



MOSM: Multi-objective optimized switch migration for controller load balancing in software-defined internet of vehicles[☆]

Yong Liu^a, Bin Xu^a, Quanze Liu^a, Qian Meng^b

^a School of Information Science and Technology, Hangzhou Normal University, Hangzhou, 311121, China

^b School of Mathematics, Hangzhou Normal University, Hangzhou, 311121, China

ARTICLE INFO

Keywords:

Software-defined networking
Load balancing
Switch migration
Intelligent transportation systems
Internet of vehicles

ABSTRACT

The Software Defined Internet of Vehicles (SD-IoV) offers effective enhancements in resource utilization, service quality, and management simplification. However, the randomness and burstiness of traffic in IoV lead to load imbalances among multiple controllers. Existing work focuses on the switch migration strategy, which effectively achieves load balance by migrating switches from overloaded to underloaded controllers. However, most schemes only use the flow request rate to measure controller load, decreasing measurement accuracy, and the switch with the highest flow request rate is selected for migration, which degrades the load balancing performance. Moreover, they mainly select the controller with a lower load as the target controller, making it easy to overload the target controller and incur high migration cost. To address the problems above, we propose the Multi-Objective Optimized Switch Migration (MOSM) scheme. MOSM improves load measurement and controller partition accuracy using multiple metrics and introduces the Coefficient of Variation (CV) to determine the load weight for each metric. Meanwhile, when selecting the switches to be migrated, MOSM improves the load balancing performance by evaluating the controller overhead from processing flow request messages and the updating of flow entries. Furthermore, MOSM selects the optimal target controller by minimizing migration cost and maximizing load balancing rate after migration, which reduces the migration cost and prevents overloading the target controller. Finally, we verified MOSM's efficacy in performance evaluation. Compared to existing methods, MOSM improves the load balancing rate by 31% and reduces the migration cost by 33% on average.

1. Introduction

As the cornerstone of Intelligent Transportation Systems (ITS), the Internet of Vehicles (IoV) aims to provide a wide range of services, ranging from mobile Internet access to road safety [1–3]. In typical IoV scenarios, vehicles equipped with dedicated On-Board Units (OBUs) can establish Vehicle-to-Vehicle (V2V) connections with adjacent peer vehicles [4]. Additionally, vehicles can also receive data services from infrastructures, such as Road Side Units (RSUs), Wi-Fi Access Points (APs), and cellular Base Stations (BS), enabling Vehicle-to-Infrastructure (V2I) communication [5]. Through these communication processes, both vehicle drivers and passengers can enjoy various IoV-based applications and services via the Internet [6]. Despite the immense potential of IoV in future intelligent transportation systems,

several challenges need to be addressed, such as the heterogeneity of the vehicular network, security, and efficiency in resource utilization. Furthermore, the coupling of the data plane and control plane in traditional architectures also adds complexity to addressing the aforementioned challenges.

Software Defined Networking (SDN) differs from traditional networks by separating the control and data planes, enabling centralized network control, programmability, and flexibility [7]. SDN has been successfully applied in data centers [8], wireless mesh networks [9], and cellular networks [10].

The integration of SDN and the Internet of Vehicles (IoV) forms the Software Defined Internet of Vehicles (SD-IoV), enhancing network controllability, flexibility, resource utilization, service quality, and network optimization [11]. Fig. 1 presents a generalized SD-IoV architecture,

[☆] This work was supported by the National Natural Science Foundation of China under Grant 62402154 and Grant 62302136, the Zhejiang Provincial Natural Science Foundation of China under Grant No. LMS26F020039 and Grant No. LQ22F010001, the Hangzhou City Natural Science Foundation of China under Grant 2025SZRJ0187, and the Starting Research Funds from the Hangzhou Normal University under Grant 4115C50221204137 and Grant 4085C50220204093.

* Corresponding author.

E-mail addresses: yongliu@hznu.edu.cn (Y. Liu), 2024111011023@stu.hznu.edu.cn (B. Xu), 2022112011053@stu.hznu.edu.cn (Q. Liu), mq@hznu.edu.cn (Q. Meng).

<https://doi.org/10.1016/j.future.2026.108614>

Received 13 November 2025; Received in revised form 11 May 2026; Accepted 18 May 2026

Available online 22 May 2026

0167-739X/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

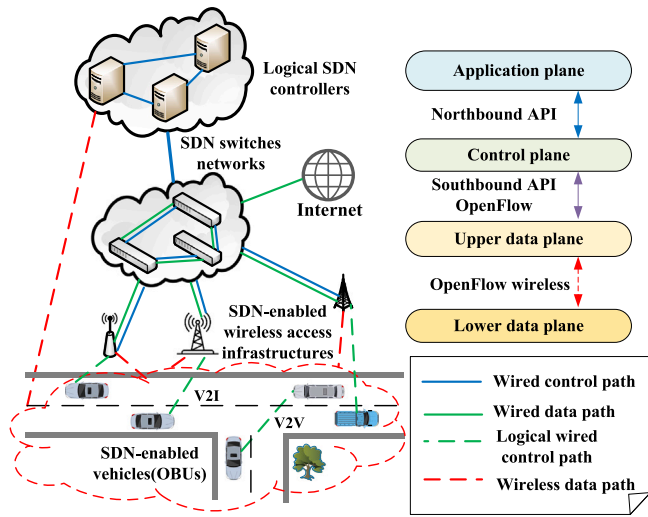


Fig. 1. Generalized SD-IoV architecture.

where the SDN controller manages the network and communicates with the SDN switch network. Data packets from Internet servers are processed by SDN switches, with unmatched packets forwarded to the controller for processing and routing. SDN-enabled wireless access facilities, like Wi-Fi APs, cellular BSs, and RSUs, provide wireless interfaces to vehicles, which also serve as relays in Vehicle-to-Vehicle scenarios. The architecture has two types of transmission paths: the data path (wired and wireless links for data transmission) and the control path (for network management and feedback).

The SD-IoV architecture consists of four planes: the application plane, control plane, upper data plane (wired network), and lower data plane (SDN-enabled vehicles or wireless data plane). The control plane, with multiple distributed and centralized controllers, improves network reliability and scalability. However, due to the mobility of vehicles in the road network and variations in connectivity patterns, different controllers may have varying numbers of vehicle connections and data processing tasks. In scenarios where there are dense areas of vehicles or sudden increases in vehicle traffic, certain controllers may become overloaded while others have lighter loads. This leads to the issue of load imbalance among multiple controllers.

Therefore, researchers have proposed the switch migration strategy, which can accomplish load balancing based on switch granularity by migrating switches from the overloaded controller to the underloaded controller. For example, the work in [12,13] can significantly reduce the load on an overloaded controller by migrating the switch to the controller with the lightest load. However, most existing work simply measures controller load by flow request rate, resulting in insufficient accuracy. Also, the switch with the highest flow request rate is often selected as the switch to be migrated, which leads to inefficient load balancing performance. Furthermore, the controller with the lower load is always selected as the target controller, ignoring the migration cost and making it easy to overload the target controller.

To address the aforementioned shortcomings, we designed the MOSM scheme. MOSM measures the controller load using multiple metrics to improve the accuracy of the load measurement and the controller partition. In the selection of the candidate switches, MOSM further evaluates the controller overhead caused by the update of flow entries, thereby enhancing load balancing performance. Finally, when selecting the target controller, MOSM considers the migration cost and the controller load distribution after migration to optimize the switch migration process.

In summary, the main contributions of this paper are as follows:

- Different from the existing schemes that only use the flow request rate to measure the controller load, we select CPU, bandwidth and memory utilization for load measurement. In addition, we adopt the coefficient of variation method to further determine the load weights of various metrics and improve the accuracy of controller partition and load measurements.
- Unlike present work, which chooses the switch with the highest flow request rate as a candidate switch, we take into account not only the controller overhead from processing flow request messages but also the overhead from updating flow entries to enhance the load balancing performance.
- Different from the existing work, which directly selects the controller with the lower load as the target controller, we select the target controller by minimizing the migration cost and further consider the load balancing rate of the control plane after migration to avoid overloading the target controller.
- To verify the feasibility of our scheme, we implement and evaluate MOSM on the Mininet platform. The experimental results demonstrate the effectiveness of MOSM. Compared with existing schemes, the load balancing rate of MOSM is increased by an average of 31%, and the migration cost is reduced by an average of 33%.

The rest of this paper is organized as follows: Section 2 gives the current related work and the motivation of this paper. Section 3 describes the MOSM scheme in detail. The experimental evaluation is carried out in Section 4. Finally, we summarize the main results in Section 5.

2. Motivation and related work

2.1. Motivation

In real network environments, the random and bursty nature of network traffic can lead to load imbalance issues among SD-IoV controllers. To address this problem, switch migration has been proposed as an effective method. This method allows the continuous monitoring of controller loads in real time. When an imbalance is identified, switches are dynamically migrated from heavily loaded controllers to those with lighter loads, thereby balancing the load at the switch granularity level. In Fig. 2(a), the control plane is composed of three controllers, each with its own load capacity. The network is divided into three subdomains, and each controller manages a specific portion of the SDN switch network. The data plane includes seven switch nodes, with each node connected to its corresponding master controller. Let us assume that initially, the loads on controllers C_1 , C_2 , and C_3 reach 40%, 70%, and 30% of their respective capacities. Notably, controller C_2 exhibits a considerably higher load than the other two controllers, leading to a load imbalance among the multiple controllers and consequent network performance degradation. To achieve dynamic load balancing among multiple controllers, we continuously monitor the load distribution in the control plane and accurately identify the overloaded controller C_2 and the underloaded controllers C_1 and C_3 . We dynamically migrate the switch S_4 , which is managed by the overloaded controller C_2 , to the subdomain network managed by controller C_3 . After the migration, the mapping between controllers and switches is depicted in Fig. 2(b). At this point, the loads on controllers C_1 , C_2 , and C_3 are 40%, 50%, and 50%, respectively, resulting in a more balanced load distribution among the controllers. Therefore, we propose the MOSM scheme to deal with the problem of uneven loads in the distributed control plane. This scheme selects CPU utilization, bandwidth utilization, and memory utilization to evaluate the load of the current controller jointly, which can accurately measure the controller load by introducing the coefficient of variation method for determining the load weight of each metric. To improve load balancing performance, the additional processing overhead of the controller from

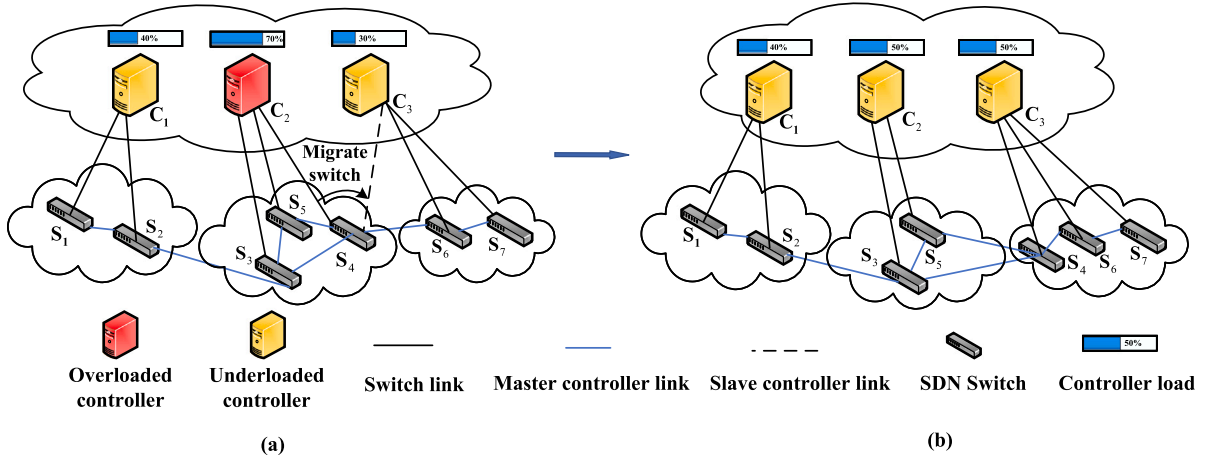


Fig. 2. (a) The load is imbalanced; (b) After switch migration.

updating flow entries is precisely considered when selecting candidate switches. Moreover, to avoid overloading the target controller and reduce migration cost, we determine the optimal target controller by minimizing migration cost while maximizing the load balancing rate after migration.

2.2. Related work

The control plane plays a central role in the SDN as a bridge between the applications in the upper layer and the networks in the lower layer. But the inherent single point of failure and insufficient processing resources of a single centralized controller significantly reduce the reliability and availability of SDN. To address the shortcomings of the single centralized controller, the Distributed SDN Control (DSC) has been proposed in [14]. Its implementations include Hyper-Flow, Kandoo, Orion [15–17], and so on, which deploy multiple controllers in the network. To address the load imbalance among multiple controllers, existing solutions are mainly categorized into two categories. The first type of scheme achieves load balancing by optimizing the number and location of controllers deployed in the network. The authors of [18] defined the dynamic controller placement problem, and proposed a dynamic controller placement strategy by combining two heuristic algorithms to optimize the number and location of multi-controller deployments. Through the proposed deployment algorithm, the flow rule formulation time can be minimized, and the required communication overhead among controllers reduced, but the algorithm takes a long time to converge. Lange [19] proposed a framework for Pareto-based Optimal Controller placement, POCO. To adapt to large-scale networks, POCO is extended with heuristics that improve the computational speed of large-scale networks. However, this comes at the cost of decreased algorithmic accuracy. Zhang [20] introduced an adaptive bacterial foraging optimization algorithm for the multi-objective optimization controller placement problem, which effectively attains low latency between the controllers and the switches, enhancing the overall high availability of the network. The work in [21] proposed a reliable controller placement strategy, which not only improves the reliability of the network but also reduces the packet loss rate and improves the overall transmission efficiency of the network.

While the previous works can help to improve the performance of distributed control planes, the majority of them rely on a static mapping between switches and controllers. To adapt to dynamic changes in network traffic, researchers have proposed a switch migration method that involves dynamically changing the mapping between switches and controllers based on changes in controller load. This allows for controller load segmentation and migration based on switch granularity. Dixit [22] proposed the *ElastiCon* architecture and designed

an elastic and scalable controller pool based on dual thresholds. The size of the controller pool can be contracted and expanded according to the current network load. The work presented in [23] designed a controller load balancing algorithm based on dynamic threshold, which dynamically adjusts controller load thresholds and migrates switches between heavily loaded and lightly loaded controllers, thereby ensuring load balance in the control plane. The authors of [13] proposed a switch migration strategy based on response time and used the response time to measure the controller load. This strategy is able to identify load imbalances more quickly and migrate switches promptly. The authors of [24] proposed an algorithm for randomly selecting switches for migration, but this may quickly lead to load imbalance among the controllers. The work in [25] proposed the concept of migration efficiency. They used a simulated annealing algorithm to select optimal target controllers to achieve controller load balancing and lower migration cost. Mokhtar [26] proposed a switch migration algorithm based on a multi-level threshold that divided the controller load into several progressive levels according to the threshold. When the load of the controller is significantly different from that of the other controllers, the corresponding switch migration operation is executed. Guo et al. [27] proposed *RetroFlow*, a failure recovery scheme for SD-WANs under controller failures. By dynamically switching switches to traditional routing mode during controller outages, *RetroFlow* reduces the load on active controllers and restores flow programmability. The approach emphasizes switch-level mapping and topology adaptation but does not provide fine-grained optimization at the flow level. In *RetroFlow++* [28], Guo et al. introduced a resilient control recovery mechanism for SD-WANs experiencing one or more controller failures. The method leverages hybrid SDN switches that support both SDN and legacy routing modes. Upon controller failures, *RetroFlow++* selectively configures a subset of offline switches to operate in legacy mode to alleviate control load, while maintaining SDN mode for other switches to preserve flow programmability. To further accommodate offline switches, active controllers can release control resources by transitioning some of their managed switches to legacy mode. The scheme jointly optimizes switch-controller mapping to minimize communication overhead, considering propagation delay and flow count, while ensuring that each offline flow retains at least one programmable switch along its path. This multi-objective optimization is formulated as the NP-hard OCMSC problem and solved using a heuristic algorithm.

Existing switch migration schemes for controller load measurement rely solely on the flow request rate, resulting in lower accuracy. Additionally, these schemes overlook the processing overhead caused by flow entry updates, which degrades load balancing performance. Furthermore, selecting the controller with the lowest load as the target can lead to higher migration costs and potential overload. Although

Table 1
Comparison of MOSM with representative switch migration schemes.

Scheme	Load metrics	Target selection criteria	Optimization objective
ElastiCon [22]	Flow request rate	Closest underloaded controller	Load balancing only
Scheme [13]	Response time	Lowest-load controller	Load balancing only
Scheme [23]	Flow request rate	Threshold-based selection	Load balancing via dynamic threshold
Scheme [24]	Flow request rate	Zero-sum game competition	Master controller re-election
EASM [25]	Flow request rate	Simulated annealing optimization	Migration efficiency
Scheme [26]	Multi-level load classification	Progressive level adjustment	Continuous load balancing
RetroFlow [27]	Flow count & controller residual capacity	Delay-aware selection	Minimize overhead while ensuring flow programmability
RetroFlow++ [28]	Flow count & controller residual capacity	Minimize communication overhead	Minimize overhead while ensuring flow programmability
MOSM (proposed)	CPU, bandwidth, memory (CV-weighted)	Minimize CER (cost/benefit ratio)	Multi-objective: load balance + migration cost + flow entry update overhead

the aforementioned works represent significant progress in distributed SDN control planes, they primarily rely on a static mapping between switches and controllers and do not fully address the complexities of large-scale networks, such as varying resource utilization across controllers and the impact of flow entry updates. Moreover, these methods tend to focus on optimizing either placement or migration of switches, without integrating multiple performance metrics for comprehensive load balancing.

Our work builds on these frameworks and introduces several key improvements. We extend existing methods by incorporating three performance metrics: CPU, bandwidth, and memory utilization, to more accurately capture the true load of each controller, addressing the limitations of using only flow request rate. We also introduce the coefficient of variation method, which helps assign appropriate weights to each metric based on its variance and contribution to the overall load, enabling more precise load balancing. Additionally, existing approaches largely ignore the processing overhead of flow entry updates during switch migrations. By incorporating this overhead into our decision-making process, we enhance load balancing performance, considering not only immediate migration costs but also the long-term impact on controller performance. Unlike existing methods, which often select the controller with the lowest load as the target, our approach optimizes this selection by evaluating both migration cost and controller capacity, minimizing migration costs and ensuring a more balanced load distribution post-migration. These improvements significantly enhance the performance, scalability, and adaptability of distributed SDN control planes, especially in large-scale and dynamic network environments. Table 1 summarizes the key differences between MOSM and representative switch migration schemes in the literature, highlighting the unique aspects of our approach.

3. The design of MOSM

In this section, we provide a detailed description of the MOSM, and the notations are summarized in Table 2. Firstly, we represent the SDN network topology using an undirected graph G , which consists of a set of controllers $\hat{C} = \{C_1, C_2, C_3, \dots, C_{N_C}\}$ and a set of switches $\hat{S} = \{S_1, S_2, S_3, \dots, S_{N_S}\}$, where N_C and N_S denote the number of controllers and switches, respectively. Each controller $C_i \in \hat{C}$ manages a corresponding domain network, and the set of switches in the domain network is denoted by $\hat{S}_{C_i}$, which satisfies $\bigcup_{C_i \in \hat{C}} \hat{S}_{C_i} = \hat{S}$. We use the matrix R to represent the connection relationship between the switch $S_k \in \hat{S}$ and the controller $C_i \in \hat{C}$. The binary variables in the matrix are described as follows:

$$r_{ki} = \begin{cases} 1, & \text{if } S_k \in \hat{S}_{C_i}, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The OpenFlow protocol allows switches to establish connections with multiple controllers, and each controller can define its role with

Table 2
Summary of notations.

Notion	Description
$O_{CPU}(C_i)$	The CPU utilization of controller C_i .
V_j	The coefficient of variation of the metric j .
μ_j	The weight of metric j .
$Load_{C_i}$	The load of controller C_i .
$Load_{mean}$	The average load of all controllers in the network.
τ_i	The partition factor of controller C_i .
$f_{S_k}^1$	The controller's overhead for processing the Packet-In messages from the flows that first reach the switch S_k .
FR_{S_k}	The set of flow entries removed by S_k .
$f_{S_k}^2$	Additional processing overhead caused by the update of flow entries.
LBR_{S_k, C_j}	The load balancing rate after migration.
MC_{S_k, C_j}^1	Controller load change cost during migration.
MC_{S_k, C_j}^2	Message exchange cost during migration.
MC_{S_k, C_j}	The total migration cost during migration.
CER_{S_k, C_j}	The cost-effectiveness ratio during migration.

different modes of operation. The OpenFlow protocol provides three controller roles: master, slave, and equal [29].

- **OFPCR_ROLE_MASTER**: this role grants the controller full access to the switch, allowing the controller to obtain switch state information and configure the switch. However, it is important to note that each switch can only have a maximum of one master controller.
- **OFPCR_ROLE_SLAVE**: this role grants the controller read-only access to the switch. By default, the controller does not receive asynchronous messages from the switch, except for port state messages.
- **OFPCR_ROLE_EQUAL**: this default role grants the controller complete access to the switch and ensures equality with other controllers in the same role. Specifically, the controller can issue commands to the switch and retrieve its status report.

Since the master controller is exclusive to a single switch, we stipulate that the switch can establish connections to multiple controllers in the network, but can only be managed by one master controller, as described below:

$$\sum_{i=1}^{N_C} r_{ki} = 1. \quad (2)$$

The overview architecture of the MOSM is shown in Fig. 3 and implemented using a hierarchical design. The lower layer includes

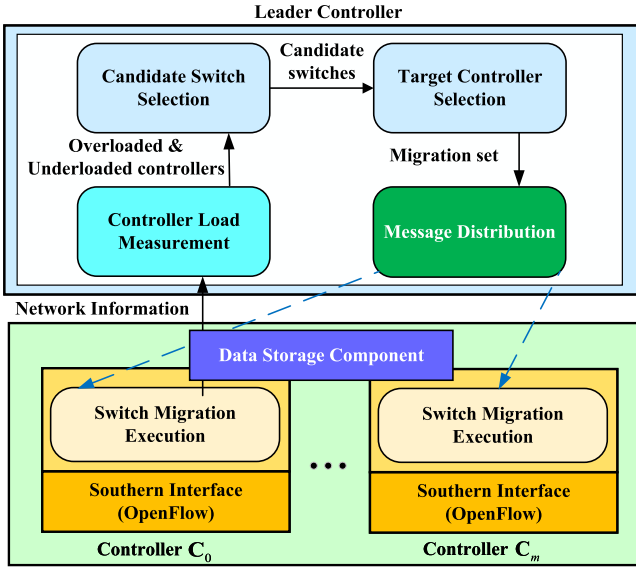


Fig. 3. The overview architecture of MOSM.

common controllers with three main functions. The first function is to manage the corresponding subdomain network. The second function is to periodically send the network information to the data storage component of the leader controller, and the third function is to receive load balancing decisions from the leader controller and perform specific switch migration operations. Each common controller runs an identical copy of the code. The Leader controller consists of four main parts: The controller load measurement module measures load and partitions the overloaded and underloaded controller sets, which are then passed as input parameters to the candidate switch selection module. After the candidate switch selection module processes the data, it outputs the set of candidate switches, which is then fed into the target controller selection module. Finally, the migration set is obtained and input to the migration distribution module, which distributes migration commands to the lower-level controllers to execute the actual switch migration operations.

MOSM is designed to reduce its reliance on specific wireless topology configurations in SD-IoV environments. Its migration decisions are primarily driven by controller-side aggregate load metrics, including CPU, bandwidth, and memory utilization, as well as switch-level processing overhead. These variables characterize the control-plane load induced by aggregated vehicular traffic, rather than directly capturing individual vehicle mobility events or instantaneous wireless-link fluctuations. Consequently, provided that the wired infrastructure connecting controllers and switches remains stable, the proposed scheme is expected to be less sensitive to short-term wireless topology variations. This design property motivates the evaluation of MOSM using fixed backbone topologies, while dynamic and bursty traffic patterns are used to reflect essential load variations in the control plane. Nevertheless, the current evaluation does not explicitly validate MOSM under continuously changing vehicular wireless topologies, which will be considered in future work.

3.1. Load measurement

In SDN networks, the controller's resource utilization directly reflects its load. As the number of requests it processes increases, the amount of resources used by the controller grows proportionally. Therefore, we use three resource utilization metrics to measure controller load in this scheme, namely CPU, bandwidth and memory utilization.

These metrics are denoted by $O_{CPU}(C_i)$, $O_{BW}(C_i)$, and $O_{MEM}(C_i)$, respectively, and defined as follows:

$$O_{CPU}(C_i) = \frac{t_{work}}{t_{work} + t_{idle}}. \quad (3)$$

$$O_{BW}(C_i) = \frac{n_{C_i} \times size_{C_i}}{bandwidth_{C_i}}. \quad (4)$$

$$O_{MEM}(C_i) = \frac{m_{used}}{m_{total}}. \quad (5)$$

Where t_{work} and t_{idle} denote the working and idle time of the CPU, respectively. n_{C_i} , $size_{C_i}$, and $bandwidth_{C_i}$ represent the number of packets sent and received by the controller C_i , the average size of the packets, and the bandwidth of the controller, respectively. Here, m_{used} and m_{total} denote the used memory space and the total memory capacity, respectively.

As the number of processed requests rises, the controller uses more resources; however, not all resources are equally impacted. Therefore, to accurately measure the controller load, the Coefficient of Variation (CV) [30] method is used to determine the weight of each metric in the controller load measurement. The coefficient of variation method, which is also referred to as the information weight method, can directly use the information in each metric to determine the corresponding weight, which is an objective weighting method. The basic idea of the method is that the volatility of the data can be used to measure the weights [30]. Metrics with a higher coefficient of variation carry more information and are assigned a higher weight. The coefficient of variation method involves the following steps for determining the weights:

First, we assume that a set of data contains m metrics and n samples to be evaluated, and obtain an $n \times m$ evaluation matrix $X = [x_{ij}]_{n \times m}$, where x_{ij} denotes the value corresponding to the metric j of the evaluation sample i .

Next, the average value A_j and standard deviation S_j of each metric can be calculated by the following two equations:

$$A_j = \frac{1}{n} \sum_{i=1}^n x_{ij}. \quad (6)$$

$$S_j = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - A_j)^2}. \quad (7)$$

Then the coefficient of variation V_j is computed as:

$$V_j = \frac{S_j}{A_j}. \quad (8)$$

The weight of each evaluated metric μ_j is then determined using:

$$\mu_j = \frac{V_j}{\sum_{j=1}^m V_j}. \quad (9)$$

After obtaining the resource utilization and its corresponding weights, the controller load $Load_{C_i}$ can be obtained by the following equation:

$$Load_{C_i} = \mu_1 \times O_{CPU}(C_i) + \mu_2 \times O_{BW}(C_i) + \mu_3 \times O_{MEM}(C_i). \quad (10)$$

Where μ_1 , μ_2 and μ_3 indicate the load weights corresponding to the above three metrics, which are all within $[0, 1]$ and satisfy $\mu_1 + \mu_2 + \mu_3 = 1$. Besides, to complete the controller partitioning, we compute the average load of all controllers in the network denoted by $Load_{mean}$, as follows:

$$Load_{mean} = \frac{\sum_{i=1}^{N_C} Load_{C_i}}{N_C}. \quad (11)$$

Using $Load_{mean}$, the partition factor τ_i for each controller can be obtained as follows:

$$\tau_i = \frac{Load_{C_i}}{Load_{mean}}. \quad (12)$$

Algorithm 1 Load Measurement

Input: Controller set $\hat{C}$;
Output: Overloaded controller set $\hat{C}_O$, Underloaded controller set $\hat{C}_U$;

```

1: Initialization:  $\hat{C}_O \leftarrow \emptyset, \hat{C}_U \leftarrow \emptyset, Sum=0$ ;
2: for  $C_i \in \hat{C}$  do
3:   Obtain  $Load_{C_i}$  according to Eq. (10);
4:    $Sum = Sum + Load_{C_i}$ ;
5: end for
6: Calculate  $Load_{mean}$  according to Eq. (11);
7: for  $C_i \in \hat{C}$  do
8:   Get  $\tau_i$  using Eq. (12);
9:   if  $\tau_i > 1$  then
10:     $\hat{C}_O \leftarrow C_i$ ;
11:   else if  $\tau_i < 1$  then
12:     $\hat{C}_U \leftarrow C_i$ ;
13:   end if
14: end for
15: return  $\hat{C}_O, \hat{C}_U$ ;

```

Depending on the partition factor τ_i , controllers can be classified into three categories:

- $\tau_i < 1$: an indication that the current controller is underloaded and classified as the underloaded controller.
- $\tau_i = 1$: an indication that the current controller is in a load balancing state and does not need to participate in load balancing decisions.
- $\tau_i > 1$: an indication that the current controller is overloaded and needs to be classified as an overloaded controller.

By following the steps outlined above, we can separate the sets of overloaded and underloaded controllers from the network and use them as inputs for the subsequent load balancing decisions. The pseudocode of the load measurement algorithm is given in Algorithm 1. First, we traverse all the controllers, and then we calculate the load of each controller according to Eq. (10). Once the traversal is complete, we can determine the mean load $Load_{mean}$. We then assign the controller to the corresponding controller set based on the partition factor τ_i , which is obtained using Eq. (12). Once the partitioning is completed, the algorithm returns the overloaded controller set $\hat{C}_O$ and the underloaded controller set $\hat{C}_U$, which will be used as inputs for Algorithms 2 and 3.

3.2. Candidate switch selection

After the controller partition, the appropriate switches need to be selected from the subdomain network managed by the overloaded controller and migrated to the underloaded controller. However, most existing strategies for switch selection rely solely on the flow request rate of switches, disregarding the additional processing overhead due to the updating of flow entries. As a result, it is difficult to accurately evaluate the impact of each switch on the controller processing overhead during candidate switch selection. In SDN networks, switches complete packet forwarding by matching flow entries. However, in real networks, the use of static routing methods can lead to path congestion and even network collapse due to dynamic changes in network traffic. Hence, timely updating of flow entries in the switches is necessary for routing optimization and improving the overall network performance. When a packet arrives at the switch, the switch first matches the packet to the matching fields in the flow entry. If the match is successful, the switch will process the packet based on the action field. Otherwise, the switch encapsulates the Packet-In message according to the packet and sends it to the controller. The controller then formulates appropriate forwarding rules for the packets, consuming certain controller processing resources.

Algorithm 2 Candidate Switch Selection

Input: Overloaded controller C_i ;
Output: Set of candidate switches F ;

```

1: Initialization:  $F = \emptyset, FR_{S_k} = \emptyset, f_{S_k}^1 = 0, f_{S_k}^2 = 0$ ;
2: Fetch  $\hat{S}_{C_i}$  based on  $C_i$ ;
3: for  $S_k \in \hat{S}_{C_i}$  do
4:   Obtain  $FR_{S_k}$  based on  $S_k$ ;
5:   Get  $f_{S_k}^1$  using Eq. (13);
6:   for  $s \in FR_{S_k}$  do
7:     Get  $flow_s$  according to Eq. (14);
8:      $f_{S_k}^2 = f_{S_k}^2 + flow_s$ ;
9:   end for
10:  Get  $F_{S_k}$  using Eq. (17);
11:  Add  $S_k$  to  $F$  along with  $F_{S_k}$ ;
12: end for
13: return  $F$ ;

```

Therefore, we consider the impact of flow entry updates on the controller processing overhead when selecting the candidate switches. In the OpenFlow protocol, the type field indicates the type of packets transmitted by the OpenFlow protocol. OpenFlow v1.3 defines a total of 29 packet types, mainly divided into three types: Controller-to-Switch Messages, Asynchronous Messages, and Symmetric Messages. When a flow entry is removed, the switch sends a FLOW-REMOVED message to the controller, which is an asynchronous message containing statistics about the currently removed flow entry. The candidate switch selection algorithm aims to construct the candidate switch set F , which contains the switch S_k and the controller processing overhead F_{S_k} . The processing overhead F_{S_k} consists of two parts: (1) The controller's overhead for processing the Packet-In messages from the flows that first reach the switch S_k , denoted by $f_{S_k}^1$; (2) The additional overhead $f_{S_k}^2$ caused by the update of flow entries. The overhead $f_{S_k}^1$ can be obtained by the following equation:

$$f_{S_k}^1 = m_{packet-in} \times avg_{packet-in}. \quad (13)$$

Where $m_{packet-in}$ denotes the number of Packet-In messages processed by the switch S_k in period t , and $avg_{packet-in}$ denotes the average size of Packet-In messages in bytes. The additional processing overhead caused by the update of the flow entry is denoted by $flow_s$, which can be obtained by the following equation:

$$flow_s = R_s \times avg_{packet-in} \times avg_{update}. \quad (14)$$

Where avg_{update} denotes the average time required to update a flow entry and R_s denotes the average packet matching rate of the flow entry s , which can be obtained by the following equation:

$$R_s = \frac{Count_{packet}}{duration_s}. \quad (15)$$

Where $duration_s$ denotes the survival time of the flow entry s , and $Count_{packet}$ denotes the number of packets matched by s during its survival time. Therefore, the additional processing overhead $f_{S_k}^2$ can be obtained by the following equation:

$$f_{S_k}^2 = \sum_{s=1}^{|FR_{S_k}|} flow_s. \quad (16)$$

Where $|FR_{S_k}|$ represents the total number of flow entries removed by S_k in period t . The total processing overhead F_{S_k} can be obtained by the following equation:

$$F_{S_k} = f_{S_k}^1 + f_{S_k}^2. \quad (17)$$

The detailed procedure of the candidate switch selection algorithm is shown in Algorithm 2.

Algorithm 3 Target Controller Selection**Input:** Underloaded controller set $\hat{C}_U$, Set of candidate switches F ;**Output:** Set of target controller and switch TS ;1: Initialization: $TS = \emptyset, CERSet = \emptyset, minCER = +\infty$;2: **for** $C_j \in \hat{C}_U$ **do**3: **for** $S_k, F_{S_k} \in F$ **do**4: Obtain LBR_{S_k, C_j} according to Eq. (21);5: Obtain MC_{S_k, C_j} according to Eq. (24);6: Obtain CER_{S_k, C_j} according to Eq. (25);7: Add CER_{S_k, C_j} to $CERSet$;8: **end for**9: **end for**10: **for** $CER_{S_k, C_j} \in CERSet$ **do**11: **if** $CER_{S_k, C_j} < minCER$ **then**12: $minCER = CER_{S_k, C_j}$;13: $TS = \langle S_k, C_j \rangle$;14: **end if**15: **end for**16: return TS ;**3.3. Target controller selection**

In this algorithm, we select the target controller and switch by minimizing the cost-effectiveness ratio (CER), which refers to the ratio of the migration cost to the obtained effect of load balancing. We aim to reduce the migration cost while achieving the optimal load balancing rate after migration. By the following equation, we can obtain the total processing overhead of controller C_m in period t , denoted by OH_{C_m} :

$$OH_{C_m} = n_{C_m} \times avg_{packet-in}. \quad (18)$$

Where n_{C_m} denotes the total number of Packet-In messages processed by controller C_m in period t . After S_k is migrated to C_j from C_i , the total processing overhead of overloaded controller C_i and underloaded controller C_j is updated by the following two equations:

$$OH'_{C_j} = OH_{C_j} + F_{S_k}. \quad (19)$$

$$OH'_{C_i} = OH_{C_i} - F_{S_k}. \quad (20)$$

The load balancing rate after migration, LBR_{S_k, C_j} can be obtained from the following equation:

$$LBR_{S_k, C_j} = 1 - \frac{\sum_{d=1}^{N_C} |OH'_{C_d} - OH'_{mean}|}{N_C \times OH'_{mean}}. \quad (21)$$

Where OH'_{mean} denotes the average overhead after migration.

In the process of switch migration, not only the load balancing rate among controllers should be considered, but also the migration cost should be reduced to diminish the negative impact of switch migration. The migration cost consists of two parts, which are controller load change cost $MC^1_{S_k, C_j}$ and message exchange cost $MC^2_{S_k, C_j}$. When switch S_k is taken over by controller C_j , C_j will handle the requests generated by S_k , and the load of C_j will change accordingly, resulting in the load change cost $MC^1_{S_k, C_j}$, which can be obtained from the following equation:

$$MC^1_{S_k, C_j} = F_{S_k} \times |h_{S_k, C_i} - h_{S_k, C_j}|. \quad (22)$$

Where h_{S_k, C_i} and h_{S_k, C_j} denote the logical distance required for S_k to reach C_i and C_j , respectively. When performing switch migration, the controller generates a portion of the communication overhead by sending the migration request to complete the corresponding load balancing operation, and the message exchange cost $MC^2_{S_k, C_j}$ can be obtained by the following equation:

$$MC^2_{S_k, C_j} = avg_{mig} \times Count_{mig} \times h_{C_j}. \quad (23)$$

Where avg_{mig} denotes the average packet size during migration, and $Count_{mig}$ denotes the total number of migration packets. The total migration cost can be obtained from the following equation:

$$MC_{S_k, C_j} = MC^1_{S_k, C_j} + MC^2_{S_k, C_j}. \quad (24)$$

Finally, we can further calculate the CER_{S_k, C_j} using the following equation:

$$CER_{S_k, C_j} = \frac{MC_{S_k, C_j}}{LBR_{S_k, C_j}}. \quad (25)$$

For each overloaded controller, when selecting the target controller and switch, we first traverse the underloaded controller set $\hat{C}_U$ obtained by Algorithm 1 and the candidate switch set F obtained by Algorithm 2. Next, according to Eqs. (21) and (24), the load balancing rate and the migration cost after the migration are obtained, respectively, and the cost-effectiveness ratio of this migration is also obtained. Finally, we return a pair of target controller and switch that minimize the cost-effectiveness ratio. The pseudocode of the target controller selection algorithm is shown in Algorithm 3.

To evaluate the scalability of the MOSM scheme in dense IoV environments, we analyze the computational complexity of Algorithm 3. In the worst case, the algorithm has a time complexity of $O(N_C \times N_S)$, where N_C is the number of controllers and N_S is the total number of switches. This upper bound arises because the algorithm may evaluate every combination of underloaded controllers (up to N_C) and candidate switches (up to N_S).

In typical SD-IoV deployments, the actual number of evaluation iterations is substantially reduced by the candidate switch filtering mechanism described in Algorithm 2, which eliminates switches that contribute negligibly to the overload condition. To demonstrate feasibility, consider three representative network scales: a small network ($N_C = 10, N_S = 500$) with a worst-case evaluation count of approximately 5000; a medium network ($N_C = 20, N_S = 2000$) with approximately 40,000 evaluations; and a dense network ($N_C = 50, N_S = 5000$) with approximately 250,000 evaluations. Even in the densest configuration, executing 250,000 simple arithmetic operations requires less than 10 ms on modern CPUs. Since load balancing is triggered solely when a controller is detected as overloaded (typically at intervals ranging from seconds to minutes), this computational overhead remains negligible. Furthermore, the number of evaluations is further reduced by the candidate switch filtering step, which prevents the initiation of unnecessary migrations during transient load spikes. Consequently, MOSM scales efficiently in terms of computational overhead and remains practical for deployment in large-scale and dynamic IoV networks.

3.4. Switch migration execution

The input to the switch migration execution algorithm includes the set of overloaded controllers and the set of underloaded controllers. The algorithm begins by sorting the overloaded controllers and treating the controller with the highest load preferentially. The candidate switch selection algorithm is then called to obtain the set of candidate switches F . Then, the sets F and $\hat{C}_U$ are fed into the target controller selection algorithm to obtain the target controller C_i and the switch S_k . Next, these are combined into a triplet with the overloaded controller C_i and added to the migration set MS . After that, the overloaded and underloaded controller sets will be updated. After all the overloaded controllers have completed the migration decision, the final migration set can be obtained and fed into the migration execution module to complete the migration operation. The pseudocode of the migration execution algorithm is shown in Algorithm 4.

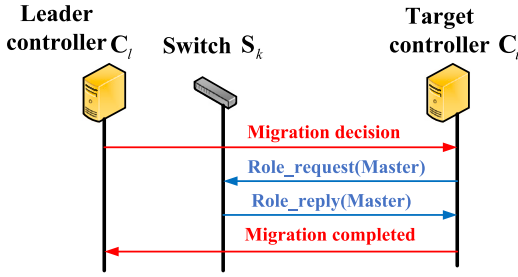
In the OpenFlow protocol, a single switch can connect to only one master controller but multiple slave controllers. Meanwhile, the switch is not given the right to decide the role of the controller, only the right to be informed. After the target controller, whose role is slave, is selected, it will actively send a ROLE-REQUEST message to the

Algorithm 4 Switch Migration Execution**Input:** Overloaded controller set $\hat{C}_O$, Underloaded controller set $\hat{C}_U$;**Output:** Migration set MS ;

```

1: Initialization:  $MS = \emptyset$ ;
2: Sort  $\hat{C}_O$  in descending order;
3: for  $C_i \in \hat{C}_O$  do
4:   if  $\hat{C}_U \neq \emptyset$  then
5:      $F \leftarrow \text{Algo. 2}(C_i)$ ;
6:      $C_i, S_k \leftarrow \text{Algo. 3}(\hat{C}_U, F)$ ;
7:      $MS \leftarrow \langle C_i, S_k, C_i \rangle$ ;
8:      $\hat{C}_O.\text{remove}(C_i)$ ;
9:      $\hat{C}_U.\text{remove}(C_i)$ ;
10:  end if
11: end for
12: return  $MS$ ;

```

**Fig. 4.** The specific execution procedure of the switch migration.

switch to shift its role. When the switch receives the ROLE-REQUEST message, it will shift the target controller to master. Since the master controller is exclusive, the original master controller will be converted to a slave controller. After the role transition, the switch will send a ROLE-REPLY message to inform the target controller that the role transition is finished and the switch migration is complete.

The specific switch migration execution procedure is shown in Fig. 4. In the initial state, the leader controller C_l detects the load imbalance and decides to migrate switch S_k to target controller C_t . Then, it informs controller C_t to perform the switch migration operation. After receiving the migration operation command, C_t sends a ROLE-REQUEST message to switch S_k . Upon receiving the request, S_k updates the role of C_t to master while automatically transforming the role of the initial controller into slave.

In the switch migration process, it is essential to ensure that active flows, those being processed or in transit, are not disrupted. MOSM addresses this by incorporating a flow state preservation mechanism during the role transition between controllers. When a target controller C_t is selected to take over switch S_k , the original leader controller C_l synchronizes the active flow entries and their associated state information with C_t via the control channel. This synchronization occurs before the role change begins, ensuring C_t has an up-to-date view of all ongoing flows. During migration, packets from active flows are temporarily buffered at the switch until C_t confirms the completion of the role transition. Once C_t receives the ROLE-REPLY message from S_k , it assumes master control and processes the buffered packets. This mechanism ensures that no active flow is dropped or interrupted, preserving service continuity and enhancing the reliability of the load balancing process.

3.5. Theoretical analysis

In this section, we provide a theoretical analysis of the MOSM algorithm and discuss its limitations as a heuristic method. We first examine the theoretical foundations of the algorithm, analyzing its performance

boundaries and stability. Then, we explore its nature as a heuristic approach, outlining the applicable scenarios and its limitations.

The MOSM algorithm aims to balance controller load and migration cost through a heuristic multi-objective decision process. Specifically, MOSM evaluates candidate switch-controller migration pairs using the cost-effectiveness ratio, which jointly considers the load balancing rate after migration and the migration cost. This design enables MOSM to obtain feasible and cost-effective migration decisions within a limited decision time, which is important for large-scale and dynamic SD-IoV environments. It should be noted that MOSM is a heuristic method rather than an approximation algorithm with a formal performance guarantee. Since no theoretical approximation ratio or optimality bound is derived in this work, MOSM does not guarantee global optimality. Its effectiveness is therefore evaluated empirically under different traffic intensities and network topologies in Section 4.

MOSM employs heuristic methods to address load balancing and migration, relying on approximate solutions that provide satisfactory outcomes within limited timeframes. While this approach is efficient for large-scale dynamic networks, it does not guarantee global optimality. In extreme cases of network topology changes or severe load imbalances, MOSM may fail to identify the best migration strategies. Additionally, the algorithm's performance is sensitive to parameter settings, such as load weighting coefficients and migration strategies, which can significantly impact the results.

In conclusion, while MOSM provides efficient solutions for most network scenarios, further optimization or integration with other techniques may be required for optimal performance in specialized cases.

In practical SD-IoV environments, delays in the reporting of controller load statistics may affect the real-time decision-making accuracy of MOSM. We analyze the sensitivity of the system in two key aspects:

- **Load Measurement Phase:** MOSM uses the coefficient of variation method to determine the weights for CPU, bandwidth, and memory utilization. These weights are calculated based on the overall statistical characteristics of preliminary experimental data and remain fixed during operation, without updating with real-time data. Consequently, short-term delays in data reporting do not cause fluctuations in the weight values, ensuring consistency in the load evaluation benchmark. However, delays can cause discrepancies between the reported raw metric values (i.e., O_{CPU} , O_{BW} , O_{MEM}) and the actual values used to calculate the controller load (Eq. (10)). The impact of this discrepancy on load classification (overload/underload) depends on the delay duration and the rate of load change. Specifically, let the real load of the controller be $L(t)$, and the reported load be $L(t - \delta)$, where δ is the delay. If the load variation within the interval $[t - \delta, t]$ is smaller than the difference between the overload threshold and the current load, the classification result will remain unchanged, and the migration decision will not be affected.
- **Target Controller Selection Phase:** The migration decision relies on the processing overhead OH_{C_m} of each controller and the candidate switch overhead F_{S_k} (Eqs. (18)–(24)). When these data are delayed, the calculated load balancing rate LBR_{S_k, C_j} and migration cost MC_{S_k, C_j} may deviate from the actual values. Since MOSM uses the CER as the optimization objective, which is the ratio of migration cost to load balancing rate, the sensitivity of this ratio's error to delay depends on the relative rate of change between the two. Analysis shows that when the data delay differences among all controllers are smaller than a sampling period and the load variation rate is low, the CER ranking remains consistent with the ideal synchronized case, meaning that the selected migration pair is likely to remain consistent with the decision obtained under the ideal synchronized case.

In summary, MOSM exhibits a certain level of robustness against moderate statistical delays, maintaining decision stability when delays are small and load variations are gradual. Extreme delays or sharp load fluctuations may result in suboptimal migrations, but the algorithm will not fail entirely.

4. Performance evaluation

The scheme is implemented using the Ryu [31] controller and the Mininet [32] simulation platform. Ryu is an open-source SDN/OpenFlow controller implemented in Python and supports the OpenFlow v1.3 protocol [33] adopted in the experiments. Mininet is a lightweight virtualization simulation tool that enables the rapid construction of large-scale SDN prototype systems on computers with limited resources. In the experiment, four controllers are independently deployed on four virtual machines to manage all switches in the network, and Mininet is used to simulate the network environment. We deployed four controllers in the experiment, and the configuration of the experimental devices was as follows: AMD Ryzen 5 processors with 16 GB RAM and Intel i7 processors with 16 GB RAM.

To generate network traffic, we used iPerf [34] to emulate UDP flows under different traffic intensities. To capture the stochastic and bursty characteristics of IoV traffic, the flow-request arrival process was modeled as a non-homogeneous Poisson process [35] with a time-varying rate $\lambda(t)$. For each tested traffic intensity, λ_0 denotes the nominal average flow-request rate. The rate $\lambda(t)$ followed a 100-s periodic cycle, consisting of an 80-s normal state and a 20-s burst state. The arrival rates in the two states were set to

$$\lambda_N = \frac{\lambda_0}{(1 - p_b) + p_b R}, \quad (26)$$

and

$$\lambda_B = R\lambda_N, \quad (27)$$

respectively, where $p_b = 0.2$ is the burst duty ratio and $R = 4$ is the burst-to-normal ratio. Under this setting, the time-averaged arrival rate over one cycle equals λ_0 , and the peak-to-average ratio is $\lambda_B/\lambda_0 = 2.5$. Each experiment lasted 400 s with a sampling interval of $\Delta t = 5$ s. In each interval, the number of flow requests was sampled from a Poisson distribution with mean $\lambda(t)\Delta t$, and the corresponding UDP flows were generated using iPerf. To ensure a fair comparison, the same traffic trace was used for all compared schemes under each traffic intensity and topology. The average sizes of Packet-In and migration messages were set to $avg_{packet-in} = 1620$ bytes and $avg_{mig} = 92$ bytes, respectively. The link bandwidth was set to 1000 Mbps.

For load measurement in the MOSM framework, we adopted the coefficient of variation (CV) method to determine the weights of CPU, bandwidth, and memory utilization based on pre-experimental data. These weights are fixed during subsequent comparative experiments to maintain evaluation fairness. The resulting weights — 0.22, 0.03, and 0.75 for CPU, memory, and bandwidth, respectively — are representative values obtained from the pre-experiment phase [30].

To assess the adaptability of our scheme across different network environments, we selected four real network topologies from the Internet Topology Zoo [36]. The topologies, which progressively increase in size, are shown in Table 3. These topologies are ideal for evaluating control plane load balancing for two main reasons. First, the load imbalance addressed in this paper is primarily caused by traffic burstiness and uneven flow request distribution, which are independent of the specific wireless topology. The aggregate traffic from vehicles is transmitted via fixed infrastructure (RSUs, BSs) and forwarded through the wired network to the controllers, abstracting the dynamics of individual vehicle movements in the bursty traffic generation process. Second, by using standardized topologies from the Internet Topology Zoo, we ensure fair comparisons with existing work and facilitate the reproducibility of our results. While integrating mobility simulators and wireless network models would enhance evaluation in realistic vehicular environments, the current setup effectively captures the essential load dynamics and validates MOSM's core mechanisms. Extending the evaluation to mobility-aware scenarios is a promising direction for future research.

These topologies allow us to evaluate the performance of the MOSM scheme under varying network configurations, ensuring that the solution is robust across different real-world scenarios.

Table 3

Topology features.

Topology Feature	Eenet	Aarnet	Cesnet	Canerie
Number of Links	16	24	22	41
Number of Nodes	13	19	23	32

4.1. Performance metrics and comparison schemes

To demonstrate the feasibility of our scheme, we selected three metrics to evaluate its performance: (1) Response time: when there is a load imbalance, the average response time of the controller will be significantly higher. Since the response time directly affects the overall network quality of service, we define response time as the time interval between the time the Packet-In message arrives at the controller and the time the controller replies with a Packet-Out or Flow-Mod to the switch [13]; (2) Load balancing rate [37]: the load balancing rate is the most effective metric for measuring switch migration performance. A higher load balancing rate indicates a more even load distribution and better load balancing performance. The definition of load balancing rate refers to Eq. (21); (3) Migration cost: since the switch migration incurs additional processing overhead for the controller, when the controller is overloaded, performing the switch migration operation will further consume the processing resources of the controller. Therefore, it is necessary to reduce the migration cost during the load balancing process.

To effectively evaluate the performance of MOSM, we select four schemes for comparison, and the main features of these schemes are as follows:

- Static [38]: introduces the Controller Placement Problem (CPP) and methods to select controller number and location. Unlike this static approach, MOSM uses multi-metric optimization (CPU, memory, bandwidth) to enable dynamic switch migration and improved load balancing.
- Random [39]: proposes the ElasticCon elastic controller pool adjusting dynamically to traffic. MOSM extends this with multi-metric evaluation to reduce migration costs and maintain real-time load balance.
- Remained [12]: focuses on dynamic switch migration based on controller load. MOSM further considers traffic, latency, and network topology to enhance migration precision and efficiency.
- RetroFlow++ [28]: a resilient control recovery scheme for SD-WANs under controller failures. It dynamically configures offline switches to legacy or SDN mode and selectively releases control resources from active controllers to accommodate more offline switches. The scheme jointly optimizes switch-controller mapping to minimize communication overhead while maintaining flow programmability.
- MOSM: multiple performance metrics are used to measure controller load and the cost-effectiveness ratio is considered as the optimization objective when making the migration decisions.

4.2. Performance evaluation

4.2.1. The effectiveness of MOSM

To verify the effectiveness of the MOSM, we test it under the Eenet topology. First, we increase the load on the controller C_0 and record the change in load over time under MOSM and static schemes, respectively. The test duration is 400 s, and the sampling is performed every 5 s. The controller load distribution under Static scheme is shown in Fig. 5(a). We can observe that the load on controller C_0 is much higher than that of the other three controllers, and there is a severe load imbalance. The load variation of each controller over time when MOSM is employed is

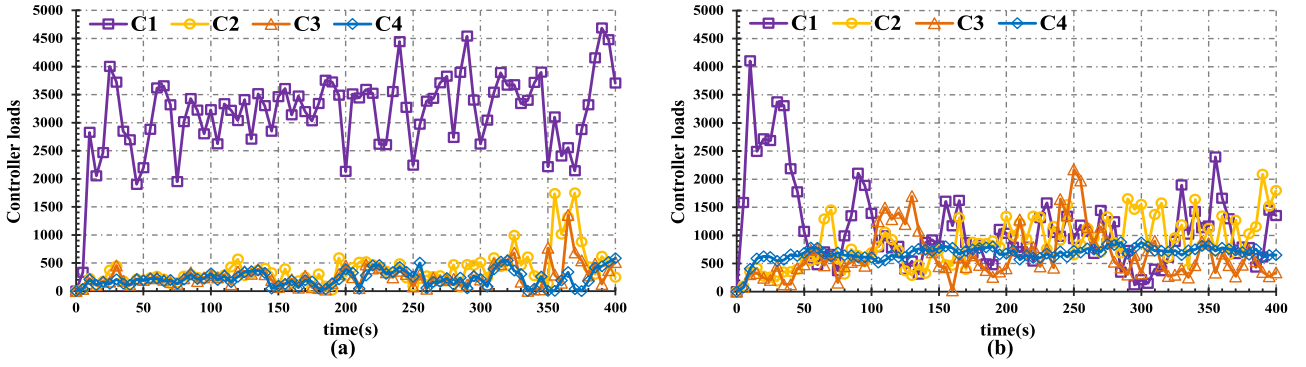


Fig. 5. (a) Load distribution under Static scheme; (b) Load distribution under MOSM scheme.

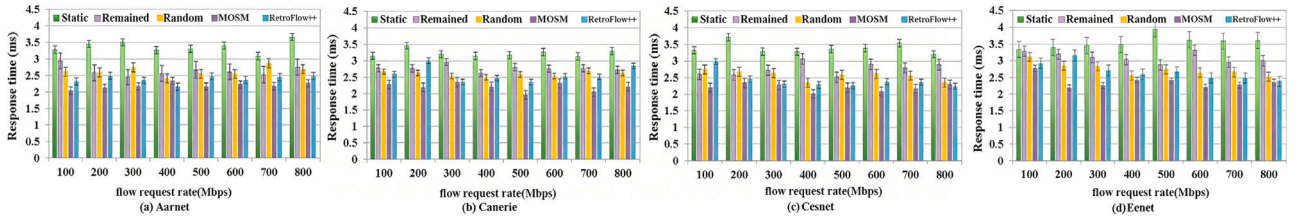


Fig. 6. Response time of different topologies and schemes.

shown in Fig. 5(b). It can be seen that the MOSM can quickly detect the overloaded controller C_0 and transfer a portion of the load to the three underloaded controllers via switch migration, effectively achieving load balancing.

4.2.2. Response time

When the controller is overloaded, its response time will be significantly higher than that of the underloaded controller. Therefore, load balancing among the controllers can effectively reduce the response time of the controllers and improve the quality of the overall network. We illustrate the average response time variation of different load balancing schemes with varying flow request rates across four different network topologies in Fig. 6. These results demonstrate MOSM's ability to adapt to various network topologies. It shows that the MOSM is able to maintain a relatively stable response time, and the average response time of MOSM is also lower compared to the other four schemes. Since the static mapping scheme does not employ any load balancing strategy, the response time of the overloaded controller increases significantly when the load is not balanced among the controllers. Thus, the Static scheme has the highest average response time. The Random scheme selects the closest controller as the migration domain. It overlooks the target controller's remaining capacity, making it easy to overload. The Remained scheme, on the other hand, only evaluates the target controller's remaining capacity and ignores the overall load distribution of the network. The RetroFlow++ does not fully consider the dynamic controller load and the additional processing overhead of switches, which can limit its performance under peak traffic conditions. In contrast, MOSM considers the additional processing overhead caused by the candidate switch to the target controller, which can more accurately measure the impact of each switch on the controller load. In addition, MOSM can evaluate the load status of each controller more accurately and improve the load balancing performance through multiple performance indicators. Under peak load conditions, MOSM achieved a response time reduction of 31%, 20%, 15%, and 19.5% compared to the Static, Random, Remained, and RetroFlow++ schemes, respectively.

4.2.3. Load balancing rate

To accurately compare the load balancing performance of each scheme, we introduce the load balancing rate. Since the static scheme

does not employ any load balancing strategy, we exclude the static scheme from the comparison. The variation of the average load balancing rate of the four load balancing schemes under different topologies is shown in Fig. 7. Since the Random scheme directly selects the nearest underloaded controller as the target controller, which is most likely to overload the target controller, the load balancing rate of this scheme fluctuates wildly under the four topologies. In contrast, the load balancing rates of the Remained scheme and MOSM schemes are more stable than the Random scheme. The MOSM uses the coefficient of variation method to determine the utilization weight, which enables accurate partitioning of overloaded and underloaded controllers. In addition, MOSM considers the load balancing rate as one of the optimization metrics when making the migration decision, which can make the network load distribution more balanced after the migration. As can be seen from Fig. 7, the load balancing rate of MOSM can be maintained above 0.5 in the four topologies with gradually increasing sizes, and the load balancing rate of MOSM changes the least as the flow request rate increases. In the four topologies, the load balancing rate of the MOSM scheme is on average 30% and 31% higher than that of the Random and Remained schemes, respectively. Furthermore, compared to the RetroFlow++ scheme, MOSM achieves a 4.3% higher load balancing rate by leveraging fine-grained flow-level controller assignment and multi-metric optimization.

In MOSM, load measurement is based on weights assigned to CPU, bandwidth, and memory utilization, which are determined using the coefficient of variation (CV) method [30]. Bandwidth is assigned the highest weight because of its pronounced variability across traffic conditions. To justify this design choice, we analyze its influence under different traffic intensities based on the results shown in Fig. 7. Under low traffic conditions, controller loads remain well below capacity, so moderate variations in the bandwidth weight have minimal effect on load classification. Under high traffic conditions, bandwidth becomes the dominant bottleneck. Assigning it a higher weight ensures that controllers experiencing bursty flows are promptly identified as overloaded. The observed stability of the load balancing rate across all topologies and flow intensities, such as MOSM maintaining a load balancing rate above 0.5 even under peak loads, confirms that the weight selection is effective and robust. Moreover, the consistent ranking of variability from bandwidth to CPU to memory across traffic scenarios

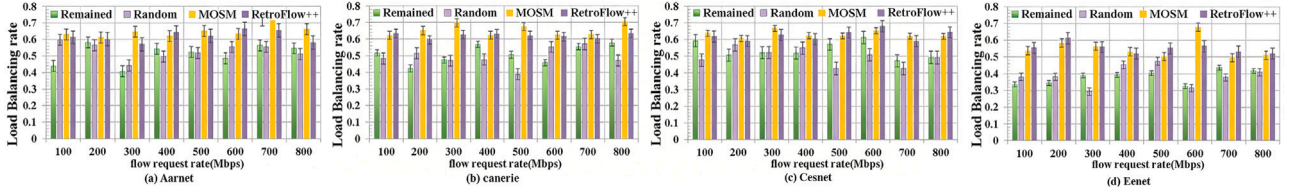


Fig. 7. Load balancing rate of different topologies and schemes.

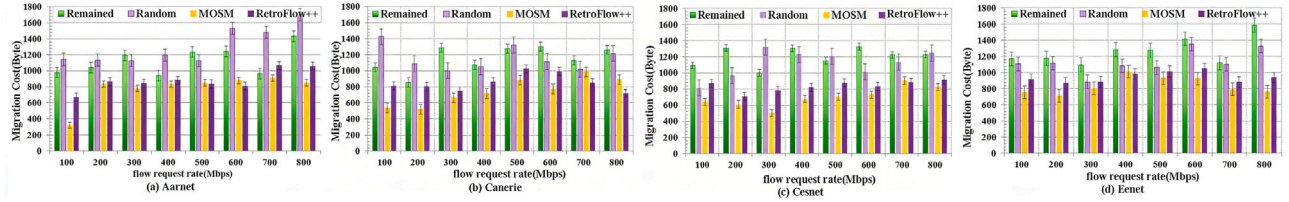


Fig. 8. Migration cost of different topologies and schemes.

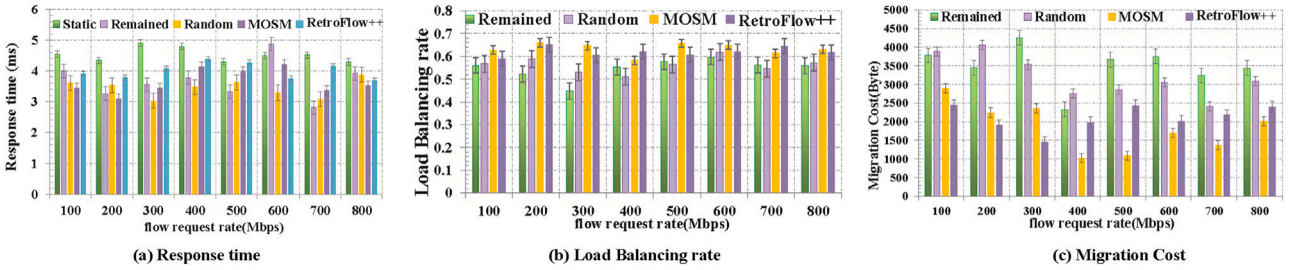


Fig. 9. Real-traffic experimental scenarios on the Canerie topology.

indicates that fixed weights generalize well, eliminating the need for recalibration for each scenario.

4.2.4. Migration cost

We compare the migration cost of the four switch migration schemes, and the result is shown in Fig. 8. Since the Static scheme does not involve switch migration, it is excluded from the migration cost comparison. MOSM considers migration cost as one of the optimization metrics when selecting the target controller and switch, which significantly reduces the migration cost during load balancing and mitigates the negative impact on the network. However, the Remained scheme and Random scheme do not consider the migration cost when making migration decisions, so their required migration costs are considerably higher than MOSM. Lastly, experimental data from the different schemes shows that the MOSM scheme reduces the migration cost by 33% and 32%, respectively, compared to the Random and Remained schemes, and achieves an 18.7% lower migration cost compared to the RetroFlow++ scheme.

4.2.5. Scaling experiments under real-world traffic

To further assess the effectiveness of the proposed MOSM scheme under more realistic network conditions, supplementary experiments are conducted using real-world traffic traces on the largest topology, Canerie (32 nodes). This topology is selected due to its relatively large scale, which better reflects the complexity of practical SD-IoV deployments.

The real-world traffic dataset utilized in these experiments consists of publicly available packet traces from the MAWI (Measurement and Analysis on the WIDE Internet) working group [40], which capture actual backbone traffic with diverse characteristics. This dataset is particularly suitable for emulating IoV scenarios, as it exhibits key traffic features prevalent in vehicular networks, including high burstiness

resulting from abrupt changes in vehicle density, mixed protocol types encompassing TCP/UDP for various V2X applications, and dynamic flow patterns driven by vehicle mobility and fluctuating communication demands. A 15-minute trace from 14:00 to 14:15 on March 1, 2026, is extracted, which contains a rich mixture of these behaviors and thus provides a realistic benchmark for evaluating SD-IoV controller load balancing.

In the experimental setup, the pcap file is replayed into the Mininet-emulated Canerie topology using tcpreplay [41], a tool that supports packet trace replay at adjustable speeds. The same Ryu controllers and OpenFlow v1.3 configuration as in the synthetic traffic experiments are employed. All other parameters, including link bandwidth (1000 Mbps) and average packet sizes, remain unchanged. During the replay, the same performance metrics are collected: response time, load balancing rate, and migration cost.

Fig. 9(a) illustrates the average response time of the evaluated schemes under real-world traffic. Consistent with the synthetic traffic experiments, MOSM achieves the lowest response time across all flow intensities. This advantage stems from its accurate load measurement and multi-objective migration decisions. Under peak load, MOSM reduces response time by 19.5% compared to RetroFlow++ and by over 40% relative to the Static scheme. Response times under real traffic are slightly higher than those observed in the simulated scenarios, attributable to increased traffic variability and burstiness. Nevertheless, the relative improvement achieved by MOSM remains consistent under the more variable real-world conditions.

Fig. 9(b) presents the load balancing rate of each scheme under real-world traffic. MOSM achieves a load balancing rate consistently above 0.58 with minimal fluctuations as flow intensity increases, outperforming all other schemes. This performance surpasses that of the other evaluated schemes. The results demonstrate that MOSM's multi-metric load measurement and cost-effectiveness optimization generalize effectively to real traffic patterns characteristic of IoV dynamics.

Fig. 9(c) compares the migration cost across the evaluated schemes. MOSM demonstrates a clear advantage over RetroFlow++, achieving a 14.7% reduction in migration cost and thereby lowering the resource overhead typically associated with load balancing. Under real-world traffic, migration costs are slightly higher than those observed in simulated experiments, attributable to more frequent flow entry updates triggered by diverse traffic patterns. Nonetheless, MOSM effectively minimizes this overhead by incorporating both load variation and message exchange costs into its CER metric.

Overall, these real-world experiments validate that MOSM maintains its superior load balancing performance and low migration overhead even under the complex and bursty traffic conditions characteristic of actual network environments, thereby demonstrating its practical applicability in real SD-IoV deployments.

5. Conclusion

In this paper, we propose the MOSM scheme to solve the load imbalance in the distributed control plane in SD-IoV. In this scheme, we select three performance metrics to accurately measure the controller load: CPU, bandwidth, and memory utilization. Moreover, the coefficient of variation is introduced to determine the load weight of each metric. At the same time, when selecting candidate switches, we further consider the controller overhead caused by flow entry updates to improve load balancing performance. And in the selection of the target controller, we optimize the migration process by minimizing the migration cost and maximizing the load balancing rate after migration. Finally, the effectiveness of MOSM is demonstrated through experiments conducted in the Mininet virtualized environment by comparing it with the Static scheme. To further evaluate the load balancing performance of our scheme, we selected two existing schemes for comparison. The test results show that, compared with the existing schemes, the load balancing rate and migration cost of MOSM are increased by 31% and reduced by 33% on average, respectively.

In future work, we plan to integrate MOSM with vehicular mobility simulators and wireless network models to assess its performance in realistic Vehicle-to-Everything scenarios, considering dynamic topology changes and handovers. Although MOSM has demonstrated strong load balancing performance in experiments, the statistical latency of controller load data may still impact real-time decision-making in practical deployments. To address this, future work will explore the combination of prediction mechanisms and asynchronous update strategies to enhance the robustness and adaptability of MOSM in environments with latency.

CRediT authorship contribution statement

Yong Liu: Writing – review & editing, Supervision. **Bin Xu:** Writing – review & editing. **Quanze Liu:** Writing – original draft. **Qian Meng:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

- [1] Victoria Stodden, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, Michela Taufer, Enhancing reproducibility for computational methods, *Science* 354 (6317) (2016) 1240–1241.
- [2] Dylan Chapp, Kento Sato, Dong H. Ahn, Michela Taufer, Record-and-replay techniques for HPC systems: A survey, *Supercomput. Front. Innov.* 5 (1) (2018) 11–30.
- [3] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese Van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, Jeffrey Vetter, The future of scientific workflows, *Int. J. High Perform. Comput. Appl.* 32 (1) (2018) 159–175.
- [4] Xiang Li, Lingyun Lu, Wei Ni, Abbas Jamalipour, Dalin Zhang, Haifeng Du, Federated multi-agent deep reinforcement learning for resource allocation of vehicle-to-vehicle communications, *IEEE Trans. Veh. Technol.* 71 (8) (2022) 8810–8824.
- [5] Saeid Iranmanesh, Forough Shirin Abkenar, Abbas Jamalipour, Raad Raad, A heuristic distributed scheme to detect falsification of mobility patterns in internet of vehicles, *IEEE Internet Things J.* 9 (1) (2021) 719–727.
- [6] Nishant Sharma, Naveen Chauhan, Narottam Chand, Security challenges in internet of vehicles (IoV) environment, in: 2018 First International Conference on Secure Cyber Computing and Communication, ICSCCC, IEEE, 2018, pp. 203–207.
- [7] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, Haiyong Xie, A survey on software-defined networking, *IEEE Commun. Surv. Tutorials* 17 (1) (2014) 27–51.
- [8] Umme Zakia, Hanene Ben Yedder, Dynamic load balancing in SDN-based data center networks, in: 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON, IEEE, 2017, pp. 242–247.
- [9] Huawei Huang, Peng Li, Song Guo, Weihua Zhuang, Software-defined wireless mesh networks: architecture and traffic orchestration, *IEEE Netw.* 29 (4) (2015) 24–30.
- [10] Abbas Bradai, Kamal Singh, Toufik Ahmed, Tinku Rasheed, Cellular software defined networking: a framework, *IEEE Commun. Mag.* 53 (6) (2015) 36–43.
- [11] Chen Jiacheng, ZHOU Haibo, Zhang Ning, Yang Peng, Gui Lin, Shen Xuemin Sherman, Software defined internet of vehicles: architecture, challenges and solutions, *J. Commun. Inf. Netw.* 1 (1) (2016) 14–26.
- [12] Pantelimon-Teodor Tivig, Eugen Borcoci, Marius Vochin, Dynamic offloading the SDN control plane In Large Area networks by condition-aware migration of forwarding devices, in: 2022 25th International Symposium on Wireless Personal Multimedia Communications, WPMC, IEEE, 2022, pp. 205–210.
- [13] Jie Cui, Qinghe Lu, Hong Zhong, Miaomiao Tian, Lu Liu, A load-balancing mechanism for distributed SDN control plane using response time, *IEEE Trans. Netw. Serv. Manag.* 15 (4) (2018) 1197–1206.
- [14] Fetia Bannour, Sami Souihi, Abdelhamid Mellouk, Distributed SDN control: Survey, taxonomy, and challenges, *IEEE Commun. Surv. Tutorials* 20 (1) (2017) 333–354.
- [15] Yashar Ganjali, Amin Tootoonchian, *{HyperFlow}*: A distributed control plane for *{OpenFlow}*, in: 2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN 10), 2010.
- [16] Soheil Hassas Yeganeh, Yashar Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 2012, pp. 19–24.
- [17] Yonghong Fu, Jun Bi, Kai Gao, Ze Chen, Jianping Wu, Bin Hao, Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks, in: 2014 IEEE 22nd International Conference on Network Protocols, IEEE, 2014, pp. 569–576.
- [18] Md Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, Raouf Boutaba, Dynamic controller provisioning in software defined networks, in: Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013), IEEE, 2013, pp. 18–25.
- [19] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, Marco Hoffmann, Heuristic approaches to the controller placement problem in large scale SDN networks, *IEEE Trans. Netw. Serv. Manag.* 12 (1) (2015) 4–17.
- [20] Bang Zhang, Xingwei Wang, Min Huang, Multi-objective optimization controller placement problem in internet-oriented software defined network, *Comput. Commun.* 123 (2018) 24–35.
- [21] Tao Hu, Jianhui Zhang, Lujia Cao, Jie Gao, A reliable controller deployment strategy based on network condition evaluation in SDN, in: 2017 8th IEEE International Conference on Software Engineering and Service Science, ICSESS, IEEE, 2017, pp. 367–370.
- [22] Advait Abhay Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, Ramana Kompella, Elasticcon: an elastic distributed sdn controller, in: Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2014, pp. 17–28.
- [23] Songzhou Li, Zhenghua Xin, Xu Xu, Zhiwei Zhang, Load balancing algorithm of SDN controller based on dynamic threshold, in: 2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science, ACCTCS, IEEE, 2023, pp. 517–520.

- [24] Guozhen Cheng, Hongchang Chen, Game model for switch migrations in software-defined network, *Electron. Lett.* 50 (23) (2014) 1699–1700.
 - [25] Tao Hu, Julong Lan, Jianhui Zhang, Wei Zhao, EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking, *Peer-To-Peer Netw. Appl.* 12 (2) (2019) 452–464.
 - [26] Hamza Mokhtar, Xiaoqiang Di, Ying Zhou, Alzubair Hassan, Ziyi Ma, Shafiu Musa, Multiple-level threshold load balancing in distributed SDN controllers, *Comput. Netw.* 198 (2021) 108369.
 - [27] Zehua Guo, Wendi Feng, Sen Liu, Wenchao Jiang, Yang Xu, Zhi-Li Zhang, RetroFlow: Maintaining control resiliency and flow programmability for software-defined WANs, in: *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.
 - [28] Zehua Guo, Songshi Dou, Sen Liu, Wendi Feng, Wenchao Jiang, Yang Xu, Zhi-Li Zhang, Maintaining control resiliency and flow programmability in software-defined WANs during controller failures, *IEEE/ACM Trans. Netw.* 30 (3) (2022) 969–984.
 - [29] Hong Zhong, Jinshan Xu, Jie Cui, Xiuwen Sun, Chengjie Gu, Lu Liu, Prediction-based dual-weight switch migration scheme for SDN load balancing, *Comput. Netw.* 205 (2022) 108749.
 - [30] Hervé Abdi, Coefficient of variation, *Encycl. Res. Des.* 1 (5) (2010) 169–171.
 - [31] Fujita Tomonori, Introduction to ryu sdn framework, *Open Netw. Summit* (2013) 1–14.
 - [32] Karamjeet Kaur, Japinder Singh, Navtej Singh Ghuman, Mininet as software defined networking testing platform, in: *International Conference on Communication, Computing & Systems, ICCCS*, 2014, pp. 139–142.
 - [33] Carlos Fernandez, Jose L. Munoz, Software defined networking (SDN) with OpenFlow 1.3, open vSwitch and ryu, *UPC Telemat. Dep.* (2015).
 - [34] Chung-Hsing Hsu, Ulrich Kremer, IPERF: A framework for automatic construction of performance prediction models, in: *Workshop on Profile and Feedback-Directed Compilation (PFDC)*, Paris, France, 1998, p. 20.
 - [35] John Frank Charles Kingman, *Poisson Processes*, vol. 3, Clarendon Press, 1992.
 - [36] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, Matthew Roughan, The internet topology zoo, *IEEE J. Sel. Areas Commun.* 29 (9) (2011) 1765–1775.
 - [37] Kshira Sagar Sahoo, Deepak Puthal, Mayank Tiwary, Muhammad Usman, Bibhudatta Sahoo, Zhenyu Wen, Biswa PS Sahoo, Rajiv Ranjan, ESMLB: Efficient switch migration-based load balancing for multicontroller SDN in IoT, *IEEE Internet Things J.* 7 (7) (2019) 5852–5860.
 - [38] Brandon Heller, Rob Sherwood, Nick McKeown, The controller placement problem, *ACM SIGCOMM Comput. Commun. Rev.* 42 (4) (2012) 473–478.
 - [39] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, Ramana Kompella, Towards an elastic distributed SDN controller, *ACM SIGCOMM Comput. Commun. Rev.* 43 (4) (2013) 7–12.
 - [40] Kenjiro Cho, Koushirou Mitsuya, Akira Kato, Traffic data repository at the *{WIDE}* project, in: *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.
 - [41] Tcpreplay, 2026, [Online]. Available at: <https://tcpreplay.appneta.com/>. (Accessed March 2026).
- Yong Liu**, Corresponding author, Hangzhou Normal University. He received the Ph.D. degree in Communication and Information System from Xidian University, Xi'an, China, in 2021. He is currently an associate professor at Hangzhou Normal University. His research interests include data center networks, software-defined networking, and optical interconnected networks.
- Bin Xu**, Hangzhou Normal University. He received the B.S. degree in Computer Science and Technology from Zhejiang Ocean University, Zhejiang, China, in 2024. He is currently working toward the M.S. degree in Computer Science and Technology with the School of Information Science and Technology, Hangzhou Normal University, Hangzhou, China. His research focuses on load optimization in software-defined networking.
- Quanze Liu**, Hangzhou Normal University. He received the M.S. degree in Software Engineering with the School of Information Science and Technology, Hangzhou Normal University, Hangzhou, China, in 2024. His research interests include data center networks and software-defined networking.
- Qian Meng**, Hangzhou Normal University. She received the M.S. degree with the Department of Mathematics from Hangzhou Normal University, Hangzhou, China, in 2016, and the Ph.D. degree with the Department of Telecommunication Engineering from Xidian University, Xi'an, China, in 2020. She is currently a Lecturer with the Department of Mathematics in Hangzhou Normal University. Her research interests include information security and applied cryptography.