TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 08/13 pp 440–452
DOI: 10.26599 / TST.2018.9010084
Volume 23, Number 4, August 2018

Hybrid Routing by Joint Optimization of Per-Flow Routing and Tag-Based Routing in Software-Defined Networks

Gongming Zhao, Liusheng Huang*, Ziqiang Li, and Hongli Xu

Abstract: In recent years, Software-Defined Networks (SDNs) have become a promising technology to improve network utilization. However, limited flow table size and long deployment delays may result in low network performance in large-scale networks and a poor user experience. While a typical solution to this issue is routing aggregation (i.e., wildcard routing), the aggregation feasibility problem and reduced network performance may be encountered. To address this dilemma, we first design a novel wildcard routing scheme, called the Tag-based Rule Placement Scheme (TRPS). We then formulate a Hybrid Routing by Joint optimization of Per-flow routing and Tag-based routing (HR-JPT) problem, and prove its NP-hardness. An algorithm with a bounded approximation factor is designed for this problem, and the proposed methods are implemented on a Mininet platform. Extensive simulation results show that our methods are efficient for wildcard/hybrid routing. For example, our proposed tag-based wildcard rule placement scheme can reduce the number of required rules by about 65% on average compared with previous wildcard routing methods. Our proposed hybrid routing algorithm can increase network throughput by about 43% compared with existing hybrid routing solutions.

Key words: Software Defined Networks (SDNs); load balancing; per-flow routing; tag-based routing; flow table size constraint; deployment delay constraint

1 Introduction

The concept of Software-Defined Network (SDN) is a new networking paradigm that decouples the control and data forwarding planes of network devices[1, 2]. More specifically, the controller constitutes the control plane of an SDN, and determines the forwarding path of each flow with a centralized control manner. SDN switches constitute the data plane of an SDN and the response for data forwarding of each flow. Because the controller features global visibility and full control capacity over the whole network, SDN users can composite application programs run on top of the controller to monitor and manage entire networks (e.g., traffic engineering, heavy hitter identification, and proper routing[2]) in an efficient and centralized manner. Thus, SDN has been used in different fields, such as campus networks and data center networks.

The growth of Internet services has promoted the popularity of many large-scale data intensive applications (e.g., video conferences, cloud services, and financial data analysis). As a result, large-scale networks are increasingly experiencing burst flows. For example, in a practical datacenter network with 1500 server operational clusters, the average arrival rate can reach 10^5 flows per second for core switches[3], which means, if per-flow

• Gongming Zhao, Liusheng Huang, Ziqing Li, and Hongli Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou 215123, China. E-mail: zgm1993@mail.ustc.edu.cn; lshuang@ustc.edu.cn; lzqrush@mail.ustc.edu.cn; xuhongli@ustc.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2017-08-05; revised: 2018-01-18; accepted: 2018-01-30

routing is performed in this situation, tens of thousands flow rules may be required on each switch. However, due to the high price and energy consumption of Ternary Content Addressable Memory (TCAM) units, an SDN switch usually contains only a few thousand flow rules[4]. Recent testing results show a 3.3 ms delay when inserting a single flow rule into a flow table on a commodity switch[5]. Deployment delays are critical for many applications, e.g., the authors of Ref. [6] showed that a 100 ms delay causes a 1% drop in revenue at Amazon and a 400 ms delay causes a 5%–9% decrease in traffic at Google. Thus, considering flow table and deployment delay constraints, *per-flow routing is impractical in large-scale networks*.

To solve the flow table and deployment delay constraints problem, the classical design principle is destination host-based aggregate routing (i.e., wildcard routing). Unfortunately, as many datacenter networks contain millions of virtual or physical hosts, this scheme also requires tens of thousands of TCAM rules to deploy host-based wildcard routing in large-scale networks[7]. Thus, the deployment delay and table size constraints in large-scale networks cannot be solved by destination host-based aggregate routing, especially for core switches, which may process a vast number of flows. Besides, the network performance (e.g., load balancing) cannot be guaranteed by aggregate routing.

This paper first studies wildcard routing and proposes a novel scheme called the Tag-based Rule Placement Scheme (TRPS). We then focus on the Hybrid Routing by Joint optimization of Per-flow routing and Tag-based routing (HR-JPT) problem. To the best of our knowledge, our work is the first to deploy hybrid routing by joint optimization of per-flow routing and tag-based routing under flow table size and deployment delay constraints for load balancing. The main contributions of this paper are as follows:

(1) We propose a TRPS for wildcard routing. This scheme uses a tag to aggregate flows to overcome the prefix/suffix-based flow aggregation constraint and combines proactive rule placement and reactive rule placement. Here, the rules between switches are pre-deployed and the tag of flows are reactively installed to significantly reduce the deployment delay and number of rules required for wildcard routing, especially for core switches.

(2) We formally define HR-JPT problem and prove its NP-hardness. An approximation algorithm, Rounding-based Route Joint Deployment (RRJD), with a bounded approximation factor is proposed to solve this problem.

(3) We implement the proposed TRPS solution and its hybrid routing algorithm on a Mininet platform. Extensive simulation results show that our proposed tag-based wildcard rule placement scheme can reduce the number of required rules by about 65% on average compared with previous wildcard routing methods and that our proposed hybrid routing algorithm can increase network throughput by about 43% compared with existing hybrid routing solutions.

The rest of this paper is organized as follows. Section 2 illustrates the TRPS mechanism for wildcard routing. The network model and the definition of the HR-JPT problem are introduced in Section 3. We propose the RRJD to deal with the HR-JPT problem and analyze its approximate performance in Section 4. The testing results are given in Section 5 and Section 6 discusses related works. We conclude this paper in Section 7.

2 TRPS

As shown in Fig. 1, a typical datacenter network architecture consists of three-level trees of switches. A three-tiered design can typically support tens of thousands of terminals[8]. Under this circumstance, the bottleneck of the datacenter networks usually is the core switches, which encounter a massive number of flows[9]. Due to the development of Open vSwitch and Overlay technologies, the resource shortages of edge switches have been eased[10], which means the switches v_1, v_2, v_3, v_4, v_5, and v_6 are highly likely to encounter resource shortages and deployment delay problems in Fig. 1. Our proposed scheme mainly concentrates on solving the deployment delay and flow table constraints of core switches (or internal switches).

An Openflow-based flow rule mainly contains match fields, priority, counters, and actions. Match fields match against packet headers and consist of in_port, $vlan\_id$, eth_dst, eth_src, ip_dst, and ip_src, among others. Note that, with the update of the Openflow protocol, the number of supported match fields is increased and some reserved match fields are maintained. Thus, we can define an

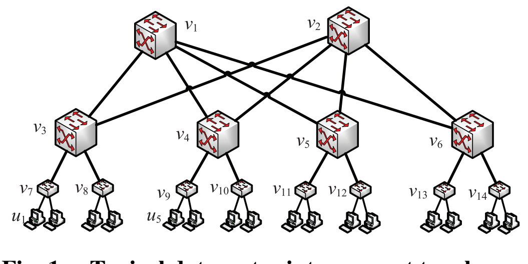


Fig. 1 Typical datacenter interconnect topology.

$SwID$ field (by choosing one reserved match field) as a new match field to perform the TRPS. The work in Ref. [9] showed that the operation of adding or deleting a tag is easy for the controller, so the tag-based method is practical. The scheme details are as follows.

The scheme first installs the proactive switch-level routing rules. The controller discovers the entire network topology using the discovery interface in Openflow[11]. The controller then assigns a unique ID (e.g., v_1, v_2, v_3) for each switch and computes the shortest path of each switch pair. Next, the controller sends flow-mod messages[11] to all of the switches to install switch-level routing rules. In other words, we assume the network contains m switches, for each switch, $m-1$ switch-level rules are installed for all other switches based on $SwID$. Note that, the number of switches is much smaller than the number of terminals in a network and much smaller than the number of flow rules[7], which means the number of proactive rules is far fewer than the number of flow rules on each switch.

The proposed TRPS mechanism builds reactive routing rules when the ingress switch reports the packet header of a flow to the controller. This process may be summarized as follows: (1) The controller first floods the APR request to all the edge switches using packet-out messages[11] and locates the destination terminal. (2) The controller then chooses one optimal path for this flow and sends flow-mod messages to install rules on the corresponding switches. The path selection algorithm will be addressed in the next two sections of this paper. (3) If the controller performs per-flow routing, all of the switches along with the selected path should be reactively installed with one rule for this flow via a method identical to that for traditional per-flow routing. (4) If the controller performs tag-based routing, a command is sent to the ingress switch to install one rule, which matches flows with this destination terminal and adds this egress switch's ID to the corresponding packet headers. Thus, all of the matched flows can be forwarded by proactive tag-based routing rules. The controller also sends a command to the egress switch to install one rule, which matches flows with the destination terminal, deletes the ID, and forwards matched flows to the port that connects to the terminal.

A few points should be emphasized here: (1) With the development of virtualization technology, the edge switches of datacenters have been mainly changed to Open vSwitches, which means the resource shortages of edge switches have been eased[10]. (2) The priorities of per-flow routing rules are higher than those of tag-based routing rules, which means we can perform per-flow routing for some specific flows. (3) The $SwID$ field is transparent to users and can easily be implemented on today's SDN architectures.

To obtain a better understanding of this mechanism, an example is presented in Fig. 2. The controller first assigns four unique $SwID$: v_1, v_2, v_3, v_4 to corresponding switches. Next, the controller installs the tag-based switch-level routing rules into the switches. We only list the dst and $SwID$ match fields in Fig. 2 for simplicity and only analyze switch v_1 to illustrate switch-level rules. v_1 installs three switch-level rules for the three other switches. For

Flow Table (v_1)

dst	*SwID*	Action
*	$SwID = v_2$	Output=2
*	$SwID = v_3$	Output=3
*	$SwID = v_4$	Output=2
dst = 3.0.0.1		Output=3
dst = 2.0.0.1		$SwID = v_4$

Flow Table (v_4)

dst	*SwID*	Action
*	$SwID = v_1$	Output=4
*	$SwID = v_2$	Output=4
*	$SwID = v_3$	Output=3
dst = 3.0.0.1		Output=2
dst = 2.0.0.1		Del SwID Output=1

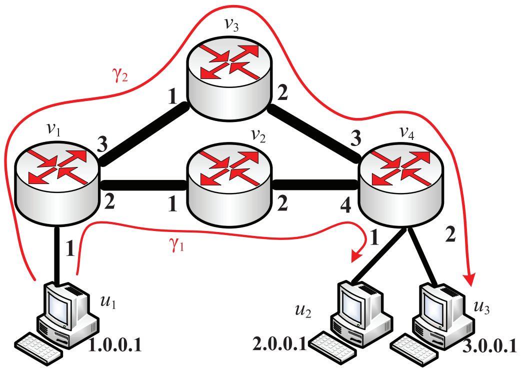


Fig. 2 Illustration of the tag-based rule placement scheme. The two plots on the left denote the flow tables of switches v_1 and v_4. The $SwID$ rules are pre-deployed when the topology is created. The dst rules are reactively deployed based on the controller commands. When v_1 receives packets destined for u_2 with IP address 2.0.0.1, this switch will set $SwID = v_4$ for the flow and forward this flow based on switch level. When v_1 receives packets destined for u_3 with IP address 3.0.0.1, this switch will forward the flow based on per-flow level.

example, the rule "$SwID = v_4, output = 2$" denotes that flows labeled $SwID = v_4$ will be forwarded to port 2. Note that all flows labeled with this $SwID$ have the same egress switch v_4, which will be introduced later. After all the proactive rules are installed, three proactive rules are found on each switch and two flows arrive: one from 1.0.0.1 to 2.0.0.1 (denoted by γ_1) and the other from 1.0.0.1 to 3.0.0.1 (denoted by γ_2). We assume that the controller decides to forward γ_1 by tag-based routing and forward γ_2 by per-flow routing. Thus, the controller only needs to interact with the ingress switch v_1 and egress switch v_4 to install tag-based routing rules. For the ingress switch v_1, we install a rule that matches flows with destination 2.0.0.1, this rule attaches $SwID$ to v_4 so that they can be forwarded by tag-based routing rules. For the egress switch v_4, we install a rule that forwarded flows to the port 1, which matches flows with destination 2.0.0.1, so that these flows can reach the destination terminal. The controller needs to interact with v_1, v_3, and v_4 to install three rules for γ_2, similar to the traditional per-flow routing rules.

By building rules on switch levels, TRPS presents the following advantages compared with other schemes:

• **Reduction of the number of required flow rules.** The number of switches is always much smaller than the number of hosts in a network. For example, assuming n switches and m hosts ($m \gg n$), if we build rules on the host level, a maximum of $m \times (m-1)$ rules is needed for the core switch. Even we build destination terminal-based rules, the core switch also needs m rules. By using TRPS, however, only a maximum of $n-1$ rules is needed for the core switches.

• **Decreases in deployment delay.** TRPS combines proactive routing with reactive routing. Since switch-level routing has been deployed in advance, the controller only needs to install two rules on the ingress and egress switches for each request. Letting ψ denote the average length of all the paths, ψ clearly increases along with the growing network and is usually greater than 5. The traditional scheme usually needs to install ψ rules on average for each request. As such, our proposed wildcard routing scheme is efficient in reducing deployment delays.

• **Relief of the controller load.** In our TRPS, the controller only needs to send flow-mod commands to the ingress and egress switches. Fewer packet-in messages[11] are also sent by the switches to the controller. For example, terminal u_1 sends packets to terminal u_5 in Fig. 1 via the traditional scheme, and the controller sends flow-mod commands to v_7, v_3, v_1, v_4, and v_9 to install flow rules. If the flow rule is deployed on v_7 before v_3, switch v_7 can then forward this flow to switch v_3. However, since switch v_3 has not deployed the specified flow rule for this flow, or cannot find the matched rule, v_3 will report the packet header to the controller, which results in several packet-in calls. In this example, the switch may encounter five (the length of path) packet-in calls at most. Using the TRPS, however, only one packet-in call (on the ingress switch v_7) and at most two packet-in calls (on the ingress switch v_7 and the egress switch v_9) are encountered. Thus, TRPS can greatly decrease the communication between switches and the controller and retain the stability of the latter.

In summary, TRPS has the potential to achieve smaller deployment delays and maintain stable network performance with limited TCAM rules.

3 Definition of HR-JPT

In this section, we first introduce the network and flow models and then illustrate the motivation of hybrid routing. Finally, we define the HR-JPT problem and prove its NP-hardness.

3.1 Network and flow models

An SDN typically consists of three device sets: a terminal set, $U = \{u_1, ..., u_m\}$, with $m = |U|$; an SDN switch set, $V = \{v_1, ..., v_n\}$, with $n = |V|$; and a cluster of controllers. The controllers response to route selection of all the flows and do not participate in packet forwarding in a network. These switches and terminals comprise the data plane of an SDN. Thus, in view of the data plane, the network topology can be modeled by a directed graph $G = (U \cup V, E)$, where E is the link set in the network. For ease of expression, let $c(e)$ and $T(v)$ denote the capacity of link $e \in E$ and the number of available rules of switch $v \in V$ in graph G, respectively.

A set of bursty flows, denoted by $\Gamma = \{\gamma_1, ..., \gamma_{|\Gamma|}\}$, arrive in the network. By collecting flow statistics information from switches, the controller can estimate the size (or intensity) of each flow $\gamma \in \Gamma$ as $f(\gamma)$. Similar to many previous works[12, 13], in this paper, we adopt the unsplittable flow mode because of its simplicity, which means each flow can be forwarded through only one forwarding path to save flow rules and reduce the complexity of flow management.

3.2 Motivation for hybrid routing

TRPS can save flow rules resources, reduce deployment delays, and relieve the load of the controller to a greater extent than other routing schemes. However, TRPS is a wildcard routing scheme, which means many flows may be aggregated and forwarded through the same path. Thus, this scheme may cause link congestion and decrease network performance (e.g., network throughput, transmission delay, and packet loss).

To address this dilemma, we can leverage the strength of per-flow routing to improve network performance. A flow can be forwarded through an individual forwarding path by per-flow routing, so we can select some (elephant) flows to be forwarded through per-flow routing to improve network performance and other flows to be forwarded through tag-based wildcard routing to save flow table resources, decrease deployment delays, and relieve the controller load.

In the next section, we study the HR-JPT problem.

3.3 Hybrid routing by the HR-JPT problem

For each flow $\gamma \in \varGamma$, we explore a set of feasible paths $\mathcal{P}_\gamma$ from source to destination. The tag-based path $h(\gamma)$ also belongs to the set $\mathcal{P}_\gamma$ (i.e., $h(\gamma) \in \mathcal{P}_\gamma, \forall \gamma \in \varGamma$). Note that, each tag-based path (i.e., $h(\gamma)$) can be shared by many flows and is pre-deployed on the switch level (which may be the shortest path from ingress switch to egress switch) as illustrated in Section 2. For ease of expression, let $\mathcal{P}'_\gamma = \mathcal{P}_\gamma - h(\gamma)$ denote the feasible per-flow paths set of flow γ, and p_{ie} denote the set of ingress and egress switches of path $p \in \mathcal{P}_\gamma$. Our objective is to select one optimal path for each flow to achieve network load balancing under flow table size and deployment delay constraints.

We formulate the HR-JPT problem into a non-linear program as follows. Let variable $y_\gamma^p \in \{0,1\}$ denote whether flow γ selects the feasible path $p \in \mathcal{P}_\gamma$. z_v^u denotes whether a tag-based flow rule must be installed for the terminal u on switch v. H is the union of all $h(\gamma)$. Because the deployment delay is linearly associated with the number of rules that will be installed on each switch, we can use a conversion factor $\omega(v)$ to combine the two constraints as one constraint for simplicity. For example, the number of available rules $T(v)$ on switch v is 2000 and we want to be able to forward these flows within 3.3 s (denoted by T_0). About 3.3 ms (denoted by t_0) is necessary to insert a rule[5], which means, considering the flow table constraint, we can only install 2000 rules at most. Considering the deployment delay constraint, we can only install 3.3 s $\div$ 3.3 ms $=$ 1000 rules at most. Thus, let $\omega(v) = 0.5$ satisfy both constraints. In other words, we set

$$\omega(v) = \min\left\{\frac{T_0}{t_0 \cdot T(v)}, 1\right\}, \quad \forall v \in V \tag{1}$$

HR-JPT solves the following problem:

$$\min \quad \lambda,$$

$$\text{s.t.}\begin{cases}\sum\limits_{p\in\mathcal{P}_\gamma} y_\gamma^p = 1, & \forall \gamma \in \varGamma;\\ y_\gamma^p \leqslant z_v^{d(p)}, & \forall v \in p_{ie}, p \in H;\\ \sum\limits_{\gamma\in\varGamma}\sum\limits_{v\in p:p\in\mathcal{P}'_\gamma} y_\gamma^p + \sum\limits_{u\in U} z_v^u; \leqslant \omega(v)\cdot T(v), & \forall v \in V;\\ \sum\limits_{\gamma\in\varGamma}\sum\limits_{e\in p:p\in\mathcal{P}_\gamma} y_\gamma^p f(\gamma) \leqslant \lambda \cdot c(e), & \forall e \in E;\\ y_\gamma^p \in \{0,1\}, & \forall p, \gamma;\\ z_v^u \in \{0,1\}, & \forall u \in U, v \in V\end{cases} \tag{2}$$

The first set of equations means that each flow will be assigned a feasible path from source to destination. Thus, for each flow $\gamma \in \varGamma$, only one path $p \in \mathcal{P}_\gamma$ can be chosen (i.e., $y_\gamma^p = 1$) as the final forwarding path. The second set of inequalities denotes that, if a flow $\gamma \in \varGamma$ chooses $h(\gamma)$ as its forwarding path, then the ingress and egress switches of this forwarding path $h(\gamma)$ should install a tag-based rule for destination u (i.e., $z_v^u = 1$). As illustrated in Section 2, we only need to install rules on the ingress and egress switches of the tag-based path. The third set of inequalities denotes the flow table and the deployment delay constraints. The fourth set of inequalities expresses that the traffic load on each link e does not exceed the $\lambda \cdot c(e)$, where λ is the load balancing factor. Our objective is to minimize λ.

Theorem 1 The HR-JPT problem defined in Eq. (2) is an NP-hard problem.

Proof We consider a special example of the HR-JPT problem, in which no flow table and deployment delay constraints exist. We are then able to deploy the routes of all the flows in an SDN to achieve load balancing under a link capacity constraint. In other words, the flow in the network becomes an unsplittable multi-commodity flow with minimum congestion problem[14], which is NP-hard. Since the multi-commodity flow problem is a special case of our problem, the HR-JPT problem is also NP-hard. ■

4 Description of the HR-JPT Problem Algorithm

Due to its NP-hardness, HR-JPT problem is difficult to optimally solve. In this section, we first present an approximation algorithm RRJD to solve the HR-JPT problem (Section 4.1) and then analyze the approximate performance of this algorithm (Section 4.2). Finally, we give the complete algorithm description (Section 4.3).

4.1 Approximation algorithm to solve the HR-JPT problem

This section presents the RRJD algorithm to solve the HR-JPT problem. To solve this problem in polynomial time, we first relax the integer program to a linear program as in Eq. (3).

$$\min \quad \lambda,$$
$$\text{s.t.}\begin{cases}\sum\limits_{p\in\mathcal{P}_\gamma} y_\gamma^p=1, & \forall\gamma\in\varGamma;\\ y_\gamma^p\leqslant z_v^{d(p)}, & \forall v\in p_{ie}, p\in H;\\ \sum\limits_{\gamma\in\varGamma}\sum\limits_{v\in p:p\in\mathcal{P}'_\gamma} y_\gamma^p+\sum\limits_{u\in U} z_v^u\leqslant\omega(v)\cdot T(v), & \forall v\in V;\\ \sum\limits_{\gamma\in\varGamma}\sum\limits_{e\in p:p\in\mathcal{P}_\gamma} y_\gamma^p f(\gamma)\leqslant\lambda\cdot c(e), & \forall e\in E;\\ y_\gamma^p\in[0,1], & \forall p,\gamma;\\ z_v^u\in[0,1], & \forall u\in U, v\in V\end{cases} \tag{3}$$

By relaxing this assumption, y_γ^p and z_v^u are fractional, which means we assume that each flow can be split and forwarded through multiple paths. Since Eq. (3) is a linear program, we can solve it in polynomial time with a linear program solver (e.g., CPLEX[15]). Assume that the optimal solution is denoted by $\widetilde{y}_\gamma^p,\forall\gamma,p$ and the optimal result is denoted by $\widetilde{\lambda}$. As the linear program is a relaxation of the HR-JPT problem, $\widetilde{\lambda}$ is a lower-bound result for this problem.

More specifically, for each flow $\gamma\in\varGamma$, we select a feasible path $p\in P_\gamma$ with the probability of $\widetilde{y}_\gamma^p$ for flow γ. If $\exists p\in P_\gamma, \widehat{y}_\gamma^p=1$, flow γ selects $p\in P_\gamma$ as its finally route path. For tag-based routing, $\widehat{x}_v^u=\max\{\widehat{y}_\gamma^p,\forall d(p)=u, v\in p_{ie}, p\in H,\gamma\in\varGamma\}$. In this manner, we have determined the final route paths for all flows. The RRJD algorithm is formally described in Algorithm 1.

We now discuss the time complexity of the RRJD algorithm. The first step mainly solves the linear program. Since the linear program contains a polynomial number of variables, polynomial times are required to solve this program. The second step uses randomized rounding for route selection, with the time complexity $f\cdot|\varGamma|$, where f is the maximum number of feasible paths for all flows and $|\varGamma|$ is the number of flows. As a result, the total time complexity of RRJD is polynomial.

4.2 Approximate performance analysis

We give two well-known lemmas for probability analysis.

Theorem 2 (Chernoff Bound) Given n independent

Algorithm 1 RRJD: Rounding-based Route Joint Deployment

```
1: Step 1: Solving the Relaxed HR-JPT Problem
2: Construct a linear program $LP_1$ based on Eq. (2)
3: Obtain the optional solution $\widetilde{y}$
4: Step 2: Route Selection for Load Balancing
5: Derive an integer solution $\widehat{y}_\gamma^p$ by randomized rounding
6: for each switch $v\in V$, each terminal $u\in U$ do
7:     $\widehat{x}_v^u=\max\{\widehat{y}_\gamma^p,\forall d(p)=u, v\in p_{ie}, p\in H,\gamma\in\varGamma\}$
8:     if $\widehat{x}_v^u=1$ then
9:         Install a tag-based rule on switch $v$ for destination $u$
10:    end if
11: end for
12: for each flow $\gamma\in\varGamma$ do
13:     for each routh path $p\in P_\gamma$ do
14:         if $\widehat{y}_\gamma^p=1$ then
15:             Appoint path $p$ for flow $\gamma$
16:         end if
17:     end for
18: end for
```

variables: $x_1,x_2,...,x_n$, where $\forall x_i\in[0,1]$. Let $\mu=\mathbb{E}[\sum_{i=1}^n x_i]$. Then, $\mathbf{Pr}\left[\sum\limits_{i=1}^n x_i\geqslant(1+\epsilon)\mu\right]\leqslant \mathrm{e}^{\frac{-\epsilon^2\mu}{2+\epsilon}}$, where ϵ is an arbitrarily positive value.

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

We define a variable α as follows:

$$\alpha=\min\{\min\{\frac{\widetilde{\lambda}c_{\min}}{f(\gamma)},\gamma\in\varGamma\},\min\{T(v),v\in V\}\} \tag{4}$$

Link Capacity Constraint. The load of link e from each flow γ is defined as a variable $x_{e,\gamma}$. We have

$$\mathbb{E}\left[\sum_{\gamma\in\Gamma}x_{e,\gamma}\right]=\sum_{\gamma\in\Gamma}\mathbb{E}\left[x_{e,\gamma}\right]=\sum_{\gamma\in\Gamma}\sum_{e\in p:p\in\mathcal{P}_\gamma}\widetilde{y}_\gamma^p f(\gamma)\leqslant\widetilde{\lambda}c(e) \tag{5}$$

Combining Eq. (5) and the definition of α in Eq. (4), we have

$$\begin{cases}\dfrac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}c(e)}\in[0,1],\\ \mathbb{E}\left[\sum_{\gamma\in\Gamma}\dfrac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\right]\leqslant\alpha\end{cases} \tag{6}$$

Then, by applying Theorem 2, we assume that ρ is an arbitrary positive value. Thus

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{e,\gamma}\cdot\alpha}{\widetilde{\lambda}\cdot c(e)}\geqslant(1+\rho)\alpha\right]\leqslant\mathrm{e}^{\frac{-\rho^2\alpha}{2+\rho}} \tag{7}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\geqslant(1+\rho)\right]\leqslant \mathrm{e}^{\frac{-\rho^2\alpha}{2+\rho}}\leqslant\frac{\mathcal{F}}{n^2} \quad (8)$$

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

The solution for Eq. (8) is expressed as

$$\rho\geqslant\frac{\log\frac{n^2}{\mathcal{F}}+\sqrt{\log^2\frac{n^2}{\mathcal{F}}+8\alpha\log\frac{n^2}{\mathcal{F}}}}{2\alpha},\quad n\geqslant 2 \quad (9)$$

Lemma 1 The proposed RRJD algorithm achieves the approximation factor of $\frac{4\log n}{\alpha}+3$ for link capacity constraints.

Proof Set $\mathcal{F}=\frac{1}{n^2}$. Equation (8) is transformed into

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\geqslant(1+\rho)\right]\leqslant\frac{1}{n^4},\rho\geqslant\frac{4\log n}{\alpha}+2 \quad (10)$$

By applying Lemma 3, we have

$$\mathbf{Pr}\left[\bigvee_{e\in E}\sum_{\gamma\in\Gamma}\frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\geqslant(1+\rho)\right]\leqslant$$
$$\sum_{e\in E}\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{x_{e,\gamma}}{\widetilde{\lambda}\cdot c(e)}\geqslant(1+\rho)\right]\leqslant$$
$$n^2\cdot\frac{1}{n^4}=\frac{1}{n^2},\quad\rho\geqslant\frac{4\log n}{\alpha}+2 \quad (11)$$

Note that the third inequality holds, a maximum of n^2 links exist in a network with n switches. The approximation factor of our algorithm is $\rho+1=\frac{4\log n}{\alpha}+3$. ■

Flow Table Constraint. Note that the approximate performance analysis of the **deployment delay constraint** is identical to the analysis of the flow table constraint, thus we omit it here to preserve space. Similar to that for the link capacity constraint, we define random variables δ, $t_{v,\gamma}$, and $\mathcal{F}$. Similar to Eqs. (5)–(8), we have

$$\delta\geqslant\frac{\log\frac{n}{\mathcal{F}}+\sqrt{\log^2\frac{n}{\mathcal{F}}+8\alpha\log\frac{n}{\mathcal{F}}}}{2\alpha},\quad n\geqslant 2 \quad (12)$$

We give the approximation performance as follows.

Lemma 2 After a rounding process, the total number of flow rules on any switch v will not exceed the constraint $T(v)$ by a factor of $\frac{3\log n}{\alpha}+3$ for the HR-JPT problem.

Proof Set $\mathcal{F}=\frac{1}{n^2}$. Apparently $\mathcal{F}\to 0$ when $n\to\infty$. With respect to Eq. (12), we set

$$\delta=\frac{\log\frac{n}{\mathcal{F}}+\log\frac{n}{\mathcal{F}}+4\cdot\alpha}{2\cdot\alpha}=$$
$$\frac{6\log n+4\cdot\alpha}{2\cdot\alpha}=\frac{3\log n}{\alpha}+2 \quad (13)$$

Then Eq. (13) guarantees $1+\delta=\frac{3\log n}{\alpha}+3$ and $\mathcal{F}=\frac{1}{n^2}$, which concludes the proof. ■

Approximation Ratio. With these analyses, we know the approximation factors for the link capacity and flow table constraints (deployment delay constraint) are $\frac{4\log n}{\alpha}+3$ and $\frac{3\log n}{\alpha}+3$, respectively. By using the proposed RRSD approximate algorithm, we can scale flows by a factor of $\frac{4\log n}{\alpha}+3$ to satisfy the link capacity constraint and by a factor of $\frac{3\log n}{\alpha}+3$ to satisfy the flow table constraint (deployment delay constraint). For example, let $\alpha=100$, $n=100$, then, the approximation factors for the link capacity and flow table constraints (deployment delay constraint) are 3.08 and 3.06, respectively. Thus, our RRSD approximate algorithm can achieve the constant bi-criteria approximation for the HR-JPT problem.

4.3 Complete RRJD algorithm description

While the RRJD algorithm can almost achieve the bi-criteria approximation performance for the HR-JPT problem, the randomized rounding mechanism may not guarantee that the flow table size constraint (or deployment delay constraint) is always met. Here, we describe the complete RRJD algorithm to satisfy the flow table size constraint (deployment delay constraint). The complete algorithm is formally described in Algorithm 2.

The complete version of the algorithm consists of three steps, the first two of which are identical to those in Algorithm 1. The third step will remove some per-flow rules so that the flow table size constraint (deployment delay constraint) is satisfied on all switches. Initially, V' denotes the set of switches that violate the flow table size constraint (deployment delay constraint). We rank all of these switches in decreasing order of deployment delay. For each switch $v\in V'$, the set of flows that pass through switch v and use the per-flow rules is denoted by Γ_v^f. We rank these flows in decreasing order of the value $\widetilde{y}_\gamma^{h(\gamma)}$, which indicates the probability of being forwarded by tag-based rules. For each flow $\gamma\in\Gamma_v^f$, we remove the per-flow

Algorithm 2 Complete RRJD Algorithm Description

```
1: Step 1: Same as that in Algorithm 1
2: Step 2: Same as that in Algorithm 1
3: Step 3: Removing Some Rules
4: Put all switches that violate the flow table size and deployment delay
   constraints in set $V'$
5: while $V' \neq \varnothing$ do
6:     Select a switch $v \in V'$ with the maximum deployment delay (the
       maximum number of required flow rules)
7:     The flows that pass through switch $v$ and use the per-flow rules
       are denoted by $\Gamma_v^f$
8:     Rank $\gamma \in \Gamma_v^f$ in the decreasing order of $\widetilde{y}_\gamma^{h(\gamma)}$
9:     for each flow $\gamma \in \Gamma_v^f$ do
10:        Remove the per-flow rules on all switches along the per-flow
           path of flow $\gamma$ and forwarded by tag-based path
11:        if The constraints on switch $v$ is satisfied then
12:            break
13:        end if
14:    end for
15:    $V' = V' - \{v\}$
16: end while
```

rules on all switches along the per-flow path of flow γ and forwarded by the tag-based path until the constraints on switch v are satisfied. We then remove switch v from set V'. This iteration is terminated when no other switch exists in set V'.

5 Simulation Results

In this section, we first introduce the simulation settings and performance metrics (Section 5.1). We then compare the proposed method with previous methods by running extensive simulations (Section 5.2). Note that our simulations are executed on the Mininet platform[16] with a POX controller, which is a widely-used simulator specified for SDN.

5.1 Performance metrics and setting

The proposed scheme and algorithm focus on datacenter networks, so we choose two typical datacenter topologies:

(1) The first topology, in the simulation, denoted by topology (a), is the fat-tree topology[7], which is widely used in many datacenter networks. The fat-tree topology has 16 core switches, 32 aggregation switches, 32 edge switches, and 128 servers. Similar to Ref. [17], we assume that each server hosts 15 Virtual Machines (VMs) to simulate a realistic scenario. Thus, the total number of terminals is 1920.

(2) The second topology, denoted by (b), is a two-dimensional HyperX topology[18]. This topology can make better use of available bisection bandwidths than the famous fat-tree topology, thus, it is also widely used in datacenter networks[19]. The topology contains 81 access switches and 1620 terminals.

Each simulation is executed 100 times, and the numerical results are averaged. We use the power law for the flow-size distribution, where 20% of all flows account for 80% of traffic volume[20].

Since this paper studies both wildcard routing and hybrid routing, we let TRPS denote all of the flows that are forwarded by tag-based routing. The RRJD denotes the hybrid routing by joint per-flow routing and tag-based routing. We compare these two proposed methods with four other methods:

(1) The first method is Per-Flow Routing (PFR). We adopt the multi-commodity flow method using randomized rounding for unsplittable flows in an SDN. Note that the method may drop some flows to satisfy flow table and deployment delay constraints.

(2) The second method is wildcard routing. We perform the widely-used OSPF protocol, which involves terminal-based routing.

(3) The third method is DomainFlow (DFW). DFW divides the network into two parts: one part using wildcard rules and the other part using PFR rules. This benchmark is mainly compared with RRJD.

(4) The last method is the optimal result for the linear program LP_1 based on Eq. (2), denoted by OPT. Since LP_1 is the relaxed version of the HR-JPT problem, OPT is a lower-bound for HR-JPT.

We mainly adopt five different metrics for performance measurement. The first two metrics are the **maximum and average numbers of required flow rules** on all switches. After executing these algorithms, we can obtain the number of flow rules used on each switch, and compute the maximum/average numbers of flow rules used on all switches. To measure network performance, the metrics **Link Load Ratio (LLR)** and **Network Throughput (NT)** are used. LLR can be obtained by measuring the traffic load $l(e)$ of each link e. Then, LLR is defined as $LLR = \max\{l(e)/c(e), e \in E\}$. NT can be obtained by measuring the total traffic amount through the network. The last metric is **communication overhead** to/from the controller. When new flows arrive at the switch, and no matched flow entry is produced, the switch will send packet-in messages to the controller, and the controller will send flow-mod messages to the corresponding switches for flow entry installment. The communication overhead is also called control overhead.

5.2 Simulation evaluation

We run six groups of experiments on two topologies to test the five different metrics of these algorithms. The flow table constraint is 5 000[4]. The first two groups of experiments observe the maximum/average number of required flow rules by increasing the number of flows. The results are shown in Figs. 3–6. Due to the flow table constraint, the maximum number of flow rules is 5 000. Both figures show that wildcard routing (i.e., OSPF and TRPS) requires fewer rules than the other methods. In fact, our proposed TRPS wildcard routing scheme requires a fewer number of rules than OSPF. For example, when 40×10^4 flows exist in topology (b), TRPS can reduce the maximum/average number of required rules by about 20%/65% compared with OSPF. Note that, because TRPS

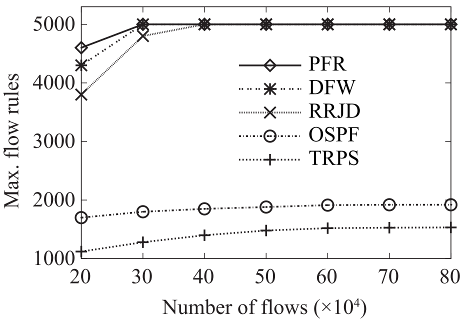


Fig. 3 Maximum number of flow rules versus number of flows for topology (a).

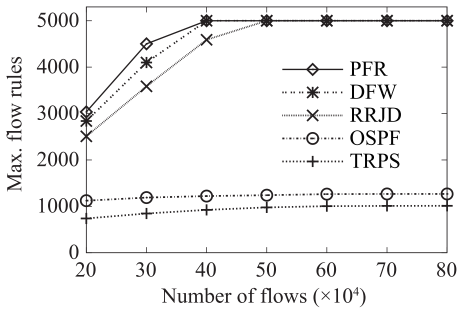


Fig. 4 Maximum number of flow rules versus number of flows for topology (b).

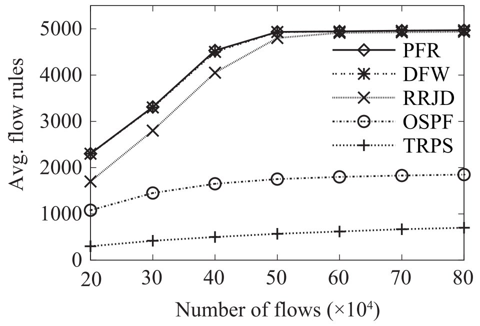


Fig. 5 Average number of flow rules versus number of flows for topology (a).

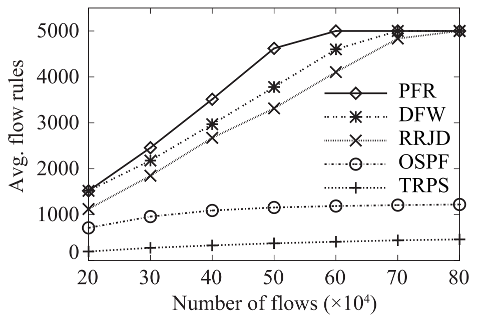


Fig. 6 Average number of flow rules versus number of flows for topology (b).

can significantly reduce the number of rules for the core switches and have less impact on the edge switches, the performance of the average number of required rules is much better. These four figures indicate that TRPS requires a fewer number of rules than OSPF.

The third group of experiments observes the NT by changing the deployment delay constraint. The default number of flows is 40×10^4. The results in Figs. 7 and 8 indicate that RRJD can achieve better NT than DFW. For example, when 40×10^4 flows exist in topology (a), our proposed RRJD scheme can improve NT by about 43% compared with DFW. The performance of PFR is poor due

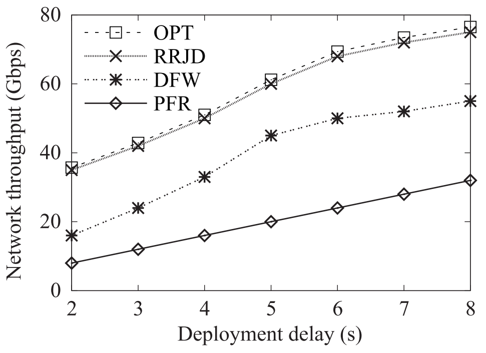


Fig. 7 Network throughput versus deployment delay for topology (a).

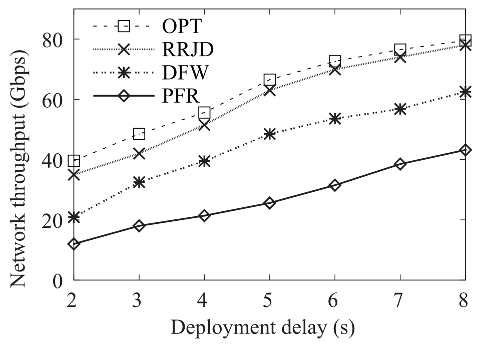


Fig. 8 Network throughput versus deployment delay for topology (b).

to the low speed of the method.

As illustrated in Figs. 9 and 10, the fourth group of experiments measures the change in NT by increasing the number of flows. When 60×10^4 flows exist in topology (a), our proposed RRJD can improve the NT by about 30%/90% compared with DFW/PFR while using a similar number of rules. While the NT of OSPF and TRPS are similar, our proposed TRPS can reduce the required number of rules by about 65% on average (illustrated in Fig. 5). The poor performance of PFR indicates that the method is impractical for large-scale networks.

The results of the fifth group of experiments are presented in Figs. 11 and 12. PFR has an exceptionally

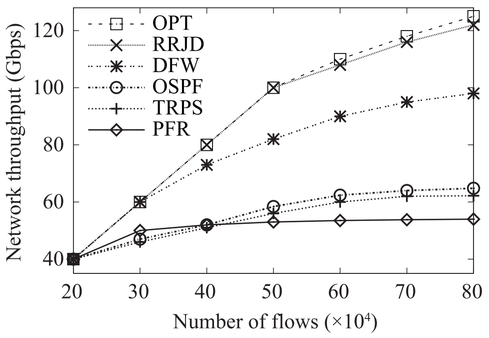


Fig. 9 Network throughput versus number of flows for topology (a).

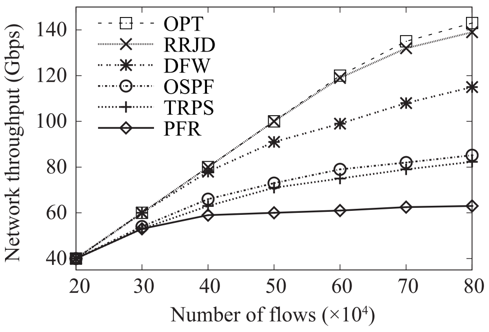


Fig. 10 Network throughput versus number of flows for topology (b).

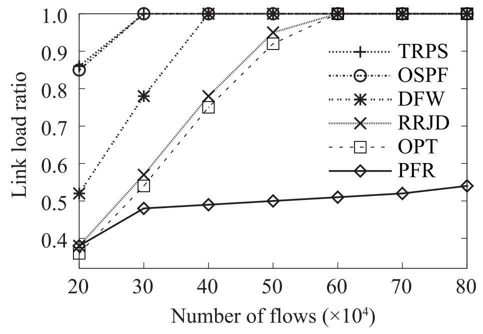


Fig. 11 Link load ratio versus number of flows for topology (a).

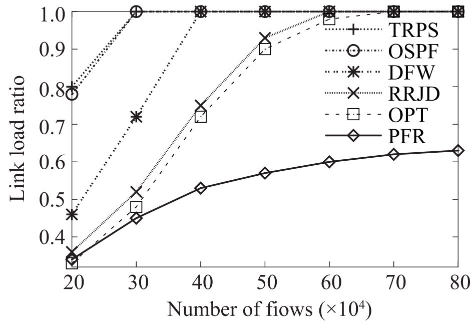


Fig. 12 Link load ratio versus number of flows for topology (b).

low LLR because of the flow table constraint and its dropping of many flows. Our proposed RRJD algorithm can reduce LLR by about 23% compared with DFW while using a similar number of rules. All three figures show that the performances of OPT and RRJD are very similar, which means the proposed algorithm can elegantly solve the HR-JPT problem and obtain a feasible solution that is close to that produced by the lower-bound OPT.

The last group of simulations evaluates the communication overhead between the controller and switches. The results in Figs. 13 and 14 show that wildcard routing (OSPF and TRPS) can greatly reduce the control

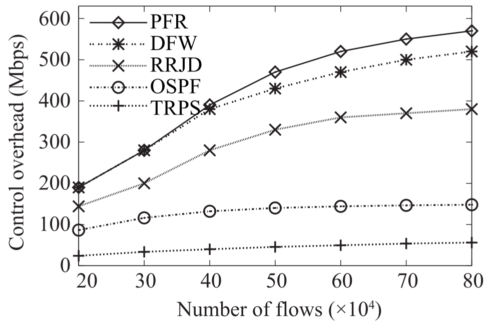


Fig. 13 Communication overhead versus number of flows for topology (a).

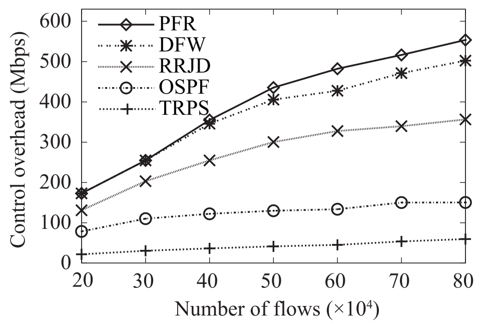


Fig. 14 Communication overhead versus number of flows for topology (b).

overhead because many flows can be forwarded by wildcard rules without being reported to the controller. For example, when 60×10^4 flows exist in topology (a), our proposed TRPS can reduce communication overhead by about 75% compared with OSPF. Moreover, our proposed RRJD can reduce overhead by about 30% compared with DFW.

From these simulations, we can make some conclusions. (1) Our proposed TRPS (the version including only tag-based wildcard routing) is better than OSPF (terminal-based wildcard routing). For example, TRPS can reduce the number of required rules by about 65% on average and reduce the overhead by about 75% compared with OSPF. (2) Our proposed RRJD increases NT by about 43% (or 30%) compared with DFW by changing the number of rules (or changing the deployment delay). Moreover, RRJD can reduce the communication overhead by about 30% and reduce the LLR by about 23% compared with DFW. (3) The performance of our proposed RRJD is similar to that of the lower-bound OPT, which means the approximation algorithm is efficient in solving the NP-hard problem.

6 Related Work

Since routing is a critical issue to achieve better network performance in an SDN, many related works to handle the routing problem have been reported. An obvious way to address the problem is to deploy one individual rule for each flow to provide fine-grained route selection. Al-Fares et al.[21] designed a dynamic flow scheduler for datacenter networks that sets up a new TCAM rule for every new flow in the network. However, as networks experience more and more flows while commodity switches only contain a few thousand TCAM rules[4], PFR is impractical to use in large-scale networks.

Some works are devoted to aggregate traffic (i.e., wildcard routing). iSTAMP[22] uses some of the TCAM rules for aggregation traffic, and similar works have been studied by Refs. [23, 24]. Unfortunately, the aggregation feasibility problem in a network was encountered in these works. Some studies have considered wildcard routing. References [5, 14], for example, adopted destination terminal-based aggregate routing. However, as many current datacenter networks contain millions of virtual or physical end terminals, this scheme also requires tens of thousands TCAM rules to deploy terminal-based wildcard routing in large-scale networks[7]. Previous works suffer from poor network performance due to the many flows that are forwarded via the same path, causing link congestion.

To achieve balance between network performance and flow table constraints, the combination of PFR and wildcard routing has been proposed. Devoflow[4] combined pre-deployed wildcard rules and dynamically-established exact rules, and DomainFlow[25] divided the network into two parts, one part using wildcard rules and another part using exactly matching rules. However, these works did not detail how default paths are deployed and only mainly adopted the OSPF protocol for wildcard routing. To date, the performance of the OSPF protocol cannot be guaranteed.

This paper focuses on wildcard routing and proposes the TRPS. We also study the HR-JPT problem. To the best of our knowledge, our work is the first to deploy the HR-JPT problem under flow table constraints for load balancing.

7 Conclusion

In this paper, we proposed a novel TRPS for wildcard routing and studied the HR-JPT problem. The test results show the high efficiency of both the TRPS and HR-JPT. In the future, we will consider an update scheme and online algorithm.

Acknowledgment

This paper was supported by the National Natural Science Foundation of China (Nos. 61472383, 61472385, and U1301256), and the Natural Science Foundation of Jiangsu Province in China (No. BK20161257).

References

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, Ethane: Taking control of the enterprise, *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 1–12, 2007.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, Openflow: Enabling innovation in campus networks, *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, The nature of data center traffic: Measurements & analysis, in *ACM SIGCOMM Conference on Internet Measurement 2009*, Chicago, IL, USA, 2009, pp. 202–208.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, Devoflow: Scaling flow management for high-performance networks, *ACM SIGCOMM Computer Communication Review*, vol. 41,

no. 4. pp. 254–265, 2011.

[5] H. Huang, S. Guo, P. Li, B. Ye, and I. Stojmenovic, Joint optimization of rule placement and traffic engineering for qos provisioning in software defined network, *IEEE Transactions on Computers*, vol. 64, no. 12, pp. 3488–3499, 2015.

[6] C. Wei, R. Buffone, and R. Stata, System and method for website performance optimization and internet traffic processing, US Patent 8112471, Feb. 7, 2012.

[7] X. Lu and Y. Xu, Sfabric: A scalable sdn based large layer 2 data center network fabric, in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*, 2015, pp. 57–58.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat, A scalable, commodity data center network architecture, in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[9] S. Banerjee and K. Kannan, Tag-in-tag: Efficient flow table management in sdn switches, in *International Conference on Network and Service Management*, 2014, pp. 109–117.

[10] A. Wang, Y. Guo, F. Hao, T. Lakshman, and 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*, 2014, pp. 403–414.

[11] Open Flow Switch, Specification, https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, June 25, 2012.

[12] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, Dynamic scheduling of network updates, in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 539–550.

[13] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, Abstractions for network update, in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2012, pp. 323–334.

[14] S. Even, A. Itai, and A. Shamir, On the complexity of time table and multi-commodity flow problems, in *Foundations of Computer Science, 1975., 16th Annual Symposium on*, 1975, pp. 184–193.

[15] ILOG, IBM, CPLEX, V12. 1, Users Manual for Cplex, 2009.

[16] Mininet Team, Mininet overview, http://mininet.org/overview, 2017.

[17] K. Kannan and S. Banerjee, Compact tcam: Flow entry compaction in tcam for power aware sdn, in *International Conference on Distributed Computing and Networking*, 2013, pp. 439–444.

[18] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, Hyperx: Topology, routing, and packaging of efficient large-scale networks, in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, p. 41.

[19] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, Past: Scalable ethernet for data centers, in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, 2012, pp. 49–60.

[20] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, The nature of data center traffic: Measurements & analysis, in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, 2009, pp. 202–208.

[21] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, Hedera: Dynamic flow scheduling for data center networks, in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 10, 2010, p. 19.

[22] M. Malboubi, L. Wang, C.-N. Chuah, and P. Sharma, Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp), in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 934–942.

[23] Z. Hu and J. Luo, Cracking network monitoring in dcns with sdn, in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, 2015, pp. 199–207.

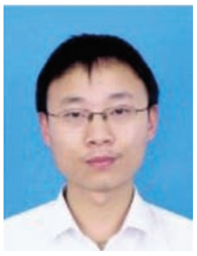
[24] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, Plug-n-serve: Load-balancing web traffic using openflow, *ACM Sigcomm Demo*, vol. 4, no. 5, p. 6, 2009.

[25] Y. Nakagawa, K. Hyoudou, C. Lee, S. Kobayashi, O. Shiraki, and T. Shimizu, Domainflow: Practical flow management method using multiple flow tables in commodity switches, in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 399–404.

Gongming Zhao received the BS degree from Shandong University, China, in 2015. He is currently a PhD candidate in computer science at the University of Science and Technology of China. His main research interests are software defined network and Internet of Things.

Ziqiang Li received the BS degree from Chongqing University of Post and Telecommunications, Chongqing, China, in 2015. He is currently a master student in computer science at the University of Science and Technology of China. His main research interest is software defined networks.

Liusheng Huang received the MS degree in computer science from University of Science and Technology of China in 1988. He is currently a professor and PhD supervisor of the Department of Computer Science and Technology at the University of Science and Technology of China. He has published 6 books and more than 200 papers. His research interests are in the areas of Internet of Things and information security.



Hongli Xu received the PhD degree in computer science 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 over 70 papers, and held about 30 patents. His main research interest is software defined networks, cooperative communication, and vehicular ad hoc network.