Joint Virtual Switch Deployment and Routing for Load Balancing in SDNs

Xuwei Yang, Hongli Xu, *Member, IEEE*, Liusheng Huang, *Member, IEEE*, Gongming Zhao, Peng Xi, and Chunming Qiao, *Fellow, IEEE,*

***Abstract*— To better serve a diversity of flows, load balancing is crucial to ensure operational efficiency. However, previous works for load balancing have several disadvantages: 1) limited applicability with sub-flow scheduling (e.g., LetFlow); 2) hash collision (e.g., ECMP); or 3) transient network congestion due to reactive scheduling for traffic dynamics (e.g., Hedera and DevoFlow). An important reason for the above disadvantages is that it is difficult to provide fully fine-grained flow control for load balancing in an SDN as the flow table size of each SDN switch is usually limited. Inspired by the fact that a virtual switch (vswitch) has more powerful processing capacity and more flow entries compared with a physical switch, the previous work (e.g., Presto) deploys one vswitch for each ingress switch, and achieves the load balancing through efficient flow routing. However, this mechanism may lead to high cost and not well deal with topology asymmetry. Thus, this paper proposes to achieve the load balancing by incrementally deploying a certain number of vswitches in an SDN. We formulate the joint optimization of vswitch deployment and routing (JVR) problem as an integer linear program, and prove its NP-hardness. A rounding-based algorithm with bounded approximation factors is proposed to solve the JVR problem. We implement the proposed algorithm on an SDN testbed for experimental studies and use simulations for large-scale investigation. The experimental results and simulation results show high efficiency of our algorithm. For example, our proposed algorithm can reduce the link load ratio by about 41.5% compared with ECMP by deploying a small number of virtual switches.**

***Index Terms*— Software defined networks, load balancing, virtual switch deployment, rounding, approximation.**

I. Introduction

A TYPICAL SDN consists of a logical controller in the control plane and a set of switches in the data plane. The controller monitors the network and determines the forwarding path of each flow. The switches execute different operations (*e.g.*, forwarding or dropping) for flows based on the rules installed by the controller. Since the controller is able to provide centralized control for each flow through the header-packet reporting mechanism [1], an SDN can implement fine-grained flow management and route control, which help to improve the network resource utilization compared with traditional networks [2].

With these advantages, software defined networking has been widely used in different scenarios, such as data centers [3] and WANs [2]. Due to versatility and universality of network applications, software defined networks should support an increasingly diverse set of workloads, ranging from small latency-sensitive flows (*e.g.*, search or RPCs [4]) to bandwidth-hungry large flows (*e.g.*, video, big data analytics, or VM migration). To better serve a diversity of flows, load balancing is crucial to ensure operational efficiency and suitable application performance. Thus, many previous works have been studied on this issue. There are three main schemes for load balancing, *i.e.*, sub-flow routing, multi-path forwarding, and reactive flow rerouting. However, we will demonstrate that all three schemes may not always work well for SDN scenarios.

The first scheme for load balancing is based on sub-flow routing or splittable traffic, *e.g.*, LetFlow [5] and Hermes [6]. Though these solutions are able to provide fine-grained traffic control, and can achieve better load balancing, this scheme has two critical disadvantages. First, this scheme permits flow traffic to be splittable. However, it does not always work for various scenarios. For example, the window adaptation of TCP flows may be adversely affected if packets of the same flow follow different paths, which limits the applicability of these solutions. Second, due to splittable traffic, it also increases the management difficulty for flow traffic.

To avoid the splittable traffic, the second scheme for load balancing is multi-path forwarding, *e.g.*, ECMP [7], based on flow hashing. This mechanism distributes the traffic of different flows on multiple equal-cost paths. Different from the first scheme (*i.e.*, sub-flow scheduling), the second scheme requires that one flow will be forwarded just through a single path. However, it may perform poorly in asymmetric topologies, which are common in today's networks due to heterogeneous network components or link/device failures [8], [9]. To deal with this weakness, the weighted version of ECMP, called WCMP [9], is designed. For example, Zhou *et al.* [9] assign different weights for paths based on link load distribution. While ECMP or WCMP are applied in an SDN, the group table is necessary to support multi-path forwarding. However, both two methods do not consider

Manuscript received October 3, 2017; revised February 5, 2018; accepted February 27, 2018. Date of publication March 12, 2018; date of current version May 21, 2018. This work was supported in part by NSFC under Grant 61472383, Grant U1709217, Grant 61728207, and Grant 61472385, and in part by the Natural Science Foundation of Jiangsu Province in China under Grant BK20161257. *(Corresponding author: Hongli Xu.)*

X. Yang, H. Xu, L. Huang, G. Zhao, and P. Xi are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with the Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China (e-mail: issacyxw@mail.ustc.edu.cn; xuhongli@ustc.edu.cn; lshuang@ustc.edu.cn; zgm1993@mail.ustc.edu.cn; xipeng@mail.ahnu.edu.cn).

C. Qiao is with the Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY 14260 USA (e-mail: qiao@computer.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2815379

the detailed method to install group entries for multi-path routing. Since the number of group entries is often less than the number of flow entries, and the number of operation rules (*i.e.*, action buckets specified in OpenFlow) supported by each group entry is limited [10], Zhao *et al.* [11] study the joint optimization of flow/group tables for load balancing in the complex setting of large-scale SDNs. However, these solutions have two main disadvantages. First, since the multi-path forwarding is usually implemented based on flow hashing, it causes congestion when hash collisions occur [12], [13]. Second, these methods route flows often based on the traffic prediction, which may not be exactly accurate. Due to traffic dynamics, the network may still be load imbalance.

The third scheme of load balancing is implemented by reactive flow rerouting, *i.e.*, combining the default paths and per-flow paths. Specifically, the controller pre-deploys default paths using wildcard rules for all flows, and then reroutes some elephant flows for better performance, such as Hedera [12], DevoFlow [14], Planck [13], and HS [15]. These methods can achieve load balancing with a limited size of flow table on each switch. Under this scheme, when a new flow arrives at a switch, it will be directly forwarded to the destination through the default path without reporting to the controller. Thus, due to traffic uncertainty, the network congestion can not be avoided. To alleviate the congestion, these approaches are fundamentally reactive to congestion by rerouting flows, which poses additional flow entry operations on switches and requires extra network infrastructure for real-time traffic counting [13].

An important reason for the above disadvantages is that it is difficult to provide fully fine-grained flow control for load balancing in an SDN as the flow table size of each SDN switch is often limited [14], [16]. Inspired by the fact that a virtual switch (vswitch or software-implemented switch) has more powerful processing capacity and more flow entries for fine-grained control compared with a physical switch [17], Presto [18] enhances the load balancing based on the virtual switch. Specifically, all the edge switches are implemented using vswitches, while the core switches are physical switches. This makes a strong case for moving network load balancing functionality out of the core network hardware and into the software-based edge. However, Presto also results in two main disadvantages. First, this method is usually fit for the structured topology, *e.g.*, Fat-Tree [19], in which only parts of switches are ingress switches, while for unstructured topologies, *e.g.*, HyperX, it requires to deploy one vswitch for each physical switch, which leads to a higher deployment cost and worse scalability. Second, Presto uses the proactive routing scheme (*e.g.*, ECMP), which may not well deal with the network asymmetry [5]. To this end, we propose to deploy some vswitches associated with some (not all) physical switches. When a flow arrives at a vswitch, the controller can provide reactive (or fine-grained) route control for this flow, while other flows are forwarded through the proactive (or coarse-grained) control. The advantages of our proposed method are as follows:

1) Since our method does not require the multi-path forwarding or flow hashing, the hash collision can be avoided.
2) When a new flow arrives at a vswitch, the controller can dynamically determine the route path for this flow. Thus, transient link congestion can also be avoided.
3) Since our method only deploys a certain number of vswitches in the network, it is scalable and fit for both structured and unstructured topologies.

The main contribution of this paper is as follows. Different from Presto, we first introduce a novel solution of duplicate vswitch deployment while not breaking the legacy network topology. We then formulate the joint vswitch deployment and routing (JVR) problem as an integer linear program, and prove its NP-hardness. A rounding-based algorithm with bounded approximation factors is proposed to solve the problem. Some practical issues are discussed to enhance our load balancing mechanism. We implement the proposed algorithm on an SDN testbed for experimental studies and use simulations for large-scale investigation. The experimental results and simulation results show high efficiency of our proposed algorithm. For example, our algorithm reduces the link load ratio by about 41.5% compared with ECMP, and achieves similar routing performance compared with Presto by deploying a small number of virtual switches.

II. Preliminaries

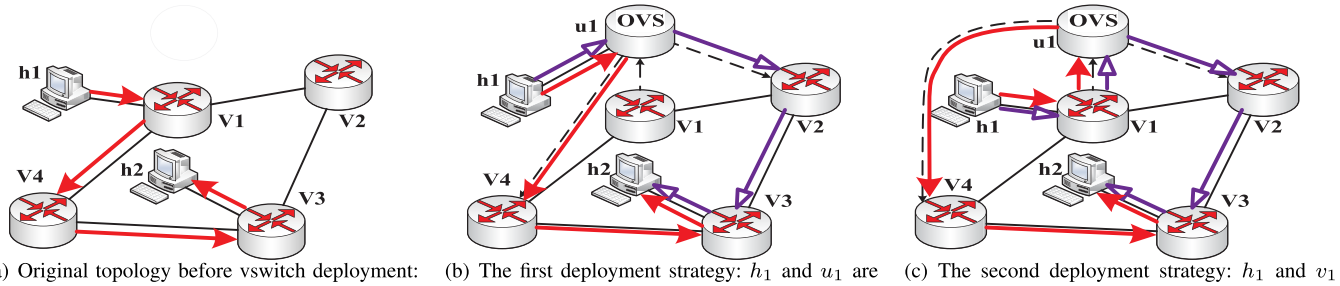
A. Network and Flow Models

An SDN typically consists of a logically-centralized controller and a set of physical switches (or called pswitches), $V = \{v_1, \ldots, v_n\}$, $n = |V|$. These pswitches comprise the data plane of an SDN before deployment. Thus, the network topology of the data plane can be modeled by $G' = (V, E')$, where $E' = \{e'_1, \ldots, e'_l\}$ is the set of directional links connecting pswitches. Note that the controller may be a cluster of controllers, which helps to balance the processing overhead among individual controllers. Since we focus on load balancing in the data plane, the number of controllers will not significantly impact our problem. Thus, we assume that there is only one controller for ease of description. Some key notations are listed in Table I.

The flow set in the SDN is denoted by $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$, with $m = |\Gamma|$. Through long-term observation or statistics collection which is supported by OpenFlow protocol, for each flow $\gamma_i \in \Gamma$, the controller can get the information including the ingress switch $s(\gamma_i)$, the egress switch $d(\gamma_i)$, the traffic throughput, the flow duration, *etc.* It is worth noting that, the traffic intensity $f(\gamma_i)$ can be measured by its traffic throughput divided by flow duration. We assume that all flows are unsplittable for simplicity and ease of flow management [11]. Under the practical scenarios, flow traffic may vary dynamically. To deal with this situation, we will design dynamic flow routing mechanism in Section IV-C.

B. Two Strategies of vSwitch Deployment

In addition to the pswitches, there is another category of devices in the data plane called virtual switch, which is based on software implementation, *e.g.*, OVS [20]. Since a vswitch



(a) Original topology before vswitch deployment: h_1 and v_1 are connected.

(b) The first deployment strategy: h_1 and u_1 are connected.

(c) The second deployment strategy: h_1 and v_1 are connected.

Fig. 1. Two strategies of vswitch deployment. There are four pswitches $\{v_1, v_2, v_3, v_4\}$ and two hosts $\{h_1, h_2\}$ in the original network. We will deploy vswitch u_1 as a duplicate of v_1. Solid lines, dashed lines and thick lines denote links in the original network, the incremental links after vswitch deployment and paths from h_1 to h_2, respectively.

TABLE I
KEY NOTATIONS

Symbol	Semantics
V	a set of pswitches, $V = \{v_1, ..., v_n\}$
U	a set of vswitches
E'	a set of links before vswitch deployment
E	a set of links after vswitch deployment
G'	network topology before vswitch deployment
G	network topology after vswitch deployment
$N(v)$	the neighbor set of switch $v \in V$ in G'
$e(v_i, v_j)$	a link from $v_i \in V \bigcup U$ to $v_j \in V \bigcup U$
$c(e)$	the capacity of link $e \in E$
$l(e)$	the traffic load on link $e \in E$
Γ	a set of flows, $\Gamma = \{\gamma_1, ..., \gamma_m\}$
$s(\gamma)$	the ingress switch of flow $\gamma \in \Gamma$
$d(\gamma)$	the egress switch of flow $\gamma \in \Gamma$
$f(\gamma)$	the traffic intensity of flow $\gamma \in \Gamma$
$\mathcal{P}_\gamma$	a feasible path set for flow $\gamma \in \Gamma$
θ_γ	the default path for flow γ
$p_{v_i}^{v_j}$	the default path from $v_i \in V$ to $v_j \in V$
k	the maximum number of vswitches to be deployed

has a large amount of flow entries [17], it has more fine-granularity control ability compared with a pswitch, and helps to improve the performance of flow scheduling in SDNs [18].

We introduce the duplicate vswitch deployment approach. Assume that the neighbor set of pswitch v_i in an SDN is denoted by $N(v_i)$. When a vswitch u_i is deployed as a duplicate of pswitch v_i, this vswitch will be connected to v_i and its neighboring physical switches in set $N(v_i)$. There are two strategies of duplicate vsiwtch deployment in an SDN, illustrated through an example in Fig. 1. The original network, in Fig. 1(a), contains four pswitches $\{v_1, v_2, v_3, v_4\}$ and two hosts $\{h_1, h_2\}$. We will deploy vswitch u_1 as a duplicate of pswitch v_1.

1. For the first deployment strategy, this pswitch v_i connects with these hosts via the duplicated vswitch u_i. As shown in Fig. 1(b), host h_1 connects to u_1, and u_1 will connect to three pswitches $\{v_1, v_2, v_4\}$. Under this case, the controller may choose the path (*i.e.*, $h_1 \rightarrow u_1 \rightarrow v_2 \rightarrow v_3 \rightarrow h_2$) or the other path (*i.e.*, $h_1 \rightarrow u_1 \rightarrow v_4 \rightarrow v_3 \rightarrow h_2$) from h_1 to h_2 according to the state of the current network. This strategy breaks the connection between v_i and its associated hosts.
2. The second deployment strategy adds (1) a directional link from v_i to u_i and (2) directional links from u_i to all the pswitches in $N(v_i)$. This is an alternative version of the first strategy. As shown in Fig. 1(c), u_1 will connect with three pswitches $\{v_1, v_2, v_4\}$. Under this case, the controller should install a rule on pswitch v_1 so that all flows from h_1 will be directly forwarded to vswitch u_1, and may choose the path (*i.e.*, $h_1 \rightarrow v_1 \rightarrow u_1 \rightarrow v_2 \rightarrow v_3 \rightarrow h_2$) or the other path (*i.e.*, $h_1 \rightarrow v_1 \rightarrow u_1 \rightarrow v_4 \rightarrow v_3 \rightarrow h_2$) from h_1 to h_2.

Obviously, the second strategy will not break the original connection between v_i and attached hosts, which is convenient for vswitch deployment. In the following, we adopt the second strategy in problem definition and algorithm description. In fact, our proposed algorithm will work for both two strategies. It is worth noting that, the experiment results in Section V-B shows that the additional delay caused by a vswitch is small and will not significantly affect the user's QoS.

C. Impact of Flow Routing on vSwitch Deployment

One may think that a natural way is to deploy vswitches on the locations that can control more flows or traffics, which is of benefit to routing performance in general. However, it is not always the case for the following reasons. First, the routing performance depends on the network topology and traffic distribution. It may be inefficient if we ignore the impact of network topology. Second, besides network topology and traffic distribution, flow routing also significantly impacts the network performance [2]. Third, the natural way may lead to routing performance reduction and can not guarantee the bounded approximation performance. Therefore, it is of significance to study the joint optimization of vswitch deployment and routing for load balancing in SDNs.

D. Problem Definition

In this section, we give the definition of the joint vswitch deployment and routing (JVR) problem. To enhance the load balancing, we first deploy k virtual switches (*e.g.*, OVS) as duplicates of chosen pswitches, where k is a predefined constant and determined by the deployment budget. In addition, the controller needs to install a wildcard rule that matches all flows generated from its adjacent terminals on each chosen

pswitch and route all these flows to duplicate vswitch. The final topology after vswitch deployment is denoted by G.

Due to the limited size of a flow table on a pswitch, we assume that the controller has deployed default paths on these pswitches [14], [18]. For simplicity, the default path from $v_i \in V$ to $v_j \in V$ is denoted by $p_{v_i}^{v_j}$, which will be further discussed in Section IV-A. There are two routing strategies for new-arrival flows. (1) If a flow arrives at a vswitch, this vswitch reports its header packet to the controller, which will choose one feasible path for this flow. (2) Otherwise, this flow will be forwarded through the default path. For flow γ that arrives at a vswitch, its ingress and egress switches are denoted by $s(\gamma)$ and $d(\gamma)$. For the ingress switch $s(\gamma)$, the deployed duplicate vswitch is denoted by u, and the neighbor switch set is $N(s(\gamma))$. We construct a feasible path set $\mathcal{P}_\gamma$ for flow γ as follows: for each pswitch $v \in N(s(\gamma))$, we add the path $e(s(\gamma), u) + e(u, v) + p_v^{d(\gamma)}$ to $\mathcal{P}_\gamma$ if this path is loop-free. Then, the controller will choose a feasible path $p \in \mathcal{P}_\gamma$ for this flow. We measure the traffic load $l(e)$ of link e. Assume that the capacity of link e is denoted by $c(e)$. The link load ratio is defined as $\lambda = \max\{\frac{l(e)}{c(e)},\ e \in E\}$. Our objective is to minimize the maximum link load ratio, *i.e.*, $\min\ \lambda$.

To formulate the JVR problem, we construct a super-graph based on the original topology. Specifically, for each pswitch v_i, we add a duplicate vswitch u_i. As described in Section II-B, we also add (1) a directional link from v_i to u_i, and (2) some links from vswitch u_i to each neighbor pswitch of v_i. After vswitch deployment, the link set in the super-graph is denoted by E^s. θ_γ denotes the default path of flow γ. We give the formulation of JVR as follows:

$$\min\ \lambda$$
$$S.t.\ \begin{cases} \sum_{v\in V} x_v \le k & \\ \sum_{p\in\mathcal{P}_\gamma} y_\gamma^p = 1, & \forall\gamma\in\Gamma \\ \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma} y_\gamma^p \le x_{s(\gamma)}, & \forall\gamma\in\Gamma \\ \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in\mathcal{P}_\gamma} y_\gamma^p & \\ \quad \cdot f(\gamma) \le \lambda\cdot c(e), & \forall e\in E^s \\ x_v \in \{0,1\}, & \forall v\in V \\ y_\gamma^p \in \{0,1\}, & \forall p\in\mathcal{P}_\gamma,\ \forall\gamma\in\Gamma \end{cases} \tag{1}$$

where x_v denotes whether there deploys a duplicate vswitch for pswitch v or not, and y_γ^p denotes whether the flow γ will select the path $p \in \mathcal{P}_\gamma$ or not. The first inequality means that at most k vswitches will be deployed in the network. The second set of constraints tells that each flow γ must select one path in feasible path set $\mathcal{P}_\gamma$. The third set of constraints means that the controller can dynamically choose a feasible path for flow γ if there deploys a duplicate vswitch for its ingress switch $s(\gamma)$. The fourth set of inequalities measures the traffic load on each link e. The objective is to achieve the load balancing on the links, that is, $\min\ \lambda$.

Theorem 1: The JVR problem is NP-hard.

We can show that the multi-commodity flow (MCF) with minimum congestion problem [21] is a special case of the JVR problem. Thus, the JVR problem is NP-hard too.

III. Algorithm Design

A. Rounding-Based Algorithm for JVR

In this section, we design a rounding-based algorithm RBVR for vswitch deployment. To solve the integer linear program in Eq. (1), the algorithm first constructs a linear program as a relaxation of the JVR problem. Specifically, we relax the variables $\{x_v\}$ and $\{y_\gamma^p\}$ to be fractional. The number of vswitches is permitted to be fractional, and the traffic of each flow γ can be arbitrarily split on feasible paths. We formulate the linear program LP_1.

$$\min\ \lambda$$
$$S.t.\ \begin{cases} \sum_{v\in V} x_v \le k & \\ \sum_{p\in\mathcal{P}_\gamma} y_\gamma^p = 1, & \forall\gamma\in\Gamma \\ \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma} y_\gamma^p \le x_{s(\gamma)}, & \forall\gamma\in\Gamma \\ \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in\mathcal{P}_\gamma} y_\gamma^p & \\ \quad \cdot f(\gamma) \le \lambda\cdot c(e), & \forall e\in E^s \\ x_v \in [0,1], & \forall v\in V \\ y_\gamma^p \in [0,1], & \forall p\in\mathcal{P}_\gamma,\ \forall\gamma\in\Gamma \end{cases} \tag{2}$$

Since Eq. (2) is a linear program, the first step of RBVR solves it in polynomial time with a linear program solver. Assume that the optimal solutions for Eq. (2) are denoted by $\{\widetilde{x}_v\}$ and $\{\widetilde{y}_\gamma^p\}$, and the optimal result is denoted by $\widetilde{\lambda}$. As Eq. (2) is a relaxation of the JVR problem, $\widetilde{\lambda}$ is a lower-bound result for JVR.

The second step will determine how to deploy vswitches and select one of the feasible paths for every flow. We obtain integer solutions $\{\widehat{x}_v\}$ and $\{\widehat{y}_\gamma^p\}$ with $v \in V$ and $p \in P_\gamma$ using the randomized rounding method [22]. For each ingress switch v, the algorithm sets $\widehat{x}_v$ to 1 with probability $\widetilde{x}_v$ independently, which means that a vswitch will be deployed for switch v. Otherwise, $\widehat{x}_v$ is set to 0, which means no vswitch will be deployed for switch v. We then select a path for flow γ according to the rounding result of $\widehat{x}_{s(\gamma)}$. There are two cases for the ingress switch $s(\gamma)$. (1) If there is no vswitch for $s(\gamma)$ (or $\widehat{x}_{s(\gamma)} = 0$), we only select the default path θ_γ for flow γ. (2) If there deploys a vswitch for $s(\gamma)$ (or $\widehat{x}_{s(\gamma)} = 1$), we randomly choose one path $p \in \mathcal{P}_\gamma$ as the route path of flow γ with probability $w(p)$. Specifically, if p is the default path θ_γ, its probability $w(p)$ is $\frac{\widetilde{x}_{s(\gamma)}+\widetilde{y}_\gamma^{\theta_\gamma}-1}{\widetilde{x}_{s(\gamma)}}$. Note that $\widetilde{x}_{s(\gamma)} + \widetilde{y}_\gamma^{\theta_\gamma} - 1 \ge 0$ because of the third set of inequalities in Eq. (2). Otherwise, the probability $w(p)$ of path p is $\frac{\widetilde{y}_\gamma^p}{\widetilde{x}_{s(\gamma)}}$. By the end of this step, we have determined the vswitch deployment and flow routing. The RBVR algorithm is formally described in Alg. 1.

B. Performance Analysis

This section analyzes the approximation performance of the proposed RBVR algorithm. We first give two famous theorems for probability analysis.

Theorem 2 (Chernoff Bound): Given n independent variables: $z_1, z_2, \ldots, z_n$, where $\forall z_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n z_i]$.

Algorithm 1 RBVR: Rounding-Based Algorithm for JVR

1: **Step 1: Solving the Relaxed JVR Problem**
2: Construct a linear program LP_1 in Eq. (2)
3: Obtain the optimal solutions $\{\widetilde{x}_v\}$ and $\{\widetilde{y}_\gamma^p\}$
4: **Step 2: Deploying vswitches for load balancing**
5: Derive integer solutions $\{\widehat{x}_v\}$ and $\{\widehat{y}_\gamma^p\}$ with $v \in V$ and $p \in \mathcal{P}_\gamma$
6: **for** each $v \in V$ **do**
7: Set $\widehat{x}_v$ to 1 with probability $\widetilde{x}_v$
8: **for** each $\gamma \in \Gamma$ **do**
9: **if** $\widehat{x}_{s(\gamma)} = 0$ **then**
10: Set $\widehat{y}_\gamma^{\theta_\gamma}$ to 1 and y_γ^p to 0, where $p \in \mathcal{P}_\gamma \setminus \theta_\gamma$
11: **if** $\widehat{x}_{s(\gamma)} = 1$ **then**
12: Set $\widehat{y}_\gamma^{\theta_\gamma}$ to 1 with probability $\frac{\widetilde{x}_{s(\gamma)}+\widetilde{y}_\gamma^{\theta_\gamma}-1}{\widetilde{x}_{s(\gamma)}}$ or set $\widehat{y}_\gamma^p$, $p \in \mathcal{P}_\gamma \setminus \theta_\gamma$, to 1 with probability $\frac{\widetilde{y}_\gamma^p}{\widetilde{x}_{s(\gamma)}}$
13: Deploy a vswitch for switch v if $\widehat{x}_v = 1$ and Select path p for flow γ if $\widehat{y}_\gamma^p = 1$

Then, $\mathbf{Pr}\left[\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{z_i} \geq (\mathbf{1}+\epsilon)\mu\right] \leq \mathbf{e}^{\frac{-\epsilon^2\mu}{2+\epsilon}}$, where ϵ is an arbitrary positive value.

Theorem 3 (Union Bound): Given a countable set of n events: $A_1, A_2, \ldots, A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup \ldots \cup A_n) \leq \sum_{i=1}^{n} \mathbf{Pr}(A_i)$.

We then give the approximation performance for vswitch constraint.

Theorem 4: After the rounding process, the number of deployed vswitches will not exceed the vswitch constraint k by a factor of $\frac{2\log n}{k} + 3$.

Proof: We use a variable φ_v to denote whether a vswitch is deployed for pswitch v or not. According to Line 7 of the RBVR algorithm, we set $\widehat{x}_v$ to 1 with probability $\widetilde{x}_v$. Otherwise, $\widehat{x}_v$ is set to 0. Thus, the expectation $\mathbb{E}[\varphi_v]$ is $\widetilde{x}_v$. The expected number of required vswitches is:

$$\mathbb{E}\left[\sum_{v\in V} \varphi_v\right] = \sum_{v\in V} \mathbb{E}[\varphi_v] = \sum_{v\in V} \widetilde{x}_v \leq k \tag{3}$$

As φ_v is a 0-1 integer variable, we can directly apply Theorem 2. Assume that σ is an arbitrary positive value. It follows:

$$\mathbf{Pr}\left[\sum_{v\in V} \varphi_v \geq (1+\sigma)k\right] \leq e^{\frac{-\sigma^2 k}{2+\sigma}} \tag{4}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{v\in V} \varphi_v \geq (1+\sigma)k\right] \leq e^{\frac{-\sigma^2 k}{2+\sigma}} \leq \mathcal{H} \tag{5}$$

where $\mathcal{H}$ is the function of network-related variables (such as the number of switches n, *etc.*) and $\mathcal{H} \to 0$ when the network size grows.

The solution for Eq. (5) can be expressed as:

$$\sigma \geq \frac{\log\frac{1}{\mathcal{H}} + \sqrt{\log^2\frac{1}{\mathcal{H}} + 8\ k\log\frac{1}{\mathcal{H}}}}{2k} \tag{6}$$

Set $\mathcal{H} = \frac{1}{n^2}$. Apparently $\mathcal{H} \to 0$ as $n \to \infty$. With respect to Eq. (6), we set

$$\begin{aligned}\sigma &= \frac{\log\frac{1}{\mathcal{H}} + \log\frac{1}{\mathcal{H}} + 4k}{2k} \\ &= \frac{4\log n + 4k}{2k} = \frac{2\log n}{k} + 2\end{aligned} \tag{7}$$

Then Eq. (7) is guaranteed with $1+\sigma = \frac{2\log n}{k} + 3$. That means, after the rounding process, the total number of deployed vswitches will not exceed k by a factor of $\frac{2\log n}{k} + 3$. ■

In the following, we analyze the approximation performance for the link capacity constraint. We first show that the algorithm can guarantee that the controller will choose one path for each flow even with randomized rounding.

Lemma 5: The controller will choose a route path for each flow.

Proof: As described in the second step of the CRBVR algorithm, for each flow γ, there are two cases for its ingress switch $s(\gamma)$. On one hand, there is no duplicate vswitch for $s(\gamma)$ or $\widehat{x}_s(\gamma) = 0$, the algorithm will assign the default path for flow γ. On the other hand, one duplicate vswitch is deployed for $s(\gamma)$ or $\widehat{x}_{s(\gamma)} = 1$. We will show that the total probability assigned for all feasible paths of each flow is 1, which ensures a route path for this flow.

As described in the second step of the CRBVR algorithm, the probability that the controller selects path θ_γ as the route is

$$\begin{aligned}\mathbf{Pr}\left[\widehat{y}_\gamma^{\theta_\gamma} = 1\right] &= \mathbf{Pr}\left[\widehat{y}_\gamma^{\theta_\gamma} = 1 \mid \widehat{x}_{s(\gamma)} = 1\right] \cdot \mathbf{Pr}\left[\widehat{x}_{s(\gamma)} = 1\right] \\ &\quad + \mathbf{Pr}\left[\widehat{y}_\gamma^{\theta_\gamma} = 1 \mid \widehat{x}_{s(\gamma)} = 0\right] \cdot \mathbf{Pr}\left[\widehat{x}_{s(\gamma)} = 0\right] \\ &= \frac{\widetilde{x}_{s(\gamma)} + \widetilde{y}_\gamma^{\theta_\gamma} - 1}{\widetilde{x}_{s(\gamma)}} \cdot \widetilde{x}_{s(\gamma)} + 1 \\ &\quad \cdot (1 - \widetilde{x}_{s(\gamma)}) = \widetilde{y}_\gamma^{\theta_\gamma}\end{aligned} \tag{8}$$

Similarly, the probability that the controller selects path $p \in P_\gamma \setminus \theta_\gamma$ as the route is

$$\begin{aligned}\mathbf{Pr}\left[\widehat{y}_\gamma^{p} = 1\right] &= \mathbf{Pr}\left[\widehat{y}_\gamma^{p} = 1 \mid \widehat{x}_{s(\gamma)} = 1\right] \cdot \mathbf{Pr}\left[\widehat{x}_{s(\gamma)} = 1\right] \\ &= \frac{\widetilde{y}_\gamma^{p}}{\widetilde{x}_{s(\gamma)}} \cdot \widetilde{x}_{s(\gamma)} = \widetilde{y}_\gamma^{p}\end{aligned} \tag{9}$$

According to Eqs. (8) and (9), we have:

$$\begin{aligned}\sum_{p\in\mathcal{P}_\gamma} \mathbf{Pr}\left[\widehat{y}_\gamma^{p} = 1\right] &= \mathbf{Pr}\left[\widehat{y}_\gamma^{\theta_\gamma} = 1\right] + \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma} \mathbf{Pr}\left[\widehat{y}_\gamma^{p} = 1\right] \\ &= \widetilde{y}_\gamma^{\theta_\gamma} + \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma} \widetilde{y}_\gamma^{p} = \sum_{p\in\mathcal{P}_\gamma} \widetilde{y}_\gamma^{p} = 1\end{aligned} \tag{10}$$

Eq. (10) means that one path $p \in \mathcal{P}_\gamma$ should be selected for flow γ. ■

Lemma 6: The RBVR algorithm can guarantee that the expected traffic load on each link e from γ after the second step is same as the solution $\widetilde{l}(e,\gamma)$ of the linear program LP_1.

Proof: Let variable $\eta_{e,\gamma}$ denote the total traffic load of link e from flow γ. We use $\widetilde{l}(e,\gamma)$ to denote the traffic load of link e from γ by the result of the linear program LP_1.

Moreover, τ_e^p means whether link e belongs to path p or not. The expectation of variable $\eta_{e,\gamma}$ is:

$$\begin{aligned}\mathbb{E}[\eta_{e,\gamma}] &= \mathbf{Pr}\left[\widehat{y}_\gamma^{\theta_\gamma}=1\right]\cdot\tau_e^p + \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma}\mathbf{Pr}\left[\widehat{y}_\gamma^{p}=1\right]\cdot\tau_e^p \\ &= \widetilde{y}_\gamma^{\theta_\gamma}\cdot\tau_e^{\theta_\gamma} + \sum_{p\in\mathcal{P}_\gamma\setminus\theta_\gamma}\widetilde{y}_\gamma^{p}\cdot\tau_e^p = \sum_{p\in\mathcal{P}_\gamma}\widetilde{y}_\gamma^{p}\cdot\tau_e^p \\ &= \widetilde{l}(e,\gamma) \end{aligned} \tag{11}$$

Note that the second equality holds according to Eqs. (8) and (9). Eq. (11) shows that the expected traffic load of each link after the second step is same as the solution of the linear program LP_1. ∎

Assume that the minimum capacity of all links is denoted by $c^{min}(e)$. We define a constant value α as follows:

$$\alpha = \min\{\frac{\widetilde{\lambda}\cdot c^{\min}(e)}{f(\gamma)}, \gamma\in\Gamma\} \tag{12}$$

Under many practical application scenarios, the flow intensity is usually much less than the link capacity, because the flow intensity is not more than the corresponding host-switch link capacity [16], [23]. Thus, it is reasonable to assume that $\alpha \gg 1$.

Theorem 7: The proposed RBVR algorithm guarantees that the total traffic on any link $e\in E$ will not exceed the traffic of the fractional solution by a factor of $\frac{3\log n}{\alpha}+3$.

Proof: The traffic load of link e after the first step is denoted by $\widetilde{l}(e)$. By the definition, variables $\eta_{e,\gamma}$ with $\gamma\in\Gamma$ are mutually independent. According to Eq. (11), the expected traffic load on link e is:

$$\mathbb{E}\left[\sum_{\gamma\in\Gamma}\eta_{e,\gamma}\right] = \sum_{\gamma\in\Gamma}\mathbb{E}[\eta_{e,\gamma}] = \sum_{\gamma\in\Gamma}\widetilde{l}(e,\gamma) = \widetilde{l}(e) \tag{13}$$

By the fourth set of inequalities in Eq. (2), we have:

$$\widetilde{l}(e) = \sum_{\gamma\in\Gamma}\sum_{e\in p: p\in\mathcal{P}_\gamma}\widetilde{y_\gamma^p}\cdot f(\gamma) \le \widetilde{\lambda}\cdot c(e) \tag{14}$$

Combining Eqs. (13), (14) and the definition of α in Eq. (12), we have:

$$\begin{cases}\dfrac{\eta_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}c(e)}\in[0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma}\dfrac{\eta_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\right]\le\alpha.\end{cases} \tag{15}$$

By applying Theorem 2, assume that ρ is an arbitrary positive value. It follows:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{\eta_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\ge(1+\rho)\alpha\right]\le e^{\frac{-\rho^2\alpha}{2+\rho}} \tag{16}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{\eta_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\ge(1+\rho)\right]\le e^{\frac{-\rho^2\alpha}{2+\rho}}\le\frac{\mathcal{F}}{n^2} \tag{17}$$

where $\mathcal{F}$ is the function of network-related variables (such as the number of switches n, *etc.*) and $\mathcal{F}\to 0$ when the network size grows.

The solution for Eq. (17) can be expressed as:

$$\rho \ge \frac{\log\frac{n^2}{\mathcal{F}} + \sqrt{\log^2\frac{n^2}{\mathcal{F}} + 8\alpha\log\frac{n^2}{\mathcal{F}}}}{2\alpha}, \quad n\ge 2 \tag{18}$$

Set $\mathcal{F}=\frac{1}{n^2}$. Eq. (17) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{\eta_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\ge(1+\rho)\right]\le\frac{1}{n^4}, \text{where}\quad \rho=\frac{4\log n}{\alpha}+2 \tag{19}$$

By applying Theorem 3, we have,

$$\begin{aligned}&\mathbf{Pr}\left[\bigvee_{e\in E}\sum_{\gamma\in\Gamma}\frac{\eta_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\ge(1+\rho)\right] \\ &\le\sum_{e\in E}\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{\eta_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\ge(1+\rho)\right] \\ &\le[\frac{1}{2}n(n-1)+n^2]\cdot\frac{1}{n^4} \\ &\le\frac{3}{2}n^2\cdot\frac{1}{n^4}=\frac{3}{2n^2},\quad \rho=\frac{4\log n}{\alpha}+2\end{aligned} \tag{20}$$

Note that the third inequality holds, because there are at most $\frac{1}{2}n(n-1)$ links in the original network G' and n^2 incremental links when there are n vswitches in an SDN. Eq. (20) means that the proposed RBVR algorithm can guarantee that the total traffic on any link $e\in E$ will not exceed the fractional solution by a factor of $1+\rho=\frac{4\log n}{\alpha}+3$. ∎

Approximation Factors: Following from our analyses, the vswitch constraint will not be violated by a factor of $\frac{2\log n}{k}+3$, and the link capacity will hardly be violated by a factor of more than $\frac{4\log n}{\alpha}+3$ by routing a full percentage of flows on each chosen path. It means that the algorithm can achieve the optimal solution, violating the vswitch constraint by a factor $\frac{2\log n}{k}+3$ and the link capacity constraint by a factor $\frac{4\log n}{\alpha}+3$ at most, which is also called as bi-criteria approximation [16]. By using the traffic controlling method, the intensity of each flow can be limited to a specific value, so that the network congestion can be avoided.

We should address that, in most situations, the RBVR algorithm can reach almost the constant bi-criteria approximation. For example, let $\widetilde{\lambda}$ be 0.4 (with a moderate value). Consider a large-scale network with $n=1000$ switches, so that $\log n\approx 10$. The link capacity of today's networks will be a bandwidth of 1Gbps at least. Observing the practical flow traces, the maximum intensity of a flow may reach 1Mbps or 10Mbps. Under two cases, $\frac{c^{\min}(e)}{f(\gamma)}$ will be 10^3 and 10^2. The approximation factors for the link capacity constraint are 3.1 and 4, respectively. Since k is usually at least 100 if $n=1000$, the approximation factor for the virtual switch constraint is 3.2. In other words, our RBVR algorithm can achieve almost the constant bi-criteria approximation for the JVR problem in many network situations.

C. Complete RBVR Algorithm Description

Though the RBVR algorithm can obtain the bi-criteria approximation performance for the JVR problem, the vswitch

Algorithm 2 CRBVR: Complete Rounding-Based Algorithm for JVR

Step 1: Same as that in RBVR
Step 2: Same as that in RBVR
Step 3: Removing the redundant vswitches
Put all switch v with $\widehat{x}_v = 1$ into set $V^{'}$
if $|V^{'}| \leq k$ **then**
  algorithm terminates.
else if $|V^{'}| > k$ **then**
  Sort all the switches in set V' in the decreasing order of value $\widetilde{x}_v$
  Deploy vswitches for k switches with the largest $\widetilde{x}_v$

constraint may not be fully satisfied after the randomized rounding process. We give the complete algorithm, called CRBVR, so as to satisfy this constraint.

The CRBVR algorithm consists of three main steps. The former two steps of CRBVR are same as those in Alg. 1. The third step will remove some vswitches so as to satisfy the vswitch number constraint. At the beginning of the third step, let V' denote the set of pswitches that have been deployed duplicate vswitches. If $|V'| \leq k$, the algorithm terminates. However, if $|V'| > k$, we retain k vswitches for duplicate deployment. Specifically, the algorithm sorts all the pswitches in set V' in the decreasing order of $\widetilde{x}_v$, and just chooses k pswitches with the largest $\widetilde{x}_v$. The CRBVR algorithm is described in Alg. 2.

IV. Practical Issues for System Implementation

In this section, we give the detailed description of some practical issues for system implementation. Our system mainly consists of three main modules, including system configuration, port traffic statistics collection and dynamic flow routing (*Section IV-B-IV-C*). Moreover, we also discuss how to support multicast in our system (*Section IV-D*).

A. System Configuration

To construct the default paths for flows, we regard all flows between any two pswitches as a macroflow. We adopt the OSPF method [24] to select routes for those macroflows, and install flow entries for those default paths. As specified by the Openflow standard [1], each flow entry includes two fields: idle_timeout and hard_timeout. These two fields control the removal of a flow entry from the flow table. A flow entry will be removed, if no packet has been matched by this entry after a given number of seconds, specified by its non-zero idle_timeout field. A non-zero hard_timeout field causes the flow entry to be removed after a given number of seconds, regardless of how many packets it has matched. If both parameters are set to zero, this flow entry is considered to be permanent, and will be removed only by the controller. To implement the default paths, the controller installs flow entries to each pswitch and sets the idle_timeout and hard_timeout field of each flow entry as zero. Then, we determine k pswitches for duplicated vswitch deployment using the CRBVR algorithm, and build connections between vswitches and pswitches as described in Section II-B. Moreover, for each switch v with a duplicate vswitch u, the controller installs a rule for each connected host on switch v so that all flows from this host will be directly forwarded to vswitch u. We should note that, if there is a flow whose source and destination hosts connect to a same pswitch, the controller needs to install a rule with higher priority on this pswitch to match this flow in order to route it to corresponding port in a loop-free fashion.

B. Port Traffic Statistics Collection

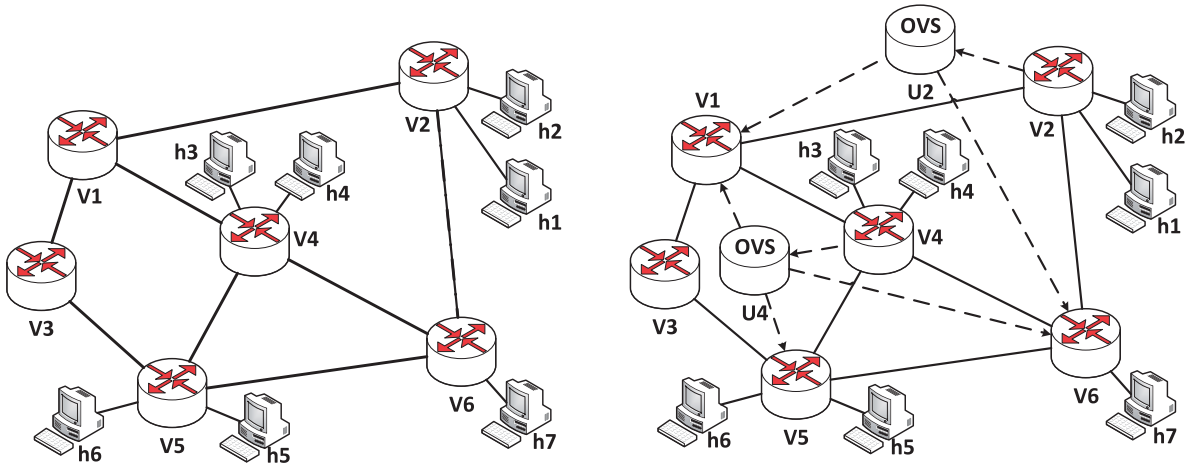
During system running, the controller should master the real-time traffic load on each link to better deal with traffic dynamics. The openflow standard specifies the OFPT_PORT_STATUS interface for port traffic statistics collection. Since each link connects with two ports on two switches, we can collect port statistics only from a subset of switches to reduce the controller overhead. To this end, we use the minimum set cover algorithm [25] for port traffic statistics collection. Specifically, the controller determines from which switches the port statistics information will be collected and sends a set of the OFPT_PORT_STATUS requests to specified switches. After receiving these requests, the switch will report the statistics information of all ports to the controller. As a result, the controller can know the accurate traffic information of all ports (links).

C. Dynamic Flow Routing

When a new flow arrives at a vswitch, this vswitch reports the header-packet to the controller, which will dynamically determine its route path. In this paper, we introduce a simple and efficient routing mechanism, in which the controller chooses the least-congestion route path with flow-table size constraint for each new-arrival flow. Specifically, the controller knows the link traffic load (or the link load ratio) by collecting the port statistics information. For each switch, since the flow table is updated by the controller, the controller can derive the number of occupied flow entries and know whether this switch can accommodate this flow or not. For each path p, the congestion of this path is the maximum load ratio of all links on this path. The controller adopts the Dijkstra method [25] to explore the route path with the least congestion for this flow, and installs rules on switches along this path.

D. Multicast Traffic Routing

The multicast mechanism in SDNs is usually implemented using a joint flow table and group table. The match field of a flow entry matches corresponding multicast traffic, and the instructions domain points to the group entry that specifies the actions bucket for multicast traffic. Each action in the bucket specifies the operations for traffic, such as forwarding traffic to a port. It is assumed that the original network can handle multicast traffic by deploying flow table entries and group table entries on pswitches. To support multicast on vswitches, the controller needs to install a wildcard rule matching all flows from the adjacent hosts on the pswitch's flow table. Then, the controller installs proper flow entries and



(a) SDN testbed before vswitch deployment.

(b) SDN testbed after deployment of vswitches u_2 and u_4.

Fig. 2. Our SDN testbed consists of 6 logically physical switches and 7 hosts before vswitch deployment in (a). We deploy two vswitches u_2 and u_4 as duplicates of pswitches v_2 and v_4, shown in (b). Solid lines and dashed lines denote links in the original network and the incremental links after vswitch deployment, respectively.

group entries on the vswitch to route multicast traffic. Thus, the vswitch can also support multicast traffic.

V. Performance Evaluation

In this section, we evaluate our proposed algorithm through both the testbed implementation and the network simulator [26].

A. Performance Metrics and Benchmarks

This paper studies how to deploy vswitches and route flows for load balancing. We adopt three main metrics for performance evaluation. After vswitch deployment, when a flow arrives at a vswitch, the controller can provide fine-grained control for this flow. We call this flow as a controllable one. The first metric is the *number of controllable flows* (NCF). To evaluate the routing performance, we adopt link load ratio (LLR) and network throughput factor (NTF) as two metrics. During system running, we measure the traffic load $l(e)$ of each link e, and the *link load ratio* is defined as: $LLR = \max\{l(e)/c(e), e \in E\}$. The smaller LLR means better load balancing. When there occurs congestion on some links, we can only forward fractional traffic to the destination. For each flow γ, the traffic of $\delta \cdot f(\gamma)$ at most can be forwarded from source to destination with congestion avoidance, where δ is the *network throughput factor*, with $0 < \delta \leq 1$. In general, if the controller can dynamically control more flows (or with a larger NCF), the link load ratio will be reduced and the network throughput can be improved.

To evaluate how well our proposed algorithm performs, we compare with other three benchmarks. The first benchmark is the most widely used OSPF method [24]. Each switch will construct the shortest paths to all other switches. Thus, the number of required flow entries does not exceed the number of physical switches in an SDN. The second one is ECMP [7], which is widely applied in data center networks for load balancing. It needs to install flow/group entries on pswitches when there exist several equal-cost paths to the destination. Otherwise, flows will be forwarded through the OSPF paths. In the simulations, we use three equal-cost paths for each switch pair. The final one is Presto [18]. As Presto adopts the flow segmentation (*i.e.*, splittable traffic) for load balancing, it is not fit for the TCP flows and increases the additional traffic management cost. Thus, we modify Presto so that all flows are unsplittable, and the controller can determine the dynamic routes for all flows in an SDN. Espresso [27] also puts the routing selection on the powerful edge server. However Espresso improves user experience by automatic selection of the best data center location to serve a particular user, based on real-time performance measurements, which is different from the link load balancing problem for a DC or LAN addressed in this paper. As a result, we choose not to compare our proposed algorithm with Espresso quantitatively. Note that, for fairness, all four algorithms adopt the destination-based prefix-match scheme for default paths in our simulations, so that these methods require almost the same number of flow entries on all pswitches.

B. System Implementation on Platform

1) Implementation on the Platform: We implement the OSPF, ECMP, Presto and CRBVR algorithms on a small-scale testbed. Our SDN platform is logically comprised of three parts: a controller, 6 SDN-enabled physical switches and 7 virtual machines (acting as hosts) in Fig. 2(a). As we focus on the load balancing of data plane, we omit the controller in Fig. 2(a). To expand the testing topology and collect testing data conveniently, we adopt the virtualization technology for system implementation. Specifically, all 6 logically physical switches are implemented using open virtual switches (version 2.7.2 [20]), and each host is implemented using the kernel-based virtual machine (KVM). Each open virtual switch and the connected KVMs are implemented on a server with a core i5-3470 processor and 8GB of RAM. For example, $\{v_1\}$, $\{v_2, h_1, h_2\}$, $\{v_3\}$, $\{v_4, h_3, h_4\}$, $\{v_5, h_5, h_6\}$ and $\{v_6, h_7\}$ are run on 6 servers, respectively. The original network

topology is shown in Fig. 2(a). Besides, we use Ryu [28] that supports the OpenFlow v1.3 standard as the controller software running on another server with a core i5-3470 processor and 16GB of RAM. Since the controller will not participate into data forwarding, it is not explicitly included in Fig. 2(a) for simplicity.

As v_2, v_4, v_5 and v_6 directly connect to hosts, they are ingress switches in the SDN. So, vswitches will only be deployed as duplicates of these ingress switches. When a vswitch is deployed, it connects to an ingress switch and all of its neighbor switches.

2) Flows in the Network: There are roughly two categories of packets in the networks. One is UDP, the other is TCP. Our testing uses the UDP packets, and we will adopt the TCP messages for testing as a future work. When a host generates a flow, in addition to the destination IP address, a unique destination UDP port (larger than 50000) is also specified, so that the controller can distinguish these flows by the unique destination UDP port and make routing decision. Through testing on the OVS platform, we find that when the length of a UDP packet exceeds 1500 bytes, it will be split into multiple packets, and the transport protocol field cannot be analyzed by the vswitch. As a result, header information of different data packets belonging to a same flow will be reported to the controller repeatedly, which will cause massive control link overhead, and additional controller response delay. Thus, we set the size of every data packet to 1300 bytes uniformly. Moreover, there are 20% elephant flows and 80% mice flows to simulate the realistic network scenario [14]. We simulate the elephant flows and mice flows by adjusting different transmitting interval between packets of a flow. The average flow intensity is 0.85Mbps.

3) Testing Results: We run three sets of testings in the SDN platform. We first measure the additional delay of packet forwarding caused by a vswitches. Specifically, we have performed two tests to measure the delay of packet forwarding between two hosts connected with (1) a physical switch and (2) a virtual switch along with a physical switch, respectively. The packet size in the flow is 1KB uniformly. The gap between forwarding delays of two tests is the additional delay caused by a virtual switch, which is shown in Fig. 3. As the flow size increases exponentially, the additional delay increases almost linearly. When one host sends a flow contains one million packets to another, the additional delay caused by a virtual switch is 5.09ms while the total transmission time is about 8.5s. As specified by ITU G.114 [29], it will not significantly impact the user's QoS if the forwarding delay is less than 150ms. In addition, one flow may pass through at most one virtual switch as described in Fig. 1(c). Thus, we can conclude that, while there is some delay for flows passing through a virtual switch, the additional delay is small and will not significantly affect the user's QoS.

The second set of testing observes the maximum link load, which is defined as: $MLL = \max\{l(e), e \in E\}$, by changing the number of deployed vswitches in an SDN. We generate 800 flows by default in this testing. Fig. 4 shows that the maximum link load of OSPF, ECMP and Presto is about 60.7Mbps, 48.0Mbps and 25.3Mbps, respectively. The

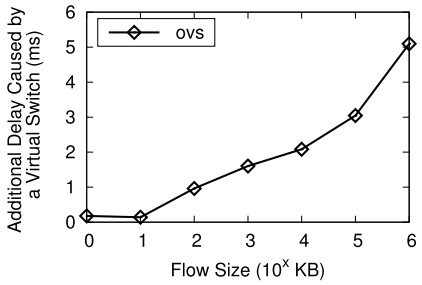


Fig. 3. Additional Delay Caused by a virtual switch (ms) vs. flow size.

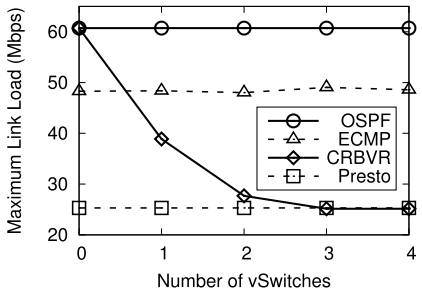


Fig. 4. Maximum Link Load vs. Number of vSwitches.

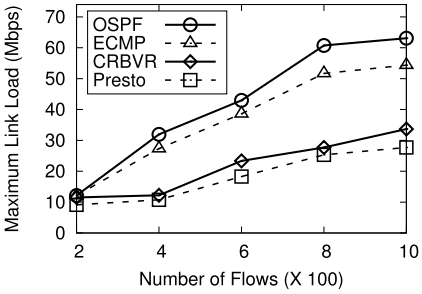


Fig. 5. Maximum Link Load vs. Number of Flows.

CRBVR algorithm decreases the maximum link load with the increasing number of deployed vswitches. Without vswitch, the maximum link load of CRBVR is almost same as that of OSPF. When only one vswitch is deployed, CRBVR can achieve a lower maximum link load compared with ECMP. When two vswitches are deployed (as duplicates of v_2 and v_4 by our testing), CRBVR can further reduce the maximum link load. Specifically, CRBVR reduces the maximum link load by about 54.4% and 42.4% compared with OSPF and ECMP respectively when two vswitches are deployed. Moreover, CRBVR can achieve a similar maximum link load compared with Presto while CRBVR only requires half number of vswitches compared with Presto.

The third testing shows the maximum link load by changing the number of flows in an SDN. We deploy two vswitches u_2 and u_4 as duplicates of pswitches v_2 and v_4 in this testing, and the topology after vswitch deployment is shown in Fig. 2(b). The testing results in Fig. 5 indicate that the maximum link load increases for all four algorithms with more flows in an SDN. Moreover, CRBVR reduces the maximum link load by about 46.6% and 39% compared with OSPF and ECMP, respectively, and achieves similar

routing performance as Presto while using only half number of vswitches.

C. Simulation Evaluation

1) Simulation Setting: We select three practical and typical topologies for our simulations. The first topology, denoted by (a), is for campus networks, and contains 100 switches and 200 servers from Monash university [30]. The second one is the Fat-Tree topology [19], which contains 80 switches (including 16 core switches, 32 aggregation switches, and 32 edge switches) and 128 servers. The third one is the two-dimensional HyperX topology [31], denoted by (c), which contains 64 (ingress) switches and 256 servers. Specifically, all switches are arranged in an 8×8 square, in which each switch is connected to other 14 switches in the same row/column and 4 servers at the same time. We should note that these topologies represent various networks with different features. Specifically, the Monash topology (a) is asymmetrical, while other two topologies are symmetrical for many data center networks. Meanwhile, Fat-Tree is structured while HyperX is unstructured. For all three topologies, each link has a uniform capacity, 10Gbps. Since the number of flow entries on each pswitch is limited, we deploy default paths on pswitches using the OSPF method for simplicity.

To simulate the practical traffic scenario, we generate three types of flows: (1) random flows, whose source and destination hosts are randomly picked; (2) server flows, which simulate the traffic between random hosts and a number of designated servers, *e.g.*, mail servers and web servers; (3) associate flows, which simulate the traffic between a subnet and a server, *e.g.*, communications between the finance department and the finance database or between a hospital and a data center that houses the patient data. Each type of flows accounts for one third of total traffic [15]. Curtis *et al.* [14] have shown that less than 20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we allocate the size for each flow according to this 2-8 distribution and the expected traffic demand of each flow is 1Mbps or 2Mbps when LLR or NTF is the metric respectively. We execute each simulation 100 times and average the numerical results.

2) Simulation Results: We mainly run three sets of simulations on three different topologies to check the effectiveness of our proposed algorithm. The first set of simulations observes the different performance metrics by changing the number of deployed vswitches. In a practical data center network with 1,500 server clusters [32], the average arrival rate reaches 100k flows per second (around 67 flows per second per server) and the duration time of most flows (more than 80%) is less than 10s, so we roughly think that each server provides 670 flows according to Queue Theory [33]. Since there are at most 256 servers on the topology in our simulations, we set the default number of flows to 160K (around 640 flows per server), to make the results of the simulation more credible. First, we observe the number of controllable flows by changing the number of deployed vswitches in an SDN. We note that the Presto system will deploy one vswitch for each ingress pswitch. As a result, it requires 100, 32 and 64 vswitches for Monash, Fat-Tree and Hyperx topologies, respectively. The simulation results are shown in Figs. 6-8. We observe that our proposed algorithm can control more flows with the increasing number of vswitches, which means more powerful control ability for flows. The increasing ratio of controllable flows is much slower with more vswitches deployed in an SDN.

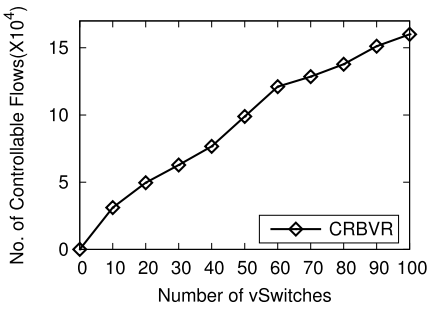


Fig. 6. Number of Controllable Flows vs. Number of vSwitches for Topology (a).

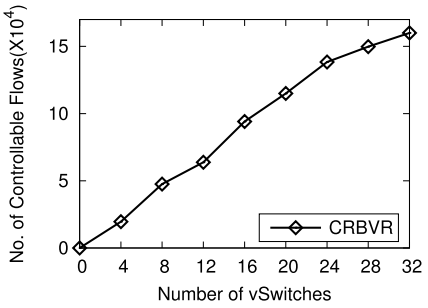


Fig. 7. Number of Controllable Flows vs. Number of vSwitches for Topology (b).

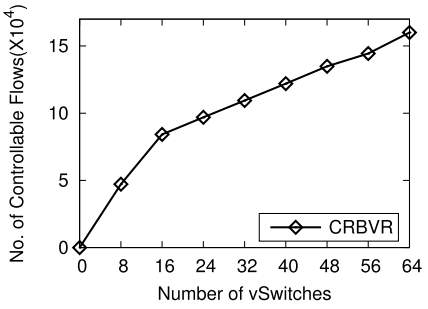


Fig. 8. Number of Controllable Flows vs. Number of vSwitches for Topology (c).

We then observe the link load ratio by changing the number of vswitches on three topologies, and the simulation results are shown in Figs. 9-11. Since the number of vswitches does not affect the number of controllable flows in the networks for OSPF, ECMP and Presto, the link load ratio almost keeps unchanged for these algorithms. When there is no vswitch in the network, all the flows will be forwarded through the default paths, and the CRBVR algorithm can not improve the link load ratio compared with OSPF. With the increasing number of deployed vswitches, the CRBVR algorithm is able to control more flows, which helps to reduce the link load ratio. For example, for topology (b), when there deploys 8 and

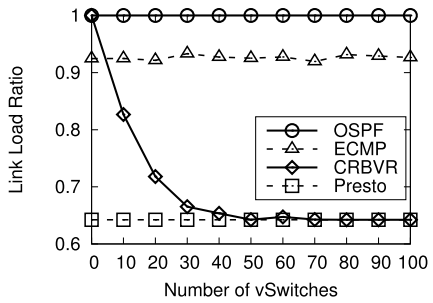


Fig. 9. Link Load Ratio vs. Number of vSwitches for Topology (a).

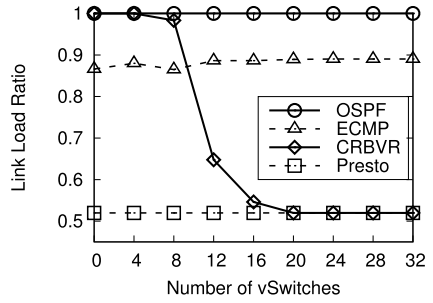


Fig. 10. Link Load Ratio vs. Number of vSwitches for Topology (b).

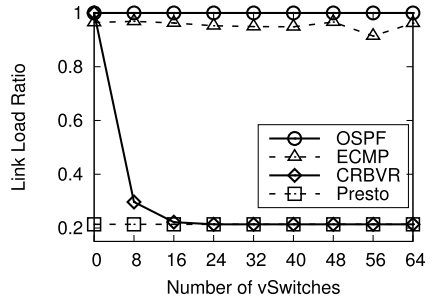


Fig. 11. Link Load Ratio vs. Number of vSwitches for Topology (c).

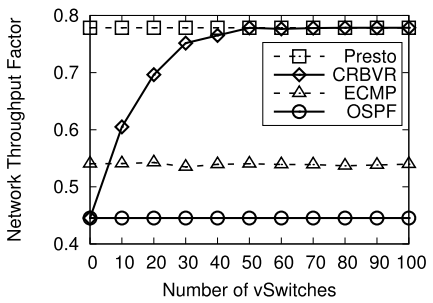


Fig. 12. Network Throughout Factor vs. Number of vSwitches for Topology (a).

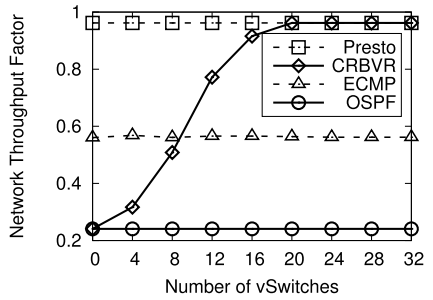


Fig. 13. Network Throughout Factor vs. Number of vSwitches for Topology (b).

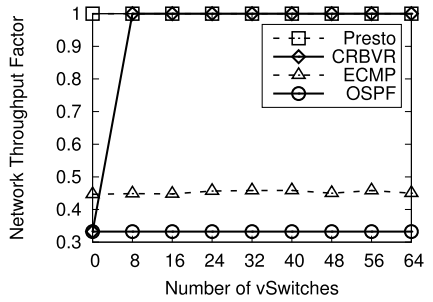


Fig. 14. Network Throughout Factor vs. Number of vSwitches for Topology (c).

16 vswitches respectively, Fig. 10 shows that the link load ratio is 98.3% and 54.6%, respectively. Moreover, the CRBVR algorithm can achieve the similar link load ratio compared with Presto by deploying a few vswitches in an SDN. For example, when we deploy 30, 16 and 16 vswitches on topologies (a), (b) and (c), respectively, both CRBVR and Presto can achieve almost the similar link load ratio while CRBVR saves the number of vswitches by about 70%, 50%, 75% compared with Presto on topologies (a), (b) and (c), respectively.

We observe the network throughput factor by changing the number of deployed vswitches. The simulation results in Figs. 12-14 show that the network throughput factor keeps almost unchanged for OSPF, ECMP and Presto. With more vswitched deployed in an SDN, the CRBVR algorithm can significantly improve the network throughput factor. For example, when there is no vswitch, the NTF of CRBVR is same as that of OSPF on three topologies, which is consistent with Figs. 9-11. When we deploy 30, 16, and 8 vswitches on topologies (a), (b), and (c) respectively, the NTF of our proposed algorithm is very close to that of Presto.

From the above simulation results, we find that our proposed CRBVR algorithm can achieve better routing performance with a small number of vswitches. In the following simulations, we deploy 30, 16, and 16 vswitches by default on three topologies, respectively.

The second set of simulations observes different performance metrics (*i.e.*, NCF, LLR and NTF) by changing the number of flows from 40K to 320K for four algorithms on three topologies. Figs. 15-17 show how the number of flows in an SDN affects the number of controllable flows. We find that the number of controllable flows is almost linearly increasing with the increasing number of flows in an SDN. For a given number (*e.g.*, 200K) of flows, the CRBVR algorithm can control 85,072 (42.5%), 100,078 (50.0%), and 106,096 (53.0%) flows on three topologies, respectively.

Figs. 18-20 show that the link load ratio rises by changing the number of flows for all these algorithms. With the increasing number of flows in an SDN, the OSPF, ECMP, CRBVR, and Presto algorithms occur congestion (*i.e.*, LLR is 1) in turn. Our CRBVR algorithm has decent link load ratio performance on all three topologies. For example, when there

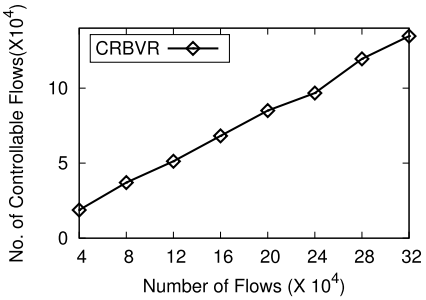


Fig. 15. Number of Controllable Flows vs. Number of Flows for Topology (a).

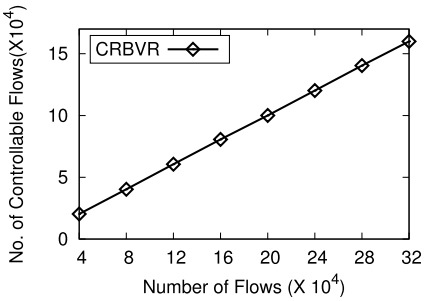


Fig. 16. Number of Controllable Flows vs. Number of Flows for Topology (b).

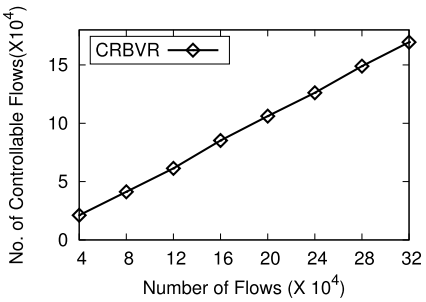


Fig. 17. Number of Controllable Flows vs. Number of Flows for Topology (c).

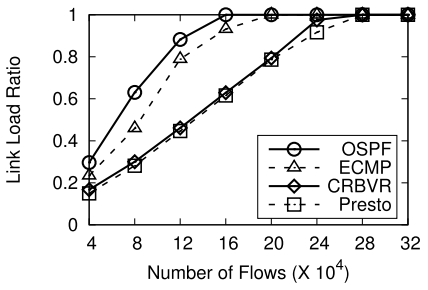


Fig. 18. Link Load Ratio vs. Number of Flows for Topology (a).

are 120K flows, the proposed CRBVR algorithm reduces link load ratio by about 47.7% compared with the OSPF method on topologies (a). Meanwhile, CRBVR can reduce the link load ratio by about 41.5% compared with the ECMP method, which requires extra group entries to implement multi-path transmission. Comparing with Presto, CRBVR only increases

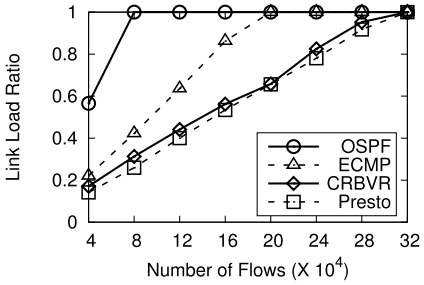


Fig. 19. Link Load Ratio vs. Number of Flows for Topology (b).

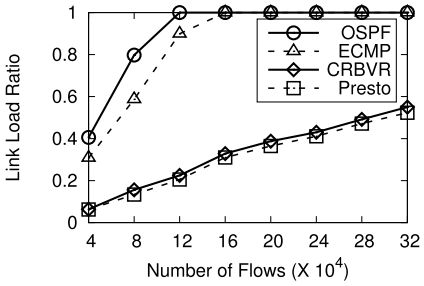


Fig. 20. Link Load Ratio vs. Number of Flows for Topology (c).

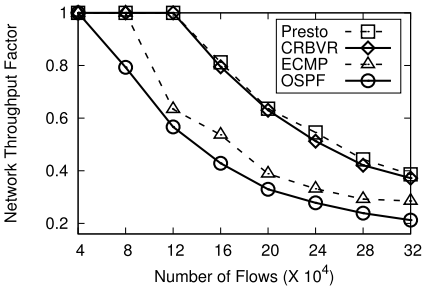


Fig. 21. Network Throughout Factor vs. Number of Flows for Topology (a).

the LLR by 3.2%, 5.0% and 5.6% on three topologies while using only 30%, 50% and 25% vswitches.

Figs. 21-23 indicate that the network throughput factor decreases with more and more flows for all algorithms on three topologies. CRBVR can perform better than OSPF and ECMP obviously. For example, when there are 160K flows on topology (a), CRBVR improves the network throughput factor by about 46.0% and 32.4% compared with the OSPF and ECMP methods, respectively. Meanwhile, the Presto method improves the network throughput factor by about only 2.1% while using 3.3 times vswitches compared with CRBVR. In a word, our CRBVR algorithm has better performance than OSPF and ECMP, and can achieve similar performance to Presto while only a small number of vswitches are deployed. We can get similar conclusions on topologies (b) and (c).

The third set of simulations observes the required number of vswitches to achieve a good tradeoff between network performance and deployment cost when there are 40K and 160K flows, respectively. As shown in Fig. 24, the link load ratio decreases with the increasing number of deployed vswitches in topology (a). In particular, when 30 vswitches are deployed, no matter 40K flows or 160K flows, CRBVR achieves similar

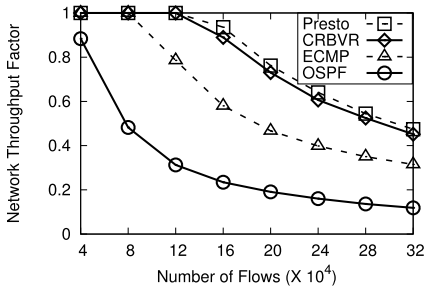


Fig. 22. Network Throughout Factor vs. Number of Flows for Topology (b).

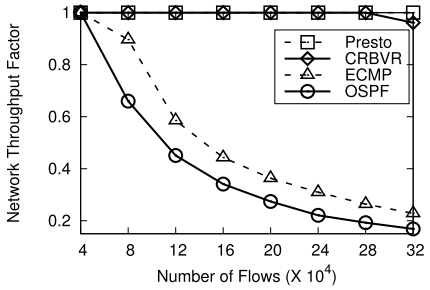


Fig. 23. Network Throughout Factor vs. Number of Flows for Topology (c).

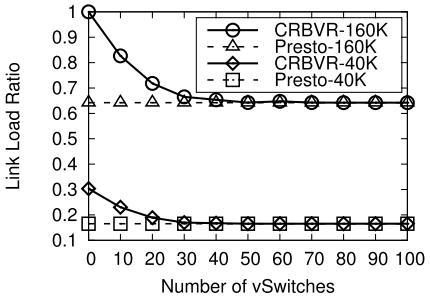


Fig. 24. Link Load Ratio vs. Number of vSwitches for Topology (a).

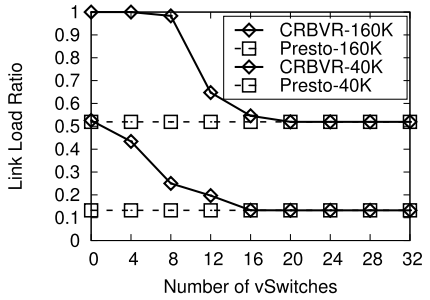


Fig. 25. Link Load Ratio vs. Number of vSwitches for Topology (b).

performance to Presto, and deploying more vswitches will not further improve the performance much. This implies that deploying 30 vswitches (which is 30% of the number of ingress switches) is the best choice for topology (a). We then exam network topologies (b) and (c). As shown in Figs. 25-26, there only deploys 16 vswitches, about 50% or 25% of the number of ingress switches in (b) and (c), respectively, to achieve a good tradeoff between performance and cost.

According to the simulation results, we can make some conclusions. First, by Figs. 6-14, as more vswitches are deployed,

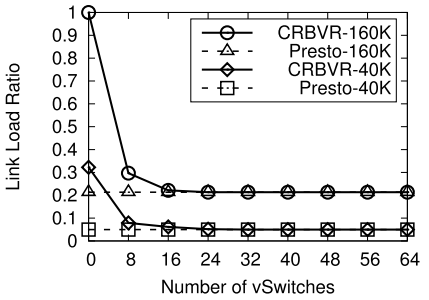


Fig. 26. Link Load Ratio vs. Number of vSwitches for Topology (c).

our CRBVR algorithm can control more flows and the network performance becomes better. Second, by Figs. 9-14 and 18-23, CRBVR has better routing performance compared with OSPF and ECMP on different topologies, which means our CRBVR algorithm can perform well on a wide range of occasions. Moreover, CRBVR can achieve similar routing performance compared with Presto while reducing the number of deployed vswitches by 70%, 50%, 75% on three topologies. Third, by Figs. 24-26, when implementing our CRBVR algorithm, we need to deploy a different number of vswitches for different topologies. However, as long as we deploy an appropriate number of vswitches in an SDN, we do not need to deploy more switches as the number of flows increases.

VI. Conclusion

In this paper, we have studied how to achieve load balancing through efficient vswitch deployment in an SDN. We formulate the joint optimization of vswitch deployment and routing problem as an integer linear program. A rounding-based algorithm with bounded approximation factors is proposed to solve the JVR problem. Some practical issues are discussed to enhance our load balancing mechanism. We implement the proposed algorithm on an SDN testbed for experimental studies and use simulations for large-scale investigation. The testing results on the SDN platform and the extensive simulation results on the Mininet show that our proposed method can reduce the link load ratio by about 41.5% compared with the ECMP method, and achieve almost the similar performance as Presto, by deploying a small number of vswitches.

References

[1] B. Pfaff *et al.*, "The openflow switch," Open Netw. Found., Menlo Park, CA, USA, Tech. Rep. ONF TS-006, 2012. [Online]. Available: http://openflowswitch.org

[2] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven wan," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[3] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. INFOCOM*, 2016, pp. 1–9.

[4] E. Haleplidis, J. H. Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for sdn," *J. Netw. Syst. Manage.*, vol. 23, no. 2, pp. 309–327, 2015.

[5] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. NSDI*, 2017, pp. 407–420.

[6] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 253–266.

[7] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[8] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.

[9] J. Zhou *et al.*, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. Ninth Eur. Conf. Comput. Syst.*, 2014, p. 5.

[10] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable ethernet for data centers," in *Proc. 8th Int. Conf. Emerg. Netw. Experim. Technol.*, 2012, pp. 49–60.

[11] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang, "Deploying default paths by joint optimization of flow table and group table in sdns," in *Proc. IEEE ICNP*, Oct. 2017, pp. 1–10.

[12] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. Netw. Syst. Design Implement. Symp. (NSDI)*, vol. 10. 2010, p. 19.

[13] J. Rasley *et al.*, "Planck: Millisecond-scale monitoring and control for commodity networks." *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 407–418, 2015.

[14] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[15] H. Xu, H. Huang, S. Chen, and G. Zhao, "Scalable software-defined networking through hybrid switching," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "On the effect of forwarding table size on SDN network utilization," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1734–1742.

[17] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up SDN control-plane using vSwitch based overlay," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experim. Technol.*, 2014, pp. 403–414.

[18] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, 2015.

[19] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[20] *Open vSwitch: Open Virtual Switch*. Accessed: Sep. 5, 2017. [Online]. Available: http://openvswitch.org/

[21] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. 16th Annu. Symp. Found. Comput. Sci.*, 1975, pp. 184–193.

[22] A. Srinivasan, "Approximation algorithms via randomized rounding: A survey," in *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN*. Polish, Warszawa, 1999, pp. 9–71.

[23] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for sdns," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10.

[24] T. Thomas, *OSPF Network Design Solutions*. London, U.K.: Pearson Education, 2003.

[25] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. North Chelmsford, MA, USA: Courier Corporation, 1998.

[26] *The Mininet Platform*. Accessed: Sep. 5, 2017. [Online]. Available: http://mininet.org/

[27] K.-K. Yap *et al.*, "Taking the edge off with Espresso: Scale, reliability and programmability for global Internet peering," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 432–445.

[28] J.-M. Kang, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Software-defined infrastructure and the SAVI testbed," in *Proc. Int. Conf. Testbeds Res. Infrastruct.*, 2014, pp. 3–13.

[29] *G.114: One-Way Transmission Time*, document G. 114, R. ITU-T and I. Recommend, 2000, vol. 18.

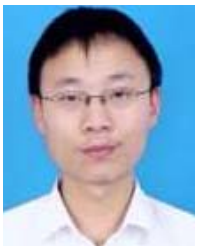
[30] *The Network Topology From the Monash University*. Accessed: Sep. 5, 2017. [Online]. Available: http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies

[31] S. Azizi, F. Safaei, and N. Hashemi, "On the topological properties of hyperx," *J. Supercomput.*, vol. 66, no. 1, pp. 572–593, 2013.

[32] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas.*, 2009, pp. 202–208.

[33] B. Gnedenko and I. Kovalenko, *Introduction to Queuing Theory* (Mathematical Modeling). Boston, MA, USA: Birkhäuser Boston, 1989.

Xuwei Yang received the B.S. degree in network engineering from Chang'an University in 2016. He is currently pursuing the M.S. degree in computer science with the University of Science and Technology of China. His main research interests include software-defined networks.



Hongli Xu (M'08) received the B.S. degree in computer science from the University of Science and Technology of China in 2002, and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2007. He is currently an Associate Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or co-authored over 70 papers, and held about 30 patents. His main research interests include software-defined networks, cooperative communication, and vehicular ad hoc network.

Liusheng Huang received the M.S. degree in computer science from the University of Science and Technology of China in 1988. He is currently a Senior Professor and Ph.D. Supervisor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or co-authored six books and over 300 journal/conference papers. His research interests are in the areas of Internet of Thing, vehicular ad hoc network, information security, and distributed computing.

Gongming Zhao received the B.S. degree in Internet of Things from Shandong University, Jinan, China, in 2015. He is currently pursuing the Ph.D. degree in computer science with the University of Science and Technology of China. His main research interests are software-defined network and Internet of Things.

Peng Xi received the B.S. degree in computer science and the M.S. degree in computer software and theory from the University of Science and Technology of China in 2004 and 2010, respectively, where he is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. He is currently a Lecturer with the School of Educational Science, Anhui Normal University. His main research interest is software-defined networks, wireless networks, and network security.

Chunming Qiao is currently a SUNY Distinguished Professor and the Chair of the CSE Department. He has been leading the Lab for Advanced Network Design, Evaluation and Research University at Buffalo, The State University of New York since 1993. His current research interests cover not only the safety and reliability of various cyber physical systems (such as transportation systems with connected and autonomous vehicles, critical infrastructures involving power grid and communications networks, and cloud services), but also algorithms and protocols for the Internet of Things, including smartphone-based systems and applications. He has published extensively with an h-index of over 69 (according to Google Scholar). Several of his papers have received the best paper awards from IEEE and Joint ACM/IEEE venues. He holds seven U.S. patents and served as a consultant for several IT and Telecommunications companies since 2000. His research has been featured in *BusinessWeek*, *Wireless Europe*, *CBC*, and *New Scientists*. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols.