Computer Networks 164 (2019) 106891

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Reducing controller response time with hybrid routing in software defined networks

Hongli Xu[a,*], Jianchun Liu[a], Chen Qian[b], He Huang[c], Chunming Qiao[d]

[a] School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China
[b] Department of Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064, USA
[c] School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215006, China
[d] Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14260-2500, USA

ARTICLE INFO

Article history:
Received 11 October 2018
Revised 5 August 2019
Accepted 26 August 2019
Available online 30 August 2019

Keywords:
Software defined networks
Hybrid routing
Controller response time
Wildcard
Approximation

ABSTRACT

In a typical software defined network (SDN), switches report the Packet-In messages of newly arrived flows to the controllers. With more and more flows arriving at a network, the controller load significantly increases, which may lead to long (or even unacceptable) controller response time. Though previous solutions, such as dynamic controller assignment, help to reduce the controller response time, they still lead to various disadvantages, such as an unacceptable controller re-assignment delay, massive communication overhead between controllers, or poor routing performance. To address this issue, *our intention is to trade data plane resource for better control plane performance*. Specifically, we propose to pre-install wildcard entries for some aggregated flows to reduce controller response time, and perform dynamic routing for new-arrival flows to optimize the network performance. We define the problem of reducing controller response time with minimum data plane resource cost, and prove its NP-hardness. We then present an efficient algorithm based on randomized rounding, and analyze that our algorithm can achieve constant bicriteria approximation under most practical situations. Some practical issues are discussed to enhance our algorithm. We have implemented the proposed algorithm on a real SDN testbed. The experimental results and the extensive simulation results show that our method can reduce the controller response time by 47%, or improve the network throughput by 56% compared with the previous solutions, even with significant traffic dynamics.

1. Introduction

With the development of information technology, the number of flows increases drastically in both cloud networks and Internet. On one hand, with the development of information technology, many novel network-based applications (*e.g.*, search [1] and content distribution [2]) are constantly emerging. Thus, the network should be able to accommodate a large number of flows for these applications. On the other hand, some hot events will attract attention of many people through networks. For example, many audiences will watch the live broadcast of some attractive sport final through the Internet. In recent years, SDN becomes a potential technology to better manage a large number of flows. An SDN is typically separated into the control plane and the data plane. The control plane consists of a logically-centralized controller, which may be a cluster of distributed controllers and is responsible for managing the whole network. The data plane consists of a set of SDN switches [3]. With more flows arriving at the network, the switches will send more Packet-In messages to the controllers, which will lead to a higher controller load.

There are three different ways to reduce the controller response time (or the controller load). The first or general way is to use multiple controllers with the static controller assignment mechanism, *e.g.*, Onix [4] and NVP [5]. Specifically, the control plane is implemented as a cluster of distributed controllers, and each switch is only connected/associated with one controller. The switches deliver the Packet-In messages to different controllers, which helps to reduce the controller load compared with the single-controller framework. However, since the traffic in the network dynamically fluctuates in space and time, some controllers may still be heavy-loaded, or even congested [6]. In fact, its infrastructure will scarcely be changed in practice once an SDN is deployed. Meanwhile, more controllers require massive communication overhead in the control plane to maintain consistency of network status [7], and increase the management complexity. Thus, it is not a feasible solution to reduce the controller response time by using the static controller assignment mechanism.

* Corresponding author.
E-mail address: xuhongli@ustc.edu.cn (H. Xu).

https://doi.org/10.1016/j.comnet.2019.106891
1389-1286/

The second method is the dynamic controller assignment mechanism [6,8], which permits each switch to dynamically associate from a "heavy-loaded" controller to a "light-loaded" one, so that the maximum controller load can be reduced. For example, the authors of [6,8] presented a near-optimal Nash stable solution for dynamic controller assignment. However, the dynamic controller assignment scheme brings some disadvantages on service continuity. In fact, traffic dynamics will often trigger the switch to change its controller assignment, which may disrupt the service continuity. For example, when bursty traffic arrives at a switch, it may change its controller assignment. During the reassignment procedure, the new-arrival packets will be recorded in the switch's buffer. Even with a short delay, the bursty traffic will lead to packet dropping, which will significantly reduce the user experience.

The final method is to pre-install entries for *all* flows in a network, also called *proactive routing scheme* [9–11]. Since each flow can match at least one pre-installed entry when arriving at a switch, the switch will not deliver any Packet-In message to the controller. Thus, the controller load is very light and the controller response time is low. However, the proactive routing can not efficiently deal with traffic and management policy dynamics. *First*, due to dynamic traffic intensity, it may result in transient congestion on some data links, for the controller can not provide dynamic route control for flows. This will lead to packet dropping and throughput reduction. Although network updates [12–15], *e.g.*, re-routing some elephant flows, can help to improve the routing performance, it increases additional operations on switches, which may interfere with other basic switch's functions, *e.g.*, entry installing and traffic measurement [9]. *Second*, for many practical applications, network operators expect to specify fine-grained policies that drive how the underlying switches forward, drop, and measure traffic. However, as the management policies for flows may change over time (*e.g.*, due to host mobility or middlebox placement), the proactive routing scheme can not well adapt to these changes.

To overcome the disadvantages of the above solutions, we propose to *trade data plane resources (e.g., flow entry and link bandwidth) for better control plane performance*. Specially, we will pre-install wildcard forwarding entries (*i.e.*, proactive routing) for partial flows (not all flows) so as to achieve the trade-off optimization between the controller response time and the resource cost in the data plane. The controllers will perform the dynamic routing for new-arrival flows, which helps to optimize the network performance, *e.g.*, throughput maximization. Compared with the previous works, our proposed method has the following advantages:

- Our method just requires static controller assignment. Thus, the communication overhead between controllers can be significantly reduced, compared with the dynamic controller assignment scheme [6].
- We pre-install wildcard entries for some aggregated flows (or macroflows [16]), which will decrease the controller response time, compared with the fully dynamic routing scheme.
- As the controller performs dynamic routing for other arrived flows, the routing performance (*e.g.*, load balancing or network throughput) will still be efficient even with traffic dynamics, which will be validated by simulations in Section 5.

In this paper, we define the problem of reducing controller response time with minimum data plane resource cost (RCRT-MR), and prove its NP-hardness. We present an efficient algorithm based on the randomized rounding method, and analyze that our proposed algorithm can achieve the constant bicriteria approximation under most practical situations. This paper discusses some practical issues to enhance the practicability of our proposed algorithm and implements the proposed method on our SDN testbed. The experimental results and the extensive simulation results show that our method can reduce the controller response time by about 47%, and improve the network throughput by about 56% compared with the previous solutions, even with significant traffic dynamics (*e.g.*, with a prediction error of 60%).

Table 1
Key notations.

Symbol	Semantics
U	A cluster of SDN controllers
V	A set of switches
E	A set of network links
$c(v_i)$	The flow table size of switch v_i
$c(e)$	The capacity of link e
α_u	The Processing capacity of controller u
$\theta_u(t)$	The load on controller u at time t
$\vartheta_u(t)$	The response time of controller u at time t
Γ	A set of macroflows
$g(\gamma)$	The number of individual flows in γ
$f(\gamma)$	The traffic size of macroflow γ
$\mathcal{P}_\gamma$	A feasible path set for macroflow γ

2. Preliminaries

This section mainly gives the preliminaries for our study. For ease of reference, we list the key notations in Table 1.

2.1. Network model

A software defined network consists of two device sets: a cluster of controllers, $U = \{u_1, \ldots, u_m\}$ with $m = |U|$ and a set of SDN switches, $V = \{v_1, \ldots, v_n\}$, with $n = |V|$. The controllers are responsible for making decisions for all flows, including route selection and statistics collection. Each switch $v_i \in V$ is connected to a fixed controller $u_j \in U$, and its flow table size is denoted as $c(v_i)$. The network topology from a view of the data plane can be modeled by a connected graph $G = (V, E)$, where E is the set of links connecting switches. For each link $e \in E$, its forwarding capacity is denoted as $c(e)$.

2.2. Controller response time model [6]

We consider a discrete time model where the length of each time slot matches the timescale at which Packet-In requests of each switch can be precisely recorded. In an SDN, coordination among multiple controllers is necessary to install this path. To simplify the problem formalization, we assume that each controller remains a fraction of capacity for Packet-In processing. For each controller $u \in U$, its processing capacity is denoted as α_u in terms of the number of requests/flows, which can be handled (including route computing and rule distributing/installing) using its reserved capacity in one time unit [6]. The set of its connected switches is denoted as $\mathcal{S}_u$. The number of Packet-In messages (one for each newly arrived flow), which will be delivered to the associated controller by switch v in slot t, is denoted as $\zeta_v(t)$. Then, the load of controller u is $\theta_u(t) = \sum_{v \in \mathcal{S}_u} \zeta_v(t)$.

By applying the Little's law [6,17], the average sojourn time on controller u is $\frac{1}{\alpha_u - \theta_u(t)}$. Given that the time for computing a single-source route is $O(n^2)$ [18], where n is the number of switches in an SDN, the average response time of controller u can be expressed as [6]:

$$\vartheta_u(t) = \frac{1}{\alpha_u - \theta_u(t)} \cdot O(n^2) \tag{1}$$

Without confusion, we omit parameter t in these variables.

2.3. System workflow

In the traditional SDN network, header packets of all new flows will be encapsulated and reported to the controller. Though this strategy can provide fine-grained control for each flow [19], it increases the controller load and its response time, resulting in poor user experience. In this paper, we propose to reduce the controller response time by deploying some wildcard rules for a partial set of flows.

The workflow of our system is as follows. The time is divided into fixed-period slots. At the beginning of each time slot, the controller will collect flow statistics information from switches and predict the flow traffic in the current time slot. We then choose a subset of flows, and install wildcard rules for them. When a packet arrives at a switch, it will be (1) measured and (2) matched with all flow entries. If there is a matching entry, this packet will be directly forwarded to a certain port, specified by the operation field in this entry. Otherwise, the switch will report the packet to the controller, which determines the route path and installs rules on switches along its route path. In this paper, we mainly focus on how to choose a suitable subset of flows for proactive routing to achieve the trade-off optimization between the controller response time and resource cost in the data plane. The details of some practical issues will be discussed in Section 4.

2.4. Definition of the RCRT-MR problem

In this paper, we focus on reducing the controller response time by pre-installing wildcard rules for some aggregated flows, which will trade some resource (*e.g.*, flow entry and link bandwidth) in the data plane. We define the problem of reducing controller response time with minimum data plane resource cost (RCRT-MR). Due to traffic dynamics, it is difficult to predict the arrival and traffic information of each individual flow. To make our solution feasible and practicable, we consider a coarse-grained flow scheme, called macroflow [16], which aggregates a set of flows matching with a wildcard rule. For example, all flows from switches v_i to v_j can be aggregated into a macroflow. In fact, our proposed algorithm can also be applied for other definitions of macroflows. For example, we can also define the macroflow as the aggregated flows with the same hierarchical IP addresses [20].

We divide the time into fixed-period slots (*e.g.*, 1 min for the data center network [21] or 5 min for the backbone network [22]). We should note that Hong et al. [23] have shown that there is the low impact of performance dips in practice when there are more flows and the update frequency is 5 min. At the beginning of each time slot, we predict the traffic information for a set of macroflows, denoted as Γ, via per-flow measurement during the last time slot, which will be discussed in Section 4. Assume that each macroflow is denoted as $\gamma = (g(\gamma), f(\gamma))$, where $g(\gamma)$ is the number of individual flows in this macroflow, and $f(\gamma)$ is its traffic size. Similar to [23], the controller has constructed a set of feasible paths $\mathcal{P}_\gamma$ for each macroflow γ, and chooses one feasible path from $\mathcal{P}_\gamma$ for this macroflow. $\mathcal{P}_\gamma$ is determined based on the management policies and performance objectives. Some previous works [10,14] have shown that only a certain number of the best feasible paths under a certain performance criterion, such as having the shortest number of hops or having the large capacities [22], are enough to achieve the better network performance. Since traffic prediction error is unavoidable, we will discuss the impact of macroflow traffic prediction error on the network performance in Section 3.4, and give the simulation results in Section 5.3.

To reduce the controller response time (or controller load), the controller performs proactive routing for some (not all) macroflows by pre-installing wildcard entries. Specifically, we will determine a subset of macroflows, denoted as Γ', from the set Γ and select a feasible path for each macroflow $\gamma \in \Gamma'$ for proactive routing. When other flows arrive at the network, the controller will perform dynamic routing for them. With more and more flows arriving at a network, the arrival intervals between flows become shorter. However, long controller response time may result in request blocking at switches, and poor user experience. Therefore, we expect that the response time should not exceed a threshold, denoted as η_0, which is determined by practical application requirements. Meanwhile, the load threshold on controller u is denoted as α'_u. By Eq. (1), it follows $\frac{1}{\alpha_u - \alpha'_u} \cdot O(n^2) \le \eta_0$, where α_u is the capacity of controller u, and n is the number of switches in the network. Thus, we compute the controller load threshold as:

$$\alpha'_u \le \alpha_u - \frac{1}{\eta_0} \cdot O(n^2) \tag{2}$$

Moreover, to remain some slack controller capacity for processing unpredictable flows, we expect that the estimated load of controller u should not exceed $\delta \cdot \alpha'_u$, where δ is a constant and explained in Section 4.3. Note that δ will be updated with system running.

After pre-installing wildcard entries for some macroflows, we consider the resource cost in the data plane, including switch resource cost and link resource cost. On one hand, for each switch, we compute the flow table utilization ratio as the number of pre-installed flow entries divided by its flow table size. Then, we derive the maximum flow table utilization ratio, denoted as λ_t, among all switches. On the other hand, to achieve better routing performance, we expect that the link bandwidth cost for proactive routing should be minimized so that the network can accommodate more flows. We compute the traffic load on each link e, and its link load ratio. We can derive the maximum load ratio, denoted as λ_e, among all links. Since flow table and link bandwidth are two types of important resources in the data plane, we define the resource cost μ of the data plane as:

$$\mu = \beta \cdot \lambda_e + (1 - \beta) \cdot \lambda_t \tag{3}$$

where β represents the relative weight of each type of resource. Our objective is to minimize the resource cost in the data plane, *i.e.*, min μ.

We formulate the RCRT-MR problem as follows:

$$\min \ \mu$$

$$S.t. \begin{cases} \sum_{p \in \mathcal{P}_\gamma} y_\gamma^p + \phi_\gamma = 1, & \forall \gamma \\ \sum_{e \in p: p \in \mathcal{P}_\gamma} y_\gamma^p \cdot f(\gamma) \le \lambda_e \cdot c(e), & \forall e \\ \sum_{v \in p: p \in \mathcal{P}_\gamma} y_\gamma^p \le \lambda_t \cdot c(v), & \forall v \\ \zeta_v = \sum_{s(\gamma)=v} \phi_\gamma \cdot g(\gamma), & \forall \gamma \\ \theta_u = \sum_{v \in \mathcal{S}_u} \zeta_v \le \delta \cdot \alpha'_u, & \forall u \\ \mu = \beta \cdot \lambda_e + (1 - \beta) \cdot \lambda_t, & \\ y_\gamma^p \in \{0, 1\}, & \forall \gamma, p \\ \phi_\gamma \in \{0, 1\}, & \forall \gamma \end{cases} \tag{4}$$

y_γ^p denotes whether the controller will pre-deploy path p for macroflow γ or not, and ϕ_γ denotes whether macroflow γ will choose the dynamic routing scheme or not. The first set of inequalities means that the controller will perform the proactive routing scheme (*i.e.*, pre-installing wildcard entries) or the dynamic routing scheme for each macroflow $\gamma \in \Gamma$. The second and third sets of inequalities express the link bandwidth cost and flow table cost for proactive routing. The fourth and fifth sets of equations denote that the load of controller u should not exceed the threshold $\delta \cdot \alpha'_u$, where $s(\gamma)$ denotes the ingress switch of γ, $g(\gamma)$ is the predicted number of individual flows in this macroflow γ and α'_u is derived by Eq. (2). The sixth set of equations expresses the data plane resource cost.

Theorem 1. *The RCRT-MR problem is NP-hard.*

Proof. We consider a special case of the RCRT-MR problem, in which there is no constraint on the flow table size (*i.e.*, $c(v) = \infty$), no dynamic routing (*i.e.*, $\alpha'_u = 0$), and $\beta = 1$. Then, this special case becomes an unsplittable multi-commodity flow (MCF) with minimum congestion problem [24], which is NP-hard. Since MCF is a special case of our problem, the RCRT-MR problem is NP-hard too. □

3. Algorithm design for LCRT-MR

Due to NP-hardness, it is difficult to optimally solve our RCRT-MR problem. We first give a rounding-based algorithm for this problem (Section 3.1), and analyze its approximation performance (Section 3.2). Then, we give the complete algorithm description (Section 3.3). Finally, we discuss the impact of prediction error on the network performance (Section 3.4).

3.1. Algorithm description for the RCRT-MR problem

In this section, we describe a rounding-based wildcard entry pre-installing (RWP) algorithm for low controller response time with minimum resource cost in the data plane. Since the formalization of RCRT-MR in Eq. (4) is an integer linear program, it is difficult to solve this problem directly. The algorithm consists of two main steps, relaxing the problem and rounding to the integer solution, respectively. The RWP algorithm is formally described in Algorithm 1.

Algorithm 1 RWP: Rounding-based Wildcard Entry Pre-installing.

Step 1: Solving the relaxed RCRT-MR problem
Construct LP_1 as Relaxed RCRT-MR
Obtain the optimal solution $(\widetilde{y}^p_\gamma, \widetilde{\phi}_\gamma)$
Step 2: Choosing macroflows for proactive routing
Derive an integer solution $\widehat{y}^p_\gamma$ by randomized rounding
for each macroflow $\gamma \in \Gamma$ **do**
  $\widehat{\phi}_\gamma = 1 - \sum_{p\in\mathcal{P}_\gamma} \widehat{y}^p_\gamma$
  if $\widehat{\phi}_\gamma = 0$ **then**
    for each feasible path $p \in \mathcal{P}_\gamma$ **do**
      if $\widehat{y}^p_\gamma = 1$ **then**
        Install wildcard entries on switches along p

To solve the problem formalized in Eq. (4), our algorithm first constructs a linear program as a relaxation of the RCRT-MR problem. More specifically, RCRT-MR assumes that the controller will choose one feasible with proactive routing or not pre-install wildcard entries for each macroflow. By relaxing this assumption, traffic of each macroflow $\gamma \in \Gamma$ is permitted to be splittable and routed through a path set $\mathcal{P}_\gamma$. We formulate the following linear program LP_1.

$$\min \quad \mu$$

$$S.t. \begin{cases} \sum_{p\in\mathcal{P}_\gamma} y^p_\gamma + \phi_\gamma = 1, & \forall\gamma \\ \sum_{e\in p: p\in\mathcal{P}_\gamma} y^p_\gamma \cdot f(\gamma) \le \lambda_e \cdot c(e), & \forall e \\ \sum_{v\in p: p\in\mathcal{P}_\gamma} y^p_\gamma \le \lambda_t \cdot c(v), & \forall v \\ \zeta_v = \sum_{s(\gamma)=v} \phi_\gamma \cdot n(\gamma), & \forall\gamma \\ \theta_u = \sum_{v\in\mathcal{S}_u} \zeta_v \le \delta \cdot \alpha'_u, & \forall u \\ \mu = \beta \cdot \lambda_e + (1-\beta)\cdot\lambda_t, & \\ y^p_\gamma \ge 0, & \forall\gamma, p \\ \phi_\gamma \ge 0, & \forall\gamma \end{cases} \tag{5}$$

Note that variables y^p_γ and ϕ_γ are integral in Eq. (4), but fractional in Eq. (5). Since LP_1 is a linear program, we can solve it in polynomial time using a linear program solver. Assume that the optimal solution for LP_1 is denoted by $\{\widetilde{y}^p_\gamma, \widetilde{\phi}_\gamma\}$, and the optimal result is denoted by $\widetilde{\mu}$. As LP_1 is a relaxation of our RCRT-MR problem, $\widetilde{\mu}$ is the lower-bound result for RCRT-MR. Using the randomized rounding method [25], we derive the integral solution, denoted by $\{\widehat{y}^p_\gamma, \widehat{\phi}_\gamma\}$, for $\forall\gamma \in \Gamma$ and $\forall p \in \mathcal{P}_\gamma$. Specifically, $\widehat{y}^p_\gamma$ will be set as 1 with the probability of $\widetilde{y}^p_\gamma$, $\forall\gamma \in \Gamma$. If $\widehat{y}^p_\gamma = 1$, $\exists\gamma \in \Gamma$, it means that the controller will pre-install wildcard entries for macroflow γ on all switches along the path p. $\widehat{\phi}_\gamma$ can be derived by the first equation of Eq. (4). If $\widehat{\phi}_\gamma = 1$, it means that we will perform the dynamic routing scheme for all individual flows in this macroflow.

Now, we give a scenario of our randomized rounding method. For example, there are three potentially options for a macroflow γ, *e.g.*, (p_1, p_2, ϕ), where p_i denotes that the macroflow will be routed by the pre-installed entries and ϕ means that the macroflow will take the dynamic routing scheme. The final solutions for the situation are denoted by {0.2, 0.4, 0.4}. This means that the interval [0, 1] has been divided into four parts, (0, 0.2), (0.2, 0.6), and (0.6, 1.0). We then randomly choose a value from 0 to 1. Assume that the random value is 0.3. Since this value lies in (0.2, 0.4), the second feasible path p_2 will be chosen for macroflow. If the random value is 0.8, the fourth one ϕ will be selected.

3.2. Performance analysis

We analyze the approximate performance of the proposed RWP algorithm. Assume that the minimum capacity of all the links is denoted by $c_{\min}$. We define a constant χ as follows:

$$\chi = \min\{c(v), v \in V; \frac{\alpha'_u}{g(\gamma)}, s(\gamma) \in \mathcal{S}_u, \gamma \in \Gamma, u \in U; \frac{c_{\min}}{f(\gamma)}, \gamma \in \Gamma\} \tag{6}$$

Under most practical situations (*e.g.*, data center network [21], backbone network [22] and campus network [26]), the flow intensity is usually much less than the link capacity. For example, $c_{\min}$ = 100Mbps, and $s(\gamma)$ = 4Mbps for high-definition video conference. It follows that $\chi \ge 1$. Since RWP is a randomized algorithm, we compute the expected controller load and the expected data plane resource cost. We give the Chernoff bound for probability analysis.

Lemma 2 (Chernoff Bound)**.** *Given n independent variables: $x_1, x_2, \ldots, x_n$, where $\forall x_i \in [0, 1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n x_i]$. Then,* $\mathbf{Pr}[\sum_{i=1}^n x_i \ge (1+\epsilon)\mu] \le e^{\frac{-\epsilon^2\mu}{2+\epsilon}}$, *where ϵ is an arbitrary positive value.*

Controller Load Performance. Before analyzing the controller load performance, we define a random variable z^u_γ to denote how many Packet-In messages will be reported to the controller u from macroflow γ.

Definition 1. For each controller $u \in U$, and each macroflow $\gamma \in \Gamma$ with $s(\gamma) \in \mathcal{S}_u$, random variable z^u_γ is defined as follows:

$$z^u_\gamma = \begin{cases} g(\gamma), & \textit{with probability } \widetilde{\phi}_\gamma \\ 0, & \textit{otherwise} \end{cases} \tag{7}$$

By the definition, variables z^u_γ, with $s(\gamma) \in \mathcal{S}_u$, are mutually independent. The expected load on controller u is:

$$\mathbb{E}\left[\sum_{v\in\mathcal{S}_u}\sum_{s(\gamma)=v} z^u_\gamma\right] = \sum_{v\in\mathcal{S}_u}\sum_{s(\gamma)=v} \mathbb{E}[z^u_\gamma] = \sum_{v\in\mathcal{S}_u}\sum_{s(\gamma)=v} \widetilde{\phi}_\gamma \cdot g(\gamma) \le \delta \cdot \alpha'_u \tag{8}$$

Combining Eq. (8), we can explore a variable χ so that:

$$\begin{cases} \frac{z_\gamma^u \cdot \chi}{\delta \cdot \alpha_u'} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma} \frac{z_\gamma^u \cdot \chi}{\delta \cdot \alpha_u'}\right] \le \chi. \end{cases} \tag{9}$$

Then, by applying Lemma 2, assume that τ is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma, s(\gamma)\in\mathcal{S}_u} \frac{z_\gamma^u \cdot \chi}{\delta \cdot \alpha_u'} \ge (1+\tau)\chi\right] \le e^{\frac{-\tau^2\chi}{2+\tau}} \tag{10}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma, s(\gamma)\in\mathcal{S}_u} \frac{z_\gamma^u}{\delta \cdot \alpha_u'} \ge (1+\tau)\right] \le e^{\frac{-\tau^2\chi}{2+\tau}} \le \frac{\mathcal{F}}{m}, \tag{11}$$

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

The solution for Eq. (11) can be expressed as:

$$\tau \ge \frac{\log\frac{m}{\mathcal{F}} + \sqrt{\log^2\frac{m}{\mathcal{F}} + 8\cdot\chi\log\frac{m}{\mathcal{F}}}}{2\cdot\chi}, \quad m \ge 2 \tag{12}$$

By setting suitable values of parameters τ and $\mathcal{F}$, we can derive the approximation performance on the controller load.

Lemma 3. *After the rounding process, the load of controller u will not exceed the constraint $\delta\cdot\alpha_u'$ by a factor of $\frac{3\log m}{\chi}+3$.*

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

$$\tau = \frac{\log\frac{m}{\mathcal{F}} + \log\frac{m}{\mathcal{F}} + 4\cdot\chi}{2\cdot\chi} = \frac{3\log m}{\chi} + 2 \tag{13}$$

Thus, it follows

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{u\in U}\sum_{\gamma\in\Gamma, s(\gamma)\in\mathcal{S}_u} \frac{z_\gamma^u}{\delta\cdot\alpha_u'} \ge (1+\tau)\right] \\ &\le \sum_{u\in U}\mathbf{Pr}\left[\sum_{\gamma\in\Gamma, s(\gamma)\in\mathcal{S}_u} \frac{z_\gamma^u}{\delta\cdot\alpha_u'} \ge (1+\tau)\right] \\ &\le m\cdot\frac{1}{m^3} = \frac{1}{m^2}, \quad \tau \ge \frac{3\log m}{\chi} + 2 \end{aligned} \tag{14}$$

Then Eq. (11) is guaranteed with $1+\tau = \frac{3\log m}{\chi}+3$ and $\mathcal{F} = \frac{1}{m^2}$, which concludes the proof. □

Link resource cost: Similar to the analysis for the controller load performance, it also follows:

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

Flow Table Resource Cost: Similar to the analysis for the controller load performance, we have:

Lemma 5. *Our algorithm can achieve the approximation factor of $\frac{3\log n}{\chi}+3$ for the flow table resource cost.*

Data plane resource cost: Now we analyze the approximation performance for the data plane resource cost. By solving the linear program LP_1, we obtain the optimal result $\widetilde{\mu}$ for the relaxed RCRT-MR problem. Assume that $\widehat{\lambda_e}$ and $\widehat{\lambda_t}$ are the maximum link load ratio and the maximum flow table utilization ratio, respectively, by the RWP algorithm. The final resource cost is expressed as:

$$\widehat{\mu} = \beta\cdot\widehat{\lambda_e} + (1-\beta)\cdot\widehat{\lambda_t}, \ \beta\in[0,1] \tag{15}$$

For ease of description, we use variable ρ to denote $\frac{4\log n}{\chi}+3$. By two Lemmas 4 and 5, we obtain the following equation, similar to Eq. (14).

$$\begin{cases} \mathbf{Pr}\left[\beta\cdot\widehat{\lambda_e} \ge \beta\cdot\rho\cdot\widetilde{\lambda_e}\right] \le \frac{1}{n^2} \\ \mathbf{Pr}\left[(1-\beta)\cdot\widehat{\lambda_t} \ge (1-\beta)\cdot\rho\cdot\widetilde{\lambda_t}\right] \le \frac{1}{n^2} \end{cases} \tag{16}$$

Then, we have

$$\begin{aligned} &\mathbf{Pr}[\widehat{\mu} \ge \rho\cdot\widetilde{\mu}] \\ &= \mathbf{Pr}\left[\beta\cdot\widehat{\lambda_e} + (1-\beta)\cdot\widehat{\lambda_t} \ge \rho\cdot(\beta\cdot\widetilde{\lambda_e} + (1-\beta)\cdot\widetilde{\lambda_t})\right] \\ &\le \mathbf{Pr}\left[\beta\cdot\widehat{\lambda_e} \ge \beta\cdot\rho\cdot\widetilde{\lambda_e}\right] + \mathbf{Pr}\left[(1-\beta)\cdot\widehat{\lambda_t} \ge (1-\beta)\cdot\rho\cdot\widetilde{\lambda_t}\right] \\ &\le \frac{1}{n^2} + \frac{1}{n^2} = \frac{2}{n^2} \end{aligned} \tag{17}$$

So our RWP algorithm achieves the approximation factor ρ or $\frac{4\log n}{\chi}+3$ for the data plane resource cost.

Approximation factor: According to the above analyses, we can conclude that:

Theorem 6. *The randomized RWP algorithm can guarantee that the data plane resource cost will hardly be violated by a factor of $\frac{4\log n}{\chi}+3$, and the controller load performance will not be violated by a factor of $\frac{3\log m}{\chi}+3$ with a high probability.*

Under most practical situations, in which the flow intensity is usually much less than the link capacity, the RWP algorithm can achieve almost the constant bicriteria approximation. For example, the link capacity of today's networks will be 10Gbps. Observing the practical flow traces, the maximum intensity of a macroflow may reach 100Mbps. Under this case, $\frac{c_{\min}}{f(\gamma)}$ is 10^2. Even in a large-scale network with 1000 switches, or $\log n = 10$, the approximation factor for the data plane resource cost is 3.4. This analysis also fits for that of the controller load constraint. In other words, our RWP algorithm can achieve the constant bicriteria approximation with a high probability for the RCRT-MR problem under many practical situations.

3.3. Complete algorithm description

Though the RWP algorithm can almost achieve the constant bicriteria approximation for the RCRT-MR problem, the randomized rounding method cannot fully guarantee that the controller response time (or controller load) constraint is always met. Now, we describe the complete RWP algorithm to meet the load constraint on each switch. The complete algorithm is formally described in Algorithm 2. For ease of description, Γ^u denotes a set

Algorithm 2 Complete RWP Algorithm Description.

1: **Step 1: The same as that in Algorithm 1**
2: **Step 2: The same as that in Algorithm 1**
3: **Step 3: Choosing more macroflows for proactive routing**
4: The set of macroflows with pre-installed wildcard entries is denoted as Γ'
5: **for** Each controller $u\in U$ **do**
6: $\theta_u = \sum_{\gamma\in\Gamma^u-\Gamma'} g(\gamma)$
7: Sort all macroflows in $\Gamma^u-\Gamma'$ according to increasing order of $\frac{f(\gamma)}{g(\gamma)}$
8: **while** $\theta_u > \delta\cdot\alpha_u'$ **do**
9: Select a macroflow γ with minimum ratio of $\frac{f(\gamma)}{g(\gamma)}$
10: Choose a route path with minimum resource cost for this macroflow, and pre-install wildcard entries
11: Update the load of controller u, *i.e.* , $\theta_u = \theta_u - g(\gamma)$

of macroflows whose ingress switches are connected to controller u, *i.e.*, $\Gamma^u = \{\gamma, s(\gamma) \in \mathcal{S}_u\}$.

The complete RWP algorithm consists of three steps. Same as Algorithm 1, the first step constructs a linear program LP_1 as the relaxation of the RCRT-MR problem, and obtain the optimal solution, denoted as $(\widetilde{y}^p_\gamma, \widetilde{\phi}_\gamma)$. The second step determines a subset of macroflows, denoted as Γ', for proactive routing. In the third step, we check whether the load constraint on each controller is met or not. For each controller u, we estimate the controller load θ_u as $\theta_u = \sum_{\gamma \in \Gamma^u - \Gamma'} g(\gamma)$, where Γ' is a set of macroflows with pre-installed wildcard entries. The algorithm sorts all macroflows in $\Gamma^u - \Gamma'$ according to increasing order of the ratio between the traffic size and the number of flows in this macroflow (*i.e.*, $\frac{f(\gamma)}{g(\gamma)}$), and check these macroflows one by one. For each macroflow γ, we choose a feasible path with minimum resource cost, pre-install wildcard entries for this macroflow, and update the controller load estimation. The iteration is terminated until the load constraint on each controller is met.

3.4. Impact of traffic prediction error

In this section, we discuss the impact of traffic prediction error on the network performance. All flows can be divided into two sets. One is using the proactive routing scheme, the other is using the dynamic routing scheme. For flows in the first set, we will install wildcard rules before their arrival. Due to traffic dynamics, there is a high traffic prediction error in the worst case, which may lead to improper route selection for those flows. When flows in the second set arrive at the network, the controller will choose the least-congestion route paths using the dynamic scheme, which helps to alleviate the negative impact of infeasible route paths for flows in the first set. In fact, some previous works have shown that it can help to improve the network performance by dynamically rerouting only partial flows in the network [27–29]. Our simulation results in Section 5.3 also show that the network performance will not be significantly reduced even with a high prediction error.

4. Practical issues for system implementation

In our system, the packet processing procedure is illustrated in Fig. 1. To predict the flow size, the switch will take the per-flow traffic measurement (Section 4.1). At the beginning of each time slot, the controller collects flow statistics information from switches. Then, we predict the flow traffic in the current time slot (Section 4.2), and determine the value of parameter δ (Section 4.3). Next, we will choose a subset of flows using the RWP algorithm and update/pre-install rules in the flow tables of SDN switches (Section 4.4). When new flows arrive at the network, the controller will dynamically determine the route paths for these flows. This section gives the detailed description of the above five modules for system implementation.

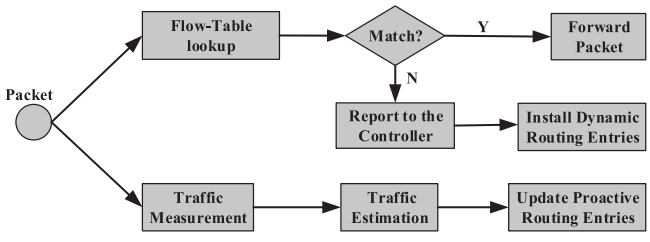


Fig. 1. Illustration of Packet Processing Procedure.

4.1. Per-flow traffic measurement

Our system needs to periodically collect the traffic information (*e.g.*, the traffic size) of each individual flow in the network to predict the macroflow information. Due to the limited size of flow tables at switches, it is impractical to measure the per-flow traffic only through flow tables. To overcome this challenge, some previous methods can be adopted for traffic measurement. For instance, flow-based measurements (*e.g.*, NetFlow [30] and sFlow [31]) could provide generic support for different measurement tasks through sampling packets. Besides, some advanced methods could support online measurement using probabilistic multiplicity counting [32] or randomized counter sharing [33] to keep up with the line speed of the modern routers. Moreover, we can obtain the average traffic size of each flow, denoted as φ.

4.2. Macroflow traffic prediction

In our design, the controller needs to predict the traffic size of macroflows based on the network traffic statistics. Since the macroflow is a coarse-grained flow management scheme, some previous traffic prediction works [34,35] can be directly applied in our system. Specifically, the authors in [34] could provide accurate prediction of traffic behavior at different time scales with less than 15% relative error. After we derive the estimated traffic size $f(\gamma)$ of each macroflow γ, the number of individual flows in this macroflow is estimated as $g(\gamma) = f(\gamma)/\varphi$. Thus, it is reasonable to assume that we can accurately predict the traffic information of macroflows. We will evaluate the impact of the traffic prediction error on the network performance in Section 5.

4.3. Estimation of parameter δ

Due to prediction error and flow uncertainty, we should remain some slack controller capacity for unpredictable flows to avoid long response time. Assume that the load of controller u in the current time slot is $\overline{\theta}_u$. We update the parameter δ as $\delta = \min\{\frac{\delta \cdot \alpha'_u}{\overline{\theta}_u},\ u \in U\}$, where α'_u is defined in Eq. (2). If δ exceeds 1, we just set δ as 1.

4.4. Update static routing entries/rules

As specified by the Openflow standard, each flow entry has two fields: idle_timeout and hard_timeout [19]. A non-zero idle_timeout field means that this flow entry will be removed after the given number (*i.e.*, the value of this field) of seconds, if no packet has been matched by this flow entry. A non-zero hard_timeout field means that this flow entry should be removed after the given number of seconds runs out, no matter there are matched packets or not. If both parameters are set to zero, this flow entry is considered to be permanent, and will be removed only by the controller.

After choosing a subset of macroflows for proactive routing, we will first delete those static routing entries which have no traffic by the prediction. Then, the controller installs flow entries to each switch and sets the idle_timeout and hard_timeout as zero.

5. Performance evaluation

In this section, we evaluatate the performance of our proposed algorithm through network simulator and testbed implementation.

5.1. Performance metrics and methodology

We adopt five main metrics for performance evaluation of our proposed algorithm: (1) the resource cost; (2) the maximum load

ratio of any controller; (3) the maximum controller response time; (4) the maximum load ratio of any link; and (5) the network throughput. During a simulation run, we first pre-install wildcard entries for some predicted macroflows with the controller response time constraint, and measure the resource cost in the data plane based on Eq. (3). The second metric measures the maximum number of Packet-In requests per controller divided by the controller processing capacity during the simulation. We compute the maximum controller response time by Eq. (1). The controller will perform the dynamic routing scheme for each new-arrival flow. In the simulations, we just choose a route path with the least traffic link load as the route path of a flow. The load ratio of a link is the traffic load divided by the link capacity. The load balancing metric is the maximum load ratio among all links. As we continuously increase the number of flows, we measure the maximum throughput that the network can support.

To evaluate how well our proposed algorithm performs, we select two other methods as benchmarks for performance comparison. The first benchmark is called the *proactive* routing scheme, *e.g.*, DevoFlow [9], in which the controller pre-installs wildcard entries for all macroflows. This method can achieve the lowest controller load. When the (potential) link congestion is detected, the controller will re-route some flows by installing fine-grained rules on switches. The second one is the *dynamic* routing scheme. When each individual flow arrives at a switch, the controller will dynamically determine its route path. Compared with the dynamic controller assignment mechanism [6], the above two methods and our RWP algorithm just require each switch to connect to a fixed controller, thus the communication delay for controller re-assignment and the control plane overhead between controllers will be reduced or even avoided. For fairness, we do not compare our algorithm with the dynamic controller assignment mechanism [6].

Our proposed algorithm for wildcard entry pre-installing takes the macroflow prediction as the input. Since the traffic prediction error is unavoidable in practice, it is necessary and meaningful to observe the impact of the traffic prediction error on the network performance. As the controller predicts the number of flows and its traffic size for each macroflow γ, there are two prediction errors. One is the prediction error (ε_1) of the number of individual flows in a macroflow, the other is that (ε_2) of the traffic size of a macroflow. Because over-estimation does not violate controller response time constraint and also its negative impact on the routing performance is insignificant, we are interested in the worst-case effect of under-estimation on the controller response time in the simulations.

5.2. Testbed evaluation

5.2.1. Implementation on the platform

We implement the dynamic routing scheme and our RWP algorithm on a small-scale testbed. Our SDN platform is mainly composed of three parts: a controller, 7 OpenFlow enabled switches and 6 terminals. For simplicity, there deploys only one controller in the SDN, and each switch is directly connected to the controller, as shown in Fig. 2. Without confusion, we do not draw the controller in the figure. We use the Ryu [36] as the controller software running on a server with a core i7-8700khttps://github.com/lyl617/SDN-RWP and 32GB of RAM. The data plane is comprised of 7 Pica8 3297 switches, which support the OpenFlow v1.3 standard [19]. We use source IP, source port and destination IP to identify a flow, so that each terminal is able to generate a number of different flows in a network. We generate 300 flows in the network and the average flow intensity is 600 Kbps. Moreover, to simulate the realistic scenario, 20% elephant flows and 80% mice flows are generated in the network.

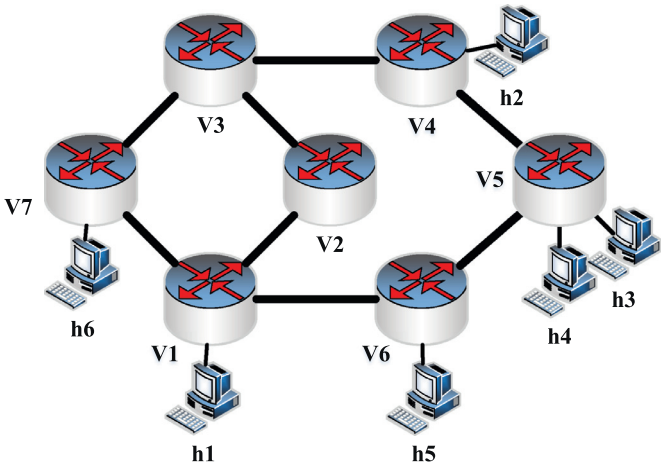


Fig. 2. Topology of the SDN Platform. We implement our proposed algorithm and other benchmarks on a real testbed. Our testbed is composed of three parts: a controller, seven switches and six terminals.

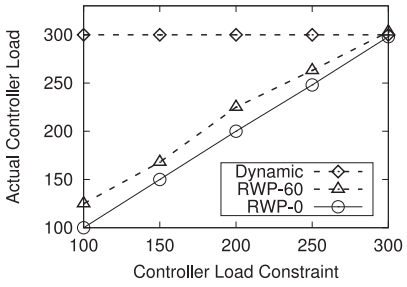


Fig. 3. Actual Controller Load vs. Controller Load Constraint.

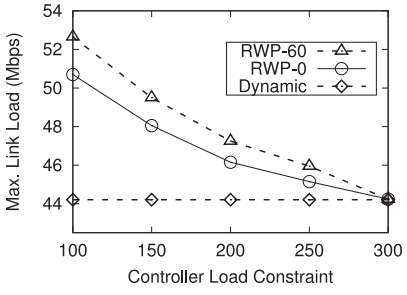


Fig. 4. Max. Link Load vs. Controller Load Constraint.

We have implemented the proposed RWP algorithm as an application in python, and integrated it with Ryu. The testing is executed under Ubuntu 16.04 LTS and the implementation of RWP with Ryu has been published at https://github.com/lyl617/SDN-RWP [37].

5.2.2. Testing results

We mainly observe the performance (including controller load and maximum link load) of our proposed algorithm even with a prediction error. The flows from a source terminal to a destination terminal is regarded as a macroflow in the testing. The testing results in Fig. 3 indicate that (1) our proposed RWP algorithm can significantly reduce the controller load compared with the dynamic routing scheme; and (2) the prediction error has an insignificant impact the controller load for our proposed algorithm. For example, when the controller load constraint is 250, the difference in actual controller load between RWP-0 and RWP-60 is less than 20. It should be noted that RWP-0 and RWP-60 will pre-install wildcard entries for different macroflows based on prediction. As shown in Fig. 4, the maximum link load will decrease with the increasing controller load constraint. That's because, with a large

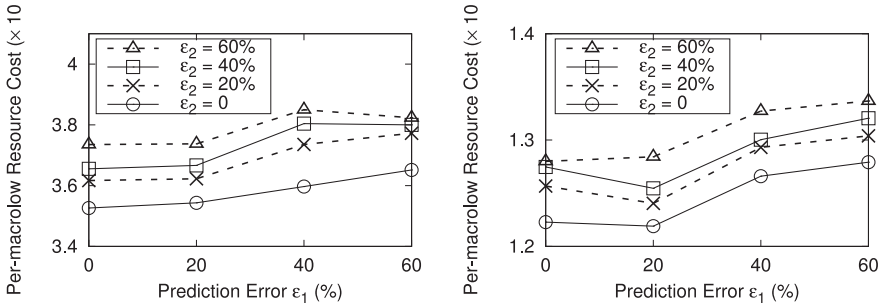


Fig. 5. Per-macroflow Resource Cost vs. Two Prediction Errors. *Left plot*: Topology (a); *right plot*: Topology (b).

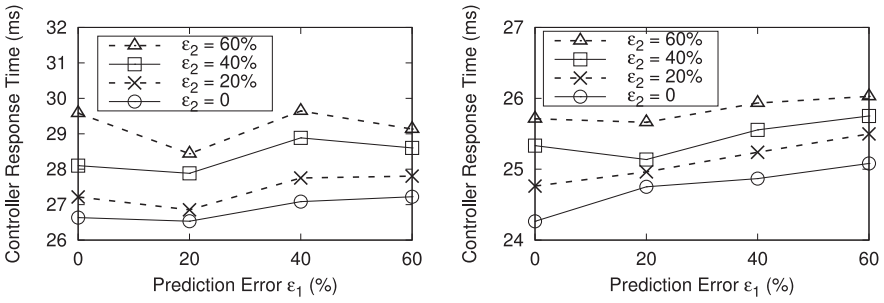


Fig. 6. Controller Response Time vs. Two Prediction Errors. *Left plot*: Topology (a); *right plot*: Topology (b).

controller load constraint, the controller has chance to dynamically choose routes for more flows, which benefits the routing performance. From these testing results, we can conclude that RWP can achieve better trade-off between controller load and routing performance than the dynamic routing, even with a large prediction error (*e.g.*, 60%).

5.3. Simulation evaluation

5.3.1. Simulation settings

We run four sets of experiments on two different topologies to evaluate the efficiency of the proposed algorithm. As running examples, our simulations select two practical and typical network topologies. The first topology, denoted as (a), is the fat-tree topology [38], which has been widely used in many data center networks. This topology contains totally 80 switches (including 16 core switches, 32 aggregation switches, 32 edge switches) and 128 servers. The second one, denoted as (b), is a campus network topology [39]. The topology (b) contains 100 switches and 200 servers. We deploy 5 controllers, and each controller connects to almost the same number of switches in both topologies. Due to the limited capacity of our simulation platform, the capacity of each link is set as 1 Gbps for both topologies. Moreover, the flow table size of each switch is set as 65,356 [40]. For the flow size, the authors of [9] have shown that less than 20% of the top-ranked flows may be responsible for more than 80% of the total traffic. Thus, we adopt this rule to allocate the size for each flow. The parameter δ and the controller capacity are set as 0.8 and 180 K [6] in all simulations, respectively. By default, the network contains 600 K flows. Several works have measured the controller response time through testbed and simulations. For example, the work [41] presents a study of SDN controller performance using four OpenFlow enabled controllers and measures minimum (least load) and maximum controller (maximum load) response time of SDN controllers. The authors in [17] study the response time behavior for Packet-In messages by changing flow arrival rate and repeat the experiments by varying the number of controllers. Both of their results show that average response times of all controllers are between 0.01s and 0.03s. Thus, without lose of generality, we choose 0.02 s as the controllerâs response time in this paper. We execute each simulation 100 times, and average the numerical results. We will analyze the performance of our proposed algorithm with different prediction errors.

5.3.2. Simulation results

The first set of simulations observes how two prediction errors ε_1 and ε_2 affect different metrics (*e.g.*, resource cost and controller response time). Fig. 5 shows that the resource cost will be generally increased with a larger prediction error. Fig. 6 shows that the controller response time generally increases as the prediction errors ε_1 and ε_2 are increasing. Two figures also indicate that the performance difference is not significant (less than 10%) even with larger prediction errors, *e.g.*, $\varepsilon_1 = 60\%$ and $\varepsilon_2 = 60\%$.

According to the evaluation results in Figs. 5 and 6, we will use a single parameter X to capture the prediction error for simplifying the performance evaluation. More specifically, we use $\overline{g}(\gamma)$ to denote the actual number of individual flows in a macroflow γ. Besides, the actual traffic size is denoted as $\overline{f}(\gamma)$. In our simulation, the predicted number $g(\gamma)$ of individual flows in the macroflow γ obeys the uniform distribution from $(1 - X\%) \cdot \overline{g}(\gamma)$ to $\overline{g}(\gamma)$, and similarly, the predicted traffic size $f(\gamma)$ obeys the uniform distribution from $(1 - X\%) \cdot \overline{f}(\gamma)$ to $\overline{f}(\gamma)$. In the following experiments and simulations, we will compare the performance of four versions of the RWP algorithms, namely RWP-0, RWP-20, RWP-40 and RWP-60, to evaluate the performance effect of the prediction error. RWP-0 represents a baseline, while RWP-X (where $X = 20$, 40 and 60) denotes the RWP algorithm with a maximum prediction error $X\%$.

The second set of simulations observes how the weight parameter β affects different metrics (*e.g.*, resource cost and controller response time). Fig. 7 shows that the resource cost does not significantly increase as the parameter β is increasing from 0.1 to 0.9, or the maximum gap of the resource cost is less than 9%. That's because the algorithm manages to minimize the resource utilization among the link bandwidth and flow table entry by efficient route

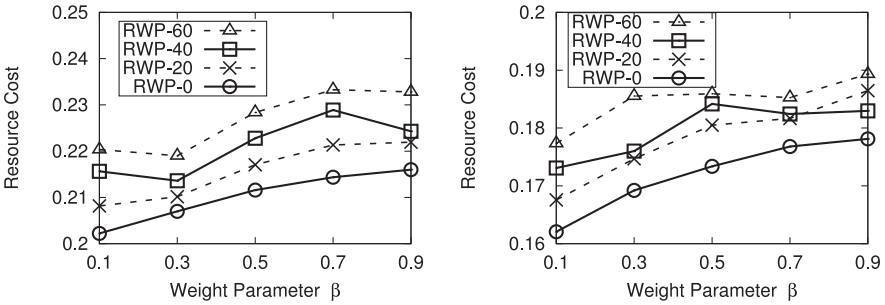


Fig. 7. Resource Cost vs. Weight Parameter β. *Left plot*: Topology (a); *right plot*: Topology (b).

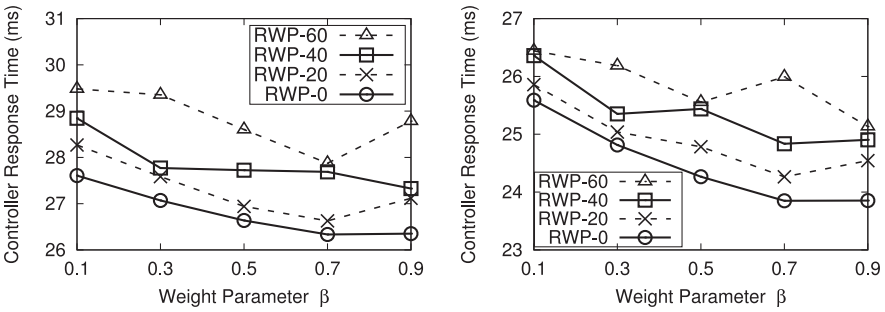


Fig. 8. Controller Response Time vs. Weight Parameter β. *Left plot*: Topology (a); *right plot*: Topology (b).

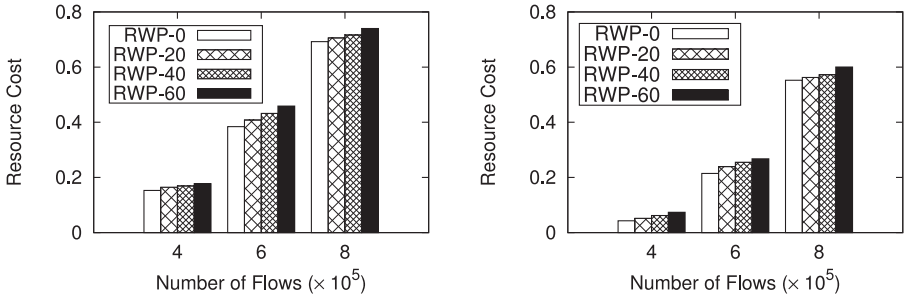


Fig. 9. Resource Cost vs. Number of Flows with CRT=0.01s. *Left plot*: Topology (a); *right plot*: Topology (b).

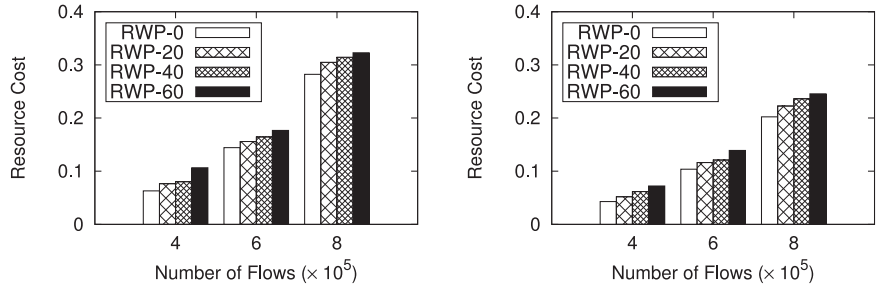


Fig. 10. Resource Cost vs. Number of Flows with CRT=0.04s. *Left plot*: Topology (a); *right plot*: Topology (b).

selection. Fig. 8 shows that the controller response time decreases as the weight parameter β is increasing. However, the decreasing rate is much smaller (less than 5% from Fig. 8). In the following, the parameter β is set as 0.5.

The third set of four simulations observes the impact of the number of flows on the different metrics (*e.g.*, resource cost and controller load ratio) with different prediction errors. Fig. 9 shows the resource cost of the data plane by changing the number of flows from 400 K to 800 K on two different network topologies, in which the controller response time (CRT) constraint is set as 0.01s. With more flows in the network, four algorithms (RWP-0, RWP-20, RWP-40 and RWP-60) need to pre-install more wildcard entries so as to meet the controller response time constraint, which leads to a higher resource cost. Fig. 10 plots the resource cost performance by changing the number of flows when the controller response time constraint is 0.04 s. This figure has the similar performance trend as Fig. 9. We can conclude from the two figures that the prediction error does not significantly impact the resource cost. For example, given 600 K flows in the network, the difference of resource cost between RWP-0 and RWP-60 is less than 6%. Fig. 11 shows that the controller load ratio is almost linearly increasing with more flows arrival in the network. Since we pre-install wildcard entries for some chosen macroflows, our proposed RWP-0 algorithm can achieve lighter controller load than the dynamic rout-

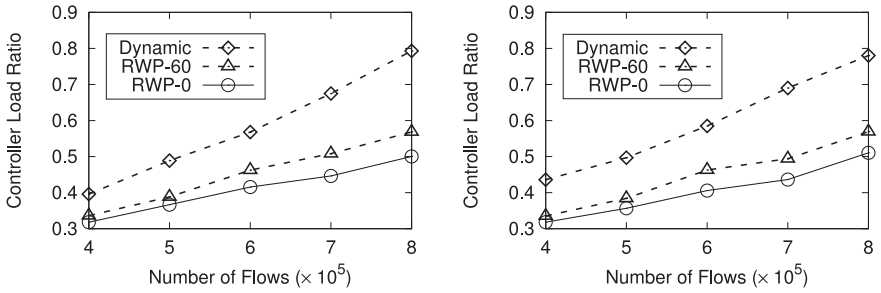


Fig. 11. Controller Load Ratio vs. Number of Flows. *Left plot*: Topology (a); *right plot*: Topology (b).

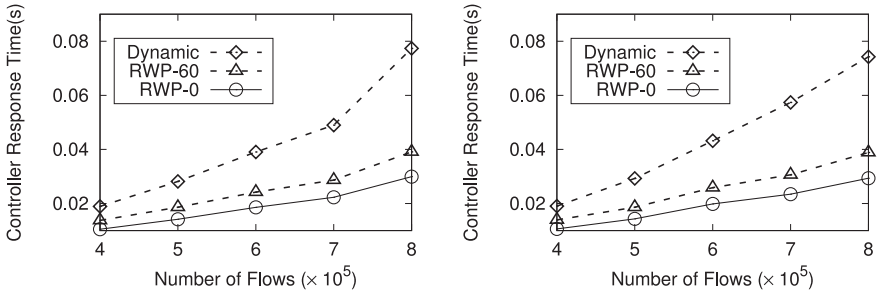


Fig. 12. Controller Response Time vs. Number of Flows. *Left plot*: Topology (a); *right plot*: Topology (b).

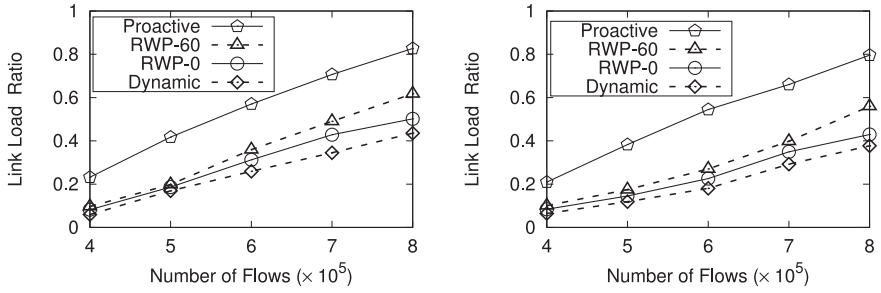


Fig. 13. Link Load Ratio vs. Number of Flows. *Left plot*: Topology (a); *right plot*: Topology (b).

ing scheme. For example, given 800 K flows in the network, the controller load ratio will reach 0.51 and 0.80 by the RWP-0 algorithm and the dynamic routing scheme, respectively. In other words, our proposed algorithm can reduce the controller load ratio by about 36% compared with the dynamic routing scheme. Heavier controller load also leads to longer controller response time, which is also validated by Fig. 12. This figure shows that our proposed algorithm can reduce the controller response time about 47–58% compared with the dynamic routing scheme. Figs. 11 and 12 also show that the performance of RWP-60 is very close to that of RWP-0.

The last set of simulations observes the routing performance of different algorithms by changing the different parameters (*e.g.*, the number of flows or the controller response time constraint) on two topologies. Fig. 13 shows that the link load ratio is increasing with more flows in a network. Since the proactive routing scheme will not dynamically adjust the routes for flows, its routing performance is worst among these solutions. For example, the left plot of Fig. 13 shows that the RWP-60 algorithm can reduce the link load ratio about 33% compared with the proactive routing scheme. Moreover, the link load ratio gap between our RWP-0 algorithm and the dynamic routing scheme is less than 5%. Fig. 14 shows that the network throughput is increasing with more flows in a network. But, the increasing ratio is slower with more flows. Besides. this figure shows that our proposed algorithm can improve the network throughput by about 56% compared with the proactive routing scheme, and achieve close network throughput compared with the dynamic scheme. Figs. 15 and 16 show the impact of the controller response time constraint on the routing performance, including link load ratio and network throughput. Even with a lower controller response time (*e.g.*, 0.01 s), our proposed routing scheme can achieve better routing performance than the proactive routing scheme. When the controller response time constraint is 0.04 s, our algorithm can achieve the similar routing performance (both link load ratio and network throughput) compared with the dynamic routing scheme, which will lead to a higher controller response time (*e.g.*, about 0.12 s).

Fig. 17 compares the routing performance of DevoFlow and our RWP algorithm. In this simulation, we gradually generate flows in the network with running time. This figure shows that the link load ratio is increasing with more flows in the network for both algorithms. At the beginning of system running, all flows follow the default paths in the DevoFlow, which leads to higher link load ratio than ours. After the controller detects the potential link congestion in the network, the controller will install some fine-grained rules to re-route some flows so as to achieve the load balancing in the network. By this figure, we can find that though DevoFlow can reduce the controller response time, it apt to be link congestion in the network and be load imbalance.

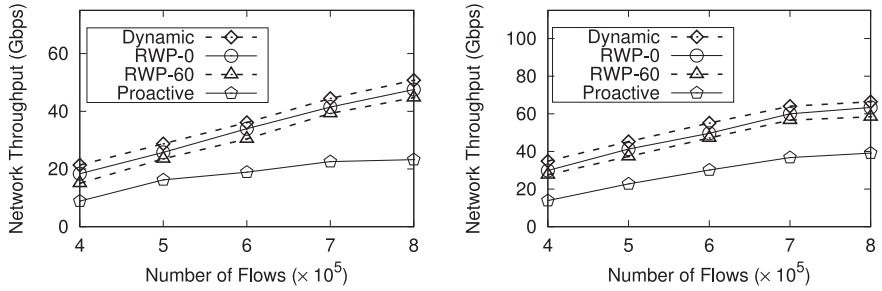


Fig. 14. Network Throughput vs. Number of Flows. *Left plot*: Topology (a); *right plot*: Topology (b).

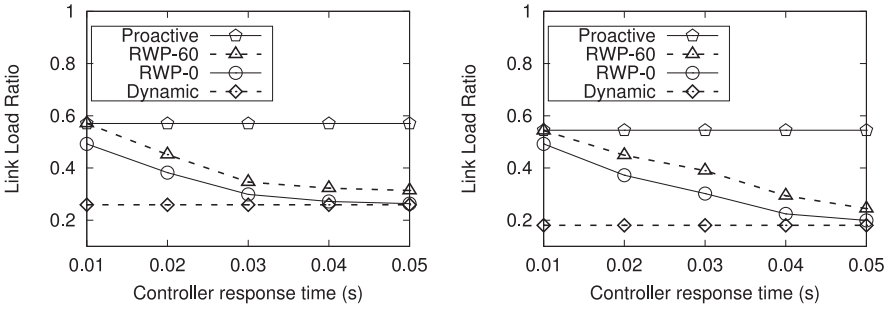


Fig. 15. Link Load Ratio vs. Controller Response Time Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

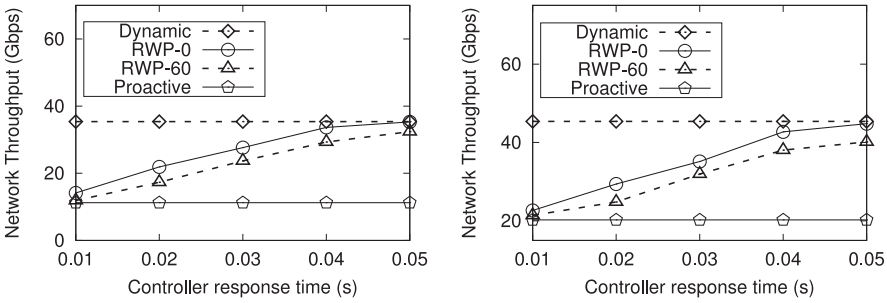


Fig. 16. Network Throughput vs. Controller Response Time Constraint. *Left plot*: Topology (a); *right plot*: Topology (b).

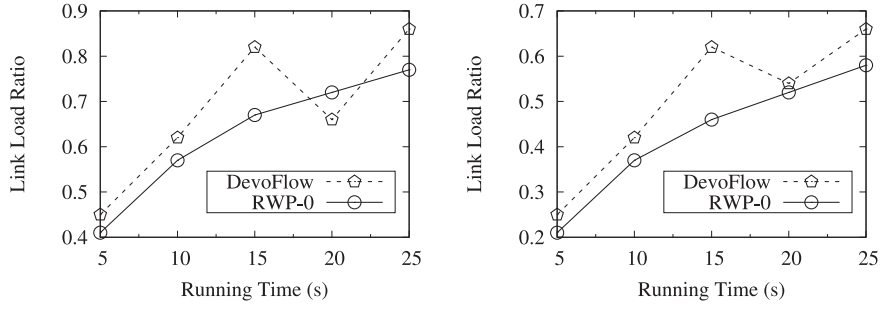


Fig. 17. Link Load Ratio vs. Running Time. *Left plot*: Topology (a); *right plot*: Topology (b).

From these simulation results, we can make three main conclusions. First, the RWP algorithm can perform much better than the dynamic routing scheme for the metrics of controller load ratio and controller response time, while achieving the close routing performance. Second, our proposed algorithm can obtain better routing performance (*e.g.*, link load ratio and network throughput) than the proactive routing scheme, while meeting the controller response time constraint. Third, all figures show that our RWP algorithm can work well for different performance metrics even with various prediction errors. Moreover, our RWP algorithm can achieve better trade-off between controller response time and routing performance by trading some data plane resource.

6. Related works

With more applications of data centers [42–44], software defined networking becomes an emerging technology for flexible resource and network management. Since the controller will provide the centralized control function for all flows, it plays an important role for an SDN. However, when the controller should determine the route paths for many flows, it may lead to long controller response time. There are three different ways for dealing with this challenge.

For the first category, a natural way is to deploy a cluster of controllers in the network. Koponen et al. [4] proposed the Onix

platform on top of which the control plane could be fulfilled as a distributed system. Specifically, control planes operated on a global view of the network, and used basic state distribution primitives provided by the platform. Paris et al. [45] addressed the challenge about the interplay between high degree of configuration flexibility and the computational limits of SDN controller logic. Dixit et al. [41] proposed ElastiCon, an elastic distributed controller architecture in which the controller pool was dynamically changed according to traffic conditions and the load was dynamically shifted among controllers. They also proposed a novel switch migration protocol for such load shifting. However, due to traffic dynamics, some controllers may be even congested, which will still increase the response time on these controllers. Furthermore, to process more flows, it is required to deploy more controllers, which will increase the additional communication/management overhead and the consistency-maintenance cost among more controllers. Moreover,

For the second category, each switch is permitted to be dynamically assigned to a controller so as to decrease the controller load. Wang et al. [6] studied a dynamic controller assignment problem so as to minimize the average response time of the controllers. In order to solve the problem efficiently, they proposed a hierarchically two-phase algorithm that integrated key concepts of both matching theory and coalitional games. The work [46] proposed a novel scheme to dynamically associate switches with controllers and dynamically devolve control of flows to switches. As described in Section 1, the dynamic controller assignment methods [6,46] bring some disadvantages on communication delay and control plane overhead.

The final category is the proactive routing scheme. In other words, the controller will pre-install wildcard rules for all flows, and dynamically adjust the routes of flows for performance optimization, such as load balancing and throughput maximization [9–11]. Using this method, the controller only installs rules for those elephant flows, the controller load is very light and the controller response time is low. Due to dynamic traffic intensity, it may result in transient congestion on some data links, for the controller can not provide dynamic route control for flows. This will lead to packet dropping and throughput reduction. As another example, the authors in [47] proposed AggreFlow, a dynamic flow scheduling scheme that pre-installs wildcard rules for some flow-sets. However, it needs extra flow entries for rerouting, while turning off some switches and links to achieve power efficiency, leading to higher link utilization than our proposed solution, with the same flow traffic.

The above solutions may lead to massive controller overhead, long communication delay, or transient network congestion. Thus, it is of vital importance to implement low controller response time while conquering the above disadvantages.

7. Conclusion

In this paper, we have studied how to reduce the controller response time with minimum data plane resource cost, and proved its NP-hardness. We have presented an efficient algorithm based on the randomized rounding method. The experimental results and extensive simulation results have shown high efficiency of our proposed algorithm. In the future, we will study some more practical issues. For example, our proposed scheme will update the wildcard forwarding entries at the beginning of each time slot. However, if this time slot is very busy, it is inappropriate to perform these operations. Thus, how to deal with this issue is a future challenge. Moreover, it is another challenge to combine our solution with service function chaining and service policies.

Declaration of Competing Interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled, Reducing Controller Response Time with Hybrid Routing in Software Defined Networks.

Acknowledgements

This research of Xu and Liu is partially supported by the National Science Foundation of China (NSFC) under Grants 61822210, U1709217, and 61936015; by Anhui Initiative in Quantum Information Technologies under No. AHY150300. The research of Qian is partially supported by National Science Foundation (NSF) Grant 1750704. The research of He Huang is partially supported by National Natural Science Foundation of China (NSFC) under Grant No. 61873177, No. 61572342

References

[1] L.A. Barroso, J. Dean, U. Holzle, Web search for a planet: the Google cluster architecture, IEEE Micro 23 (2) (2003) 22–28.
[2] S. Androutsellis-Theotokis, D. Spinellis, A survey of peer-to-peer content distribution technologies, ACM Comput. Surv. (CSUR) 36 (4) (2004) 335–371.
[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined wan, in: Proceedings of the ACM SIGCOMM, 2013.
[4] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al., Onix: A distributed control platform for large-scale production networks., in: Proceedings of the USENIX OSDI, 2010.
[5] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, et al., Network virtualization in multi-tenant datacenters, in: Proceedings of the USENIX NSDI, 2014.
[6] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: Stable matching with transfers, in: Proceedings of the IEEE INFOCOM, 2016.
[7] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, Logically centralized?: state distribution trade-offs in software defined networks, in: Proceedings of the ACM HotSDN, 2012.
[8] T. Wang, F. Liu, H. Xu, An efficient online algorithm for dynamic SDN controller assignment in data center networks, IEEE/ACM Trans. Netw. 25 (5) (2017) 2788–2801.
[9] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 41, ACM, 2011, pp. 254–265.
[10] H. Xu, H. Huang, S. Chen, G. Zhao, Scalable software-defined networking through hybrid switching, in: Proceedings of the IEEE INFOCOM, 2017.
[11] H. Xu, H. Huang, S. Chen, G. Zhao, L. Huang, Achieving high scalability through hybrid switching in software-defined networking, IEEE/ACM Trans. Netw. 26 (1) (2018) 618–632.
[12] X. Jin, H.H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, R. Wattenhofer, Dynamic scheduling of network updates, in: Proceedings of the ACM SIGCOMM, 2014.
[13] H.H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, D. Maltz, Zupdate: updating data center networks with zero loss, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 411–422.
[14] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, Real-time update with joint optimization of route selection and update scheduling for SDNS, in: Proceedings of the IEEE ICNP, 2016.
[15] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, T. Jung, H. Xu, Z. Yu, X.-Y. Li, L. Huang, et al., Joint route selection and update scheduling for low-latency update in SDNS, IEEE/ACM Trans. Netw. 25 (5) (2017) 3073–3087.
[16] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, S.A. Khayam, Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane, in: Proceedings of the IEEE Software Defined Networking, European Workshop on, 2012.
[17] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the Presented as Part of the USENIX Hot-ICE, 2012.

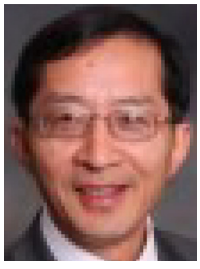
[18] S. Skiena, Dijkstra's algorithm, Addison-Wesley, MA, 1990, pp. 225–227.
[19] B. Pfaff, B. Lantz, B. Heller, et al., Openflow switch specification v1. 3.0[J], Open Netw. Found., Menlo Park, CA, USA, 2012 Tech. Rep. ONF TS-006.
[20] X. Jin, L.E. Li, L. Vanbever, J. Rexford, Softcell: Scalable and flexible cellular core network architecture, in: Proceedings of the ACM Proceedings of the CoNEXT, 2013.
[21] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, in: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, ACM, 2009, pp. 65–72.
[22] R. Cohen, L. Lewin-Eytan, J.S. Naor, D. Raz, On the effect of forwarding table size on SDN network utilization, in: Proceedings of the IEEE INFOCOM, 2014.
[23] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 43, ACM, 2013, pp. 15–26.
[24] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, in: Proceedings of the 16th Annual Symposium on Foundations of Computer Science, IEEE, 1975.
[25] P. Raghavan, C.D. Tompson, Randomized rounding: a technique for provably good algorithms and algorithmic proofs, Combinatorica 7 (4) (1987) 365–374.
[26] M. Zink, K. Suh, Y. Gu, J. Kurose, Characteristics of youtube network traffic at a campus network–measurements, models, and implications, Comput. Netw. 53 (4) (2009) 501–514.
[27] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, Minimizing flow statistics collection cost of SDN using wildcard requests, in: Proceedings of the 16th Annual Symposium on IEEE INFOCOM, 2017a.
[28] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, L. Huang, Minimizing flow statistics collection cost using wildcard-based requests in SDNS, IEEE/ACM Trans. Netw. 25 (6) (2017) 3587–3601.
[29] H. Xu, X.-Y. Li, L. Huang, Y. Du, Z. Liu, Partial flow statistics collection for load--balanced routing in software defined networks, Comput. Netw. 122 (2017) 43–55.
[30] Netflow, (http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html).
[31] M. Wang, B. Li, Z. Li, sflow: Towards resource-efficient and agile service federation in service overlay networks, in: Proceedings of the IEEE ICDCS, 2004.
[32] P. Lieven, B. Scheuermann, High-speed per-flow traffic measurement with probabilistic multiplicity counting, in: Proceedings of the IEEE INFOCOM, 2010.
[33] T. Li, S. Chen, Y. Ling, Fast and compact per-flow traffic measurement through randomized counter sharing, in: Proceedings of the IEEE INFOCOM, 2011.
[34] K. Papagiannaki, N. Taft, Z.-L. Zhang, C. Diot, Long-term forecasting of internet backbone traffic, IEEE Trans. Neural Netw. 16 (5) (2005) 1110–1124.
[35] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E.D. Kolaczyk, N. Taft, Structural analysis of network traffic flows, in: Proceedings of the ACM SIGMETRICS, 32, 2004, pp. 61–72.
[36] Ryu, (http://osrg.github.com/ryu/).
[37] lyl617, "RWP algorithm", https://github.com/lyl617/SDN-RWP.
[38] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, Wa, Usa, August, 2008.
[39] The network topology from the monash university, (http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies).
[40] H3c s5130-ei, (http://www.h3c.com/cn/Products___Technology/Products/Switches/Park_switch/S5130/S5130-EI/).
[41] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, in: Proceedings of the ACM SIGCOMM Computer Communication Review, 43, ACM, 2013, pp. 7–12.
[42] J. Guo, F. Liu, T. Wang, J.C. Lui, Pricing intra-datacenter networks with over-committed bandwidth guarantee, in: Proceedings of the USENIX ATC, 2017.
[43] J. Guo, F. Liu, J. Lui, H. Jin, Fair network bandwidth allocation in IAAS datacenters via a cooperative game approach, IEEE/ACM Trans. Netw. 24 (2) (2016) 873–886.
[44] F. Liu, J. Guo, X. Huang, J.C. Lui, EBA: efficient bandwidth guarantee under traffic variability in datacenters, IEEE/ACM Trans. Netw. 25 (1) (2017) 506–519.
[45] S. Paris, A. Destounis, L. Maggi, G. Paschos, J. Leguay, Controlling flow reconfigurations in SDN, in: Proceedings of the IEEE INFOCOM, 2016.
[46] X. Huang, S. Bian, Z. Shao, H. Xu, Dynamic switch-controller association and control devolution for SDN systems, arXiv:1702.03065 (2017).
[47] Z. Guo, S. Hui, Y. Xu, H.J. Chao, Dynamic flow scheduling for power-efficient data center networks, in: Proceedings of the IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), IEEE, 2016, pp. 1–10.

Hongli Xu received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2002 and 2007, respectively. He is currently a research Professor with the School of Computer Science and Technology, University of Science and Technology of China. He has authored or coauthored over 70 papers, and held about 30 patents. His main research interest is software]defined networks, cooperative communication, and vehicular ad hoc network.

Jianchun Liu is currently pursuing the Ph.D. degree in Computer Science at the University of Science and Technology of China. His main research interests are Edge computing, Networking for AI, SDN and IOT.

Chen Qian received the B.S. degree from Nanjing University in 2006, the M.Phil. degree from The Hong Kong University of Science and Technology in 2008, and the Ph.D. degree from The University of Texas at Austin in 2013, all in computer science. He is currently an Assistant Professor with the Department of Computer Engineering, University of California at Santa Cruz. His research interests include computer networking, network security, and Internet of Things. He has authored over 60 research papers in highly competitive conferences and journals. He is a member of the ACM.

He Huang Dr. He Huang is an associate professor in the School of Computer Science and Technology at Soochow University, P.R. China. He received his Ph.D. degree in Department of Computer Science and Technology from University of Science and Technology of China (USTC), in 2011. His current research interests include traffic measurement, spectrum auction, privacy preserving in auction, and algorithmic game theory. He is a Member of both IEEE and ACM.



Chunming Qiao directs the Lab for Advanced Network Design, Analysis, and Research (LANDER) at SUNY Buffalo. He is a recipient of many awards including the SUNY Chancellors Award for Excellence in Scholarship and Creativity. He has published many highly cited papers with an h]index of over 50 (according to Google Scholar). He has seven US patents. He pioneered research on Optical Internet in 1997, and one of his papers has been cited by more than 2,000 times. He also pioneered the work on integrated cellular and ad hoc relaying systems (iCAR) in 1999, which is recognized as the harbinger for todayfs push towards the convergence between heterogeneous wireless technologies, and has been featured in Business Week and Wireless Europe, as well as at the websites of New Scientists and CBC. These works have inspired a lot of follow]up works around the world and attracted funding from a dozen of major IT and telecommunications companies including Alcatel, Cisco, Google, NEC labs, Nokia, and Telcordia. Dr. Qiao has given a dozen of keynotes, and numerous invited talks on the above research topics. He has chaired and co]chaired a dozen of international conferences and workshops. He was an editor of several leading IEEE journals, and chaired a number of technical committees. He was elected to IEEE Fellow for his contributions to optical and wireless network architectures and protocols.