

Reducing and Balancing Flow Table Entries in Software-Defined Networks

Xuya Jia*, Yong Jiang*, Zehua Guo[†], Zhenwei Wu*

*Department of Computer Science, Tsinghua University Shenzhen Graduate School,
Tsinghua National Laboratory for Information Science and Technology(TNList). [†]ChinaCache
{jxy14, wz14}@mails.tsinghua.edu.cn, jiangy@sz.tsinghua.edu.cn, guolizihao@hotmail.com

Abstract—Software-Defined Networking (SDN) allows flexible and efficient management of networks. However, the limited capacity of flow tables in SDN switches hinders the deployment of SDN. In this paper, we propose a novel routing scheme to improve the efficiency of flow tables in SDNs. To efficiently use the routing scheme, we formulate an optimization problem with the objective to maximize the number of flows in the network, constrained by the limited flow table space in SDN switches. The problem is NP-hard, and we propose the K Similar Greedy Tree (KSGT) algorithm to solve it. We evaluate the performance of KSGT against “traditional” SDN solutions with real-world topologies and traffic. The results show that, compared to the existing solutions, KSGT can reduce about 60% of flow entries when processing the same amount of flows, and improve about 25% of the successful installation and forwarding flows under the same flow table space.

Keywords—Software-Defined Networking, MPLS, Flow table reuse, Overhead.

I. INTRODUCTION

With the development of next generation Internet, Software-Defined Networking (SDN) is widely used in enterprise WANs and data center networks [1]. The Ternary Content Addressable Memory (TCAM) is widely used in SDN switches, and it enables the fast forwarding packets in the data center network [2]. However, the capacity of TCAM in the SDN switch is limited since the TCAM is expensive, thus the flow tables of switch is limited [2][3]. The flow entries are overflowed in some core switches but underflowed in other switches, which may cause a lot of flows transmission failure [3]. Hence, designing a scheme that can efficiently reduce and balance the flow table entries to accommodate more flows in networks is a paramount problem.

In this paper, we try to solve the problem of how to reduce and balance the flow entries in different SDN switches to accommodate more flows in networks. We encapsulate Multi-Protocol Label Switching (MPLS) labels for each packet to indicate the routing information of forwarding path. Therefore, we do not need to install the flow entry to each switch of the forwarding path. However, if we add the MPLS labels to the packet only at the ingress node, it will generate too much overhead on the first link of forwarding path. Hence, we make full use of the advantages of SDN flexible center control to intelligently select nodes to install flow entry.

Although the problem of reducing and balancing the flow entries in different SDN switches is an NP-hard problem, we propose a fast algorithm called *K Similar Grade Tree* (KSGT)

to solve it under the network resource constraints. The KSGT algorithm comes from K-means algorithm [4], and it includes four main phases: 1) KSGT divides all flows into some clusters based on the coincidence degree of the forwarding paths of the flows. 2) KSGT greedily selects a small number of switches from the forwarding path of flow to install entries. 3) KSGT just greedily enumerates the possible flow entry schemes of the flows in each cluster to achieve the optimal solutions of the cluster. So the KSGT does not have high time complexity since it does not enumerate all of the solution space. 4) We use a phase function to balance the flow table entries of different switches. Hence, KSGT can effectively reduce and balance the flow table entries of different switches even though many new flows arrive in the network at the same time.

II. RELATED WORK

In order to avoid flow table overflow, some researches try to increase the usage efficiency of flow tables. In [1], *Zhu et al.* propose the dynamic timeout for SDN-based data centers. However, it may cause the controller overload due to the constraints of CPU, TCAMs, and other resources. In [2], *Cohen et al.* concentrate on satisfying global network objectives, such as the maximum flow, for maximizing network utilization in environments where the size of the forwarding table in network devices is limited. However, it can only handle the static scene.

Liang et al. [3] focus on optimizing idle_timeout value for Instant Messaging (IM) to balance flow tables and controller processing resources in OpenFlow networks. Their scheme may cause overload on the links between the controller and switches since the switches generate a large number of packet-in events per second to indicate controller processing resource cost. *Guo et al.* [5] also reduce flow entries in SDN switches, but their algorithm only handles a single flow and each packet only carries little routing information for the VLAN's limit.

In [6], *Banerjee et al.* propose *Tag-In-Tag*, an approach that exploits SDN features and replaces the flow entries with two layers of simpler and shorter tags. However, this can be troublesome for large graphs as this can lead to path explosion.

III. PROBLEM FORMULATION

In this section, we formally define the problem of how to reduce and balance flow entries in SDN switches to accommodate more flows in the networks. The latest OpenFlow

Table I
NOTATIONS.

Notation	Definition
V	The set of nodes in the network
F	The set of flows in the network
K	The number of clusters in the flow classifications
$ V $	The number of nodes in the set V
$ F $	The number of flows in the set F
v	A node in the set V
f	A flow in the set F
r_f	The arrival rate of flow f
u_v	The number of entries in the node v that have been installed before the arrival of the flows
u_{max}	The flow table capacity of a SDN switch
w_v	The number of entries to be installed in the node v by controller for all flows
y_v	$y_v = 1$ if node v is installed entry by controller; $y_v = 0$, otherwise
x_v^f	$x_v^f = 1$ if node v is installed entry for the flow f ; $x_v^f = 0$, otherwise
c_f	The vector of nodes that the flow f go through ($c_f = [x_1^f, x_2^f, \dots, x_v^f, \dots]$)
a, b, c	The serial number of nodes in vector c_f ; $c_f = [x_1^f, \dots, x_a^f, \dots, x_b^f, \dots, x_c^f, \dots]$ ($a < b < c$)
$len(c_f)$	The length of the vector c_f
p	The utilization of the flow table on switch ($p \in [0, 1]$)

protocol supports MPLS technology [7], the *Push MPLS header* actions can push new MPLS headers onto the packet. We encapsulate multiple MPLS labels for each packet that indicates the forwarding port numbers of switches on its route. For example, Fig. 1 shows our scheme for a new flow f , the controller selects switches s1 and s7 to install flow entries. When the packets of flow f enter the ingress switch s1, they are encapsulated with 5 MPLS labels, {2,2,3,2,3}, each number indicates a forwarding port of the switch on its route. The packets pop up one MPLS label as a forwarding port of switch when they pass through the switch [7].

A. System Description Let $G = (V, E)$ be a network, where V is the set of nodes, and E is the set of links. Let F denote the set of traffic flows. We use v to represent a node in the network, f to represent a flow in the network. Let r_f denote the arrival rate of flow f , and u_{max} be the flow table capacity of a SDN switch. The full list of notations is shown in Table I.

B. Constraints In our scheme, the switches forwarding packets are based on multiple MPLS labels. We give some constraints and definitions before modeling.

Definition 1. Overhead load: The MPLS labels can not be regarded as payload. We define the overhead load as $Inl(c_f)$ since the flow f is encapsulated with MPLS labels to carry the routing information [8]. The size of MPLS labels is 4 bytes, so the overhead load for flow f can be formulated as:

$$Inl(c_f) = 4 \cdot \sum_{x_a^f, x_b^f, x_c^f}^{c-a-2} (c-a-1-z) \quad (1)$$

for all $b \in (a, c)$ subject to:

$$x_a^f = 1, \quad x_b^f = 0, \quad x_c^f = 1.$$

For example, Fig. 1 shows our scheme for a flow f , we select switches s1 and s7 to install entries. Hence, $c_f = [s1, s2, \dots, s10] = [1, 0, 0, 0, 0, 1, 0, 0, 0]$, and the overhead load for each data packet is $Inl(c_f) = 4 \times (5 + 4 + 3 + 2 + 1) + 4 \times (3 + 2 + 1) = 84$. If we just install entry at the ingress switch s1 then the overhead load for each packet is $Inl(c_f) = 4 \times (9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1) = 180$.

Definition 2. Border load: We define border load φ as the tolerable overhead proportion of each packet in the network.

The smallest data packet size is 46 bytes in the network, but the size of each MPLS label is 4 bytes [8]. In Fig. 1, if we just install entry at the flow's ingress switch, the size of MPLS labels is $4 \times 9 = 36$ bytes for each packet, so the overhead load on the link between s1 and s2 is more than 12% of total load for these packets whose sizes are less than 300 bytes. Note that there are many different network packets whose sizes are between 100 bytes and 300 bytes according to the statistics in [9]. So we must select some other nodes to install entries to reduce the overhead load instead of just installing entries in the ingress switch. Hence, we define the border load ratio φ for each packet in the network to limit the overhead load.

According to the measurement method in [10], we can suppose an observation window contains S packets at time t . The average packet size at time t is defined as

$$E(t) = \sum_l \frac{n_l \cdot l}{S}, \quad (2)$$

where n_l is the number of packets with size l in the observation window. We can draw the following constraints.

Constraint 1: Equation (3) guarantees that the average overhead load proportion of each packet can not exceed the border load in the network.

$$\frac{\sum_{f=1}^{|F|} \left\{ \frac{r_f}{\sum_{f=1}^{|F|} r_f} \cdot Inl(c_f) / len(c_f) \right\}}{E(t)} \leq \varphi, \quad (3)$$

for all $b \in (a, c)$ subject to:

$$x_a^f = 1, \quad x_b^f = 0, \quad x_c^f = 1;$$

Constraint 2: Equation (4) indicates the number of entries to be installed to node v for all flows.

$$w_v = \sum_{f=1}^{|F|} x_v^f, \quad x_v^f \in \{0, 1\} \quad (4)$$

Constraint 3: Equation (5) states that whether the node v is installed entries by the controller for all flows.

$$y_v = \begin{cases} 0, & w_v = 0 \\ 1, & w_v \geq 1 \end{cases} \quad (5)$$

Constraint 4: Equation (6) ensures that the number of entries in the switch can not exceed the capacity.

$$u_v + w_v \leq u_{max}, \quad (6)$$

C. Cost Function The cost function is a piecewise function used to simulate the nonlinear impact of a link load on the network congestion in internet traffic engineering [5][6]. As shown in [5], we can use the cost function (7) to simulate the nonlinear impact of the flow table usage increase on the controller's workload.

$$g(p) = \begin{cases} p, & 0 \leq p < 1/3 \\ 3 \cdot p, & 1/3 \leq p < 2/3 \\ 10 \cdot p, & 2/3 \leq p < 9/10 \\ 70 \cdot p, & 9/10 \leq p < 1 \\ 500 \cdot p, & p = 1 \end{cases} \quad (7)$$

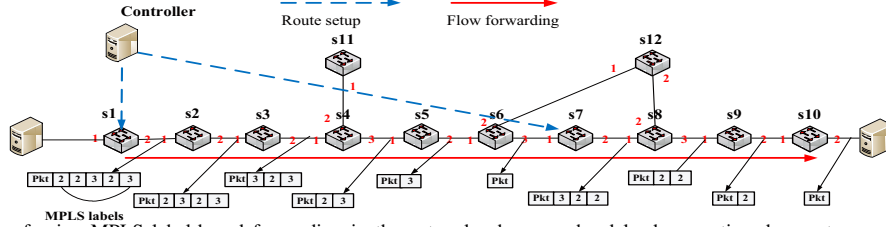


Figure 1. An example of using MPLS label-based forwarding in the network where overhead load proportion does not exceed the border load φ . The controller configures the forwarding path for a new flow c_f by installing the flow entries only at selected switches s1 and s7.

The cost function is used for all switches, and p is the utilization of the flow table on the switch. With the increasing number of the entries installed on the switches, the value p also increases.

D. Problem Formulation The problem of reducing and balancing flow table entries in SDN switches when many new flows appear at the same time in SDNs is formulated below:

$$\min \sum_{v=1}^{|V|} g\left(\frac{u_v + w_v}{u_{max}}\right) \cdot y_v, \quad (8)$$

subject to the Constraints 1 to 4.

E. Complexity Analysis The cost function $g(p)$ is a integer linear function, then our problem can be reduced to the Mixed-integer programming problem. So our optimization problem is at least as hard as Mixed-integer programming problem. The Mixed-integer programming problem is an NP-hard problem, so our optimization problem is also NP-hard.

IV. HEURISTIC ALGORITHMS

Section III shows that the problem is an NP-hard problem. We propose the KSGT algorithm to get the approximate optimal results. KSGT can effectively reduce and balance the flow table entries of switches even though many new flows arrive in the network at the same time.

A. The Outline Of The KSGT Algorithm

Algorithm 1 KSGT(K-Similar-Greed-Tree)-algorithm

```

1: Input:  $G(V, E)$  and  $c_f$ 
2: Output: The number of entries to be installed for all flows, and the corresponding nodes.
3: Phase 1:
4:  $cluster-k \ k \in [1, K] \leftarrow$  K-Similar( $G$ ) algorithm
5: Phase 2:
6: for  $k \in [1, K]$  do
7:   for flow vector  $c_f$  in  $cluster-k$  do
8:      $scheme(c_f) \leftarrow$  Greedy( $c_f$ )
9:   end for
10: end for
11: Phase 3:
12: for  $cluster-k \ k \in [1, K]$  do
13:    $scheme-cluster(k) \leftarrow$  Enumerate( $cluster-(k), 1$ )
14: end for
15: Phase 4:
16: while ( $k \geq 2$ ) do
17:   combine  $scheme-cluster(i)$  and  $scheme-cluster(i+1)$  to select some optimal schemes
18:    $k = k/2$ ;
19: end while
20: return The optimal scheme in  $scheme-cluster(k)$ 

```

The $cluster-k$ denotes the flows in the k -th cluster. The flows in the same cluster have high coincidence degree of

forwarding paths. We use the $scheme(c_f)$ to denote the two optimal schemes of installing flow entries for flow c_f , for example, s1 and s7 is a scheme for flow c_f in Fig. 1. We use $scheme-cluster(k)$ to indicate some optimal schemes for flows in the k -th cluster. The algorithm has four phases: the first phase is to classify the flows according to the number of overlapping nodes by K-Similar algorithm, which comes from K-means algorithm [4]; the second phase is greedily select two schemes for each flow; the third phase is to use the cost function to evaluate the possible schemes of the flows in each cluster to achieve some optimal solutions for every cluster; the fourth phase is to combine each of the two clusters into one until only one remains. Then we can get the optimal results by the algorithms. The computational complexity of the KSGT algorithm is $O(|F| * len(c_f) * (K + steplength))$. The space complexity is $O(|F| * len(c_f))$.

V. SIMULATION

In this section, we evaluate our solution against baseline schemes use the real network topology 1755 ISP from the Rocketfuel [11]. The flows between node pairs are dynamically generated, and the size of each flow is random [6].

A. Comparison Schemes

We propose the KSGT algorithm to solve the problem that reduces and balances the flow entries in SDN switches. To evaluate our algorithm, we compare it with other algorithms:

- **All-nodes:** The “traditional” SDN solution that the controller installs flow entries to all switches of forwarding path for all flows [12].
- **First-node:** It installs an entry only at the flow’s ingress switch [13], so the MPLS labels of each packet carry all information of forwarding path.

B. Simulation Setup We randomly set the initial flow entries number of switches which do not exceed the flow table capacity u_{max} . We assume that each switch’s flow table capacity u_{max} is equal to 5000. We evaluate the algorithms with different new flows arrival speed which range from 300 flows/second to 1100 flows/second [9], and we test each flow speed ten seconds to get the average value of all parameters.

C. Simulation Results

1) *The number of entries installed for all nodes:* Fig. 2 shows that the number of flow entries installed for all nodes increases with the different arrival rates of new flows. It is evident that the result of KSGT algorithm is smaller than the All-nodes algorithm. Compared with the All-nodes algorithm,

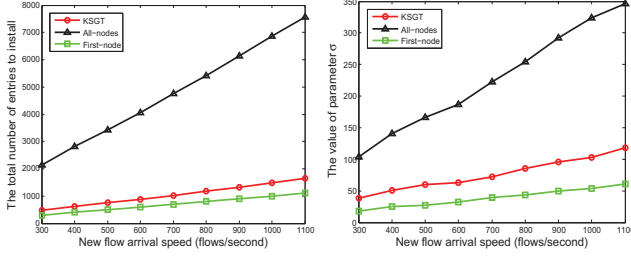


Figure 2. The number of entries installed for all nodes.

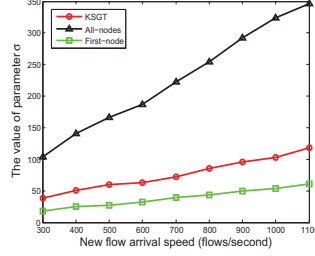


Figure 3. The value of parameter σ .

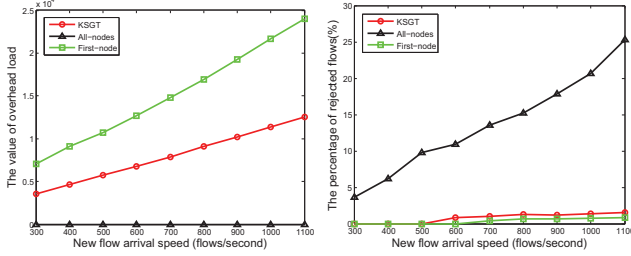


Figure 4. The value of overhead load for all flows.

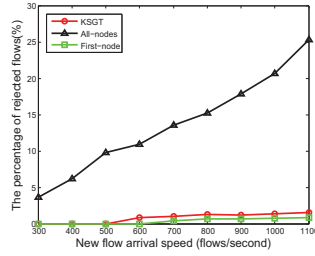


Figure 5. The percentage of rejected flows.

the KSGT algorithm can reduce the number of the flow entries about 60%. The First-node algorithm may exceed the network border load φ as the analysis in Section III and may cause too much overhead load as shown in Fig. 4. So the KSGT algorithm is more practical.

2) *The difference between the maximum and minimum flow entries to be installed:* We use σ to describe the maximum flow entries subtract the minimum flow entries, so σ can describe the balance degree of flow table of different switches. Seen from Fig. 3, we can find that the parameter σ of the KSGT algorithm is always smaller than the all-nodes algorithm. So the switches' flow tables of KSGT algorithm are more uniform than All-nodes algorithm. From Fig. 4, we can see the First-node algorithm causes too much overhead load. As it is analysed in our model, it can not be applied to the networks where the overhead load is restricted.

3) *The overhead load for all flows:* Seen from Fig. 4, the overhead load of First-node algorithm is larger than other algorithms. With the increase of the arrival rate of new flows, the overhead load gap also increases. It is worth emphasizing that there are many different network packets whose sizes are between 100 bytes and 300 bytes according to the statistics in [9]. So the average overhead load on the link of the forwarding path is about 18% of total load. So the First-node algorithm can not be widely used since the overhead load may exceed the network border load in the networks.

4) *The percentage of rejected flows:* The rejected flows mean that these flows can not be successfully transmitted because the flow tables are full. Fig. 5 shows that with the increase of the arrival rate of new flows, the All-nodes algorithm leads to more and more flows rejected. In other words, the KSGT algorithm improves the successful installation and forwarding flows by about 25% as compared with All-nodes

algorithm. So the All-nodes algorithm can not be widely used because it causes massive flows rejected. The First-node algorithm causes too much overhead load in Fig. 4. Therefore, the KSGT algorithm is more practical.

VI. CONCLUSION

In this paper, we formulate our problem as an optimization problem, and give the hardness analysis of the problem. We propose the greedy algorithm KSGT to solve the problem. Compared with the existing solutions, KSGT can reduce about 60% of flow entries when processing the same amount of flows, and improve the successful installation and forwarding flows by about 25% under the same flow table space in SDN switches.

ACKNOWLEDGMENT

This work is supported by the National Research Program of China (973) under grant No. 2012CB315803, the National Natural Science Foundation of China under grant No. 61402255, the R&D Program of Shenzhen under grant No. ZDSYS20140509172959989, No. JSGG20150512162853495, No. Shenfagai[2015]986, and JCYJ20150630170146830.

REFERENCES

- [1] H. Zhu, H. Fan, and Luo, "Intelligent timeout master: Dynamic timeout for sdn-based data centers," in *proceedings of the Integrated Network Management*. IEEE, 2015, pp. 734–737.
- [2] R. Cohen, L. Lewin-Eytan, J. S. Naor *et al.*, "On the effect of forwarding table size on sdn network utilization," in *Proceedings of IEEE INFOCOM*. IEEE, 2014, pp. 1734–1742.
- [3] H. Liang, P. Hong, J. Li, and D. Ni, "Effective idle_timeout value for instant messaging in software defined networks," in *proceedings of the Communication Workshop*. IEEE, 2015.
- [4] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, no. 3, pp. 433–439, 1999.
- [5] Z. Guo, Y. Xu, M. Cello, J. Zhang, Z. Wang, and Liu, "Jumpflow: Reducing flow table usage in software-defined networks," *Computer Networks*, vol. 92, pp. 300–315, 2015.
- [6] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow table management in sdn switches," in *proceedings of the Network and Service Management (CNSM)*. IEEE, 2014, pp. 109–117.
- [7] "Openflow switch specification 1.3.1." [Online]. Available: https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml
- [8] "Encapsulating mpls in ip or generic routing encapsulation (gre)." [Online]. Available: <http://www.hjp.at/doc/rfc/rfc4023.html>
- [9] "Packet size distribution comparison between internet links in 1998 and 2008." [Online]. Available: https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml
- [10] P. Du and S. Abe, "Detecting dos attacks using packet size distribution," in *Bio-Inspired Models of Network, Information and Computing Systems*. IEEE, 2007, pp. 93–96.
- [11] N. Spring, R. Mahajan, and Wetherall, "Measuring isp topologies with rocketfuel," in *SIGCOMM*. ACM, 2002, pp. 133–145.
- [12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, no. 2, pp. 69–74, 2008.
- [13] P. Ashwood-Smith, M. Soliman, and T. Wan, "Sdn state reduction," *IEFT draft*, 2013.