giraudo L, Palacin M, "Openflow switching: Data plane performance"[C], *IEEE International Conference on Communications* (IC-C), Cape Town, South Africa, 2010: pp1-5.

[6] Mckeown N, Anderson T, Balakrishnan H, et al, "OpenFlow: enabling in-novation in campus networks"[J], *ACM SIGCOMM Computer Communication Review,* 2008, 38(2): pp69-74.

[7] Pfaff B, Pettit J, Koponen T, et al, "Extending networking into the virtual-ization layer"[C], *8th ACM SIGCOMM Workshop on Hot Topics in Networks* (HotNets), New York, NY, 2009: pp1-16.

[8] Pfaff B, Pettit J, Koponen T, et al, "The design and implementation of Open vSwitch"[C], *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Oakland, CA, 2015: pp2-16.

[9] Congdon P T, Mohapatra P, Farrens M, et al, "Simultaneously reducing latency and power consumption in OpenFlow switches"[J], *IEEE/ACM Transactions on Networking*, 2014, 22(3): pp1007-1020.

[10] Lee B S, Kanagavelu R, Aung K M M, "An efficient flow cache algorith-m with improved fairness in software-defined data center networks"[C], *2nd IEEE International Conference on Cloud Networking* (CloudNet), San Francisco, CA, 2013: pp18-24.

[11] Shelly N, Jackson E J, Koponen T, et al, "Flow caching for high entropy packet fields"[J], *ACM SIGCOMM Computer Communication Review*, 2014, 44(4): pp151-156.

[12] Veeramani S, Mahammad S N, "Efficient IP lookup using hybrid trie-based partitioning of TCAM-based open flow switches"[J], *Photonic Network Communications*, 2014, 28(2): pp135-145.

[13] Veeramani S, Kumar M, Mahammad S K N, "Hybrid trie based partitioning of TCAM based openflow switches"[C], *IEEE International Conference on Advanced Networks and Telecommuncations Systems* (ANTS), 2013: pp1-5.

[14] Shen T, Zhang D, Li Y, et al, "A trie-based approach to fast and scalable flow recognition for OpenFlow"[J], *Lecture Notes in Electrical Engineering,* 2015, 330: pp223-230.

[15] Shen T, Zhang D, "Rule selector: A novel scalable model for high-performance flow recognition"[C], *Proceedings of IEEE Trust-com/BigDataSE/ISPA*. Tianjin, China, 2016: pp1121-1128.

[16] Singh S, Baboescu F, Varghese G, et al, "Packet classification using multi-dimensional cutting"[C], *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOM-M). New York, 2003: pp213-224.

[17] Vamanan B, Voskuilen G, Vijaykumar T N, "EffiCuts:optimizing packet classification for memory and throughput"[J], *ACM SIGCOMM Computer Communication Review,* 2010, 40(4): pp207-218.

[18] Stimpfling T, Savaria Y, Cherkaoui O, "Optimal packet classification appli-cable tothe OpenFlow context"[C], *1st edition workshop on High performance and programmable networking* (HPPN), New York, 2013: pp9-14.

[19] Qi Y, Xu L, Yang B, et al, "Packet classification algorithms: From theory to practice"[C], *28th IEEE International Conference on Computer Communications* (INFOCOM), Rio de Janeiro, Brazil, 2009: pp648-656.

[20] He P, Xie G, Salamatian K, et al, "Meta-algorithms for software-based packet classification"[C], *22nd IEEE International Conference on Network Protocols* (ICNP), Research Triangle Park, North Carolina, USA, 2014: pp308-319.

[21] Levy G, Pontarelli S, Reviriego P, "Flexible packet matching with single double cuckoo hash"[J], *IEEE Communications Magazine*, 2017, 55(6): pp212-217.

[22] Cho C H, Lee J B, Ryoo J D, "A collision-mitigation hashing scheme utiliz-ing empty slots of cuckoo hash table"[C], *19th IEEE International Conference on Advanced Communication Technology (ICACT), Bongpyeong,* South Korea, 2017: pp514-517.

[23] Yingchareonthawornchai S, Daly J, Liu A X, et al, "A sorted partitioning approach to high-speed and fast-update OpenFlow classification"[C], *Pro-ceedings of the International Conference on Network Protocols (ICNP)*, Singapore, 2016: pp1-10.

[24] Mchale L, Case J, Gratz P V, et al, "Stochastic pre-classification for SDN data plane matching"[C], *International Conference on Network Protocols*, 2014: pp596-602.

[25] Yuan D, Yang X, Shi X, et al, "Multi-protocol query structure for SDN switch based on parallel bloom filter"[C], *International Conference on In-formation and Communication Technology Convergence*, 2014: pp206-211.

[26] Lv Z, Li T, "Study and implementation of packet classification algorithms for OpenFlow"[J], *Computer Engineering and Science,* 2014, 36(5): pp860-865.

[27] Varvello M, Laufer R, Zhang F, et al, "Multilayer packet classification with graphics processing units"[J], IEEE/ACM Transactions on Networking (ToN), 2016, 24(5): pp2728-2741.

[28] Xie K, Zhao J J, Zhang D F, et al, "A fast multi-dimensional packet classification algorithm using counting bloom filter"[J], *Acta Electronica Sinica*, 2010, 38(5): pp1046-1052.

[29] Xie K, Wang X, Li W, et al, "Bloom-filter-based profile matching for proximity-based mobile social networking"[C], *13th Annual IEEE International Conference on Sensing, Communication, and Networking* (SEC-ON), London, UK, 2016: pp1-9.

[30] Fan L, Cao P, Almeida J, et al, "Summary cache: A scalable wide-area web cache sharing protocol"[J], *IEEE/ACM Transaction on Networking*, 2000, 8(3): pp281-293.

[31] Cohen S, Matias Y, "Spectral bloom filters"[C], *ACM SIGMOD International Conference on Management of Data*, 2003: pp241-252.

[32] "Network traffic traces", available at http://iptas.edu.cn/src/system.php.

Biographies

Jinyuan Zhao, is a PhD candidate of software engineering in the School of Software of Central South University, China, and also a lecturer in the School of Computer and Communication, Hunan Institute of Engineering, China. She received a M.S. degree from Central China Normal University, China, in 2007. Her current research is in software-defined networking and cloud computing.

Zhigang Hu, is a full professor in the School of Software of Central South University, China. He received a M.S. and a Ph.D. degrees in Computer Science from Central South University, China, in 1994 and 2001 respectively. His current research is in cloud computing, parallel and distributed systems. *The corresponding author. Email: zghu@csu.edu.cn

Bing Xiong, is an associate professor in the School of Computer and Communication Engineering, Changsha University of Science and Technology, China. He received a Ph.D. degree in 2009 by a master-doctorate program from School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China. His current research is in software-defined networking, future internet architecture and network security.

Keqin Li, is an IEEE Fellow, a SUNY distinguished professor of computer science in State University of New York at New Paltz, and also an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China. He received a B.S. degree in computer science from Tsinghua University, China, in 1985, and a Ph.D. degree in computer science from University of Houston, USA, in 1990. His current research is in parallel and distributed computing.