JumpFlow: Reducing Flow Table Usage in Software-Defined Networks

Zehua Guo[a], Yang Xu[b,*], Marco Cello[c], Junjie Zhang[b], Zicheng Wang[b], Mingjian Liu[b], H. Jonathan Chao[b]

[a]*School of Electronics and Information, Northwestern Polytechnical University, Xi'an, China, 710072*
[b]*Department of Electrical and Computer Engineering, New York University Polytechnic School of Engineering, New York, USA, 11201*
[c]*Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture, University of Genoa, Genova, Italy, 16145*

Abstract

The forwarding scheme in Software-Defined Networking (SDN) is usually coupled with flow table management. To reduce the redundancy in the flow tables of OpenFlow switches, some recent studies propose forwarding flows using stacked MPLS labels, in which each label in the stack indicates the forwarding decision at one hop of the forwarding route. However, using multiple MPLS labels in each packet introduces significant transmission overhead, especially in networks with large diameters.

In this paper, we propose JumpFlow, a forwarding scheme that achieves low and balanced flow table usage in an SDN by properly and reactively placing flow entries on switches. To reduce the transmission overhead, JumpFlow uses the available VLAN identifier (VID) in the packet header to carry routing information. Constrained by the limited space of the VID, a flow's complete routing information must be divided into several sections and loaded separately at different switches on the flow's forwarding route. To achieve low and balanced flow table usage, we formulate and solve the reactive flow entry placement problem. We evaluate JumpFlow against the per-hop configuration-based forwarding of OpenFlow for both unicast and multicast scenarios in a real network topology with different traffic patterns. For the unicast scenario with different new flow arrival rates, JumpFlow postpones the time when the first flow rejection occurs, reduces the flow rejection percentage by 37.06%, and reduces the control messages for route configuration by 53.52% on average. For the multicast scenario with a high new multicast group arrival rate, JumpFlow increases the ratio of accepted multicast groups by 83.90%, and reduces the ratio of average control messages for a multicast group configuration by 32.68%.

Keywords: Software-defined networking; Forwarding; Flow table management

1. Introduction

In recent years, as network applications have experienced rapid growth, Software-Defined Networking (SDN) has attracted significant attention from academia and industry [1]. With the separation of the control plane from the data plane and the logically centralized control, SDN enables flexible network control and thus provides tremendous opportunities for network innovations, such as traffic engineering [2, 3, 4], load balancing [5, 6], power saving [7, 8, 9, 10]. One key enabler of SDN is OpenFlow, which uses a flow entry-based abstraction to enable multiple network functions, such as layer 2 forwarding, layer 3 routing, and layer 2-4 admission control. Owing to the consideration of power consumption and manufacturing cost, OpenFlow switches are usually equipped with limited flow table space, which may become insufficient when there is a large number of flow entries demanded by the network [11, 12, 13]. Thus, designing a scheme that can

*Corresponding author
Email addresses: guolizihao@hotmail.com (Zehua Guo), yang@nyu.edu (Yang Xu), marco.cello@unige.it (Marco Cello), jz733@nyu.edu (Junjie Zhang), zw640@nyu.edu (Zicheng Wang), mingjian.liu1@gmail.com (Mingjian Liu), chao@nyu.edu (H. Jonathan Chao)

efficiently manage flow tables is a critical problem for practically deploying and applying SDN technology in large-scale production and data center networks.

In SDNs, flow entries can be placed in flow tables either proactively, during the network initialization, or reactively, when flows enter the network for the first time. Whereas there are some proactive flow entry placement schemes that address the problem of limited flow table space [11, 14, 15, 16], less effort has been devoted to reactive flow entry placement, which is also an important part of flow table management. On the one hand, proactively placed and reactively placed flow entries are used for different applications: proactively placed flow entries [11, 14, 15, 16] are usually used for applications that require bandwidth and/or delay guarantees (e.g., virtual private network and enterprise applications), whereas reactively placed flow entries are employed to provide best-effort service [17], such as on-demand applications (e.g., online load balancing). On the other hand, even for the same application, reactive flow entry placement provides a better chance to improve the network performance than proactive flow entry placement: as new flows enter the network, they can be directed to less congested routes or forwarded to less loaded servers based on the current network status [6].

The flow table management scheme is usually coupled with the forwarding scheme. Following the basic OpenFlow principle, the per-hop configuration-based forwarding scheme (denoted as OpenFlow) uses multiple flow entries along the route to forward a single flow and thus introduces huge redundancy in flow tables (see Section 2.1). To reduce the redundancy in flow tables, [18, 19, 20, 21] propose an MPLS label-based forwarding scheme in which switches forward packets based on multiple MPLS labels that carry the routing information (i.e., the forwarding port number of each switch on the route, similar to that of source routing). With the MPLS label-based forwarding scheme, the controller must only reactively install one flow entry in the ingress switch to encapsulate MPLS labels for each flow. Unfortunately, encapsulating the routing information in MPLS labels introduces extra transmission overhead, especially for small packets and networks with large diameters, resulting in bandwidth waste (see Section 2.2).

In this paper, we propose an efficient forwarding scheme for SDNs named JumpFlow, which makes use of the MPLS label-based forwarding scheme concept and prevents bandwidth waste. JumpFlow uses the available VLAN identifier (VID) of the packet header to carry the routing information. Considering that the VID has limited space and can carry only limited routing information, the controller must to divide a flow's complete routing information into several sections and load them on a few selected *contact switches* on the flow's forwarding route. We formulate the reactive flow entry placement problem in JumpFlow with the objective of achieving low and balanced flow table usage by properly selecting the placement and the number of contact switches. Note that JumpFlow can be applied to reactively place both exact-matched and wildcard-matched flow entries. For ease of understanding the proposed JumpFlow scheme, we present only the case of reactive exact-matched flow entry placement, and in the rest of paper, the reactive exact-matched flow entry placement will be called reactive flow entry placement. We evaluate JumpFlow against the per-hop configuration-based forwarding of OpenFlow for unicast and multicast scenarios in a real network topology with different traffic patterns. For the unicast scenario with different new flow arrival rates, JumpFlow postpones the time when the first flow rejection occurs, reduces the flow rejection percentage by 37.06%, and reduces the control messages for route configuration by 53.52% on average. For the multicast scenario with a high new multicast group arrival rate, JumpFlow increases the ratio of accepted multicast groups by 83.90%, and reduces the ratio of average control messages for a multicast group configuration by 32.68%.

The major contributions of this paper are listed as follows:

1. We summarize the existing per-hop configuration-based forwarding of OpenFlow and the MPLS label-based forwarding and analyze their advantages and disadvantages. Based on the analysis, we propose JumpFlow, a forwarding scheme that eliminates bandwidth waste in the MPLS label-based forwarding scheme by embedding the routing information in the unused fields (e.g., VID) of the packet header.
2. We include a piecewise function used in Internet traffic engineering in our cost function to achieve joint low and balanced flow table usage and then formulate and solve the reactive flow entry placement problem by properly selecting the placement and the number of contact switches for each flow.
3. We evaluate JumpFlow in a real network topology with different traffic patterns. Simulation results demonstrate that JumpFlow achieves better performance for unicast and multicast scenarios than

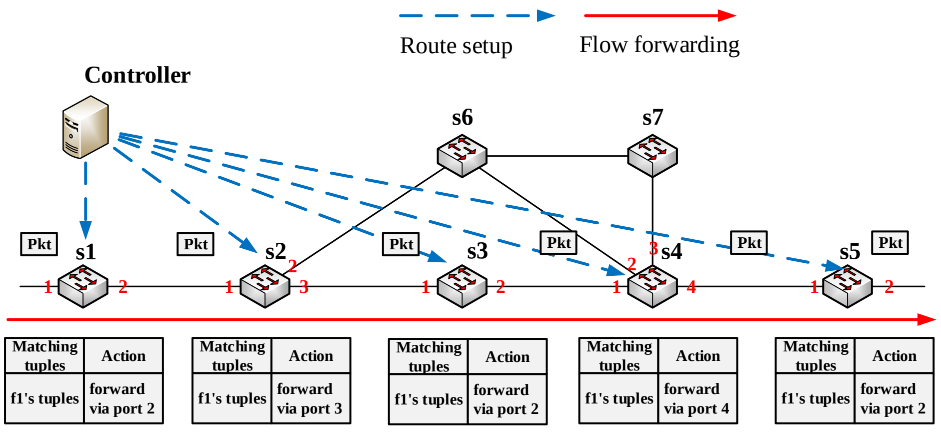


Figure 1: An example of using per-hop configuration-based forwarding of OpenFlow. The controller configures the route of a new flow $f1$ by installing a flow entry in each switch on the flow's route. Packets of flow $f1$ are forwarded by matching flow entries in corresponding switches.

baseline schemes.

The rest of the paper is organized as follows. Section 2 introduces the problem background and discusses the motivation for this paper. Section 3 presents JumpFlow's framework. Section 4 formally formulates and solves the reactive flow entry placement problem. Section 5 presents the evaluation strategy and compares JumpFlow to baseline schemes. Section 6 differentiates JumpFlow from some related schemes and discusses some concerns for JumpFlow. Section 7 reviews the related work. Section 8 concludes the paper.

2. Background and Motivation

2.1. Per-hop configuration-based forwarding of OpenFlow

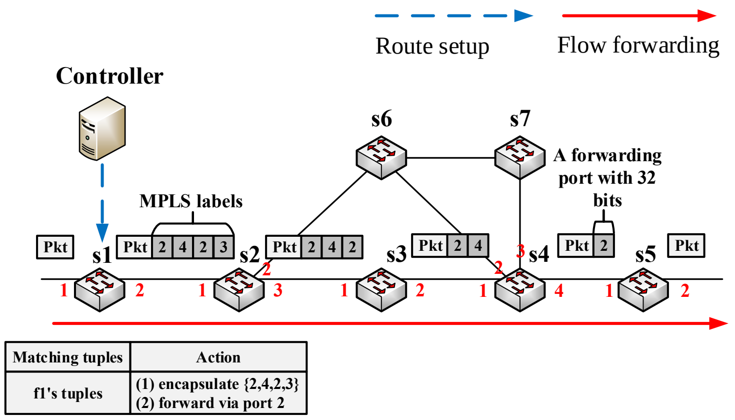
OpenFlow provides a per-hop configuration-based forwarding, which works as follows: when the first packet of a new flow enters the network from an edge switch, the switch has no specific entry in its flow table for the flow and sends a flow setup request to the controller, which will calculate a route for the flow and install flow entries coupled with specific actions in the flow tables of the switches on the route.

The above operation introduces huge redundancy in flow tables as illustrated in Figure 1. When a flow traverses five switches on its route ($s1 \to s2 \to s3 \to s4 \to s5$), the controller must install a flow entry in each of the five switches to configure such a route, totally consuming five flow entries to forward a single flow. Additionally, each packet must repeat the matching operation and the corresponding actions for each switch on its forwarding route, leading to an increase in packet latency.

2.2. MPLS label-based forwarding

Recent studies propose forwarding packets based on their multiple MPLS labels encapsulated with their routing information [20]. When a packet enters its ingress edge switch, it is encapsulated with multiple MPLS labels indicating the forwarding port numbers of other switches on its route, as in IP-in-IP encapsulation. Intermediate switches will use the MPLS labels to forward the packet to the next hop and delete the used header until the packet is delivered to its final destination. Figure 2 shows the configuration of the same route in Figure 1 with MPLS label-based forwarding. When a packet enters its ingress edge switch $s1$, it is encapsulated with four MPLS labels, {2,4,2,3}, each number indicating a forwarding port of the switch on its route. When the packet arrives at switch $s2$, it is forwarded, based on the packet's first MPLS label, to port 3 towards its next hop. Other intermediate switches work similarly to deliver the packet in the network.

In the above scheme, the controller installs only one flow entry in the ingress switch of a flow, and the matching action is performed only once. Thus, both the redundancy of flow tables and the latency of the



Matching tuples	Action
f1's tuples	(1) encapsulate {2,4,2,3} (2) forward via port 2

Figure 2: An example of using MPLS label-based forwarding. The controller configures the route of a new flow $f1$ by installing a flow entry only at the flow's ingress switch to encapsulate the MPLS labels (i.e., a sequence of forwarding port numbers of switches on the route). Packets of flow $f1$ are forwarded by matching/deleting MPLS labels.

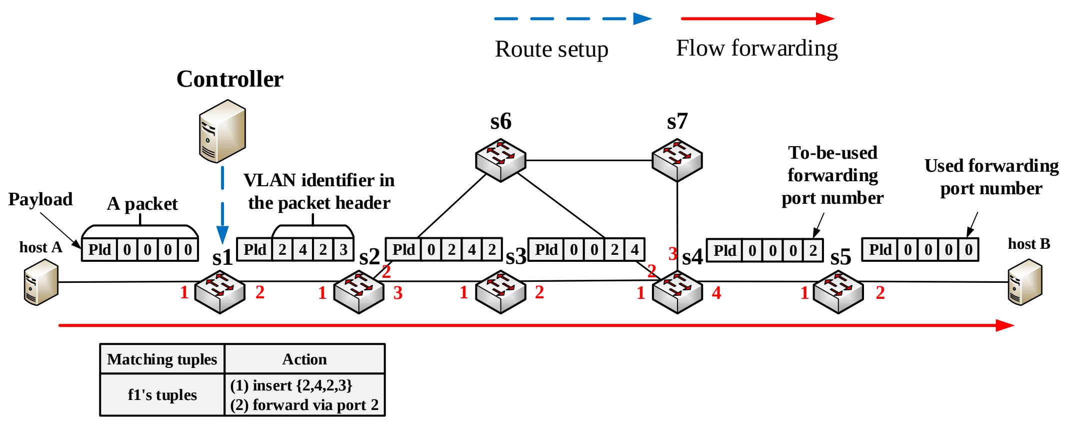
flow matching operation are significantly reduced, compared to the per-hop configuration-based forwarding that uses five flow entries and matching operations. However, the packet header encapsulation may increase the bandwidth usage to delivery packets. For a switch with 16 ports, the encoding of one forwarding port number requires 4 bits. However, one MPLS label consumes 32 bits, which is much more than that required to express the number of forwarding ports on a typical switch [20]. Thus, a 4-hop MPLS label requires 128 bits, a large waste for small packets. For a route with 10 hops, the bandwidth waste will be more severe.

2.3. Available information field in the packet header

The overhead introduced in the MPLS label-based forwarding can be avoided by making use of the available information fields in the packet header to carry routing information. For example, for a switch that supports the standard IEEE 802.1Q, we can insert routing information into the 12-bit VLAN Identifier (VID) [22]. Figure 3 shows an example in which packets are forwarded using the 12-bit VID to carry the routing information. In the figure, each switch has eight ports, and each forwarding port number consumes three bits. The ingress edge switch $s1$ inserts four forwarding port numbers into the VID. Other switches forward the packet based on the first number in the VID (i.e., the rightmost three bits of the VID) and right circular shift the VID by three bits. The details will be explained in Section 3.

2.4. Limited space of the VID

However, the above scheme is constrained by the limited space of the information field. In most cases, a switch has at most 16 ports, and 4 bits are needed for each hop. Thus, a 12-bit VID can store information for only three hops, which is not enough to carry the complete routing information. Owing to the hop limit, a packet's complete routing information must be divided into several sections and loaded at different switches on its forwarding route. The switch that loads a section of routing information for packets of a flow is called the *contact switch* of the flow. Because one loading action requires one flow entry, an intuitive scheme, called MaxHop forwarding, minimizes the number of contact switches by loading as much routing information as possible at each contact switch. Figure 4 shows an example of using the MaxHop forwarding scheme with the VID. However, without considering flow table usage, the intuitive scheme may lead to imbalanced flow table usage or, even worse, flow table overflow (see Section 4.1.2), which not only increases the burden of the controller to repeatedly update flow tables but also degrades the TCP performance by disrupting the existing flows [17]. Actually, besides the given selection of two contact switches (i.e., {$s1$, $s5$}) in Figure 4, there are three other selections: {$s1$, $s2$}, {$s1$, $s3$}, and {$s1$, $s4$} (see Figure 7). Therefore, we must design a scheme that jointly considers the placement and the number of contact switches and the balancing performance of flow tables. For simplicity, in the rest of the paper, the available information field of a packet header will be simply called a packet header.



Matching tuples	Action
f1's tuples	(1) insert {2,4,2,3} (2) forward via port 2

Figure 3: An example of packets being forwarded using the 12-bit VID to carry the routing information in a route composed of 8-port switches. The controller installs a flow entry to switch $s1$ to configure flow $f1$'s route. When flow $f1$'s packet reaches its ingress edge switch $s1$, it is equipped with its complete routing information {2,4,2,3} and forwarded via port 2 of switch $s1$. At switch $s2$, the packet is forwarded via port 3 based on its VID, and the VID is right circular shifted by three bits. Similarly, each switch forwards a packet based on the first port number in the VID and right circular shifts the VID by three bits until the packet arrives at its destination.

3. Overview of JumpFlow

JumpFlow consists of two modules: (1) flow entry placement and (2) flow forwarding. Figure 5 shows the structure of JumpFlow. In the figure, the first module resides in the controller whereas the second module is distributed in the switches. In the first module, the controller places flow entries in the switches by solving the optimization problem of reactive flow entry placement, which achieves joint low and balanced flow table usage. The optimization problem will be detailed in Section 4.

In the second module, the packet is forwarded based on the flow entries placed by the first module. Figure 6 shows the working process of the flow forwarding module in a switch. When a switch receives a packet, it first detects the VID in the packet header that carries the routing information, and it obtains the first port number used to find the corresponding entry in the shortcut table. If the first port number is zero, a lookup will be further conducted in the flow table to find the corresponding entry. The packet will be then processed based on the actions associated with the corresponding entry in the shortcut table or flow table.

4. Reactive Flow Entry Placement

The goal of reactive flow entry placement is to achieve low and balanced flow table usage with proper flow entry placement in the switches of a specific route[1].

4.1. Modeling of constrains and the cost function

4.1.1. Constraint of flow table usage

In an SDN, assume that route r from a source switch to a destination switch is composed of R switches. The flow table usage of the switches belonging to route r can be expressed through row vector[2] $\mathbf{U} = [u_1, \ldots, u_R]$, where u_j $(1 \leq j \leq R)$ is the flow table usage at switch s_j. We also express a specific flow entry placement on route r through vector $\mathbf{C} = [c_1, \ldots, c_R]$, where $c_j = 1$ if switch s_j is selected as a contact switch. Thus, the new flow table usage of switch s_j on route r is denoted as $u_j + c_j$ and should not exceed switch s_j's flow table capacity u_j^{MAX}; that is:

$$u_j + c_j \leq u_j^{MAX} \tag{1}$$

[1]In this paper, we use flow entry placement and contact switch selection interchangeably.
[2]Hereafter called "vector".

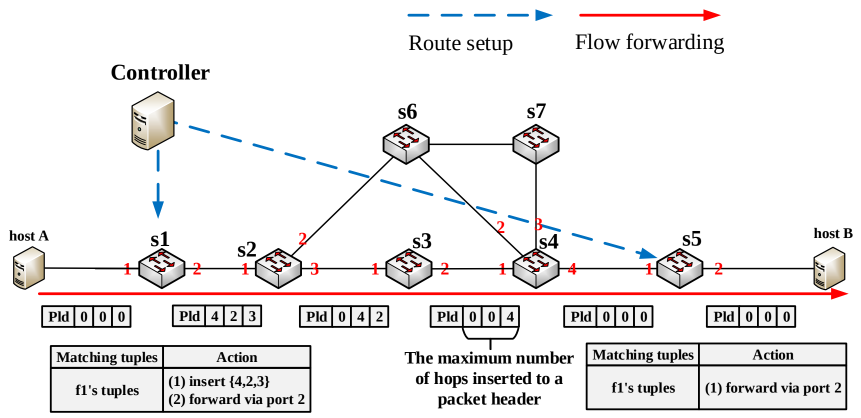


Figure 4: MaxHop forwarding with 16-port switches and the 12-bit VID. Because the VID cannot carry the complete routing information, the controller installs two flow entries in switches $s1$ and $s5$ to configure flow $f1$'s route. When flow $f1$'s packet reaches its ingress edge switch $s1$, it is equipped with its route information {4,2,3} and forwarded via port 2 of switch $s1$. At switches $s2$, $s3$, and $s4$, the packet is forwarded based on the routing information, and the corresponding port number in the packet header is set to 0. When the packet arrives at switch $s5$, the switch forwards the packet based on its flow table action.

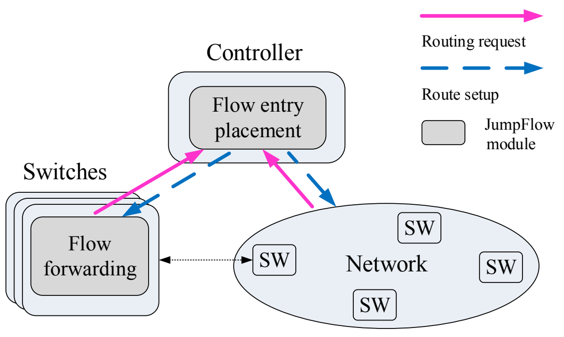


Figure 5: The structure of JumpFlow.

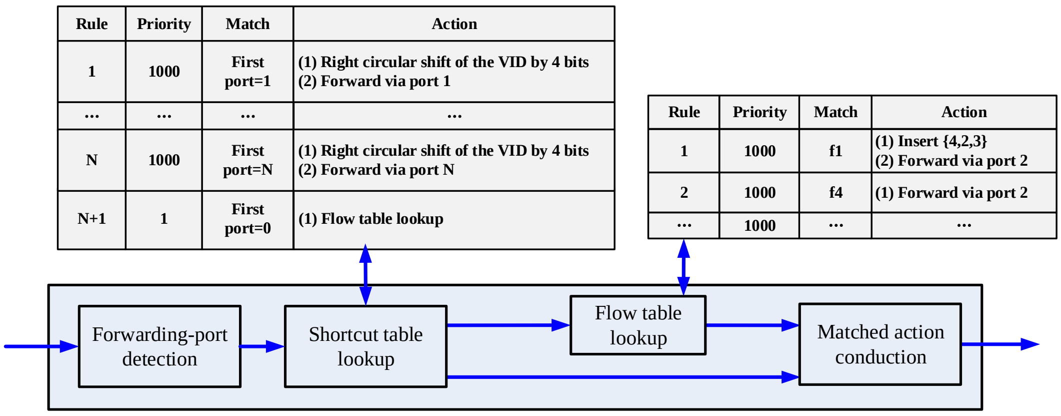
For simplicity, we assume that the flow table capacity of each switch is u^{MAX}, and the vector expression of the flow table capacities of switches on route r is $\mathbf{U^{MAX}} = [u_1^{MAX}, \ldots, u_R^{MAX}] = [u^{MAX}, \ldots, u^{MAX}]$. Therefore, with placement $\mathbf{C}$, the new flow table usage of switches on route r are $\mathbf{U} + \mathbf{C}$ and should not exceed $\mathbf{U^{MAX}}$, the flow table capacities of the switches:

$$\mathbf{U} + \mathbf{C} \leq \mathbf{U^{MAX}} \tag{2}$$

Figure 7(a) shows the flow table usage with three different contact switch selections. In the figure, the network consists of 8 switches $s1 - s8$, and a flow from host A to host B is forwarded on its shortest route $r = \{s1 \rightarrow s2 \rightarrow s3 \rightarrow s4 \rightarrow s5\}$. Figure 8(a) shows the flow table usage expressed as a vector of the switches on route r in Figure 7(a). For example, with placement $\mathbf{C}'''$, switches $s1$ and $s2$ are selected as the contact switches, and the new flow table usage is $\mathbf{U} + \mathbf{C}'' = [5\ 9\ 3\ 6\ 10\ 0]$.

4.1.2. Constraint of contact switches

We use $\mathcal{K}_R$ to denote the set of all possible placements for route r with R hops. Each placement $\mathbf{C}$ must satisfy two constraints: (1) $c_1 = 1$, the ingress edge switch of route r is always a contact switch because it must insert each packet with the first routing information; (2) there are no more than H^{MAX} consecutive zeros in $\mathbf{C}$, where H^{MAX} denotes the maximum port number of switches that can be inserted in a packet header. Figure 7 shows four possible placements for route r, and Figure 8(b) shows the corresponding vector expression of four possible placements for route r.



Rule	Priority	Match	Action
1	1000	First port=1	(1) Right circular shift of the VID by 4 bits (2) Forward via port 1
...	...	...	...
N	1000	First port=N	(1) Right circular shift of the VID by 4 bits (2) Forward via port N
N+1	1	First port=0	(1) Flow table lookup

Rule	Priority	Match	Action
1	1000	f1	(1) Insert {4,2,3} (2) Forward via port 2
2	1000	f4	(1) Forward via port 2
...	1000	...	...

Figure 6: The working process of the flow forwarding module in a switch.

4.1.3. Cost function

Because the goal of the problem is to achieve low and balanced flow table usage, the cost function should consider the balancing performance and usage of flow tables. Typically, the absolute error function is usually used to evaluate the balancing performance. A smaller value indicates a better balancing performance. However, with the absolute error function, a good balancing performance of flow tables may be achieved by implicitly increasing the flow table usage. Figure 9(a) shows the absolute error function of four placements in Figure 7. In the figure, the balancing performance of $C^{''''}$ is better than $C^{'''}$, but, as shown in Figure 9(b), $C^{''''}$ uses one more contact switch than that of $C^{'''}$. Thus, we formulate a new cost function that achieves joint low and balanced flow table usage:

$$\sum_{j=1}^{R} f\left(\frac{u_j + c_j}{u_{MAX}}\right) \tag{3}$$

where

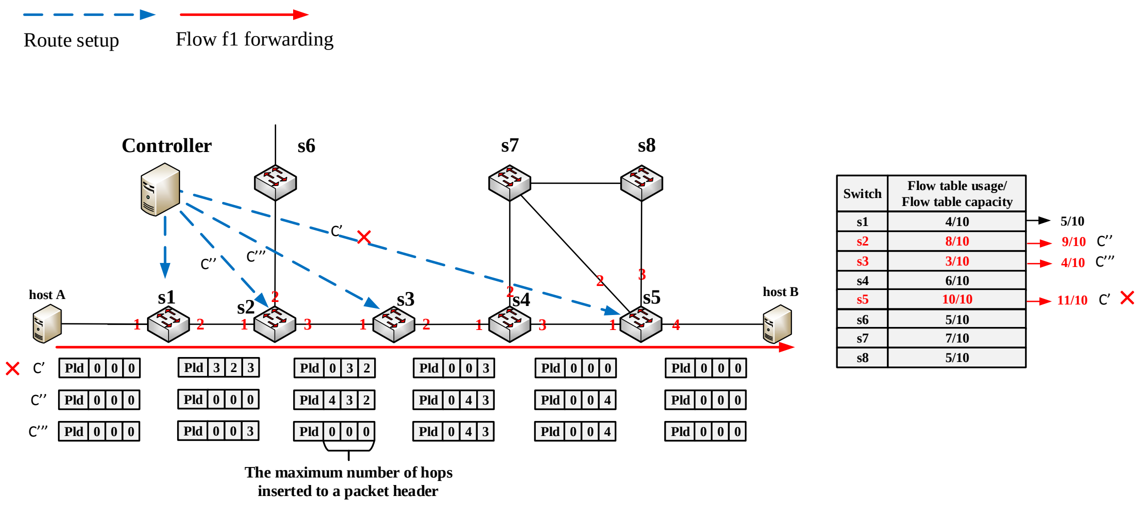
$$f(x) = \begin{cases} x & 0 \le x < 1/3 \\ 3x & 1/3 \le x < 2/3 \\ 10x & 2/3 \le x < 9/10 \\ 70x & 9/10 \le x < 1 \\ 500x & x = 1 \end{cases} \tag{4}$$

The above cost function expresses the impact of the flow table usage increase introduced by placement **C**. Equation (4) is a piecewise cost function used to simulate the nonlinear impact of a link load on the network congestion in Internet traffic engineering [23]. The higher the link load usage, the higher the chance a link would experience congestion. The idea behind Equation (4) is to send flows over a link with a small load utilization. As the utilization approaches 100%, the cost penalty increases significantly because the congested link is more sensitive to bursts. Similarly, the higher the flow table usage, the higher the possibility that a flow table would experience overflow, which significantly degrades the network performance and the controller's workload [17]. Therefore, we includes Equation (4) in our cost function to simulate the nonlinear impact of the flow table usage increase on the network performance.

The cost function can be expressed in the vector form below:

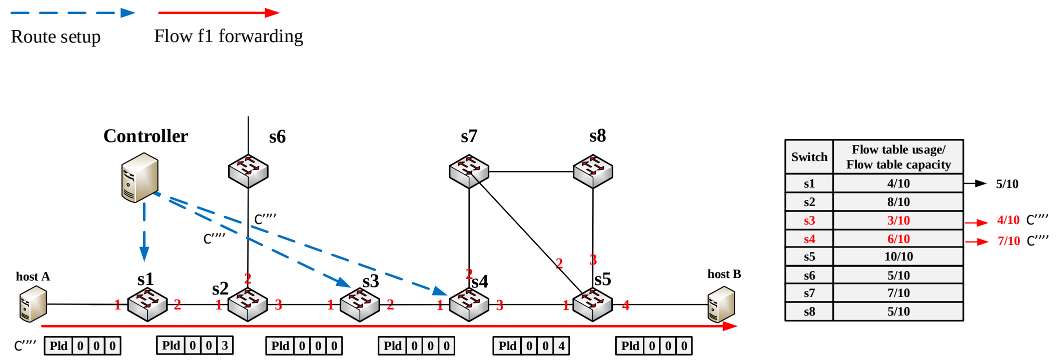
$$\left|f\left(\frac{\mathbf{U} + \mathbf{C}}{u_{MAX}}\right)\right| \tag{5}$$

where $|\mathbf{X}|$ denotes the sum of elements in vector $\mathbf{X}$.



Switch	Flow table usage/ Flow table capacity	
s1	4/10	5/10
s2	8/10	9/10 C''
s3	3/10	4/10 C'''
s4	6/10	
s5	10/10	11/10 C' ✗
s6	5/10	
s7	7/10	
s8	5/10	

(a) Selecting two contact switches with MaxHop scheme (C' and C") or considering the flow table usage (C"').



Switch	Flow table usage/ Flow table capacity	
s1	4/10	5/10
s2	8/10	
s3	3/10	4/10 C''''
s4	6/10	7/10 C''''
s5	10/10	
s6	5/10	
s7	7/10	
s8	5/10	

(b) Selecting three contact switches considering the flow table usage.

Figure 7: The flow table usage with different schemes that select contact switches for a route composed of five switches with $H^{MAX} = 3$ (i.e., the maximum number of bits that can be inserted in a packet header to represent a hop routing information equals three).

Figure 10 shows the cost function values of the three placements. We can see that placement $\mathbf{C}^{'''}$ has the lowest value, which trades off the balancing performance and usage of the flow tables.

4.2. Problem modeling

Given a route that consists of R switches, the goal is to achieve low and balanced flow table usage by properly selecting the *number* and *placement* of contact switches on the route. Therefore, the problem is formulated below:

Reactive Flow Entry Placement Problem:

$$\min_{\mathbf{C} \in \mathcal{K}_R} |f(\frac{\mathbf{U}+\mathbf{C}}{u_{MAX}})| \tag{6}$$

subject to

$$\mathbf{U}+\mathbf{C} \leq \mathbf{U^{MAX}} \tag{7}$$

$$\mathbf{U} = \begin{bmatrix}4 & 8 & 3 & 6 & 10\end{bmatrix}$$

(a) The current flow table usage of switches on route r in Figure 7

$$\begin{aligned}
\mathbf{C}^{'} &= \begin{bmatrix}1 & 0 & 0 & 0 & 1\end{bmatrix}\\
\mathbf{C}^{''} &= \begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix}\\
\mathbf{C}^{'''} &= \begin{bmatrix}1 & 0 & 1 & 0 & 0\end{bmatrix}\\
\mathbf{C}^{''''} &= \begin{bmatrix}1 & 0 & 1 & 1 & 0\end{bmatrix}
\end{aligned}$$

(b) Four placements on route r in Figure 7

$$\begin{aligned}
\mathbf{U}+\mathbf{C}^{'} &= \begin{bmatrix}5 & 8 & 3 & 6 & 11\end{bmatrix}\\
\mathbf{U}+\mathbf{C}^{''} &= \begin{bmatrix}5 & 9 & 3 & 6 & 10\end{bmatrix}\\
\mathbf{U}+\mathbf{C}^{'''} &= \begin{bmatrix}5 & 8 & 4 & 6 & 10\end{bmatrix}\\
\mathbf{U}+\mathbf{C}^{''''} &= \begin{bmatrix}5 & 8 & 4 & 7 & 10\end{bmatrix}
\end{aligned}$$

(c) The new flow table usage with four placements on route r in Figure 7

Figure 8: The vector expression of Figure 7.

Equation (7) ensures that the flow table usage of each switch on route r does not exceed its capacity. $\mathbf{C} \in \mathcal{K}_R$ states that the placement can be only selected from $\mathcal{K}_R$, the set of all possible placements of route r that satisfy two constraints of a contact switch.

4.3. Solution

To obtain the solution, we should first generate $\mathcal{K}_R$ for route r and then find the optimal placement from $\mathcal{K}_R$ by solving the above integer programming problem with existing solvers (e.g., CPLEX). Thus, the complexity of the solution comes from the method to find $\mathcal{K}_R$ and the size of $\mathcal{K}_R$. Given H^{MAX}, a possible method is to dynamically calculate $\mathcal{K}_R$ for route r based on its length R. However, the method would overwhelm the controller by high redundancy because routes with the same length R always use the same $\mathcal{K}_R$. Thus, we can precompute different sets $\mathcal{K}_R$, each of which is related to a different route length R, and store them in the controller.

4.3.1. Computing and storing $\mathcal{K}$

By analyzing the length of the shortest route of switch pairs in multiple topologies, we found that in most cases, the length of the shortest route is usually less than or equal to 10. Thus, we need to compute $\mathcal{K}_{10}$ only for the route with 10 hops, $\mathcal{K}_9$ for the route with 9 hops, $\mathcal{K}_8$ for the route with 8 hops, and so forth.

Given route r with R hops, the number of vectors included in $\mathcal{K}_R$ is denoted as $|\mathcal{K}_R|$. Considering that the ingress edge switch on route r is always the contact switch, $|\mathcal{K}_R|$ is bounded by $|\mathcal{K}_R| \le 2^{R-1}$. Because the elements of set $\mathcal{K}_R$ are vectors with R bits, the stored information in the controller is bounded by:

$$|\mathcal{K}_R| \cdot R \le 2^{R-1} \cdot R \;\; bits \tag{8}$$

For a route with R hops, we must compute $\mathcal{K}_{R-1}, \ldots, \mathcal{K}_1$, the bound of the total store capacity is:

$$\sum_{R=1}^{10} (|\mathcal{K}_R| \cdot R) \le \sum_{R=1}^{10} (2^{R-1} \cdot R) \;\; bits \sim 2.2\ KB \tag{9}$$

The storage capacity is acceptable.

$$\begin{aligned}\mathrm{AE}(\mathbf{U}+\mathbf{C}^{''}) &= \begin{bmatrix}11.6\end{bmatrix}\\ \mathrm{AE}(\mathbf{U}+\mathbf{C}^{'''}) &= \begin{bmatrix}9.6\end{bmatrix}\\ \mathrm{AE}(\mathbf{U}+\mathbf{C}^{''''}) &= \begin{bmatrix}9.2\end{bmatrix}\end{aligned}$$

(a) The absolute error of flow table usage of three placements

$$\begin{aligned}|\mathbf{C}^{''}| &= \begin{bmatrix}2\end{bmatrix}\\ |\mathbf{C}^{'''}| &= \begin{bmatrix}2\end{bmatrix}\\ |\mathbf{C}^{''''}| &= \begin{bmatrix}3\end{bmatrix}\end{aligned}$$

(b) The number of contact switches of three placements

Figure 9: The metrics for flow table balancing performance and flow table usage. For vector $\mathbf{X}$ with N elements, function $AE(\mathbf{X})$ is the absolute error of vector $\mathbf{X}$, $AE(\mathbf{X}) = \sum_{i=1}^{N} |x_i - \bar{x}|$, where $\bar{x}$ is the average value of N elements in vector $\mathbf{X}$; function $|\mathbf{X}|$ is the sum of vector $\mathbf{X}$, $|\mathbf{X}| = \sum_{i=1}^{N} x_i$.

$$|f(\frac{\mathbf{U}+\mathbf{C}^{''}}{u_{MAX}})| = 566.6$$

$$|f(\frac{\mathbf{U}+\mathbf{C}^{'''}}{u_{MAX}})| = 512.5$$

$$|f(\frac{\mathbf{U}+\mathbf{C}^{''''}}{u_{MAX}})| = 517.7$$

Figure 10: The cost function values of three placements.

4.4. Multicast scenario

In the above subsection, we mainly focus on the unicast scenario. However, JumpFlow can also reduce the flow table usage for the multicast scenario. To extend JumpFlow to the multicast scenario, we must only transform the flow entry placement problem for the multicast scenario into several independent flow entry placement problems for the unicast scenario.

To configure the forwarding routes in a multicast group (also known as multicast tree), the per-hop configuration scheme of OpenFlow should install one flow entry on each switch on the multicast tree, whereas JumpFlow installs flow entries only on selected contact switches on the multicast tree to load routing information and achieve multicast actions. A multicast action consists of two steps: (1) duplicating the multicast packet, and (2) forwarding packets via the corresponding ports. The multicast action can be achieved by the group table in the OpenFlow switch, and thus switches that perform the multicast action should be selected as contact switches. Based on the above analysis, a multicast tree can be divided into three types of unicast sections: the section from the switch connected with the sender host to a contact switch close to the switch, the section between two close contact switches, and the section from a contact switch to the switch connected with a receiver host. Therefore, the flow entry placement problem for a multicast tree can be translated into several independent flow entry placement problems for unicast sections.

We explain how to use JumpFlow for the multicast scenario with Figure 11. In the figure, a multicast group G consists of one sender, host A, and four receivers, hosts B, C, D and E. Assume that host A sends a multicast packet via route R1 ($s1 \to s2 \to s3$) to switch $s3$. At switch $s3$, the packet is duplicated in two copies that are sent toward switches $s5$ and $s8$ via routes R21 ($s3 \to s4 \to s5$) and R22 ($s3 \to s7 \to s8$), respectively. Similarly, switches $s5$ and $s8$ duplicate the received multicast packet and forward them to the

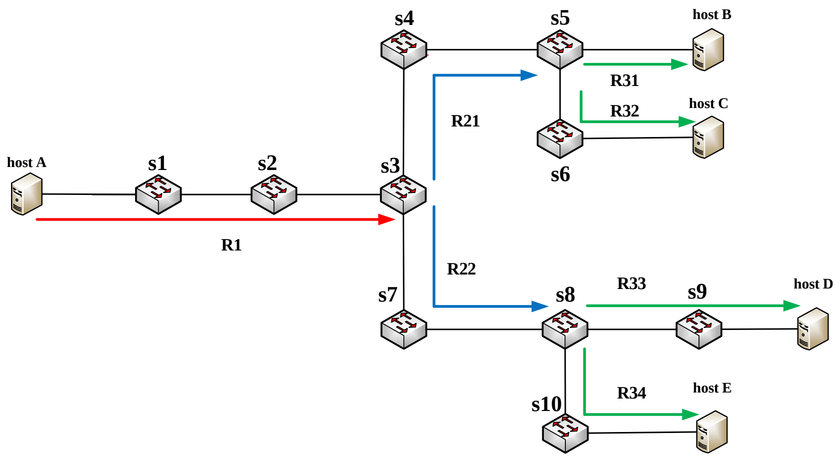


Figure 11: An example of the multicast scenario.

receivers, hosts B, C, D and E, through the corresponding routes. In Figure 11, the forwarding process includes three multicast actions, which are at switches $s3$, $s5$ and $s8$. Thus, we can solve the flow entry placement problem for multicast group G by solving seven flow entry placement problems for unicast sections $R1$, $R21$, $R22$, $R31$, $R32$, $R33$, $R34$.

5. Performance Evaluation

5.1. Comparison schemes

For comparison, we evaluate the performance of the following schemes:

- **Per-hop configuration-based forwarding of OpenFlow (OpenFlow)**: the controller installs flow entries on all switches along a route or a multicast tree.
- **MaxHop forwarding (MaxHop)**: the controller installs flow entries on a route or a multicast tree's contact switches, each of which loads the maximum amount of routing information in the VID of a packet to minimize the flow table usage.
- **JumpFlow**: the controller installs flow entries on a route or a multicast tree's contact switches, each of which loads the flexible amount of routing information in the VID of a packet to achieve low and balanced flow table usage.

5.2. Setup

In the following simulations, we use a real switch-level topology from a major tier-1 ISP in China (anonymized as ISP-China) with a real non-uniform traffic pattern to evaluate the above schemes. The link costs and link capacities are given, and the topology has 512 switches and 2362 directed links. Each switch has 16 ports, which can be encoded by 4 bits, and its flow table capacity is 2,000 flow entries. With the 12-bit VID, one contact switch can load the routing information of three hops at most.

Each flow's arrival rate and duration is driven by a Poisson distribution and uniform distribution, respectively, and each flow's destination is randomly chosen. Each flow is forwarded along its shortest route between its switch pair, and all shortest routes are pre-calculated based on the given link costs. The multicast tree is generated by the shortest route tree. In the simulations, a varying number of new flows/multicast groups are injected into the network in each time slot.

We evaluated JumpFlow's performance against the above baseline schemes. We ran our simulation 100 times with a duration of 200 time slots for each scheme under different new flow/multicast group arrival rates. A new flow/multicast group arrival rate is the average number of new flows/multicast groups entering the network per time slot. For the unicast scenario, three network-wide metrics are shown:

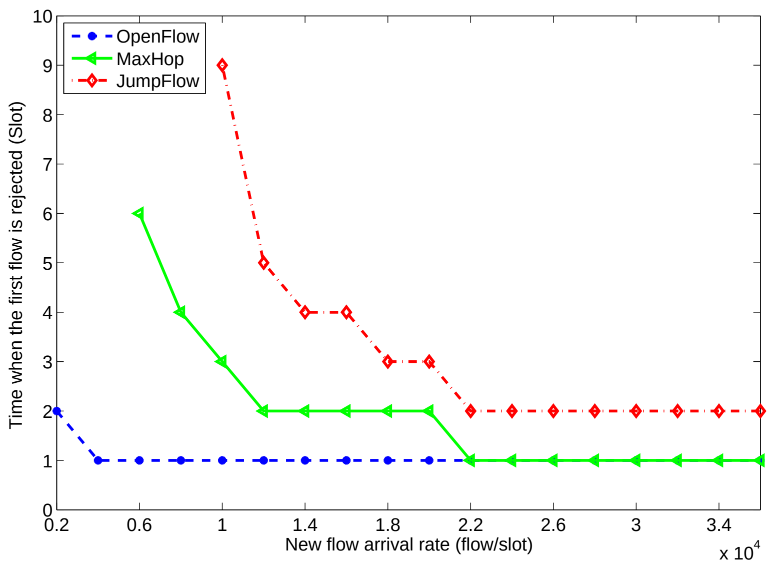


Figure 12: The time when the first flow is rejected under different new flow arrival rates.

- **Time when the first flow is rejected**: when a new flow is initialized, the controller attempts to install the flow entries to configure the forwarding route of this new flow. If the flow table usage at the corresponding switches (for OpenFlow: each switch along the route; for MaxHop and JumpFlow: contact switches along the route) do not exceed their capacities, this new flow is accepted. The first flow rejection occurs when one switch on the flow's route reaches its capacity and becomes the bottleneck switch of the route.

- **Rejected flow percentage**: the flow rejection percentage equals the number of rejected flows divided by the total number of flows generated in the simulation period.

- **Number of control messages for route configuration**: the communication overhead between the controller and switches consists of two parts: (1) routing requests that are sent from switches to the controller when the switches receive new flows without forwarding routes, and (2) control messages that are generated from the controller to install flow entries at the corresponding switches to configure new flows' routes. Under the same traffic pattern, the routing request is the same for each scheme. However, the number of control messages depends on the scheme for route configuration.

We also compare JumpFlow and MaxHop against OpenFlow for the multicast scenario. In the multicast scenario, OpenFlow strictly requires each switch on the multicast tree to be installed with one entry, and JumpFlow/MaxHop installs flow entries on contact switches selected by itself. A multicast group is accepted if the routes of all hosts in the multicast tree can be configured. For the multicast scenario, two evaluated metrics are listed below:

- **Ratio of accepted multicast groups**: the metric equals the number of accepted multicast groups of JumpFlow/MaxHop normalized by the number of accepted multicast groups of OpenFlow.

- **Ratio of average control messages of a multicast group configuration**: we define the average control messages of a multicast group configuration as the total number of control messages for configuring multicast groups' routes divided by the total number of accepted multicast groups. For JumpFlow/MaxHop, its metric equals the average control messages of a multicast group configuration normalized by that of OpenFlow.

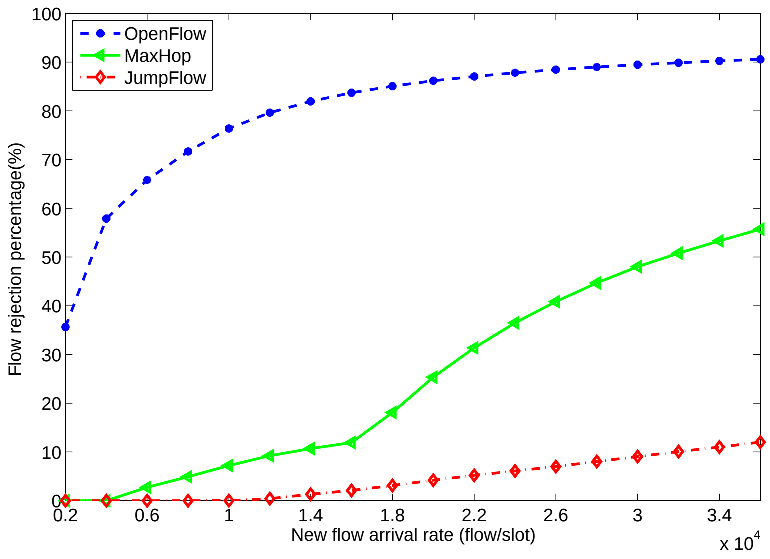


Figure 13: Flow rejection percentage under different new flow arrival rates.

5.3. Results

Figure 12 shows the time when the first flow is rejected with three schemes under different new flow arrival rates. As shown in the figure, OpenFlow rejects flows at the second time slot when the new flow arrival rate is 2,000 flows/slot, and at the first time slot as when the new flow arrival rate exceeds 2,000 flows/slot, which is much earlier than then other two proposed schemes in all rate spans. Using the per-hop configuration-based forwarding of OpenFlow, many more flow entries are consumed to forward flows than in the other two schemes. Thus, some switches could reach their capacities easily and then become the bottleneck switches on many new flows' forwarding routes.

MaxHop outperforms OpenFlow in the rate span from 2,000 to 20,000 flows/slot. MaxHop uses fewer flow entries to forward the same number of flows, and none of the switches reach their capacities at the low new flow arrival rate. When the new flow arrival rate is less than 6,000 flows/slot, MaxHop does not reject any flows. As the new flow arrival rate increases from 6,000 to 20,000 flows/slot, some switches reach their capacities and become bottleneck switches, and then MaxHop rejects some new flows. However, in that rate range, MaxHop's first rejection time is still later than that of OpenFlow.

Among all three schemes, JumpFlow performs best. JumpFlow does not incur flow rejection until the rate reaches 10,000 flows/slot, which is 4,000 flows/slot larger than MaxHop. Even if flow rejection occurs, JumpFlow postpones the time when the first flow rejection occurs at least one slot later than that of MaxHop in most rate spans except the rates of 18,000 and 20,000 flows/slot. With the balanced flow entry placement, JumpFlow significantly postpones the time when switches reach their capacities and become bottleneck switches.

Figure 13 shows the flow rejection percentage with three schemes under different new flow arrival rates. In the figure, in the new flow arrival rate range from 2,000 to 36,000 flows/slot, the flow rejection percentage of OpenFlow increases from 37.06% to 94.19%. As the new flow arrival rate increases, OpenFlow not only increases the number of bottleneck switches in the network but also extends the duration of a switch in the bottleneck status, leading to severe new flow rejection.

In the figure, MaxHop performs much better than OpenFlow. MaxHop does not reject any flows when the new flow arrival rate is less than or equal to 4,000 flows/slot, and its flow rejection percentage remains relatively low before the new flow arrival rate exceeds 16,000 flows/slot. This is because, compared to OpenFlow, MaxHop uses fewer flow entries to forward flows and creates fewer bottleneck switches. However, MaxHop cannot achieve flow table usage balancing, which plays an important role in configuring new flows' routes at a high rate. When the rate exceeds the threshold rate 16,000 flows/slot, MaxHop creates more

Table 1: Reduction of flow rejection percentage.

Scheme / Rate	MaxHop	JumpFlow
2000	37.06	37.06
4000	60.21	60.21
6000	65.61	68.44
8000	69.41	74.51
10000	71.93	79.42
12000	73.22	82.36
14000	74.1	83.86
16000	74.65	84.87
18000	69.63	85.22
20000	60.19	82.19
22000	57.94	85.14
24000	53.37	84.97
26000	49.49	84.69
28000	46.08	84.19
30000	43.11	83.61
32000	40.64	83
34000	38.39	82.41
36000	36.26	81.69

bottleneck switches than it does at the low rate, and its performance degrades significantly. From 16,000 to 36,000 flows/slot, MaxHop's rejection rate increases almost linearly and reaches the highest value of 57.92% at the rate of 36,000 flows/slot.

In the figure, JumpFlow shows good performance. Its flow rejection percentage is zero before the rate of 8,000 flows/slot and does not exceed 12.50% at the rate of 36,000 flows/slot. JumpFlow considers the current flow table usage to dynamically select the number and placement of contact switches for flow entry installment. Thus, in all rate spans, it achieves low and balanced flow table usage and creates fewer bottleneck flow tables than the other two schemes.

Table 1 shows the reduction of flow rejection percentage achieved by MaxHop and JumpFlow over OpenFlow at various new flow arrival rates. In the table, JumpFlow achieves a substantial reduction of flow rejection percentage of 37.06–85.22% for all rates. Figure 14 shows the flow rejection percentage difference between JumpFlow and MaxHop under different new flow arrival rates. In the figure, we can see that as the new flow arrival rate increases, JumpFlow's performance improves significantly.

Figure 15 shows the number of control messages for the route configuration of new flows with three schemes under different numbers of new flows. In the figure, the number of control messages increases nearly linearly as the number of new flows increases from 1,000 to 20,000. OpenFlow performs worst among the three schemes because it conducts per-hop configuration and requires more control messages than the other two schemes. Both MaxHop and JumpFlow install entries on selected switches for new flows' route configuration and thus request fewer control messages than OpenFlow. Compared to OpenFlow, JumpFlow and MaxHop consume 46.48% and 41.82% of control messages on average, respectively, to configure the routes of the same number of flows. JumpFlow requires a slightly more control messages than MaxHop to achieve balanced flow table usage and fewer bottleneck flow tables. However, JumpFlow can accept more

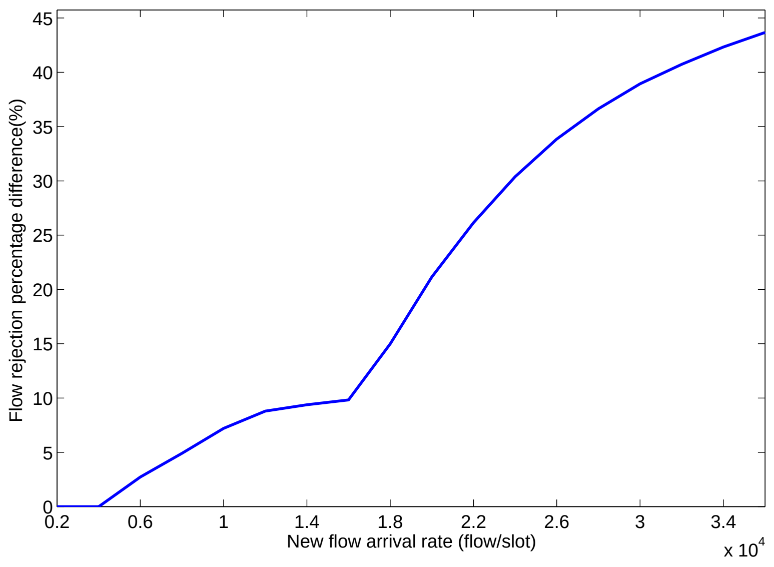


Figure 14: Flow rejection percentage difference between JumpFlow and MaxHop under different new flow arrival rates.

new flows than MaxHop, as shown in Figure 13, and thus its overall performance is much better than that of MaxHop.

We also ran simulations for the multicast scenario. In the multicast simulations, each multicast contains 80 hosts, and arrival rates of the new multicast groups are 250, 1,000 and 2,500 group/slot. Figure 16 and Figure 17 show the simulation results: the ratio of accepted multicast groups and the ratio of average control messages of a multicast group configuration.

In Figure 16 and Figure 17, when the new multicast group arrival rate is 250 group/slot, JumpFlow and MaxHop accept 9.00% and 7.64% more groups than OpenFlow, respectively, and require 14.41% and 13.52% fewer average control messages per multicast group configuration than OpenFlow, respectively. This is because-compared to OpenFlow, which strictly installs one flow entry in each switch on the multicast tree-JumpFlow/MaxHop reduces the flow table usage by installing flow entries only on select switches to conduct routing information loading and multicast actions.

The advantages of JumpFlow and MaxHop can be clearly revealed at the high new multicast group arrival rate. At the rate of 1,000 group/slot, JumpFlow and MaxHop accept 33.42% and 27.20% more groups than OpenFlow, and require 23.62% and 21.98% fewer average control messages per multicast group configuration than OpenFlow, respectively; at the rate of 2,500 group/slot, JumpFlow and MaxHop accept 83.90% and 66.31% more groups than OpenFlow, respectively, and require 32.68% and 31.28% less average control messages per multicast group configuration than OpenFlow, respectively. As the rate increases, the number of bottleneck switches in the network also increases. However, similarly to the unicast scenario, JumpFlow considers the flow table usage to select the contact switches and further reduces the possibility of involving bottleneck switches in the multicast group configuration. Hence, JumpFlow is able to accept more groups than MaxHop and thus performs better than MaxHop.

6. Discussion

6.1. Incorporating JumpFlow with Protocol Oblivious Forwarding and Protocol-Independent Forwarding

Protocol Oblivious Forwarding (POF) [24] and Protocol-Independent Forwarding (PIF) [25] are leading proposals for the SDN forwarding plane.In contrast to OpenFlow, which reqires switches to understand the protocol headers to extract the required bits that are matched with the entries in flow tables, they make the forwarding plane protocol-oblivious. With POF/PIF, a forwarding element (e.g., a switch) does not need to

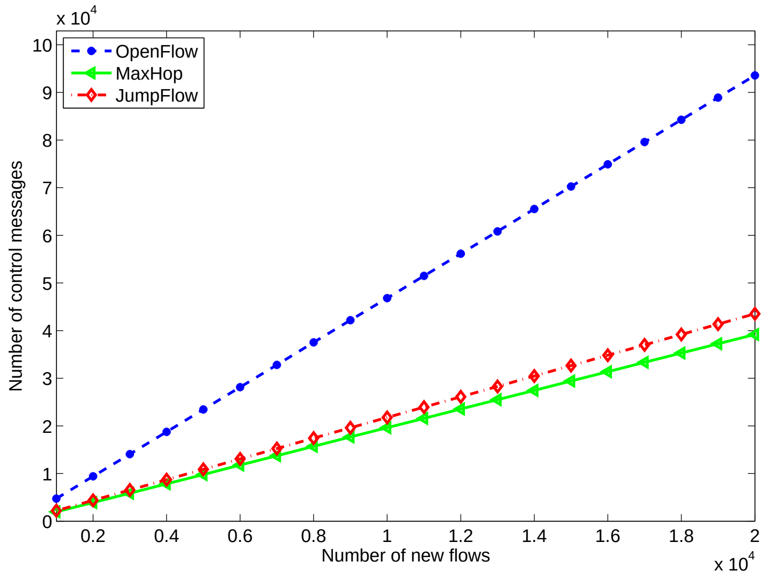


Figure 15: Number of control message for new flows' route configuration.

know the packet format in advance. The controller instructs the switch to extract and assemble the search keys from the customized packet header, to conduct the table lookups, and to execute the corresponding instructions associated with a generic flow instruction set. POF/PIF allows customization of the length of a packet header field that expresses the forwarding port based on the switch with the largest number of ports in the network. For example, in a network consisting of 16-port switches, POF/PIF requires only a 4-bit field to express a forwarding port, which is 87.5% less than that required by MPLS label-based forwarding using 32 bits to represent a hop's information. If a network supports POF/PIF, we can achieve JumpFlow with a 12-bit POF/PIF header instead of the 12-bit VID field in the conventional packet header.

Moreover, incorporating JumpFlow with POF/PIF can further improve network performance. Using JumpFlow in a network supporting POF/PIF, we can customize the number of forwarding ports in a packet header without the constraint of the 12-bit VID. For a flow, the more forwarding ports are loaded each time, the fewer flow entries are needed. We should jointly consider flow table usage, link bandwidth usage and other overhead to select the number of forwarding ports used in the packet header and the number of contact switches used to load routing information.

6.2. Comparison to segment routing

Segment routing follows the basic idea of source routing technology. To forward a packet with segment routing, the source router encodes an ordered list of the labels in the header of a packet, and the other routers must perform MPLS label forwarding based only on the labels in the packet header. However, in contrast to conventional sourcing routing, segment routing uses two types of labels: a node label and an adjacency label. A node label represents a router's loopback prefix and is globally significant within the particular domain; an adjacency label represents a link prefix, which is significant only to the local router. Within the segment routing domain, each router's forwarding table is installed with other routers' node labels and its local adjacency labels [26]. Segment routing uses interior gateway protocol (e.g., ISIS or OSPF) extensions to distribute MPLS label bindings and does not require a separate protocol, such as the Label Distribution Protocol (LDP). Thus, it improves several orders of scaling gains over conventional Resource Reservation Protocol with Traffic-engineering Extensions (RSVP-TE). However, basic segment routing requires proactive deployment of node labels and adjacency label segments [26]. The proactive flow entry deployment would require information on the traffic pattern, which is different from JumpFlow's application scenario, in which the controller reactively places entries for random and bursty traffic flows.

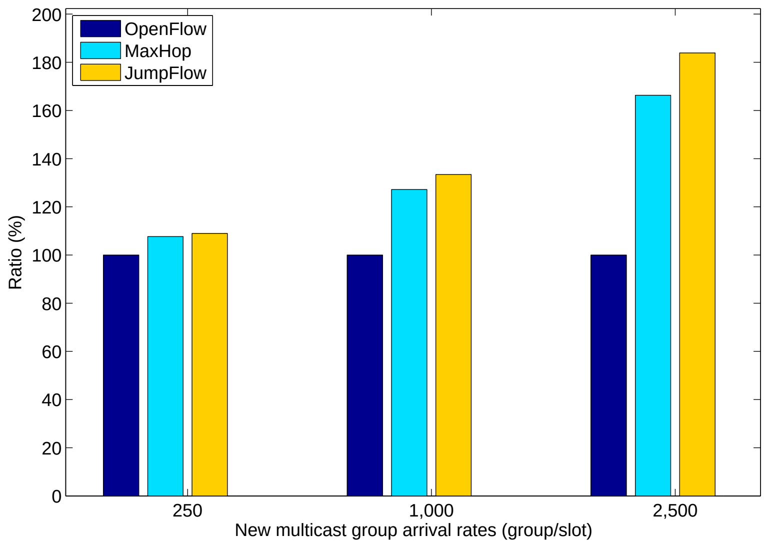


Figure 16: Ratio of accepted multicast groups.

6.3. Comparison to SlickFlow

Encoded Path [27] presents an efficient forwarding scheme for fat-tree networks: ingress edge switches embed the customized addresses of switches on the route into the Ethernet MAC fields of the packet header, and intermediate switches forward packets based on the embedded route information. SlickFlow [28] extends Encoded Path to arbitrary data center topologies and addresses data plane fault tolerance by encoding an alternative route with fixed-size segments in the packet header. Compared to SlickFlow, which does not consider the constraints of flow table space, JumpFlow focuses on reducing flow table usage in a general network scenario by efficiently selecting contact switches to load routing information for packets.

6.4. Failure recovery

JumpFlow can be extended to achieve failure recovery. When a switch receives a packet and detects a failure on the packet's route, it will notify the controller to calculate a new route from the switch to the packet's egress edge switch. If the switch that detects the failure was a contact switch on the packet's route, the controller will update the corresponding entry in the switch's flow table and install entries in other switches to configure the new route. If the switch that detects failure was not a contact switch, the switch will be selected as a new contact switch for the packet's new route and will be installed with an entry for the packet.

6.5. Security concern

JumpFlow can be extended for security cases. Conventionally, some switches may be required to perform special operations on certain packets, such as collecting flows' statistics and inspecting packets for security purposes (e.g., DDoS monitor). In such cases, JumpFlow should be enhanced with the constraint that switches for special processing should be selected as contact switches.

7. Related Work

7.1. Proactive flow entry placement

In [29], the authors formulate an offline routing optimization problem with a flow table constraint and present an approximate solution for the NP-hard problem. In [14][15][16], a given table of flow entries (containing wildcard bits) in a node is efficiently decomposed and spread in the network to achieve more

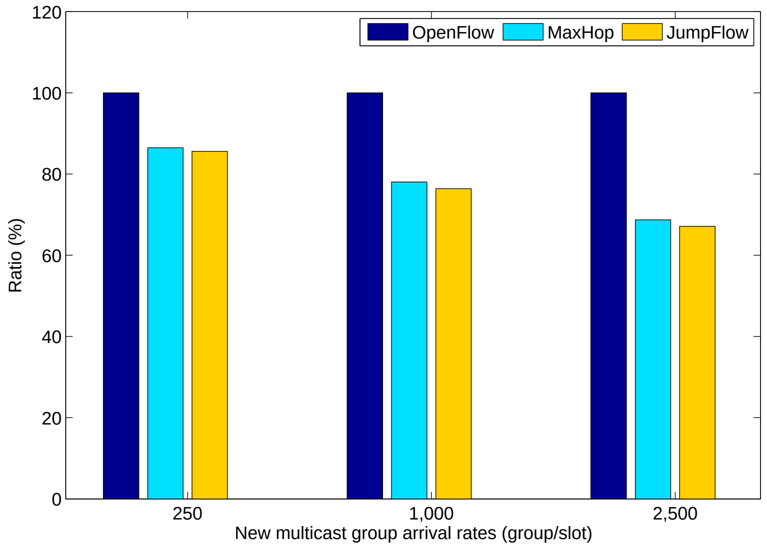


Figure 17: Ratio of average control messages of a multicast group configuration.

equally utilized tables while not violating the correctness of the original table functionality. Segment routing [26] improves several orders of scaling gains over conventional RSVP-TE by combining node and adjacency segments to express routes (see Section 6.2). However, those schemes mainly focus on applications with a given traffic demand matrix, which is an impractical assumption for applications with random and bursty traffic flows.

7.2. Reactive flow entry placement

In [2][3], the online routing schemes are presented to reduce the number of flow entries and high communication overhead between the controller and switches by various techniques. However, the flow table constraint is not explicitly introduced in these articles. Hedera [2] classifies all flows into mouse and elephant flows, and reduces entries for mouse flows with an oblivious static routing. DevoFlow [3] employs clonable wildcard-match flow entries for mouse flow routing. However, the flow table constraint is not explicitly introduced in these articles. Some current works try to use routing information encapsulation-based forwarding schemes to reduce flow table usage, such as MPLS label-based forwarding [18] [20] [21] (see Section 2.2) and SwitchReduce [19], but those schemes require extra packet headers to encapsulate routing information and could lead to bandwidth waste. Encoded Path [27] and SlickFlow [28] are similar to our works, but they concentrate on failure recovery in data center networks and are not compatible with existing networking applications (see Section 6.3). POF [24] and PIF [25] are leading proposals for SDN forwarding plane and enable improvement of the performance of JumpFlow (see Section 6.1).

7.3. Reactive flow entry caching

DIFANE [30] and Smart Rule Cache [31] resolve the dependency by generating new non-overlapped micro entries. CAB [32] proposes partitioning the field space into logical structures called buckets and dynamically caches buckets along with all associated flow entries to save flow table space. Infinite CacheFlow [33] presents a hardware-software hybrid switch design that provides large rule tables by adaptively caching smaller groups of flow entries while preserving the semantics of the network policy. Scotch [34] lowers the communication overhead between the controller and switches by using an Open vSwitch based overlay to elastically scale up the control plane capacity. AggreFlow [35] employs a coarse-grained flow scheduling scheme to achieve high power efficiency and load balancing in data center networks with a low overhead of the controller. STAR [17] mitigates the network performance degradation from the flow-table overflow by detecting the actual flow-table utilization of each switch and intelligently selecting paths for new flows from the pre-generated path-sets. Euthenia [36] prevents network congestion by extending resources in the data plane of SDNs with a server-based system. Compared to the above schemes, which generate new entries or require extra resources for entry caching, JumpFlow uses only the available resource in the packet header to reduce flow table usage without introducing other requirements.

7.4. Adaptive Timeout

In [37], the problem of memory management has been tackled by an idea of dynamic timeouts for the flow entry eviction from a switch. The idle-timeout technique usually assigns the same idle timeout for all flows-both short lived and long lived. The authors provide an adaptive approach for choosing a proper timeout for a flow.

8. Conclusion

In this paper, we propose JumpFlow, a forwarding scheme, that uses the available information field in the packet header to carry routing information and reduces flow table usage in an SDN by properly and reactively placing flow entries on the switches. The simulation results show that JumpFlow decreases the flow rejection percentage, postpones the time when the first flow rejection occurs, and improves the ratio of accepted multicast groups compared to baseline schemes. In our future work, we will extend the idea of JumpFlow with POF/PIF by jointly considering the flow table usage, bandwidth usage and other factors to conduct the route configuration.

9. Acknowledgments

We would like to thank Ruoyan Liu from New York University Polytechnic School of Engineering for his early contribution to this work as well as the anonymous reviewers, Dr. Harry Rudin and Prof. Alberto Leon-Garcia, whose comments helped us improve the paper.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM CCR 38 (2) (2008) 69–74.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: Dynamic flow scheduling for data center networks., in: NSDI '10.

[3] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, in: ACM SIGCOMM CCR, Vol. 41, ACM, 2011, pp. 254–265.

[4] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: INFOCOM'13, pp. 2211–2219.

[5] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in: HotSDN'12, ACM, 2012, pp. 1–6.

[6] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H. J. Chao, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Computer Networks 68 (2014) 95–109.

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: saving energy in data center networks, in: USENIX NSDI, 2010.

[8] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, B. Maggs, Cutting the electric bill for internet-scale systems, in: ACM SIGCOMM computer communication review, Vol. 39, ACM, 2009, pp. 123–134.

[9] Z. Guo, Z. Duan, Y. Xu, H. J. Chao, Cutting the electricity cost of distributed datacenters through smart workload dispatching, IEEE communications letters 17 (12) (2013) 2384–2387.

[10] Z. Guo, Z. Duan, Y. Xu, H. J. Chao, Jet: Electricity cost-aware dynamic workload management in geographically distributed datacenters, Computer Communications 50 (2014) 162–174.

[11] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, Simple-fying middlebox policy enforcement using sdn, in: ACM SIGCOMM'13, ACM, 2013, pp. 27–38.

[12] A. Zarek, Y. Ganjali, D. Lie, Openflow timeouts demystified, Univ. of Toronto, Toronto, Ontario, Canada.

[13] T. Benson, A. Akella, D. A. Maltz, Network traffic characteristics of data centers in the wild, in: IMC'10, ACM, 2010, pp. 267–280.

[14] Y. Kanizo, D. Hay, I. Keslassy, Palette: Distributing tables in software-defined networks, in: INFOCOM'13, pp. 545–549.

[15] N. Kang, Z. Liu, J. Rexford, D. Walker, Optimizing the one big switch abstraction in software-defined networks, in: CoNEXT'13, pp. 13–24.

[16] X. N. Nguyen, D. Saucez, C. Barakat, T. Thierry, et al., Optimizing rules placement in openflow networks: trading routing for better efficiency, in: HotSDN'14.

[17] Z. Guo, R. Liu, Y. Xu, A. Gushchin, K.-y. Chen, A. Walid, H. J. Chao, Star: Flow-table and traffic engineering in software-defined networks, Technical report.

[18] M. Soliman, B. Nandy, I. Lambadaris, P. Ashwood-Smith, Source routed forwarding with software defined control, considerations and implications, in: Proceedings of the 2012 ACM conference on CoNEXT student workshop, ACM, 2012, pp. 43–44.

[19] A. S. Iyer, V. Mann, N. R. Samineni, Switchreduce: Reducing switch state and controller involvement in openflow networks, in: IEEE IFIP Networking, 2013.

[20] P. Ashwood-Smith, M. Soliman, W. Tao, Sdn state reduction, IEFT draft.

[21] M. Soliman, B. Nandy, I. Lambadaris, P. Ashwood-Smith, Exploring source routed forwarding in sdn-based wans, in: Communications (ICC), 2014 IEEE International Conference on, IEEE, 2014, pp. 3070–3075.

[22] V. B. L. A. N. IEEE Standard 802.1Q-2003, http://standards.ieee.org/getieee802/download/802.1q-2003.pdf.

[23] B. Fortz, M. Thorup, Internet traffic engineering by optimizing ospf weights, in: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 2, IEEE, 2000, pp. 519–528.

[24] H. Song, Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 127–132.

[25] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 87–95.

[26] J. Tantsura, I. Milojevic, S. Previdi, M. Horneffer, A. Bashandy, R. Shakir, C. Filsfils, B. Decraene, S. Ytti, S. Litkowski, et al., Segment routing with is-is routing protocol, in: Draft-previdi-filsfils-isis-segment-routing-00 (work in progress),, 2013.

[27] R. M. Ramos, M. Martinello, C. E. Rothenberg, Data center fault-tolerant routing and forwarding: An approach based on encoded paths, in: Dependable Computing (LADC), 2013 Sixth Latin-American Symposium on, IEEE, 2013, pp. 104–113.

[28] R. M. Ramos, M. Martinello, C. Esteve Rothenberg, Slickflow: Resilient source routing in data center networks unlocked by openflow, in: Local Computer Networks (LCN), 2013 IEEE 38th Conference on, IEEE, 2013, pp. 606–613.

[29] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, On the effect of forwarding table size on sdn network utilization, in: INFOCOM'14, pp. 1734–1742.

[30] M. Yu, J. Rexford, M. J. Freedman, J. Wang, Scalable flow-based networking with difane, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 351–362.

[31] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, Wire speed packet classification without tcams: a few more registers (and a bit of logic) are enough, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 35, ACM, 2007, pp. 253–264.

[32] B. Yan, Y. Xu, H. Xing, K. Xi, H. J. Chao, Cab: a reactive wildcard rule caching system for software-defined networks, in: Proceedings of the third workshop on Hot topics in software defined networking, ACM, 2014, pp. 163–168.

[33] N. Katta, J. Rexford, D. Walker, Infinite cacheflow in software-defined networks, Princeton School of Engineering and Applied Science, Tech. Rep.

[34] A. Wang, Y. Guo, F. Hao, T. Lakshman, S. Chen, Scotch: Elastically scaling up sdn control-plane using vswitch based overlay, in: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, ACM, 2014, pp. 403–414.

[35] Z. Guo, S. Hui, Y. Xu, H. Chao, Aggreflow: Dynamic flow scheduling for power-efficient data center networks, Technical report.

[36] D. Zhan, S. Wang, Y. Xu, Z. Guo, H. Chao, Euthenia: Extending sdn switch resources with proxy servers, Technical report.

[37] A. Vishnoi, R. Poddar, V. Mann, S. Bhattacharya, Effective switch memory management in openflow networks, in: DEBS'14, pp. 177–188.