Exploring the Impact of Critical Programmability on Controller Placement for Software-Defined Wide Area Networks

Songshi Dou, *Graduate Student Member, IEEE, ACM*, Li Qi, Chao Yao, and Zehua Guo, *Senior Member, IEEE, Member, ACM*

***Abstract*— Control latency is a critical concern for deploying Software-Defined Networking (SDN) into Wide Area Networks (WANs). A Software-Defined WAN (SD-WAN) can be divided into multiple domains controlled by multiple controllers with a logically centralized view. The control latency is related to the placement of controllers and mappings between switches and controllers. Existing solutions usually consider the propagation delay between switches and controllers as the evaluation metric and fail to consider many important factors of dynamic network states. In this paper, we propose ProgrammabilityExplorer (PE) to optimize the control latency in SD-WAN. Inspired by the selection of critical flows, which have a critical impact on network performance, PE considers the programmability of critical flows at switches and uses this metric to decide the placement of controllers and mappings between switches and controllers. Simulation results show that PE can reduce the control latency by up to 62.3%, 27.5%, 58.3%, and 61.7% under GÉANT, Abilene, Sprintlink, and Tiscali topologies respectively, compared with baseline algorithms.**

***Index Terms*— Software-defined networking, wide area networks, critical programmability, controller placement, switch-controller mapping, control latency.**

I. Introduction

Wide Area Networks (WANs) are of great significance in the real world. It connects different types of networks (*e.g.*, data center networks, cellular networks, and metropolitan area networks) and transfers traffic among these networks to provide network applications. The emerging Software-Defined Networking (SDN) promises significant improvement in the performance of WANs with logically centralized control. Typically, a Software-Defined WAN (SD-WAN) is of large scale and is usually composed of multiple domains, each of which consists of one SDN controller to control flows traversing SDN switches placed in its domain. SDN controllers synchronize with each other to maintain a global network view [2].

The flexible flow control of SDN comes from the network programmability [3]. Once a flow traverses an SDN switch, it becomes a programmable flow and is able to change its forwarding path, which is defined as network programmability. Network programmability is empowered by the SDN controller through deploying flow entries to adjust flows' forwarding paths. With a higher network programmability, the SDN controller could frequently generate proper routing policy and further adjust forwarding paths of corresponding flows, and the SD-WAN could benefit from a lower chance of experiencing performance fluctuation under potential traffic variation.

In order to fully utilize the SDN controller's capability, an SD-WAN faces two major problems: (1) where to place the controllers and (2) how to properly assign switches to controllers. The ways we handle the above two problems make a great difference to control latency between switches and controllers. Control latency is defined as the propagation delay between switches and controllers and can be used for interaction between controllers and switches, such as collecting Traffic Matrices (TMs) from switches and applying new routing policies to switches. Since computing new routing policies does not involve the above-mentioned interaction between controllers and switches, the calculation of control latency does not include this part. Control latency is one essential concern for WAN operators. Control latency may have a negative impact on a wide range of network issues (*e.g.*, routing policy updates, load balancing, and energy efficiency) and even degrade the network performance [4]. To address the above-mentioned problems, Heller et al. [5] are the first to propose Controller Placement Problem (CPP), which mainly focuses on placing controllers at proper locations and establishing suitable switch-controller mapping.

Typically, existing solutions for minimizing control latency are typically coarse-grained location-based, and they treat each switch equally and place controllers only based on the distance between switches and controllers [5], [6], [7]. However, the number of flows that traverse switches varies significantly during traffic fluctuation in real-world situations. It is not

Manuscript received 25 June 2022; revised 14 January 2023; accepted 20 February 2023; approved by IEEE/ACM Transactions on Networking Editor K. C.-J. Lin. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB1714800, in part by the National Natural Science Foundation of China under Grant 62002019 and Grant 61802315, in part by the Zhejiang Laboratory Open Research Project under Grant K2022QA0AB02, in part by the Songshan Laboratory Fund under Grant YYJC022022009, and in part by the Beijing Institute of Technology Research Fund Program for Young Scholars. This paper was presented in part at SocialMeta'22 [1] [DOI: 10.1109/ICDCSW56584.2022.00057]. *(Songshi Dou and Li Qi contributed equally to this work.) (Corresponding author: Zehua Guo.)*

Songshi Dou, Li Qi, and Zehua Guo are with the Beijing Institute of Technology, Beijing 100081, China (e-mail: guolizihao@hotmail.com).

Chao Yao is with the Shaanxi Normal University, Xi'an 710119, China.

Digital Object Identifier 10.1109/TNET.2023.3252639

appropriate to simply treat switches at different locations as the same because some switches may have a higher chance to communicate with controllers while others just get in touch a few times. Thus, simple location-based solutions cannot guarantee low control latency, and designing the controller placement and switch-controller mapping strategies should consider the features of different flows in real-world situations.

To mitigate the negative impact of the above-mentioned problem, flow number-aware solutions [8], [9] are proposed to consider the network state in real time. Nevertheless, they neglect two important issues. First, some critical flows are more likely to experience high traffic load and network congestion, which means that these flows contribute more to the communications between switches and controllers by routing/rerouting flows to improve the network performance. Second, the frequency of communications between controllers and switches is also largely affected by the network programmability of flows at each switch. With a higher network programmability, controllers are more likely to install/ update/ delete paths of flows when facing network variation.

In this paper, we propose ProgrammabilityExplorer (PE), a novel controller placement and switch-controller mapping solution. The key design of PE is to differentiate the impact of switches on control operations by jointly considering critical flows and network programmability.

The contributions of this paper are summarized as follows:

- We formulate the Controller Placement and Switch-Controller Mapping (CPSCM) problem as an optimization problem, which aims to minimize the control latency of controllers by properly deciding the controller placement and switch-controller mapping.
- We provide a rigorous proof of the CPSCM problem to be NP-hard and propose a heuristic solution named PE to efficiently solve the problem.
- We propose a new metric named critical programmability latency to show the impact of critical programmability on control latency when deciding the suitable controller placement and switch-controller mapping strategies.
- We evaluate the performance of PE under four backbone topologies with real-world and synthetic traffic traces. Simulation results show that PE can reduce the control latency by up to 62.3%, 27.5%, 58.3%, and 61.7% under GÉANT, Abilene, Sprintlink, and Tiscali topologies respectively, compared with baseline algorithms.

The rest of the paper is organized as follows. In Section II, we introduce the background. In Section III, we introduce the system design of this paper. Section IV formulates our problem as the CPSCM problem. Section V proposes a heuristic solution to efficiently solve the problem. We evaluate and analyze the performance of PE in Section VI. Section VII introduces the related work, and Section VIII concludes this paper.

II. Background and Motivation

A. Controller Placement in SD-WANs

Controller placement means the physical placement of controllers. The controller is typically a network control software installed on a physical server or a virtual machine. An SD-WAN is usually composed of multiple network domains. Each domain contains several SDN switches and has a master SDN controller to quickly respond to the requests from the SDN switches within the domain [10]. Fig. 1 shows an example of controller placement for AT&T network. In this figure, an SD-WAN is composed of four domains, and each domain is managed by one SDN controller. The four SDN controllers maintain consistent network information via controller synchronization. Thus, both of the controllers have consistent information.

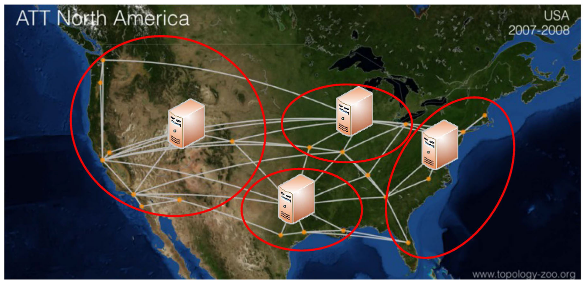


Fig. 1. An example of controller placement for AT&T network [11].

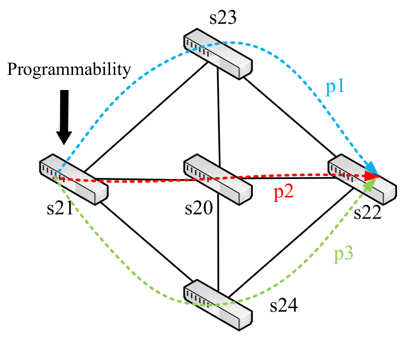
An SDN switch would request the SDN controller for appropriate forwarding actions through the control messages for each new flow, which means that the controller's control delay between the switch and the controller largely affects the flow setup time in SDNs. As a result, where to place all the SDN controllers is an important design consideration since it directly affects the control latency experienced by the SDN switches, which in turn affects a variety of network issues (*e.g.*, routing policy updates, fault tolerance, and energy efficiency). Note that WANs have their own traffic patterns, and traffic patterns are relatively stable for each network [12]. Thus, once controllers are placed, we will not relocate these controllers instead of only adjusting the switch-controller mapping strategies to accommodate the bursty traffic.

B. Network Programmability

SDN introduces network programmability to improve network performance by dynamically routing/rerouting flows under network variation. The network programmability of one flow at a switch is denoted as the number of admissible paths from the switch's next hops to the flow's destination. Fig. 2 illustrates the definition of network programmability. For flow f:$s_{21} \rightarrow s_{22}$, it has three paths (*i.e.*, $s_{21} \rightarrow s_{23} \rightarrow s_{22}$, $s_{21} \rightarrow s_{20} \rightarrow s_{22}$, and $s_{21} \rightarrow s_{24} \rightarrow s_{22}$) at s_{21}, thus its network programmability is three at s_{21}. On the switch s_{21}, the controller can determine the next hop of the flow f's forwarding path as s_{23}, s_{20}, and s_{24}, so as to adjust the forwarding path of the flow and realize the network programmability.

C. Traffic Pattern and Critical Flow

In a network, we do not need to control each flow in the network all the time. We only need to selectively change the paths of some flows to cope with network issues (*e.g.*, congestion), and other flows are always forwarded on their



Flow	Admissible paths	Programmability
f: s21→s22	p1, p2, p3	3

Fig. 2. The network programmability of flow f at switch s_{21}.

current paths. A critical flow is defined as a flow which plays an important role in determining the network performance (*e.g.*, a flow on the most congested link), and existing works show that critical flows exist in a given TM [13]. Based on our analysis of real world TMs, our previous work CFR-RL [12] proposes to select critical flows in a network by utilizing Reinforcement Learning (RL). The critical flows have a high traffic load and are more likely to experience congestion. The critical flow selection is detailed in the following subsection.

D. Critical Flow Selection

Historical TMs, measured by the SDN switches and collected by the SDN controller on a regular basis [14], are used to train for getting a selection strategy. The critical flow selection module represents the selection strategy as a neural network that links raw observations (*e.g.*, a TM) to critical flows combinations. Neural networks offer an efficient and scalable way to integrate various TMs into a selection strategy. The critical flow selection module is based on training the neural network with customized REINFORCE algorithm [15]. When the training is completed, the critical flow selection module applies the critical flow selection strategy to each real-time TM periodically and selects a small number of critical flows in it. Selecting critical flows will not introduce another burden on the network, since they are selected when initializing the network by analyzing historical TMs.

Specifically, RL utilizes a three-tuple $\langle S_t, A_t, R_t\rangle$ to denote the interaction during the training. In this three-tuple, state space S_t is equal to $\mathrm{TM_t}$, which is the TM at time step t. Action space A_t represents the space of RL agents' output actions. For state S_t, the critical flow selection module selects K critical flows. We set $K = 10\% * N * (N-1)$ for each network following our previous work [12], where N stands for the number of nodes in the topology. After getting K different critical flows for a given state S_t, we obtain the Maximum Link Utilization (MLU) u by solving the Multi-Commodity Flow (MCF) optimization problem to reroute these critical flows, which is detailed in Section II-F. Reward space R_t is used to evaluate the effectiveness of the action. Reward R_t is set as $1/u$, which reflects the network load balancing performance after rerouting critical flows.

E. The Impact of Critical Programmability

Traffic Engineering (TE) is a typical network application which aims to improve the overall Quality of Service (QoS) in the network. To be specific, given a group of network flows with certain source nodes and destination nodes, TE will reroute flows by selecting one or more paths for each flow in the network under a specific objective function (*i.e.*, minimizing the MLU in the network). However, if we reroute all flows in the network, it is of high computational complexity and would cost a long time to obtain the optimal rerouting result. In order to solve this problem, the key idea is that we only need to reroute a small set of flows (*i.e.*, critical flows) but realize near-optimal MLU performance. Thus, computational complexity can be substantially decreased since we only reroute a small set of flows.

To demonstrate the significance of critical programmability, we use the metric of TE. If only rerouting a small set of critical flows exhibits the comparable performance of optimal TE (*i.e.*, rerouting all flows), it can be proved that critical flows do play an important role in the network, and the critical programmability should be considered when designing controller placement and switch-controller mapping strategies. The typical TE solution is to optimally reroute flows by solving the MCF optimization problem for preconfigured paths, which is detailed in the following subsection (*i.e.*, Section II-F).

F. Multi-Commodity Flow Formulation for Preconfigured Paths

An SD-WAN consists of T switches and N links between switches. The utilization of link e_n ($n \in [1, N]$) cannot exceed its upper bound capacity Cap_n. $F = \{f^1, f^2, \ldots, f^l, \ldots, f^L\}$ is the set of flows. For flow $f^l \in F$, its path-set consists of K paths and is denoted as $P_l = \{p_l^1, p_l^2, \ldots, p_l^k, \ldots, p_l^K\}$. We use z_k^l to denote the percentage of traffic demand routed on path p. And V_l is used to denote the traffic demand for flow f^l. Let u denote the MLU. The objective is to minimize u:

$$obj = \min u$$

Each flow f^l can only forward on one path, *i.e.*,

$$\sum_{k=1}^{K} z_k^l = 1, \quad \forall l \in [1, L]. \tag{1}$$

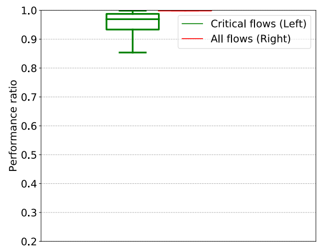
We can calculate u as follows:

$$\sum_{k=1}^{K}\sum_{l=1}^{L} z_k^l * V_l \le Cap_n * u, \quad \forall n \in [1, N]. \tag{2}$$

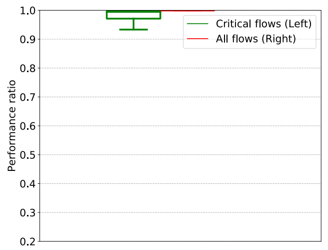
Therefore, the problem can be formulated as follows:

$$\begin{aligned} \min \quad & u \\ \text{s.t.} \quad & (1)(2), \\ & z_k^l \in \{0, 1\}, \quad \forall l \in [1, L],\ \forall n \in [1, N], \end{aligned} \tag{P}$$

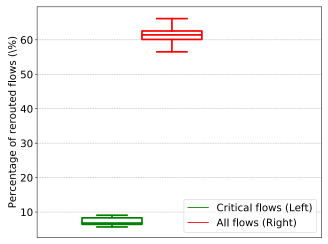
where $\{z_k^l\}$ is a binary design variable and u is a continuous design variable, $\{V_l\}$, and $\{Cap_n\}$ are given constants.



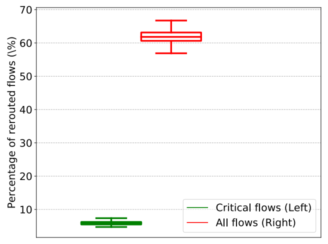
(a) The performance ratio in week 1.



(b) The performance ratio in week 2.

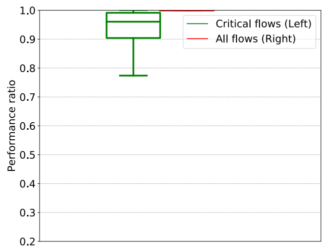


(c) The percentage of rerouted flows in week 1.

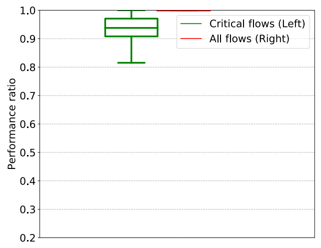


(d) The percentage of rerouted flows in week 2.

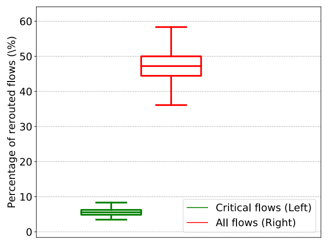
Fig. 3. The results under GÉANT topology.



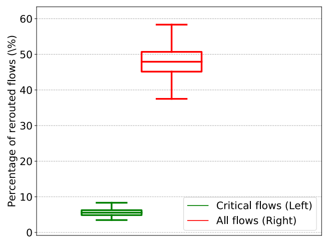
(a) The performance ratio in week 1.



(b) The performance ratio in week 2.



(c) The percentage of rerouted flows in week 1.



(d) The percentage of rerouted flows in week 2.

Fig. 4. The results under Abilene topology.

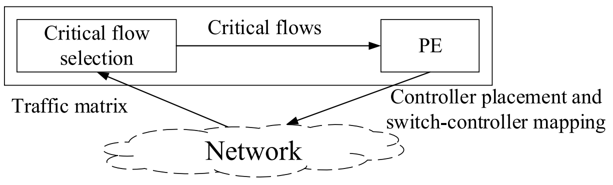


Fig. 5. System Design.

G. Observations

We use two typical backbone networks (*i.e.*, GÉANT and Abilene) to evaluate the impact of critical programmability. For GÉANT, the dataset records the TMs of 23 switches in a time slot of every fifteen minutes for a duration of four months [16]. We choose 1344 TMs in two weeks from June 2nd, 2005 to June 15th, 2005 as our dataset. The dataset of Abilene records the TMs of 12 switches in a time slot of every five minutes for a duration of six months [17]. We choose 4032 TMs in the first two weeks from May 10th, 2004 to May 16th, 2004 and July 5th, 2004 to July 11th, 2004 as our dataset. Each TM contains information on traffic demand for each flow with $N * N$ elements, where N denotes the number of nodes in each network. We use these TMs as input to optimally reroute flows (*i.e.*, calculating the routing policy to minimize MLU) by solving the MCF optimization problem. For simplicity, we use week 1-2 to represent the TMs for both GÉANT and Abilene topologies.

We use Performance Ratio (PR) to evaluate the impact of critical programmability on load balancing: $PR = \frac{P_{\text{optimal}}}{P_{\text{critical}}}$, where P_{optimal} denotes the optimal result of rerouting all flows, and P_{critical} denotes the result of only rerouting critical flows. As for non-critical flows, we reroute them by using Equal-Cost Multi-Path (ECMP) since ECMP reduces congestion probability by equally splitting traffic on equal cost paths. A higher PR indicates that critical programmability plays a more important role in network performance.

Figs. 3(a) and (b) present the results of the PR under GÉANT topology. In week 1, the medium PR of Critical flows (*i.e.*, the PR of only rerouting critical flows) is 0.97 while in week 2, the medium is 0.99, which is very close to 1. Figs. 3(c) and (d) present the percentage of rerouted flows. In week 1 and week 2, the mediums of the percentage of rerouted flows for Critical flows are both only 5%. While the mediums of All flows are both 62% in two weeks. In other words, we only reroute a small number of flows but can achieve near-optimal performance. Therefore, critical programmability plays an essential role in network performance.

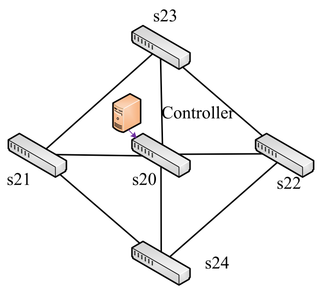
Figs. 4(a) and (b) show the results of the performance ratio of week 1 and week 2 under Abilene topology. In Fig. 4(a), the PR of Critical flows ranges from 0.77 to 1 with a medium of 0.96 in week 1, and the medium of Critical flows is 0.94 in week 2, which indicates that Critical flows achieve near-optimal performance. Figs. 4(c) and (d) present the percentage of rerouted flows. In the figure, the mediums of the percentage of rerouted flows for Critical flows are both 5% in week 1 and week 2, which is relatively lower than the mediums of All flows, *i.e.*, 47% and 48%. In a nutshell, critical programmability is very significant for network performance.

III. ProgrammabilityExplorer Overview

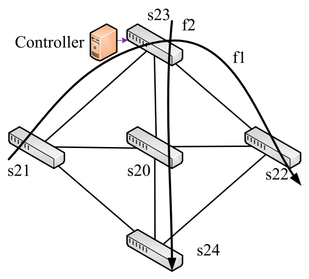
In this section, we propose to reduce the controller's control latency between switches and controllers by smartly deciding the placement of the controllers, and mapping the switches to the proper controllers.

A. System Design

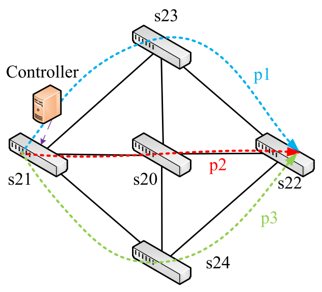
Fig. 5 shows the system design, which consists of two modules: Critical Flow Selection and PE. We generate a set of paths for each flow when the network is initialized. The critical flow selection module gets the information of flows and decides the critical flows using RL by analyzing historical TMs. The set of critical flows is then sent to the PE module. PE module smartly places controllers and maps



(a) LocationOnly solution.



(b) Flow number-aware solution. Two black lines denote two flows f_1 and f_2.



(c) Critical programmability-aware solution. The dashed lines denote three admissible paths of a critical flow $f_{critical}$ from switch s_{21} to switch s_{22}.

Fig. 6. Motivation examples.

switches to proper controllers to reduce the controller's control latency of switches. Specifically, the PE module calculates the total number of paths of critical flows that traverse a switch, which is denoted as the total programmability at the switch. Then the controllers are placed by considering the different priorities of programmability at switches and the propagation delay between locations. Following existing works [18], [19], the control resource of a controller refers to the processing ability of the controller to perform flow state pulling operations and get the network state variation without introducing extra control latency [20]. Thus, switches are mapped to controllers under the constraint of limited control resource of controllers. We formulate the CPSCM problem as an optimization problem and detail it in Section IV.

B. Motivation Examples

We use Fig. 6 to illustrate the design considerations of the controller placement and switch-controller mapping. There are five SDN switches s_{20}-s_{24} and an SDN controller C controls the five switches. Fig. 6(a) shows the result of the solution that only considers the control latency between locations. Controller C is placed at a location of s_{20}, which is the middle of this network. However, it is a coarse-grained placement solution because the network variation is not considered and we cannot simply treat different switches as the same. Fig. 6(b) considers the influence of flows, and places the controller C at the location of switch s_{23} through which flows traverse more often than other switches. But this solution treats all flows as the same, which is not quite convincing since flows are different in real-world situations as illustrated above. Fig. 6(c) exhibits our solution, p_1-p_3 are three paths of a critical flow, and the controller C is placed at the location of s_{21} which has the highest programmability of 3.

IV. Problem Formulation

In this section, we introduce how to optimally place controllers and map switches to controllers by making a description of the system, exhibiting constraints and the objective function, and formulating an optimization problem.

A. System Description

An SD-WAN consists of T switches at T locations. Among the T locations, we need to select H controllers at H locations and map the T switches to the H controllers. If the controller is placed at location j, $y_j = 1$; otherwise, $y_j = 0$. We use $x_{ij} = 1$ to denote that switch s_i is mapped to controller C_j; otherwise $x_{ij} = 0$. The network has the set of flows $F = \{f^1, f^2, \ldots, f^l, \ldots, f^L\}$. Flows from $[1, M]$ in the set F are critical flows while from $[M+1, L]$ are ordinary ones. For flow $f^l \in F$, its path-set consists of K paths and is denoted as $P_l = \{p_l^1, p_l^2, \ldots, p_l^k, \ldots, p_l^K\}$. The set of paths for F is $\mathcal{P} = \{P_1, P_2, \ldots, P_l, \ldots, P_L\}$. If path p_l^k traverses switch s_i, we have $\alpha_i^{lk} = 1$, otherwise $\alpha_i^{lk} = 0$. Note that $p_l^{k_0}$ denotes the ordinary path of flow f^l. We use D_{ij} $(D_{ij} \geq 0)$ to denote the propagation delay between switch s_i and controller C_j.

B. Constraints

1) Controller placement constraint: One switch can be only mapped to one controller only if the controller is placed. That is:

$$x_{ij} \leq y_j, \quad \forall i \in [1, T], \ \forall j \in [1, T]. \tag{3}$$

Besides, we can only place H controllers among T locations. That is:

$$\sum_{j=1}^{T} y_j = H. \tag{4}$$

2) Switch-controller mapping constraint: One switch is strictly mapped to one controller. That is:

$$\sum_{j=1}^{T} x_{ij} = 1, \quad \forall i \in [1, T]. \tag{5}$$

3) Controller resource constraint: The control resource of a controller refers to the processing ability of the controller to perform flow state pulling operations and get network state variations [18], [19]. Note that the controller has its control resource upper bound. If the controller exceeds this upper bound, extra control latency (*e.g.*, long-tail latency) will be introduced [20]. In the worst case, the network may face significant performance degradation due to cascading controller failures [21]. Thus, the control load of a controller should not exceed the controller's available control resource and can be written as follows:

$$\sum_{i=1}^{T} \left(\sum_{l=1}^{L} \alpha_i^{lk_0} * x_{ij} \right) \leq A_j * y_j, \quad \forall j \in [1, T], \tag{6}$$

where k_0 denotes the original path that flow f traverses, and A_j denotes the available control resource of controller C_j.

C. Objective Function

The objective is to minimize the control latency of controllers to control the switches that critical flows traverse, which is equal to the total propagation delay of the whole paths of critical flows between the switches and their mapped controllers. We use D_{ij} $(D_{ij} \geq 0)$ to denote the propagation delay between switch s_i and controller C_j and formulate the overhead as follows:

$$obj = \sum_{j=1}^{T}\sum_{i=1}^{T}((\sum_{l=1}^{M}\sum_{k=1}^{K}\alpha_i^{lk}) * D_{ij} * x_{ij}).$$

If we use δ_i to denote the total number that the whole paths of critical flows traverse switch s_i, which is also the critical flows' programmability at switch s_i, we can have:

$$\delta_i = \sum_{l=1}^{M}\sum_{k=1}^{K}\alpha_i^{lk}, \quad \forall i \in [1, T]. \tag{7}$$

Thus, we can formulate the objective function as follows:

$$obj = \sum_{j=1}^{T}\sum_{i=1}^{T}\delta_i * D_{ij} * x_{ij}.$$

D. Problem Formulation

The goal of our problem is to minimize the control latency between controllers and switches by selecting the location of the controllers and mapping switches to the placed controllers. Therefore, we formulate this problem as follows:

$$\begin{aligned} \min_{x,y} \quad & \sum_{j=1}^{T}\sum_{i=1}^{T}(\delta_i * D_{ij} * x_{ij}) \\ \text{s.t.} \quad & (3)(4)(5)(6)(7), \\ & y_j, x_{ij} \in \{0,1\}, \\ & \forall i \in [1,T],\ \forall j \in [1,T], \end{aligned} \tag{P'}$$

where $\{\delta_i\}$, $\{D_{ij}\}$, $\{A_j\}$, and H are constants, and $\{x_{ij}\}$ and $\{y_j\}$ are design variables. In this problem, the objective function is linear, and variables are binary integers. Thus, this problem is an Integer Linear Programming (ILP).

V. Solution

In this section, we first analyze the complexity of the CPSCM problem, and then propose our PE algorithm for solving the problem.

A. Complexity Analysis

Theorem 1: For a special case with one condition: the processing ability of controllers remains infinite, the CPSCM problem is NP-hard.

Proof: We first introduce the *p-median problem (PMP)* [22]. The PMP aims to minimize the sum of the distances of transportation costs of n users (or customers or demand points) to m facilities (or location points) such that each customer is precisely served from a facility subject to several restrictions. A typical formulation of the PMP is shown below:

$$\begin{aligned} \min_{x} \quad & \sum_{j=1}^{m}\sum_{i=1}^{n}(d_{ij} * x_{ij}) \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} = 1, \quad \forall i \in [1,n]. \\ & x_{ij} \leq y_j, \quad \forall i \in [1,n],\ \forall j \in [1,m]. \\ & \sum_{j=1}^{m} y_j = p. \\ & y_j, x_{ij} \in \{0,1\}, \\ & \forall i \in [1,n],\ \forall j \in [1,m], \end{aligned} \tag{P''}$$

where d_{ij} is the distance traveled (or cost incurred) for satisfying the demand of the user located at i from the facility located at j. Let us define two sets of decision variables: (1) $y_j = 1$, if a facility is opened in j, and 0, otherwise; (2) $x_{ij} = 1$, if customer i is served from facility located in j, and 0, otherwise. The three constraints respectively express that the demand of each user must be met, preventing any user from being supplied from a site with no open facilities, and the total number of open facilities is set to p.

It has been proved when opening multiple facilities and deciding the facility that the customer is served, the PMP is NP-hard [23]. We then prove for a special case of the condition introduced above, problem (P') and the PMP are equivalent problems. Given the condition that the processing ability of controllers remains infinite, which means that A_j in constraint (6) is infinite and we can neglect the constraint (6) anymore.

If we use w_{ij} to denote the total control latency from controller C_j to the switch s_i that the whole paths of critical flows traverse, we have

$$w_{ij} = \delta_i * D_{ij}, \quad \forall i \in [1,T],\ \forall j \in [1,T]. \tag{8}$$

Following the above condition and transformation, our CPSCM problem can be reformulated as follows:

$$\begin{aligned} \min_{z} \quad & \sum_{j=1}^{M}\sum_{i=1}^{N}(w_{ij} * x_{ij}) \\ \text{s.t.} \quad & (3)(4)(5), \\ & y_j, x_{ij} \in \{0,1\}, \quad \forall i \in [1,T],\ \forall j \in [1,T]. \end{aligned} \tag{P'''}$$

Problem (P''') aims to minimize the control latency of H controllers to T switches that the whole paths of critical flows traverse. We can treat switch s_i and controller C_j in problem (P''') as users i and facility j in the PMP. By this construction, it is easy to prove that there exists the minimum control latency by placing H controllers in T locations and mapping T switches to H controllers, if and only if there exists the optimal solution of the PMP by opening p facilities at m locations and letting n users serve for p facilities. The construction can be done in polynomial time. In problem (P'''), the mapping between switches and controllers could be many to one. Since the PMP is NP-hard when multiple users are served for a facility, and each user is performed exactly by one facility [22], problem (P''') is NP-hard. □

TABLE I
NOTATIONS

Notation	Meaning
x_{ij}	a binary design variable that denotes if switch s_i is mapped to controller C_j
y_j	a binary design variable that denotes if a controller is placed at location j
$\mathcal{W}(i)$	the total control latency of switch s_i, $\mathcal{W}(i) = \{w_{i1}, ..., w_{ij}, ..., w_{iT}\}, i \in [1, T]$
$\mathcal{C}(i)$	the set of locations by sorting $\mathcal{C} = \{C_j \mid j \in [1, T]\}$ following the ascending order of $\mathcal{W}(i)$, $i \in [1, T]$
$\mathcal{A}$	the set of the available processing capacity of controllers, $\mathcal{A} = \{A_j \mid j \in [1, T]\}$
$\mathcal{G}$	the number of flows in switches, which is denoted as $\sum_{l=1}^{L} \alpha_i^{lk_0}$, $\mathcal{G} = \{g_i \mid i \in [1, T]\}$
$\mathcal{X}$	the set of the mapping relationships between switches and controllers, $\mathcal{X} = \{(i, j) \in [1, T] \times [1, T] \mid x_{ij} = 1\}$
$\mathcal{Y}$	the set of placement of controllers, $\mathcal{Y} = \{i \in [1, T] \mid y_i = 1\}$
Δ	the set of the critical flows' programmability at switch s_i, which is calculated as the total number that the whole paths of critical flows traverse switch s_i, which is denoted at Eq. (7), $\Delta = \{\delta_i \mid i \in [1, T]\}$

Problem (P''') is a special case of the CPSCM problem and is NP-hard. Therefore, we can have the following conclusion:

Theorem 2: The CPSCM problem is NP-hard.

B. PE Algorithm

The common way to solve the above CPSCM problem is to use IP optimization solver to get the result. However, as the size of the network increases, the solution space can grow considerably, and finding a workable solution can take a long time or even be impossible. Thus, we propose a heuristic algorithm named PE to achieve a trade-off between performance and time complexity. The basic idea of PE is to first place a controller and then establish a switch-controller mapping for the controller until all controllers are placed and switches are mapped. A placement/mapping is tested following the descending placement/mapping probability. Specifically, we first solve the Linear Programming relaxation of problem (P') and sort the results in descending order to get the location of controllers following the descending probability. Then we decide the placement of each controller and test switches based on the descending order of the total number of paths that traverse the switches. Thus, we can ensure that switches with high network programmability can be tested first. If some switches remain unmapped, we test and establish switch-controller mapping relationships for these switches. Finally, we ensure that all switches are mapped. Details are summarized in Algorithm 1, and the notations used in the algorithm are listed in Table I.

Algorithm 1 PE Algorithm

```
Input: $\mathcal{C}(i)$, $A$, $\Delta$, $\mathcal{G}$;
Output: $\mathcal{X}$, $\mathcal{Y}$;
1:  $\mathcal{X} = \emptyset$, $\mathcal{Y} = \emptyset$;
2:  Generate testing mapping sets $\bar{X}^i = \{\bar{x}_k^i, k \in [1, F]\}$,
    $i \in [1, T]$ by solving the Linear Programming relaxation
    of problem (P') and sorting the results in the descending
    order;
3:  Generate vectors $I^*$ by getting the switch index from $\Delta$,
    which is sorted in the descending order;
4:  // test the switches based on the descending order of their
    total number of paths that traverse them
5:  for $i_0 \in I^*$ do
6:      for $\bar{x}_k^{i_0} \in \bar{X}^{i_0}$ do
7:          find controller ID $j_0$ of $\bar{x}_k^{i_0}$;
8:          if $\mathcal{X} \cup (i_0, j_0)$ and $\mathcal{Y} \cup j_0$ both satisfy the constraints
    in Eqs. (4) and (6) then
9:              $A_{j_0} = A_{j_0} - g_{i_0}$;
10:             $\mathcal{X} \leftarrow \mathcal{X} \cup (i_0, j_0)$;
11:             $\mathcal{Y} \leftarrow \mathcal{Y} \cup j_0$;
12:             break;
13:         end if
14:     end for
15: end for
16: // test and map the rest switches to the controller with the
    lowest control latency
17: if $|\mathcal{Y}| \neq |I^*|$ then
18:     for $i_0 \in (I^* - \mathcal{Y})$ do
19:         for $j_0 \in \mathcal{C}(i)$ do
20:             if $\mathcal{X} \cup (i_0, j_0)$ and $\mathcal{Y} \cup j_0$ both satisfy the
    constraints in Eqs. (4) and (6) then
21:                 $A_{j_0} = A_{j_0} - g_{i_0}$;
22:                 $\mathcal{X} \leftarrow \mathcal{X} \cup (i_0, j_0)$;
23:                 $\mathcal{Y} \leftarrow \mathcal{Y} \cup j_0$;
24:                 break;
25:             end if
26:         end for
27:     end for
28: end if
29: return $\mathcal{X}, \mathcal{Y}$;
```

In line 1, we initialize the set of mapping relationship $\mathcal{X}$ to be empty, and the set of controller placement $\mathcal{Y}$ to be empty at the beginning of the algorithm. In line 2, we generate vectors $\bar{X}^i = \{\bar{x}_k^i, k \in [1, F]\}$ for switch s_i ($i \in [1, T]$). We first relax binary variables in problem (P') to continuous variables and obtain the Linear Programming relaxation solution $\bar{X}_i^*$ of switch s_i. We then sort the values in $\bar{X}_i^*$ in descending order to get vectors $\bar{X}^i$. The sorting operation helps us test mappings based on their probabilities. Line 3 runs to make sure the switches with more paths traversing them are tested first, since our objective focuses on the different weights of switches when realizing the placement and mapping. In lines 5-15, we test all possible placement and mappings by rounding the decimal values in $\bar{X}^i$, $i \in [1, T]$ for switches to configure the right placement and mappings.

From line 6 to line 14, we test switch s_{i_0}. Line 7 gets the controller ID j_0 of possible mappings between the controller and switch s_{i_0}. In lines 8-13, if the placement of controller C_{j_0} and the mapping between controller C_{j_0} and switch s_{i_0} both satisfy the constraints in Eqs. (4) and (6), it is a feasible mapping. We will adjust the available control resource of controller C_{j_0}, which will control this switch, add this mapping to the set of switch-controller mapping relationships $\mathcal{X}$, and add the placement of the controller at the location j_0. In lines 17-28, if there still exist switches remaining unmapped, we need to map these switches to proper controllers. From line 19 to line 26, we test all controllers based on the ascending order of the total control latency of all paths traversing the switch s_{i_0}. If the placement of controller C_{j_0} and the mapping between controller C_{j_0} and switch s_{i_0} both satisfy the constraints in Eq. (4) and Eq. (6), it is a feasible mapping, we will change the available control resource of controller C_{j_0}, add this mapping to the set of switch-controller mapping relationships $\mathcal{X}$, and add the placement of the controller at the location j_0. In line 29, the algorithm returns the result and stops.

VI. Simulation

A. Simulation Setup

We conduct Python-based flow-level simulations and use four typical backbone topologies named GÉANT, Abilene, Sprintlink, and Tiscali from Topology Zoo [11] to evaluate the performance of PE. GÉANT network is the pan European and consists of 23 switches and 72 links. Abilene network is an educational backbone network in North America and consists of 12 switches and 30 links. Sprintlink is a network in America with 44 switches and 166 links. Tiscali network is a backbone network in Europe and consists of 49 switches and 172 links. Latitude and longitude location information for each switch is included in four topologies. SMORE is a state-of-the-art TE solution which can achieve load balancing and avoid network congestion by smartly selecting paths with dynamic weight adaptation [24]. More importantly, SMORE path-set utilizes diverse paths for robustness and is optimized for load balancing. Therefore, we use SMORE path-sets for GÉANT, Abilene, Sprintlink, and Tiscali in our simulation.

In our simulations, any two nodes have a traffic flow, and each flow is forwarded on its shortest path. Following existing works [3], [25], we set the control resource upper bound of a controller to 500. For GÉANT and Abilene, we place five and three SDN controllers, respectively. For Sprintlink and Tiscali, we place seven SDN controller clusters, and each cluster consists of three controllers to handle a large number of flows. Haversine formula [26] and the propagation speed (*i.e.*, 2 $\times 10^8$ m/s) [27] are used to calculate the propagation delay between different nodes. We use the Haversine formula to calculate the distance between the two nodes, and the details of the Haversine formula are listed as follows:

$$d = 2R\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\Phi_2-\Phi_1}{2}\right)+\cos\left(\Phi_1\right)\cos\left(\Phi_2\right)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right),$$

where R denotes the radius of the earth, d denotes the distance between two nodes, Φ_1 and Φ_2 denote the latitude of the two nodes, and λ_1 and λ_2 denote the longitude of the two nodes respectively. Thus, the propagation delay between two nodes can be calculated by dividing the distance by the propagation speed.

For GÉANT topology, the dataset records the TMs of 23 switches in a time slot of every fifteen minutes for a duration of four months [16]. We choose 2688 TMs in four weeks from June 2nd, 2005 to June 15th, 2005 and August 11th, 2005 to August 24th as our dataset. The dataset of Abilene records the TMs of 12 switches in a time slot of every five minutes for a duration of six months [17]. We choose 8064 TMs in four weeks from March 29th, 2004 to April 4th, 2004, May 3th, 2004 to May 16th, 2004, and July 5th, 2004 to July 11th, 2004 as our dataset. For simplicity, we use week 1-4 to represent the TMs in the four weeks for both GÉANT and Abilene topologies. The topology of Sprintlink and Tiscali are available from ROCKETFUEL [28], but its TMs are not available. Following existing work [12], we use a TM generation tool [29] to generate 70 synthetic exponential TMs and 70 synthetic uniform TMs for the network, and choose no more than four paths between any two nodes.

B. Comparison Algorithms

- Optimal: this is the optimal solution to the CPSCM problem. We solve the problem by using GUROBI optimization solver [30].
- LocationOnly: this kind of solutions [5], [6], [7] only considers the propagation delay between controllers and switches when minimizing the control latency.
- FlowNumber-aware: this kind of solutions [8], [9] takes the number of flows that traverse each switch into consideration to minimize the control latency between switches and controllers.
- PE: this solution takes the impact of critical programmability into consideration and decides the proper controller placement and switch-controller mapping using Algorithm 1.

C. Simulation Results

As detailed in Section II-E, critical programmability plays an important role in the network performance since we can achieve near-optimal performance by rerouting only a small number of the total traffic. We use the following three metrics to demonstrate the effectiveness of PE.

The first metric is the critical programmability latency which is also the objective function of CPSCM. Critical programmability latency is the control latency of controllers to control the switches that critical flows traverse. It is equal to the total propagation delay to control all critical flows (*i.e.*, the sum of the propagation delay between the switches that the whole paths of critical flows traverse and their mapped controllers). It can portray the impact of critical programmability on control latency when deciding the placement of the controllers and mapping switches to the placed controllers.

The second metric is the per-flow critical programmability latency, which describes the critical programmability latency for each critical flows and a lower critical programmability latency means a better result of CPSCM.

The third metric is the control latency which is induced by the installation of the flow entries. Load balancing is a major objective in networks. When rerouting flows to achieve load balancing in networks, we need to install flow entries at switches, which introduces control latency between controllers and switches. To evaluate the effectiveness of the proposed scheme in real-world situation, we optimally reroute flows by solving the MCF optimization problem for preconfigured

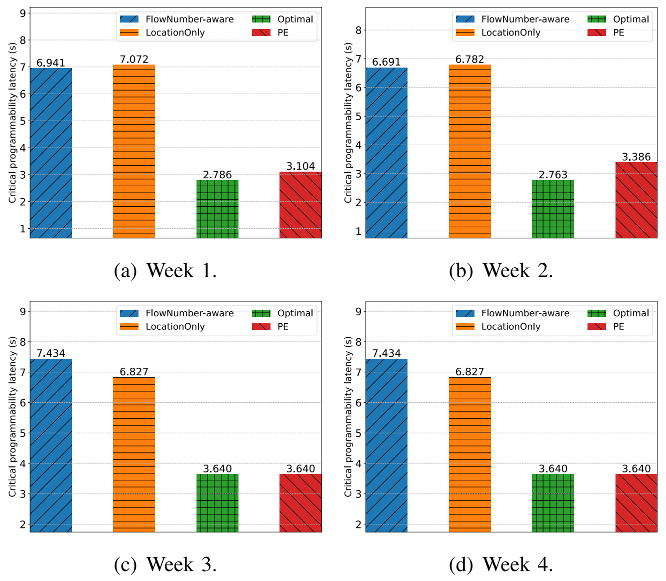


Fig. 7. The results of the total critical programmability latency under GÉANT topology. The lower, the better.

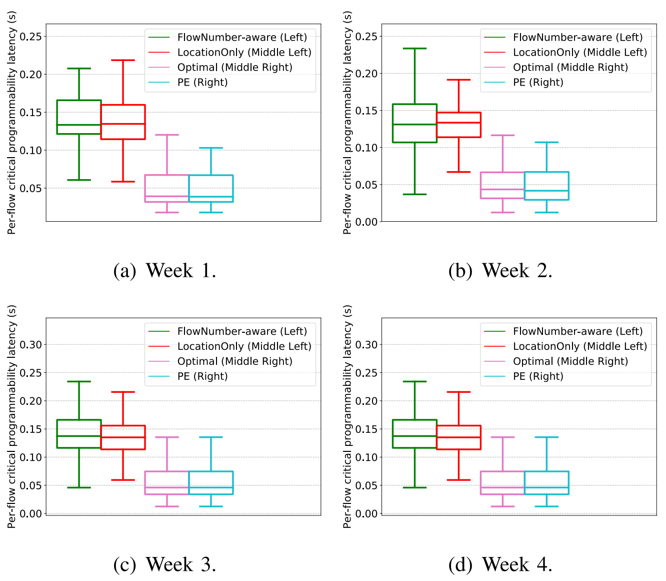


Fig. 8. The results of the per-flow critical programmability latency under GÉANT topology. The lower, the better.

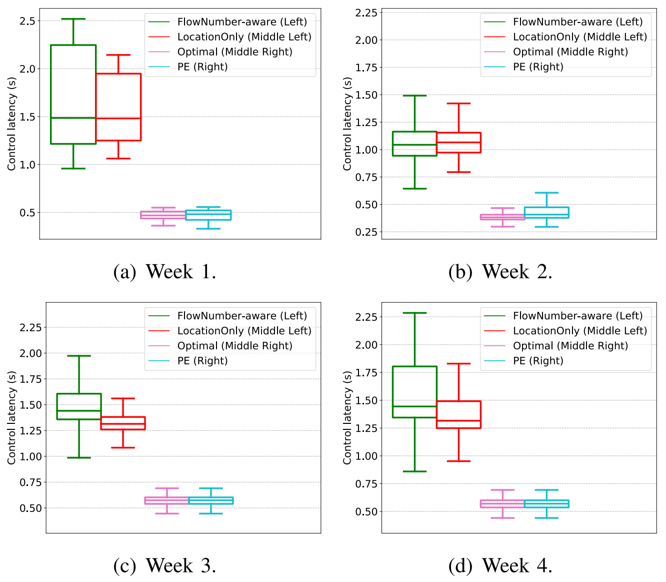


Fig. 9. The results of the control latency in real-world situation under GÉANT topology. The lower, the better.

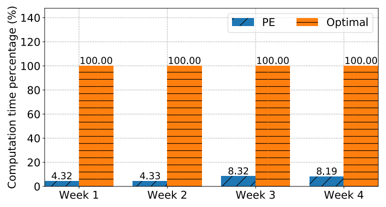


Fig. 10. The results of the computation time under GÉANT topology. The lower, the better.

paths, which is detailed in Section II-F. We calculate total control latency of all installations during each optimization. Note that all four algorithms utilize the same rerouting strategy.

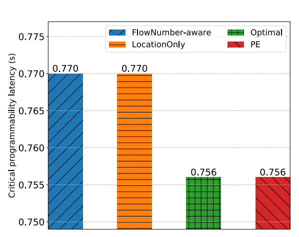
1) Under GÉANT Topology:

(1) Critical programmability latency: For GÉANT topology, Fig. 7 shows the results of critical programmability latency. Optimal performs the best by intelligently selecting the placement of controllers and mapping switches to controllers to minimize the critical programmability latency. PE's performance is near-optimal in week 1 and week 2. In week 3 and week 4, PE's performance is the same as Optimal since their placement of controllers and mapping between switches and controllers are the same. FlowNumber-aware and LocationOnly perform the worst because they cannot place the controllers at those switches with higher critical programmability, and thus their critical programmability latency is higher. Besides, the performance of the four algorithms is the same as a result of the same critical flows between week 3 and week 4.

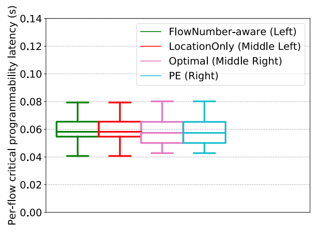
Fig. 8 shows the results of the per-flow critical programmability latency. PE and Optimal perform the best since their mediums are relatively lower than FlowNumber-aware and LocationOnly, while FlowNumber-aware and LocationOnly perform the worst. Note that in week 1 and week 2, PE's maximum and medium are slightly lower than Optimal's. This is because Optimal can minimize the total critical programmability latency but cannot ensure that each flow's critical programmability latency is minimized.

(2) Control latency in real-world situations: Fig. 9 shows the results of the control latency in real-world situation. PE's average control latency is 60.2% and 62.3% lower than LocationOnly and FlowNumber-aware, respectively. Besides, PE's performance is near-optimal in week 1. While in week 3 and week 4, PE's performance is the same as optimal since their placement of controllers and mapping between switches and controllers are the same. Note that in week 1, PE's minimum is slightly lower than Optimal's since control latency is an indirect result of total critical programmability latency, which means that Optimal's total critical programmability latency is lower as is shown in Fig. 8(a), while PE's minimum control latency is lower, as is shown in Fig. 9(a).

(3) Computation time: Fig. 10 illustrates the computation time percentage of PE and Optimal. In four weeks, PE's computation time percentage ranges from 4.32% to 8.32%

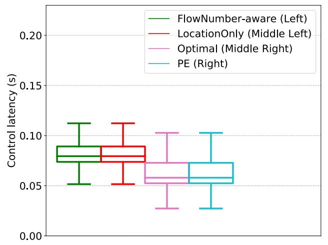


(a) Total critical programmability latency.

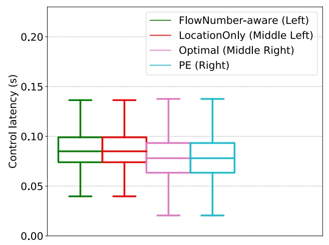


(b) Per-flow critical programmability latency.

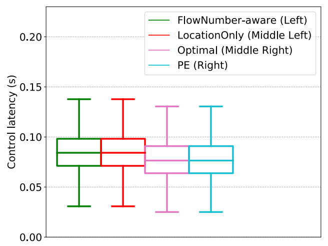
Fig. 11. The results in four weeks under Abilene topology. The lower, the better. Note that the results of week 1 to week 4 are the same since the critical flows are the same in these four weeks' TMs.



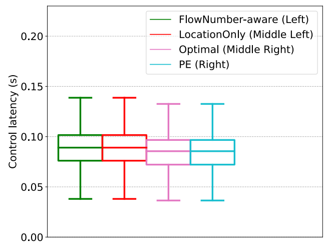
(a) Week 1.



(b) Week 2.



(c) Week 3.



(d) Week 4.

Fig. 12. The results of the control latency in real-world situation under Abilene topology. The lower, the better.

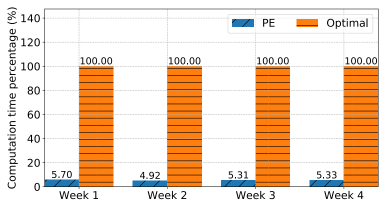


Fig. 13. The results of the computation time under Abilene topology. The lower, the better.

of Optimal and the average computation time percentage is 6.29% lower than Optimal, which demonstrates that PE reduces the computation time to a large extent.

2) Under Abilene Topology:

(1) Critical programmability latency: Fig. 11(a) shows the results of the critical programmability latency under Abilene topology. In Fig. 11(a), the critical programmability latency of FlowNumber-aware and LocationOnly are higher than Optimal and PE. This is because Optimal smartly selects the placement of controllers and establishes the mapping between switches and controllers to minimize the critical programmability latency. PE's performance is the same as Optimal because they produce the same results with regard to the placement of controllers and mapping between switches and controllers. Similarly, the placement of controllers and mapping between switches and controllers calculated by FlowNumber-aware and LocationOnly are also the same, so the critical programmability latency for them is the same. Since the Abilene topology is small and there are only three controllers in total, it's common that the placement of controllers and mapping between switches and controllers calculated by Optimal and PE, FlowNumber-aware and LocationOnly are the same respectively. Besides, since the critical flows are the same in four weeks, all four algorithms produce the same results in terms of critical programmability latency in four weeks.

Fig. 11(b) presents the results of the per-flow critical programmability latency. In Fig. 11(b), the medium of all four schemes is close to 0.06. However, PE and Optimal outperform FlowNumber-aware and LocationOnly in terms of critical programmability latency since the medium of PE and Optimal is about 0.0015 lower than the medium of FlowNumber-aware and LocationOnly, which is also shown in Fig. 11(a). Abilene topology is a small topology which only consists of 12 switches and 30 links, and the propagation delay between switches is relatively low. Thus, controller placement and switch-controller mapping strategies between different schemes do not exhibit obviously different under such a small topology compared with the other three larger topologies.

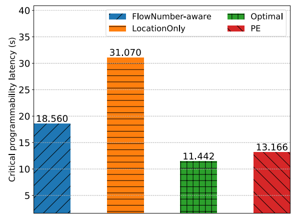
(2) Control latency in real-world situations: Fig. 12 shows the results of the control latency. In Figs. 12 (a), (b) and (d), PE's highest, medium and least are all lower than FlowNumber-aware and LocationOnly, which shows that the fine-grained solution considering the critical programmability at switches performs better than the coarse-grained solution which only considers the propagation latency between different locations. In Fig. 12 (a), our solution's medium is 27.5% lower compared with FlowNumber-aware and LocationOnly, while the mediums in Figs. 12 (b)-(d) are 7.2%, 9.6%, and 3.6% lower than FlowNumber-aware and LocationOnly respectively. The average control latency of our solution is 11.975% lower than FlowNumber-aware and LocationOnly.

(3) Computation time: We also compare the computation time of PE with Optimal and Fig. 13 presents the results of computation time. In the figure, PE's computation time percentage is 5.70%, 4.92%, 5.31%, and 5.33% of Optimal in four weeks. Thus, considering critical programmability latency and control latency in Figs. 11(a), 11(b), and 12, the result indicates that the proposed PE can achieve low control latency with relatively low computation time.

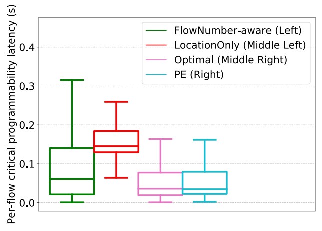
3) Under Sprintlink Topology:

(1) Critical programmability latency: Fig. 14(a) shows the critical programmability latency under Sprintlink topology. Optimal presents the best performance while FlowNumber-aware and LocationOnly perform the worst because they do not concern with the impact of critical programmability latency in terms of the placement of controllers and mapping between switches and controllers. Optimal presents the best result while PE's critical programmability latency is close to Optimal and substantially lower than FlowNumber-aware and LocationOnly, which indicates that PE shows a satisfactory result.

Fig. 14(b) presents the per-flow critical programmability latency. Optimal and PE present the best result since they take

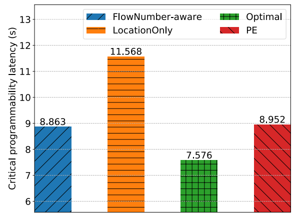


(a) Total critical programmability latency.

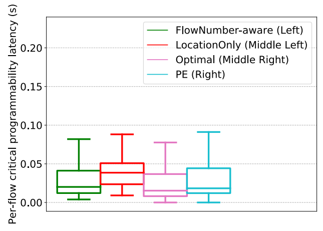


(b) Per-flow critical programmability latency.

Fig. 14. The results under Sprintlink topology. The lower, the better.



(a) Total critical programmability latency.



(b) Per-flow critical programmability latency.

Fig. 17. The results under Tiscali topology. The lower, the better.

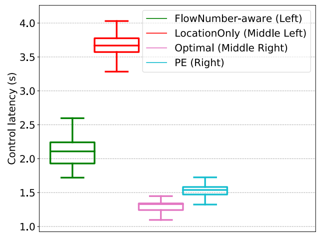


Fig. 15. The results of the control latency under Sprintlink topology. The lower, the better.

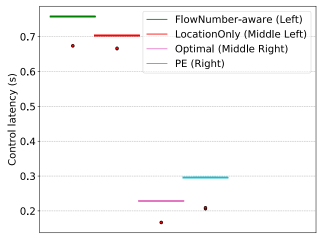


Fig. 18. The results of the control latency under Tiscali topology. The lower, the better.

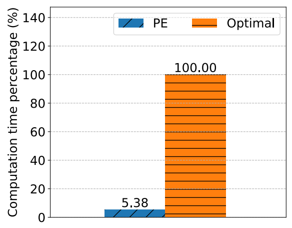


Fig. 16. The results of the computation time under Sprintlink topology. The lower, the better.

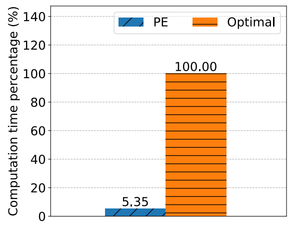


Fig. 19. The results of the computation time under Tiscali topology. The lower, the better.

the impact of critical programmability latency into consideration. On the contrary, FlowNumber-aware and LocationOnly perform the worst since they do not consider the influence of critical programmability latency.

(2) Control latency: Fig. 15 shows the control latency of the four algorithms. Optimal presents the best result. PE's performance is better than FlowNumber-aware and LocationOnly. Specifically, PE's medium is 31.0% and 27.6% lower than LocationOnly and FlowNumber-aware. FlowNumber-aware and LocationOnly perform the worst and their control latency is higher than Optimal and PE.

(3) Computation time: To show the computation efficiency, we evaluate the computation time of PE and Optimal. In Fig. 16, the computation time of PE is only 5.38% of Optimal. Therefore, PE can achieve low control latency with low computation time.

4) Under Tiscali Topology:

(1) Critical programmability latency: Fig. 17(a) shows the critical programmability latency under Tiscali topology. Optimal presents the best performance while LocationOnly performs the worst and it shows that the fine-grained solution which takes the critical programmability at switches into account performs better than the coarse-grained solution which only considers the propagation latency between different locations. PE's critical programmability latency is slightly higher than FlowNumber-aware but substantially lower than LocationOnly.

Fig. 17(b) presents the per-flow critical programmability latency. Optimal presents the best result while the mediums of PE's and FlowNumber-aware's are very close to Optimal's. On the contrary, LocationOnly performs the worst since it's coarse-grained and does not consider the impact of critical programmability latency.

(2) Control latency: Fig. 18 shows the control latency of the four algorithms. Optimal presents the best result while FlowNumber-aware and LocationOnly perform the worst. PE's performance is better than FlowNumber-aware and LocationOnly. Specifically, PE's medium is 57.9% and 61.7% lower than LocationOnly and FlowNumber-aware, respectively. Note that in most cases, the routing policy is the same, thus the control latency is the same except for some isolated points in the figure, which means that the routing policy is different for isolated points.

(3) Computation time: Fig. 19 shows the computation time between PE and Optimal. In the figure, the computation time of PE is only 5.35% of Optimal, which demonstrates PE's time effectiveness while maintaining low control latency.

5) Summary:

Table II summarizes the simulation results of four schemes under four different topologies. In this table, PE greatly reduces critical programmability latency and control latency compared with FlowNumber-aware and LocationOnly in most cases. PE achieves the trade-off between performance and time complexity by realizing near-optimal performance and

TABLE II
SUMMARY OF THE SIMULATION RESULTS

Performance Metrics / Topologies	**Critical programmability latency**	**Control latency**	**Computation time**
GÉANT	PE's performance is similar to Optimal, and can reduce about 50% critical programmability latency on average compared with FlowNumber-aware and LocationOnly	PE's performance is similar to Optimal, and can reduce about 60% control latency on average compared with FlowNumber-aware and LocationOnly	PE $\ll$ Optimal
Abilene	PE's performance is equal to Optimal, and can slightly reduce the critical programmability latency compared with FlowNumber-aware and LocationOnly	PE's performance is equal to Optimal, and can slightly reduce the control latency compared with FlowNumber-aware and LocationOnly	PE $\ll$ Optimal
Sprintlink	PE's performance is similar to Optimal, and can reduce about 58% critical programmability latency compared with LocationOnly and 29% compared with FlowNumber-aware	PE's performance is similar to Optimal, and can reduce about 58% control latency compared with LocationOnly and 31% compared with FlowNumber-aware	PE $\ll$ Optimal
Tiscali	PE's performance is similar to FlowNumber-aware, and is about 18% worse than Optimal. Compared with LocationOnly, PE can reduce about 23% critical programmability latency	PE's performance is about 28% worse than Optimal. Compared with LocationOnly and FlowNumber-aware, PE can reduce about 57% and 61% control latency respectively	PE $\ll$ Optimal

reducing over 95% computation time of Optimal. Note that the performance of the four schemes is quite similar under the Abilene topology since Abilene is a small topology and only consists of 12 switches and 30 links, and the propagation delay between switches is relatively low. Thus, and the controller placement and switch-controller mapping strategies between different schemes will not be obvious under such a small topology.

VII. Related Work

A. Controller Placement

1) Minimizing Control Latency: For minimizing the control latency in SD-WANs, Gao et al. [31] define a global latency controller placement problem with capacitated controllers, taking both the latency between controllers and the capacities of controllers into consideration, and propose a particle swarm optimization algorithm to solve the problem. NCPSO [32] is an algorithm which taking the load of controllers, propagation latency, and load balancing into account. Blenk et al. [33] introduce mixed integer programming formulations which can optimize the placement of multi-controller switches in virtualized OpenFlow-enabled SDN networks. The authors in [34] propose a utility function that jointly considers controller-switch latency, inter-controller latency, and load balancing. They use a modified cluster algorithm to minimize utility functions.

2) Maintaining Load Balancing: To maintain the load balancing performance in SD-WANs, Guo et al. [35] take controller's load diversity factor into consideration, and further propose a greedy-based algorithm, which can solve the controller placement problem in polynomial time. Cai et al. [36] consider the impact of topological potential and minimum-cost flow to minimize the average delay between switches and controllers while maintaining load balancing among controllers. Hou et al. [37] propose a hierarchical multi-controller deployment algorithm which can decrease average request delay between switches and controllers and improve the load balancing performance in each domain with high reliability and scalability. Killi and Rao [38] propose a mathematical model for the capacitated controller placement that plans ahead for the failures to avoid a drastic increase in the worst case latency and disconnections. D4CPP [39] deploys a DRL-based method that integrates the historical network data learning into the controller deployment and real time switch-controller mapping decisions. Also, it aims to achieve dynamic flow fluctuation, control latency, and load balancing simultaneously.

3) Ensuring Fault Tolerance: To ensure fault tolerance in SD-WANs, Alshamrani et al. [40] focus on ensuring resiliency under controller failure scenarios and propose a controller placement model by minimizing the worst case latency between switches and controllers. The solution in [41] decides the location of controllers using a density-based clustering algorithm and ensures network performance when some switches lose connections to the controllers. Ros and Ruiz [42] propose a controller placement problem that decides the placement of controllers and mapping between switches and controllers. Furthermore, the authors propose a heuristic algorithm to efficiently solve the problem and maintain high reliability simultaneously. The authors in [43] focus on minimizing control latency and increasing reliability by using a Varna Based Optimization (VBO). Concretely, the VBO divides switches into classes A and B for exploitation and exploration, where switches in class A move toward the best solution and away from the worst solution while switches in class B interact with other switches, and their movement is decided by their respective peers.

B. Switch-Controller Mapping

1) Minimizing Control Latency: With the growth of network scale, the control plane should minimize control latency to ensure scalability. Wang et al. [44] formulates the dynamic controller placement problem and propose a two stage method based on game theory to substantially reduce the response time and balance the load of controllers. Further, the authors [45] propose an online algorithm to minimize the total control cost induced by communication between switches and controllers and maintain load balancing among controllers. Huang et al. [46] jointly consider the Controller Placement

Problem and Controller Scheduling Problem and propose a clustering-based genetic algorithm to minimize response time and increase resource utilization of controllers. Moreover, the work [47] provides a DRL-based approach to achieve the maximization of long-term system performance by leveraging control latency and load balancing.

2) Maintaining Robust Mapping: For maintaining robust mapping, Yang et al. [48] take the minimum available residual capacity of switches into consideration and formulate a non-linear integer optimization problem. They then decompose it into a two stage problem and propose Indirect-MM to increase minimum available residual capacity and reduce synchronization cost among controllers. Rather than establishing a single switch controller mapping, the solution in [49] intelligently maps a switch to multiple controllers and allocates flow requests among these controllers to guarantee robust mapping against the controller failure. With the help of hybrid SDN/legacy routing mode in commercial SDN switches, Flex-ProgrammabilityMedic [50] proposes to dynamically adapt the routing mode of offline flows to recover more offline flows and further increase the total network programmability under multiple controller failure scenarios. In [51], a quadratic programming problem is constructed to maintain robust mapping and load balancing. Moreover, they propose a heuristic algorithm that smartly migrates fractional request flows between controllers and minimizes the number of new mappings.

VIII. Conclusion

In this paper, we propose PE to minimize control latency by deciding controller placement and switch-controller mapping strategies. PE smartly selects the critical flows and calculates the programmability of critical flows at switches. PE realizes efficient placement and mapping by considering the different priorities of switches with different programmability. Simulation results verify the effectiveness and efficiency of the proposed PE.

References

[1] L. Qi, S. Dou, Z. Guo, C. Li, Y. Li, and T. Zhu, "Low control latency SD-WANs for metaverse," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jul. 2022.

[2] Z. Guo et al., "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Comput. Netw.*, vol. 68, pp. 95–109, Aug. 2014.

[3] S. Dou, G. Miao, Z. Guo, C. Yao, W. Wu, and Y. Xia, "Matchmaker: Maintaining network programmability for software-defined WANs under multiple controller failures," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108045.

[4] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in SDN," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 472–503, 1st Quart., 2020.

[5] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, Sep. 2012.

[6] P. Yi, T. Hu, Y. Hu, J. Lan, Z. Zhang, and Z. Li, "SQHCP: Secure-aware and QoS-guaranteed heterogeneous controller placement for software-defined networking," *Comput. Netw.*, vol. 185, Feb. 2021, Art. no. 107740.

[7] G. Schütz and J. A. Martins, "A comprehensive approach for optimizing controller placement in software-defined networks," *Comput. Commun.*, vol. 159, pp. 198–205, Jun. 2020.

[8] T. Hu et al., "An efficient approach to robust controller placement for link failures in software-defined networks," *Future Gener. Comput. Syst.*, vol. 124, pp. 187–205, Nov. 2021.

[9] P. Yi et al., "A safe and reliable heterogeneous controller deployment approach in SDN," *China Commun.*, vol. 18, no. 8, pp. 47–61, Aug. 2021.

[10] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Gener. Comput. Syst.*, vol. 95, pp. 681–693, Jun. 2019.

[11] *The Internet Topology Zoo*. Accessed: Mar. 1, 2023. [Online]. Available: http://www.topology-zoo.org/

[12] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "CFR-RL: Traffic engineering with reinforcement learning in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2249–2259, Oct. 2020.

[13] J. Zhang, K. Xi, M. Luo, and H. J. Chao, "Dynamic hybrid routing: Achieve load balancing for changing traffic demands," in *Proc. IWQoS*, May 2014, pp. 105–110.

[14] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and Z. Liu, "Minimizing flow statistics collection cost of SDN using wildcard requests," in *Proc. IEEE INFOCOM*, May 2017, pp. 1–9.

[15] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992.

[16] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.

[17] *Abilene Dataset*. Accessed: Mar. 1, 2023. [Online]. Available: http://www.cs.utexas.edu/yzhang/research/AbileneTM

[18] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *Proc. IEEE NOMS*, May 2014, pp. 1–8.

[19] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for OpenFlow networks," in *Proc. PAM*. Berlin, Germany: Springer, 2010, pp. 201–210.

[20] J. Xie, D. Guo, X. Li, Y. Shen, and X. Jiang, "Cutting long-tail latency of routing response in software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 384–396, Mar. 2018.

[21] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in software defined networks," in *Proc. IEEE ICNP*, Oct. 2013, pp. 1–2.

[22] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez, "The p-median problem: A survey of Metaheuristic approaches," *Eur. J. Oper. Res.*, vol. 179, no. 3, pp. 927–939, Jun. 2007.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability*, vol. 174. San Francisco, CA, USA: Freeman, 1979.

[24] P. Kumar et al., "Semi-oblivious traffic engineering: The road not taken," in *Proc. USENIX NSDI*, 2018, pp. 157–170.

[25] Z. Guo et al., "Maintaining control resiliency and flow programmability in software-defined WANs during controller failures," *IEEE/ACM Trans. Netw.*, vol. 30, no. 3, pp. 969–984, Jun. 2022.

[26] C. C. Robusto, "The cosine-haversine formula," *Amer. Math. Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[27] *Speed, Rates, Times, Delays: Data Link Parameters for CSE 461*. Accessed: Mar. 1, 2023. [Online]. Available: https://courses.cs.washington.edu/courses/cse461/99wi/issues/definitions.html

[28] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 133–145, Oct. 2002.

[29] *TMgen: Traffic Matrix Generation Tool*. Accessed: Mar. 1, 2023. [Online]. Available: https://tmgen.readthedocs.io/en/latest/

[30] *Gurobi Optimizer*. Accessed: Mar. 1, 2023. [Online]. Available: http://www.gurobi.com

[31] C. Gao, H. Wang, F. Zhu, L. Zhai, and S. Yi, "A particle swarm optimization algorithm for controller placement problem in software defined network," in *Proc. ICA3PP*. Cham, Switzerland: Springer, 2015, pp. 44–54.

[32] S. Liu, H. Wang, S. Yi, and F. Zhu, "NCPSO: A solution of the controller placement problem in software defined networks," in *Proc. ICA3PP*. Cham, Switzerland: Springer, 2015, pp. 213–225.

[33] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with SDN network hypervisors: The cost of virtualization," *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 3, pp. 366–380, Sep. 2016.

[34] C. N. Sminesh, E. G. M. Kanaga, and A. Roy, "Optimal multi-controller placement strategy in SD-WAN using modified density peak clustering," *IET Commun.*, vol. 13, no. 20, pp. 3509–3518, Dec. 2019.

[35] S. Guo, S. Yang, Q. Li, and Y. Jiang, "Towards controller placement for robust software-defined networks," in *Proc. IEEE IPCCC*, Dec. 2015, pp. 1–8.

[36] N. Cai, Y. Han, Y. Ben, W. An, and Z. Xu, "An effective load balanced controller placement approach in software-defined WANs," in *Proc. IEEE MILCOM*, Nov. 2019, pp. 361–366.

[37] X. Hou, W. Muqing, L. Bo, and L. Yifeng, "Multi-controller deployment algorithm in hierarchical architecture for SDWAN," *IEEE Access*, vol. 7, pp. 65839–65851, 2019.
[38] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1108–1111, Jun. 2016.
[39] Y. Wu, S. Zhou, Y. Wei, and S. Leng, "Deep reinforcement learning for controller placement in software defined network," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 1254–1259.
[40] A. Alshamrani, S. Guha, S. Pisharody, A. Chowdhary, and D. Huang, "Fault tolerant controller placement in distributed SDN environments," in *Proc. IEEE ICC*, May 2018, pp. 1–7.
[41] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Comput. Netw.*, vol. 112, pp. 24–35, Jan. 2017.
[42] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. ACM HotSDN*, Aug. 2014, pp. 31–36.
[43] A. K. Singh, S. Maurya, N. Kumar, and S. Srivastava, "Heuristic approaches for the reliable SDN controller placement problem," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 2, Feb. 2020, Art. no. e3761.
[44] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.
[45] T. Wang, F. Liu, and H. Xu, "An Efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.
[46] V. Huang et al., "A scalable approach to SDN control plane management: High utilization comes with low latency," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 682–695, Jun. 2020.
[47] J. Chen, S. Chen, X. Cheng, and J. Chen, "A deep reinforcement learning based switch controller mapping strategy in software defined network," *IEEE Access*, vol. 8, pp. 221553–221567, 2020.
[48] X. Yang et al., "Indirect multi-mapping for burstiness management in software defined networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2059–2072, May 2021.
[49] V. Sridharan, M. Gurusamy, and T. Truong-Huu, "On multiple controller mapping in software defined networks with resilience constraints," *IEEE Commun. Lett.*, vol. 21, no. 8, pp. 1763–1766, Aug. 2017.
[50] Z. Guo, S. Dou, W. Wu, and Y. Xia, "Toward flexible and predictable path programmability recovery under multiple controller failures in software-defined WANs," *IEEE/ACM Trans. Netw.*, early access, Jan. 26, 2023, doi: 10.1109/TNET.2022.3227423.
[51] F. Al-Tam and N. Correia, "Fractional switch migration in multi-controller software-defined networking," *Comput. Netw.*, vol. 157, pp. 1–10, Jul. 2019.

Songshi Dou (Graduate Student Member, IEEE) received the B.S. degree from North China Electric Power University, Beijing, China, in 2019, and the M.S. degree from the Beijing Institute of Technology, Beijing, in 2022. His research interests include software-defined networking, network function virtualization, and data center networks. He is a Graduate Student Member of ACM.

Li Qi received the B.S. degree from the School of Automation, Beijing Institute of Technology, Beijing, China, in 2021, where he is currently pursuing the M.S. degree. His research interests include software defined networking and traffic engineering.

Chao Yao received the B.Sc. degree in telecommunications engineering and the Ph.D. degree in communication and information systems from Xidian University, Xi'an, China, in 2007 and 2014, respectively. He was a Visiting Student with the Center for Pattern Recognition and Machine Intelligence (CENPARMI), Montreal, Canada, from 2010 to 2011. His research interests include feature extraction, handwritten character recognition, machine learning, and pattern recognition.

Zehua Guo (Senior Member, IEEE) received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, the M.S. degree from Xidian University, Xi'an, and the Ph.D. degree from Northwestern Polytechnical University. He was a Research Fellow with the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York, NY, USA, and a Research Associate with the Department of Computer Science and Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. His research interests include programmable networks, such as software-defined networking, network function virtualization, machine learning, and network security. He is a Member of ACM. He is a Senior Member of the China Computer Federation, the China Institute of Communications, and the Chinese Institute of Electronics. He is serving as the TPC Member for several journals and conferences, such as *Computer Communications* (Elsevier), AAAI, IWQoS, ICC, ICCCN, and ICA3PP. He is an Associate Editor of IEEE SYSTEMS JOURNAL and the *EURASIP Journal on Wireless Communications and Networking* (Springer) and an Editor of the *KSII Transactions on Internet and Information Systems*.