Flowtable-Free Routing for Data Center Networks: A Software-Defined Approach

Yi Ren, Tsung-Han Tsai, Ji-Cheng Huang, Cheng-Wei Wu, and Yu-Chee Tseng
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

***Abstract*—The paradigm shift toward SDN has exhibited the following trends: (1) relying on a centralized and more powerful controller to make intelligent decisions, and (2) allowing a set of relatively dumb switches to route packets. Therefore, efficiently looking up the flowtables in forwarding switches to guarantee low latency becomes a critical issue. In this paper, following the similar paradigm, we propose a new routing scheme called KeySet which is flowtable-free and enables constant-time switching at the forwarding switches. Instead of looking up long flowtables, KeySet relies on a residual system to quickly calculate routing paths. A switch only needs to do simple modular arithmetics to obtain a packet's forwarding output port. Moreover, KeySet has a nice fault-tolerant capability because in many cases the controller does not need to update flowtables at switches when a failure occurs. We validate KeySet through extensive simulations by using general as well as Facebook fat-tree topologies. The results show that the KeySet outperforms the KeyFlow scheme [1] by at least 25% in terms of the length of the forwarding label. Moreover, we show that KeySet is very efficient when applied to fat-trees.**

***Index Terms*—Data Center, Network Protocol, Routing, SDN, Switching.**

I. Introduction

Software Defined Networking (SDN) is an emerging network architecture that separates traditional switching functions into a control plane and a data plane. The control plane is responsible for configuring switches and specifying forwarding paths for data flows. The data plane is simply responsible for forwarding packets. In such designs, a network controller has a global view of the network and is able to efficiently define a collection of forwarding rules for switches to follow. Accordingly, the switches with the rules can perform simple actions such as forwarding, flooding, and directing packets to the network controller. However, providing fine-grained forwarding rules leads to large amounts of rules deployed in the switches.

Openflow [2] is a popular protocol for the message exchange between the network controller and switches in SDN. In Openflow switches, forwarding rules are implemented in Ternary Content-Addressable Memory (TCAM), which can compare an incoming packet to all the rules at the same time, reducing switching delay significantly. However, the TCAMs address a tradeoff between flowtable size and deployment cost. In general, the TCAMs introduce ten times more expensive [3] and 100 times greater power consumption [4] compared with traditional RAM with the same size. This cost constrain leads to small TCAMs in most switches, even there are large ones in the market. Moreover, TCAMs use ternary format for rules storage, which causes the well-know range expansion problem. That is, a flow rule leads to a large number of TCAM entries [5]. Therefore, cutting the sizes of flowtables is an essential issue.

Intensive studies [6]–[9] have dedicated to reducing the size of flowtables. Those approaches can be generally divided into two categories: (1) decreasing the total number of flow entries, or (2) cutting the sizes of flow entries. References [6], [7] take the former approach. However, packets missing their corresponding flow entries need to be sent to the controller for fetching their entries, introducing extra delays. Another direction is to reduce the size of each flow entry [8], [9]. A shorter label is used to replace the MAC address in a flowtable. Using a shorter label not only reduces flow entry size, but also enables the packets with the same forwarding path to share one label, decreasing the number of flow entries accordingly. However, such designs suffer from large flowtable update cost when there are switch failures or major network topology changes.

So, instead of reducing flowtable size, is it possible to remove flowtable from switches? In this paper, we propose a new routing scheme called KeySet, which is flowtable-free and enjoys constant-time switching. KeySet removes the overhead of fetching large new flow entries from the controller. Instead, switches only need to conduct very simple modular arithmetic to obtain packets' output ports. This simplifies the switch hardware requirements. The main idea of KeySet is to employ two or more sets of modular numbers at switches and allow the controller to adaptively select a set for the switching purpose. This gives more flexibility to choose a better label. We validate KeySet through extensive simulations and compare it with KeyFlow [1]. Simulation results show that KeySet outperforms the KeyFlow scheme by at least 25% in terms of the length of forwarding label. Moreover, we show that KeySet is very efficient when being applied to fat-tree topologies.

The remainder of the paper is organized as follows. Section II reviews related work. Section III introduces preliminaries and problem statement. KeySet is proposed in Section IV, followed by simulation results in Section V. Section VI draws conclusions.

II. Related Work

Intensive research has focused on reducing the requirements of TCAM by cutting flowtables. Solutions can generally be divided into three categories, as reviewed below.

A. Reducing the Number of Flow Entries

Reference [6] proposes a distributed scheme, in which a large flowtable is decomposed into small ones and then distributed across the network. It well balances the sizes of the tables across the network and reduces the total number of flow entries. The algorithm, however, depends on the length of the shortest path in the network and does not work well when the length is small since not all available switches are taken into consideration. The work [7] improves [6] by caching the most popular rules in the small TCAM. The rest small amount of cache miss traffic are handled by software. The algorithm minimizes the number of rules for a single switch significantly. However, maintaining a subset of flowtables introduces extra fetching delay between switches and the controller if packets miss their corresponding flow entries.

B. Reducing the Sizes of Flow Entries

The main idea is to use short structured labels to replace unstructured MAC addresses. Reference [8] proposes to use MAC addresses as virtual addresses to minimize TCAM usage. Reference [9] proposes to use destination MAC address as a universal label. The ARP caches of hosts can be exploited as an ingress label table to fill destination MAC addresses. Such a design reduces the number of flow entry and the size of each flow entries, without increasing the packet header. However, the path label assignment is NP-complete.

C. Flowtable-Free Routing

References [1], [10] are most related to our work. In [10], the proposed scheme does not require header modification in each router. A label is computed by the Chinese Remainder Theorem and appended to each packet. The label actually "encodes" the forwarding port information of each router traversed by a path (refer to the details in Sec. III. A). Based on [10], reference [1] further shows how to replace the MAC address by the label. So no extra header is needed. However, the label could be very large when the forwarding path is long or the number of nodes in the network is large.

III. Preliminaries and Problem Statement

In this section, we first briefly introduce the Chinese Remainder Theorem (CRT), which is the basis of our flowtable-free routing scheme. Then we formally define our problem.

A. CRT and Flowtable-Free Routing

Theorem 1. *[11] Let $p_1, p_2, \cdots, p_n$ be n pairwise relatively primes, and $r_1, r_2, \cdots, r_n$ be arbitrary integers. There exists a unique integer $\mathcal{X}$ such that $0 \leq \mathcal{X} < \prod_{i=1}^{n} p_i$ that solves the following system of congruence equations:*

$$\begin{cases} \mathcal{X} \equiv r_1 \,(\text{mod } p_1) \\ \mathcal{X} \equiv r_2 \,(\text{mod } p_2) \\ \quad\vdots \\ \mathcal{X} \equiv r_n \,(\text{mod } p_n). \end{cases} \tag{1}$$

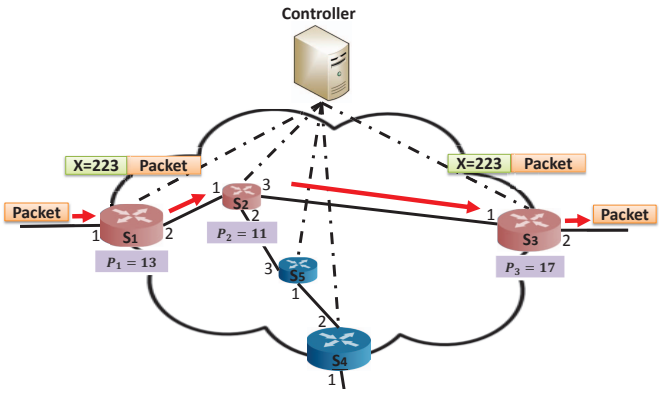

Fig. 1: An example of flowtable-free routing in KeyFlow.

The value of $\mathcal{X}$ can be calculated as

$$\mathcal{X} = (\sum_{i=1}^{n} c_i \cdot r_i) \,(mod\ P), \tag{2}$$

where $P = \prod_{i=1}^{n} p_i$, $c_i = Q_i U_i$, $Q_i = \frac{P}{p_i}$, and $U_i = Q_i^{-1} \,(mod\ p_i)$. Here, U_i is the multiplicative inverse of Q_i under modulo p_i.

References [1], [10] exploits CRT for flowtable-free routing. The basic idea is to encode forwarding information of a packet into a label (i.e., $\mathcal{X}$) and replace the MAC address of a packet with the label. Fig. 1 illustrates an example, where a packet needs to go through switches S_1, S_2 and S_3. Each switch is pre-assigned with a unique prime number (13, 11 and 17 for S_1, S_2 and S_3, respectively). Since the controller knows that the output ports at S_1, S_2 and S_3 are 2, 3 and 2, respectively, it will compute $\mathcal{X} = 223$ as the label of the packet by Theorem 1. Then, each switch uses this integer 223 and its pre-assigned prime number to get its output port. At S_1, it will compute its output port as $r_1 = 2 \equiv \mathcal{X} \pmod{13}$. Similarly, at S_2 and S_3, they will compute their output ports as $r_2 = 3 \equiv \mathcal{X} \pmod{11}$ and $r_3 = 2 \equiv \mathcal{X} \pmod{17}$, respectively.

Clearly, the magic number $\mathcal{X}$ satisfies the congruence system:

$$\begin{cases} \mathcal{X} \equiv r_1 \,(\text{mod } 13) \\ \mathcal{X} \equiv r_2 \,(\text{mod } 11) \\ \mathcal{X} \equiv r_3 \,(\text{mod } 17). \end{cases}$$

Therefore, $\mathcal{X}$ can be used by switches to route packets without looking up flowtables. Modular arithmetic can be done at switches very quickly. Only a prime is needed to be stored in each switch. Also, the controller does not need to update flowtables when some switches fail.

B. System Model and Problem Statement

We consider a SDN network domain consisting of a controller and a set $\mathcal{S}$ of n switches, where $\mathcal{S} = \{S_i \mid i = 1, 2, \ldots, n\}$. In [1], [10], in order to make the CRT congruence system solvable, n integers $p_1, p_2, \cdots, p_n$ are needed to be

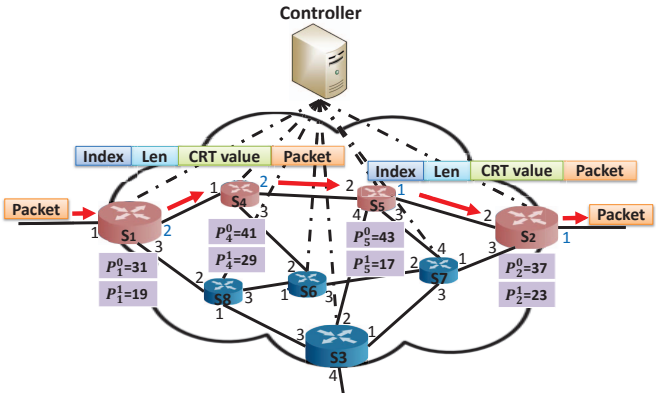

Fig. 2: A packet forwarding example using KeySet.

pairwise relatively primes for these n switches. We make the following observations on [1], [10]:

- Observation 1: $\mathcal{X}$, which is attached to each packet, can be considered as an overhead. Since $0 \leq \mathcal{X} < \prod_{i=1}^{n} p_i$, a shortcoming is that $\mathcal{X}$ will enlarge quickly as n increases.
- Observation 2: To ensure the CRT congruence system to be always solvable, it only needs to guarantee that the primes of the switches passed by each routing path are pairwise relative prime. That is, a prime may be shared by two switches as long as no routing path traverses there two switches. Therefore, using duplicated primes is possible. This potentially reduces the value of $\mathcal{X}$.
- Observation 3: When two switches in a routing path have the same output ports, they can use the same prime. Consider Fig. 1 as an example, S_1 and S_3 both send out the packet at output port 2. If we change S_3's prime to 13 (the same as p_1's), the congruence system is still solvable (then $\mathcal{X}$ becomes 80). This may further reduce the value of $\mathcal{X}$.

IV. The Proposed KeySet Solution

Following the above observations, we propose a new scheme called KeySet. The basic idea is to keep two (or even more) sets of primes for higher flexibility and allow duplicate primes in the network domain as long as there is no hazard for route selection.

A. Generating Prime Sets

KeySet exploits multiple sets of primes to solve the problem of $\mathcal{X}$ being too large. To simplify our presentation, we use two sets of primes to show how it works. The controller first prepares two sets, P^0 and P^1, where P^0 consists of n primes and P^1 consists of $\frac{n}{\alpha}$ primes, where α is a preset value (say $\alpha = 2$). Note that primes in P^0 and P^1 may overlap. A selected prime of the switch is bound to be more than the total number of ports of the switch. Intuitively, P^0 is the same as the prime set in KeyFlow. The goal of P^0 is to guarantee the worst-case scenario solvable such that every packet is "routable", while P^1 is to help reduce the size of $\mathcal{X}$ whenever possible.

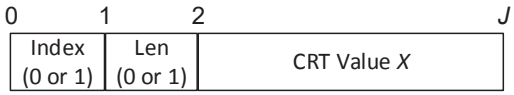

Fig. 3: Forwarding label definition.

An example is shown in Fig. 2. Suppose that there are 8 switches and 16 ports for each switch. The first 8 primes greater than 16 are assigned to 8 switches in such a way that each prime in P^0 will be given to a unique switch, but each prime in P^1 will be given to 2 switches. Then, we set $P^0 = \{17, 19, 23, 29, 31, 37, 41, 43\}$, and $P^1 = \{17, 19, 23, 29\}$. A packet wants to go through switches S_1, S_4, S_5 and S_2. If we adopt the KeyFlow scheme, that is, only use the prime set P^0, we compute the integer $\mathcal{X} = 1573500$ by Theorem 1. By using our proposed KeySet scheme, the integer $\mathcal{X}$ can be reduced to 203321.

The assignment could be optimized (if there is more information available) but here we simply adopt a random assignment.

B. Calculating Forwarding Labels

Our solution also relies on a forwarding label attached on each packet to achieve fast switching. The forwarding label format is shown in Fig. 3. Index is to choose a prime set: 0 for P^0 and 1 for P^1. Len is to define the length of the forwarding label: 0 for full length $J_0 = 2 + \lceil lg(\prod_{p_i \in P^0} p_i) \rceil$ and 1 for the partial length $J_1 = 2 + \beta \cdot \lceil lg(\prod_{p_i \in P^0} p_i) \rceil$. Here β is a controller-defined ratio. For example, in a 224 nodes fat-tree topology, β can be 0.75 (refer to the discussion of Fig. 6 (b) in Sec. V). The last part is to carry the $\mathcal{X}$ parameter for this route.

When a packet aiming at a new destination arrives at an ingress switch of the network domain, it is sent to the controller for forwarding instructions. The controller computes the forwarding label by the following steps.

Step 1: A proper end-to-end routing path is first computed. Let the routing path go through a sequence of k switches, namely $S'_1, S'_2, \cdots, S'_k$, where S'_1 is the ingress switch and S'_k is the egress switch. Also, let the output port for S'_i be r_i, $i = 1, 2, \cdots, k$.

Step 2: Let p_i^0 and p_i^1 be the prime assignments of S'_i in P^0 and P^1, respectively. Since the controller knows the assignments of all primes, it can decide which prime set to be used by checking the following condition:

$$\mathbb{C} : \forall i, j \in \{1, 2, \cdots, k\} \text{ and } r_i \neq r_j \Rightarrow p_i^1 \neq p_j^1$$

If condition $\mathbb{C}$ is true, we choose prime set P^1 and let Index $= 1$; otherwise, we choose prime set P^0 and let Index $= 0$.

Step 3: If P^0 is selected, we build the congruence system

$$\mathcal{X} \equiv r_i \pmod{p_i^0} \text{ for } i = 1, 2, \cdots, k.$$

Otherwise, we build the congruence system

$$\mathcal{X} \equiv r_i \pmod{p_i^1} \text{ for } i = 1, 2, \cdots, k.$$

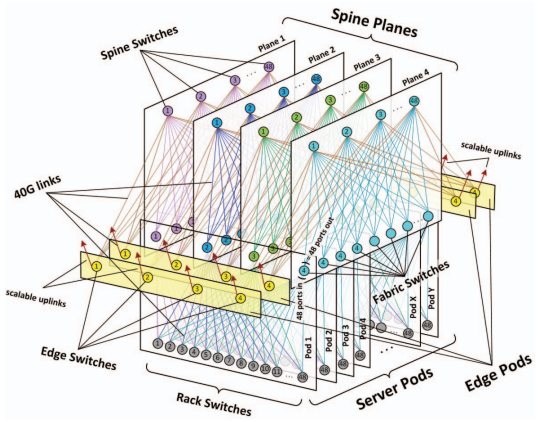

Fig. 4: Facebook data center fabric network topology [12].

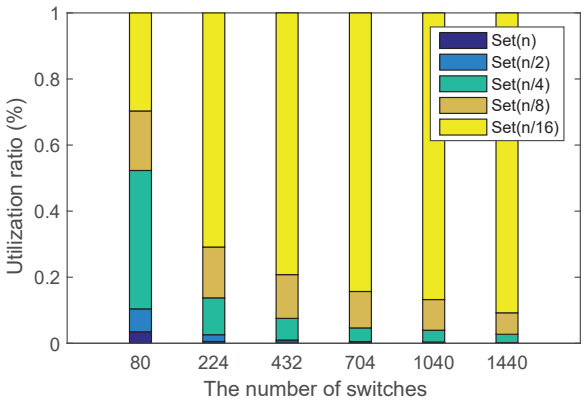

Fig. 5: Impacts of prime set size.

No matter which congruence system is used, $\mathcal{X}$ is solvable because condition $\mathbb{C}$ guarantees that whenever r_i and r_j are distinct, the corresponding primes for modular arithmetic must be distinct. Then we use Theorem 1 to find $\mathcal{X}$. Note that condition $\mathbb{C}$ integrates both observations 2 and 3 discussed above.

Step 4: Finally, the controller sets the Len based on the length of the forwarding label required: $\mathrm{Len} = 1$ if $\lceil lg\mathcal{X} \rceil \leq \beta \cdot \lceil lg(\prod_{P_i \in P^0} P_i) \rceil$ and $\mathrm{Len} = 0$ otherwise. After that, the controller appends the label on the packet header and sends it back to the ingress switch.

C. Forwarding Packets

When a packet with a forwarding label is returned from the controller to the ingress switch S'_1, the ingress switch conducts the following steps.

Step 1: The forwarding label is stored in its local memory. Note that if a different forwarding label exists correspondent to the same destination, the latter should be removed to avoid confusion.

Step 2: The switch parses the Index to decide the prime to be used and parses Len to determine the range of $\mathcal{X}$.

Step 3: The proper prime is chosen to conduct modular arithmetic on $\mathcal{X}$. This obtains the output ports. Then the switch forwards the packet to that port without modifying the packet.

Similarly, when any other non-ingress switch receives the packet, it executes Steps 2 and 3 to forward the packet. In the future, when a packet aiming at the same destination arrives at S'_1, the switch will identify the forwarding label in its memory. The label is attached to the packet, which is then forwarded by Steps 2 and 3.

Finally, at the egress switch S'_k, the packet will also be forwarded by Steps 2 and 3. The only difference is that S'_k will remove the forwarding label from the packet before sending it out.

V. Simulation Results

We have seen rapid growth of data centers. The network to support a data center also becomes extremely complex so as to inter-connect hundreds or thousands of network devices. Many mega-operators, such as Google, Yahoo!, and Facebook, have build ultra large-scale networks capable of providing high performance and low-cost connectivity among huge numbers of physical servers. We therefore investigate the possibility of applying KeySet to such data center networks. In our simulations, we construct the fat-tree topology and assign primes to each switch. After that, we use all shortest paths to calculate the length of the forwarding labels.

Fig. 4 shows the schematic of Facebook data center fat-tree fabric network topology [12]. Assume that each switch has m ports in the fat-tree topology. Then, there are 4 spine planes and m server pods depending on the number of fabric switches (say m) in each spine plane. For each fabric switch, the $\frac{m}{2}$ ports are connected to the $\frac{m}{2}$ spine switches in the same spine plane, and the other $\frac{m}{2}$ ports are connected to the $\frac{m}{2}$ rack switches in the same server pod. Totally, there are $2m$ spine switches, $4m$ fabric switches, and $\frac{m^2}{2}$ rack switches. The total number of switches is $n = 2m + 4m + \frac{m^2}{2}$. Our simulations consider such fat-tree network architectures with $n = 80, 224, 432, 704, 1040,$ and 1440 switches.

A. Impacts of α

Recall that the basic idea of KeySet is to use smaller prime sets (e.g., P^1 with $\frac{n}{\alpha}$ primes) for saving the label size, and P^0 is to support the worst-case scenario. A larger α leads to a smaller prime set and may potentially reduce the forwarding label length. However, a too small P^1 will also reduce the possibility of itself being selected to construct forwarding labels, resulting in being selected P^0 instead. Therefore, choosing a suitable α to guarantee a high utilization ratio of P^1 is desired.

Fig. 5 shows the prime sets utilization ratio of all shortest paths in the fat-tree topology. Here, the utilization ratio is defined as the ratio of the prime set chosen for conducting modular arithmetic on $\mathcal{X}$ in the simulations. We observe the impact on $\alpha = 2, 4, 8,$ and 16, respectively. We can see that prime $Set(\frac{n}{16})$ is frequently used when $n = 1440$, whereas, $Set(\frac{n}{16})$ does not work well with $n = 80$. The reason is that when n is small (say 80), $Set(\frac{n}{16})$ is too small to select distinct primes in routing path congruence systems. Based

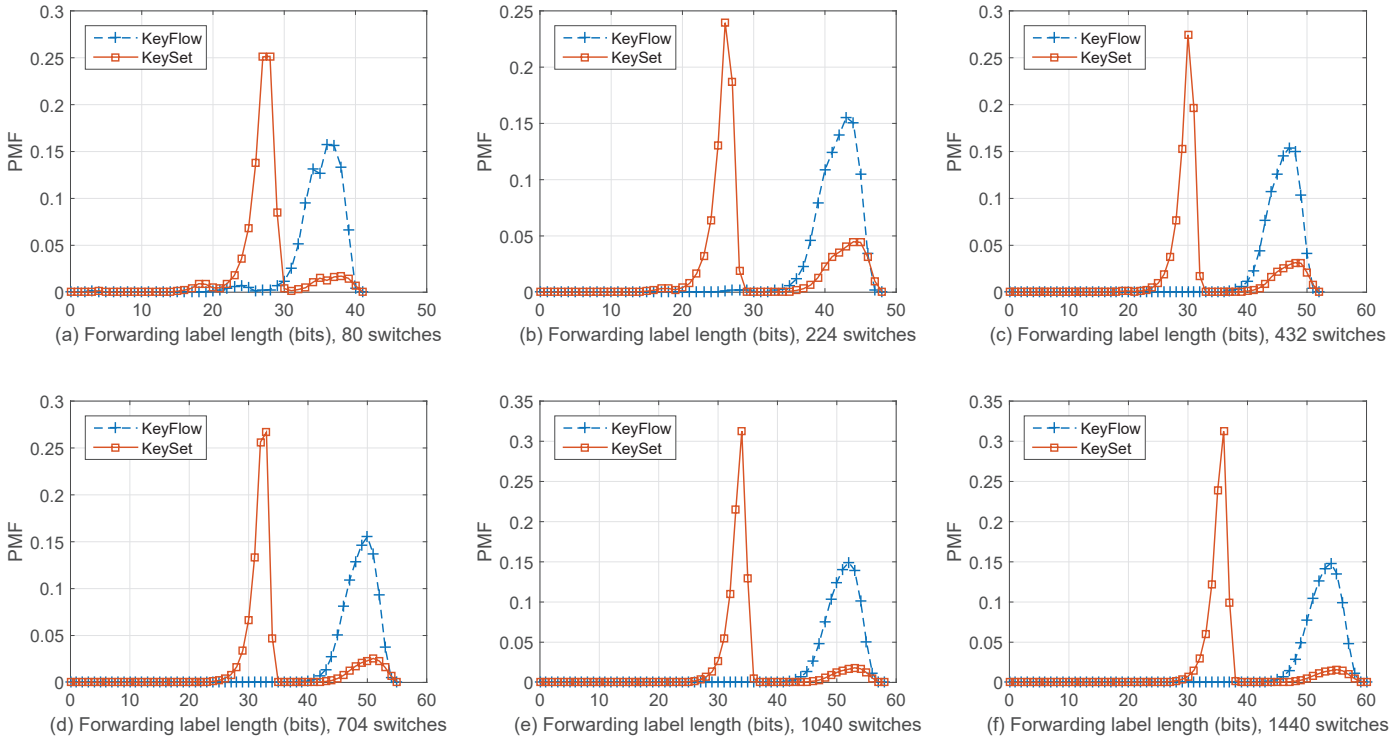

Fig. 6: The PMF distribution of forwarding label length.

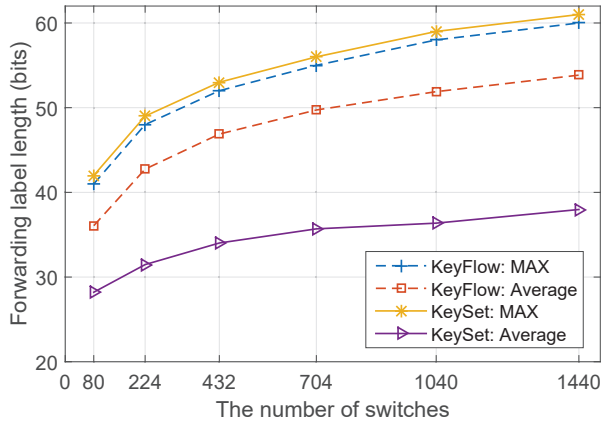

Fig. 7: Impacts of the number of switches in terms of maximum and average values.

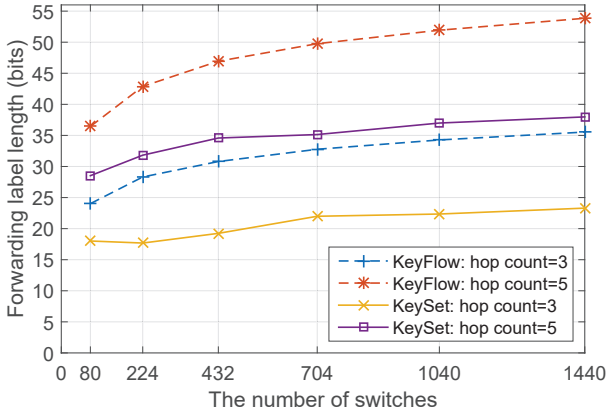

Fig. 8: Impacts of hop count.

on our results, $\alpha = 4, 16, 16, 16, 16, 16$ are recommend for $n = 80, 224, 432, 704, 1040, 1440$, respectively.

B. PMF Distribution of Forwarding Label Length

Fig. 6 illustrates the probability mass function (PMF) distribution of forwarding label length for KeyFlow and KeySet with respect to $n = 80, 224, 432, 704, 1040, 1440$, respectively. We can see that most forwarding labels of KeySet are shorter than that of KeyFlow. Specifically, in a data center with 224 switches, the labels of KeySet are around 10 bits shorter than that of KeyFlow. And the gap further enlarges to 17 bits and 22 bits with 704 switches and 1440 switches, respectively. The results clearly show that KeySet outperforms KeyFlow. Also, the results can be used for specifying β for indicating the length of forwarding labels. For example, in $n = 224$ case (see Fig. 6(b)), β could be set as 0.75 for good, which means that most labels of KeySet are less than 0.75 full length. In $n = 1440$ case (see Fig. 6(f)), $\beta = 0.58$ is a good option. We also observe that most of KeyFlow results are distributed near the maximum length (the worst-case), leading to large label length for most labels. Whereas, in KeySet, most labels are distributed far away from the worst-case scenario. This makes KeySet better than KeyFlow and also enables to specify β to further reduce label length.

C. Impacts of the Number of Switches

Fig. 7 depicts the comparison results of the forwarding label length between KeySet and KeyFlow in terms of the number

of switches n. The results of both the maximum forwarding label length (the worst-case scenario) and the average length are considered. For the maximum length, KeySet is one bit longer than KeyFlow. The reason is that KeySet uses one bit Index to select prime sets[1]. Based on the results shown in Fig. 6, the worst-case seldom happens in KeySet case, whereas, most results of KeyFlow are distributed closely to the worst-case. For the average length, the results show that KeySet outperforms KeyFlow with large gap (around 25%). Also, the gap increases as n becomes large, and the curve of KeySet is more smooth than that of KeyFlow as n increases. These mean that KeySet is scalable for large scale network topology and is better than KeyFlow especially for data centers with a large number of switches.

D. Impacts of Hop Count

According to Theorem 1, the CRT value $\mathcal{X}$ increases as the hop count grows. Thus, making the length of $\mathcal{X}$ resilient to the growth of hop count is essential for reducing forwarding label overhead. Fig. 8 demonstrates the comparison results of forwarding label length in terms of hop counts. We can see that the forwarding label length of KeySet is much shorter than that of KeyFlow. We also observe that the curves of KeySet grow slower than that of KeyFlow as n increases, meaning that KeySet is more resilient to the growth of n. Moreover, as the hop count increases from 3 to 5[2], the gap between KeySet and KeyFlow enlarges significantly, which demonstrates the advantages of KeySet.

VI. Conclusions

In this paper, we have proposed KeySet, a flowtable-free and fault-tolerant routing scheme for data center networks. KeySet reduces core network latency and simplifies switch hardware requirements. Instead of looking up flowtables in forwarding switches, KeySet enables constant-time switching in the way that switches only need to do very simple modular arithmetics to find out output ports. Also, KeySet takes advantage of two or more sets of modular values at switches and enables the controllers to adaptively select a good set for the switching purpose. We have validated KeySet through extensive simulations and compared it against KeyFlow. The simulation results show that KeySet outperforms KeyFlow by at least 25% in terms of the length of forwarding label length. In general network topology, multiple sets have potential to further reduce forwarding label length, which refer to as our future work.

References

[1] M. Martinello, M. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "KeyFlow: a prototype for evolving SDN toward core network fabrics," *IEEE Network*, vol. 28, no. 2, pp. 12 – 19, 2014.

[2] 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 Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[3] (2012) SDN system performance. [Online]. Available: http://www.pica8.com/pica8-deep-dive/sdn-system-performance/

[4] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *Proc. IEEE ICNP*, 2003.

[5] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 488–500, Apr. 2012.

[6] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE INFOCOM*, 2013.

[7] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. ACM SOSR*, 2016.

[8] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable label-switching for commodity ethernet," in *Proc. ACM HotSDN*, 2014.

[9] A. Schwabe and H. Karl, "Using MAC addresses as efficient routing labels in data centers," in *Proc. ACM HotSDN*, 2014.

[10] H. Wessing, H. Christiansen, T. Fjelde, and L. Dittmann, "Novel scheme for packet forwarding without header modifications in optical networks," *IEEE/OSA J. Lightw. Technol.*, vol. 20, no. 8, pp. 1277 – 1283, Aug. 2002.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[12] G. Berger, *Facebook fabric networking deconstructed*, Nov. 2014. [Online]. Available: http://firstclassfunc.com/facebook-fabric-networking

[1] We assume that KeyFlow also use one bit Len to indicate forwarding label length.

[2] Hop count = 5 is the maximum hop count for the fat-tree topology with 1440 nodes.