

An Efficiency Pipeline Processing Approach for OpenFlow Switch

Zhenwei Wu^{*†‡}, Yong Jiang^{*†‡}, Shu Yang^{*†‡}

^{*}Graduate School at Shenzhen, Tsinghua University, Shenzhen, China

[†]Tsinghua National Laboratory for Information Science and Technology(TNList), Tsinghua University, Beijing, China

[‡]Department of Computer Science and Technology, Tsinghua University, Beijing, China

wzw14@mails.tsinghua.edu.cn, {jiangy, yang.shu}@sz.tsinghua.edu.cn

Abstract—One primary component of OpenFlow switches is a pipeline of flow tables. Flows are directed through the pipeline by looking up a matched rule in each table. Sequentially, Packet matching starts at table 0 and continues to additional tables of the pipeline if necessary. The lookup stops when one flow entry is matched or the end of pipeline is reached. Though effective, this manner of processing is not efficient when popular rules are installed in back tables. If frequently matched rules appear earlier in the pipeline, the procedure of lookup and comparison can be improved. Unfortunately, a simple sorting algorithm is not feasible. In this paper, we formalize the problem of reducing lookup times, which is proven to be NP-hard. A heuristic approach, MILE(migrating flow rules), is proposed to minimize the average number of lookups. Experimental results show that MILE is able to reduce table lookups by 50%.

Keywords—SDN, OpenFlow Switch, Pipeline Processing

I. INTRODUCTION

Software-Defined Networking splits control plane from data plane, which enables quick deployment of innovative applications and fine-grained control over flow-level traffic. One or more logically centralized controllers [1, 2] interact with forwarding devices in an engaged fashion. OpenFlow [3] exposes interfaces for both control plane and data plane to deliver messages, commands and instructions.

Since OpenFlow 1.1[4], a pipeline of flow tables is brought in. Multiple flow tables offer flexibility but also arouse complexity[5]. One challenge is how to get the most profit by selecting and placing flow rules across a pipeline of flow tables. OpenFlow rules support flexible matching and various actions, thus more complex than conventional forwarding rules. These rules span more fields and consequentially occupy more space. This also asks for solutions to select and place the most profitable rules across the forwarding devices.

A logical OpenFlow switch contains a number of flow tables. On packets arriving, matching starts from the first table of the pipeline and goes ahead until a successful matching. The staged process would severely hinder packet processing if a matched rule resides in back tables of the pipeline. It would get even worse when such a rule is frequently matched. Several recent work has studied the problem of OpenFlow rule selection and placement [6–9], as is detailed in Section II. Their work focuses on high-level traffic management goals (e.g., access control, accounting) with minimum resources

required. Additionally, some studies propose caching popular rules [10, 11] to provide a vision of larger rule tables, aimed to relieving switches of space pressure.

In this paper we advocate an approach, MILE, to efficient rule lookup in an OpenFlow switch pipeline. Rules are re-ordered by migrating. Popular rules are placed ahead and can be matched earlier. We do not migrate an individual rule, as rules can have overlapping patterns and complex dependency relationships between each other. Instead, interdependent rules are grouped and migrated as a whole. To handle rule dependencies, we denote a list of interdependent rules as a directed acyclic graph(DAG), and design algorithms for migrating rules.

The remainder of this paper is organized as follows. Section II discusses related work and Section III provides with a formalization of the lookup reduction problem and shows that the problem is NP-hard. Our heuristic approach is detailed in section IV, including the Directed Acyclic Graph(DAG) for representing rule dependencies and algorithms for merging and migrating flow rules. Section V investigates performance of the proposed method. Finally, section VI and VII discuss the future work and summarize this paper.

II. RELATED WORK

In terms of rule placement, Palette [6] and OneBigSwitch [7] aggregate rule sets and then decide where to install them in the network. More precisely, a large rule table is decomposed into a certain number of small sub-tables and smaller sub-tables are inserted onto switches so that each sub-table is traversed at least once by incoming packets. In this manner, it ensures that one traversal of all the sub-tables is equivalent to a single lookup in the original large table. However, as is shown in [6], Palette is complicated and often produces suboptimal solutions. DIFANE [9], CacheFlow[10], and FIB caching[11] advocate classifying and caching rules. In DIFANE, rules are classified and the potentially important ones are cached at some specific devices, called authority switches. Incoming packets are firstly matched with rules in local ingress switches. If no match is found, the ingress switch will redirect packets towards those authority switches. This manner of rule placement eases communication load between the controller and the switches upon miss and also reduces the number of rules to store on ingress switches. Based on rule caching,

CacheFlow[10] proposes a hardware-software hybrid switch design aimed to provide large rule tables. As with DIFANE, CacheFlow also selects a set of important rules but caches them on the TCAM. Table misses are dealt with by software switches. Rules are not cached individually, but in group and long dependency chains are cut to length in order that smaller groups of rules are cached. [8] advocates concentrating on where to forward traffic rather than how to forward traffic. Rules are placed where the maximum amount of traffic could be delivered through the network. A rule allocation matrix is constructed in aid to maximize the total volume of the traffic that satisfies the endpoint policy. On the side of multiple flow tables, [12–14] explore the flexibility of multiple flow tables and hopefully to improve efficiency of packet processing.

These solutions either manage a single rule table or target placing rules to satisfy network-wide policies (e.g., traffic engineering, access control). Our work turns to lookup procedure of OpenFlow switches and focuses on a multi-table pipeline. By exploiting pipeline processing, we can decrease lookup times and speed up packet processing. Extra benefits can be achieved if we merge actions on a packet during pipeline processing.

III. PROBLEM FORMALIZATION

In this section, a lookup reduction problem is formalized. Major notations used are listed in Table I.

The objective function is to minimize the average lookup times through pipeline processing.

$$\sum_E w(E) * \text{lookup}(S', E) \quad (1)$$

under constraints:

$$\text{size}(S') \leq k \quad (2)$$

$$\text{forward}(S, E) \equiv A \equiv \text{forward}(S', E) \quad (3)$$

Constraint 2 ensures that the number of selected rules will not exceed flow table capacity limit. Constraint 3 ensures that the new rule set will apply the same actions as the original rule set does. The weight function $w(E)$ computes and assigns one weight to a rule based on the traffic volume it matches and the frequency it is actually matched. Note that the traffic volume a rule can match is not equal to the volume of packets it will match in real conditions and that the matching frequency adjusts the weight to reduce the deviations.

The problem of reducing lookup times can be reduced from the problem of single machine job scheduling with precedence constraints. It has been proved that single machine job scheduling problem is NP-hard, thus the lookup reduction problem is also NP-hard.

IV. HEURISTIC APPROACH

Our heuristic approach to the problem of reducing lookup times is detailed in this section. The approach is based on forwarding graph and following a forward path it will merge flow rules and create new ones.

Table I: Notations

Symbol	Description
S	A set of forwarding rules
S'	A subset of forwarding rules, $S' \subseteq S$
A	A set of processing actions
E	An equivalence class of packets, i.e. flows
$\text{forward}(S, E)$	A forwarding function that instructs how to forward a class of packets E under S . It returns a set of processing actions that will be applied to packets in E
$\text{lookup}(S, E)$	A lookup function that looks up a matched rule in S for E . It returns the number of comparisons before a matching is found
$w(E)$	A weight function that computes weight
$\text{size}(S)$	A size function that returns the number of rules in S
K	A single flow table capacity

A. Construct Forwarding Graph

While flowing through a pipeline of flow tables, flows are processed stage by stage. Each stage can be viewed as an intermediate processing point and packets are processed and forwarded by a sequence of processing points, which forms a forwarding path. Noticing that there exists an order between these points, we define a predecessor-successor relationship. For two flow rules A and B , a precedence relationship exists 1) if the flow id of the flow table where A resides is smaller than that of the flow table where B resides. The flow tables of an OpenFlow switch are uniquely identified by their numbering, starting at 0, which is denoted as flow id. 2) The instruction field of Rule A has a Goto-Table instruction and directs flows to the flow table where B resides. 3) The output of A is a subset of the input of B . The input and output are an analogy to the domain and range in mathematics. The input of a rule is a set of incoming packets to be matched. The output of a rule is a set of resulting packets.

Each flow is processed by an ordered list of rules and the ordered list is represented as a DAG. Let $G = (R, E)$ be a DAG for a rule list R , where vertices are rules and edges are the precedence relationship constraint. A DAG lists the processing stages packets undergo while flowing through the pipeline. Obviously, packets in the same flows go through the same path. A collection of forwarding paths form a forwarding graph for all the flows.

B. Merge and Migrate

Based on forwarding graph, we traverse along the forwarding path, record the instructions and actions, then merge. Each DAG is assigned a weight which is proportional to the flow space it would match. Our merging algorithm greedily selects the DAG with the largest weight (Line 5 Algorithm 1). The selected DAG is traversed. Instructions and actions at each node along the forwarding path are recorded. Finally, with information collected above, we create new rules by merging instructions and actions. Note that the OpenFlow protocol has some appointed order constraints while executing and applying instructions and actions, our merging algorithm must not violate that order. For example, the Goto-Table instruction must be executed after all the other instructions and the output

action in the action set is also executed last. In our work, for simplicity, we only support those actions and instructions that are required by OpenFlow protocol.

After merging, new rules are created. We proactively select a number of important rules to be installed at flow table 0. Then 2Q LRU replacement algorithm is applied reactively to update rules in flow table 0. Frequently hit rules at a back table are migrated ahead.

Algorithm 1 Greedily Merge Equivalence Classes

Input:

Equivalence class set E

Output:

Rule set S

```

1: MergeEquivalence( $E$ )
2:  $S \leftarrow \phi$ 
3: sort  $E$  in descending weight
4: while  $E \neq \phi$  do
5:   Select  $e$  with the largest weight from  $E$ 
6:    $r = \text{MergeRules}(e)$ 
7:    $E = E - e$ 
8:    $S = S \cup \{r\}$ ;
9: end while
10: return  $S$ 

```

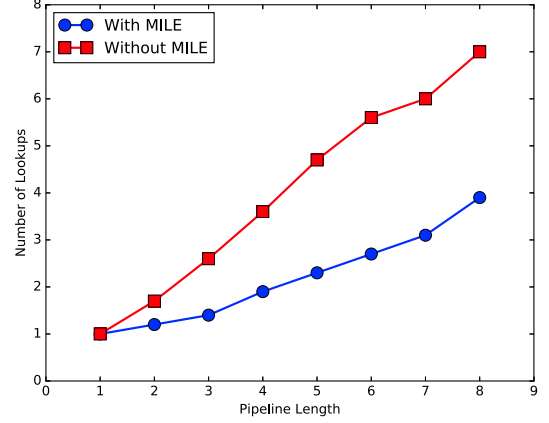
C. Update Forwarding Graph

At runtime, the controller can delete an OpenFlow rule. The rule can also expire after a timeout period. To keep consistent with these changes, our forwarding graph must get updated accordingly. An intuitive way is to reconstruct the graph from the scratch. However, in case of extensive updates, it would incur great overhead and latency for packet forwarding. It is observed that a rule can have several successor rules in next table and that the successor with the highest priority is selected while constructing forwarding graph. We can leverage this to improve efficiency. Therefore, two pointer fields are added to each node, one pointing to a list of candidate successor rules and the other pointing to a list of sibling rules. Rules in each list are almost alike except that they have different priorities. The list of rules is organized in an order of descending priority. When a rule is deleted, the first rule of its sibling list would substitute. If the successor is deleted, the highest-priority candidate rule would be selected.

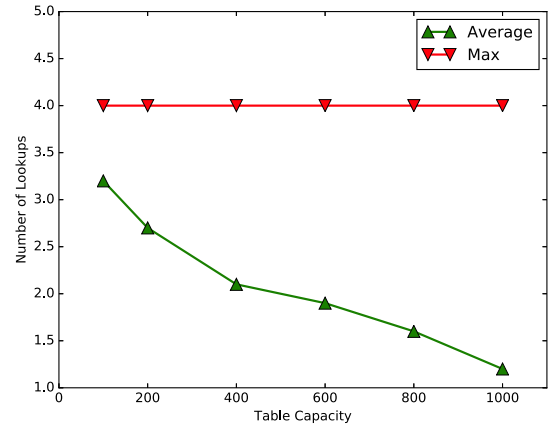
V. EVALUATION

This section evaluates our heuristic approach. We implement it in Java as a module in Floodlight controller and populate an Open vSwitch with more than 10K rules which are transformed from Stanford backbone network. Two host agent are connected to the switch and one generates a packet trace[15] with about 1 million packets sent to the other.

In our experiments, pipeline length and flow capacity are varied. With different pipe lengths and flow capacities, we collect data on packet lookup and compute average lookup times. The result is shown in Fig. 1 and Fig. 2. The cumulative



(a) Lookups Versus Pipeline Length



(b) Lookups Versus Table Capacity

Figure 1: Lookups

distribution of lookups indicates that about 50% of the lookups end within 2 stages in a pipeline. In Fig. 1a, the flow table capacity is set to 400. When pipeline length increases, the average of lookup times grows as well, but the average is almost half of the maximum lookup times. One instinctive explanation is that no flow table is empty and matches a portion of traffic. In Fig. 1b, the pipeline length is set to 4. When flow table capacity becomes larger, the average of lookup times decreases, because more space can accommodate more rules in a single flow table and no additional lookup is needed before a match is found in following tables.

VI. DISCUSSION AND FUTURE WORK

We do not consider updates upon rule insertion. Rules are installed at a certain flow table and then scanned by MILE to be linked to the sibling list or the candidate successor list. MILE needs to be improved on incremental updates so that the DAGs do not lag behind.

In addition, we will extend MILE to support other optional instructions and actions the OpenFlow protocol proposes. We will also conduct more tests with a diversity of parameters,

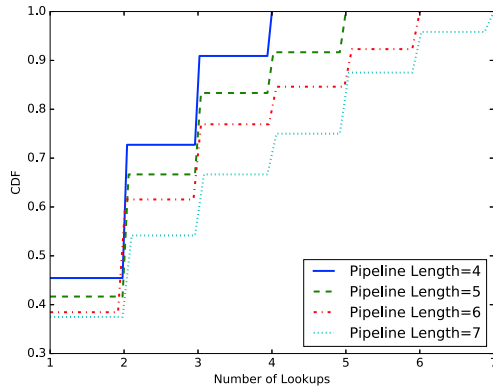


Figure 2: CDF of Lookups with Varied Pipeline Length

such as even longer pipeline length, to investigate the performance of MILE.

VII. CONCLUSION

The process of looking up a matched rule can be improved if more popular rules appear earlier. Unfortunately, a simple sorting algorithm is not feasible since the relative order of certain rules must be maintained. In this paper, The problem of determining actions on packets with a minimal number of packet lookups is formalized and proven to be NP-hard. We propose MILE, migrating flow rules between tables to minimize the average number of lookups to determine the action on a packet. Testing results show that the proposed method can reduce table lookups by 50%.

ACKNOWLEDGMENT

This work was supported in part by the National Natural Science Foundation of China under Grant 61572278, in part by the National Key R&D Program of China under Grant 2016YFB0800102, in part by the R&D Program of Shenzhen under Grant ZDSYS20140509172959989, JSGG20150512162853495, Shenfagai[2015]986.

REFERENCES

- [1] Floodlight, <http://www.projectfloodlight.org/floodlight>.
- [2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
- [3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
- [4] Open Networking Foundation, "Openflow switch specification," <http://www.openflow.org/documnets/openflow-spec-v1.1.0.pdf>.
- [5] Open Networking Foundation, "The benefits of multiple flow tables and ttps," <https://www.opennetworking.org>.
- [6] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 545–549.
- [7] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 13–24.
- [8] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Optimizing rules placement in openflow networks: Trading routing for better efficiency," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 127–132.
- [9] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 351–362.
- [10] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Infinite cache-flow in software-defined networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 175–180.
- [11] L. W. Yaoqing Liu, Vince Lehman, "Efficient fib caching using minimal non-overlapping prefixes," *Computer Networks*, vol. 83, pp. 85–99, 2015.
- [12] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu, "Domainflow: Practical flow management method using multiple flow tables in commodity switches," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: ACM, 2013, pp. 399–404.
- [13] Y. Wang, D. Tai, T. Zhang, B. Xu, L. Jin, H. Dai, B. Liu, and X. Wu, "Flowshadow: a fast path for uninterrupted packet processing in sdn switches," in *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, May 2015, pp. 205–206.
- [14] X. Sun, T. Ng, and G. Wang, "Software-defined flow table pipeline," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*, March 2015, pp. 335–340.
- [15] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, "Leveraging zipf's law for traffic offloading," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 1, pp. 16–22, Jan. 2012.