BSM-LP: Bidirectional Switch Migration With Controller Load Prediction for Software-Defined Internet of Things

Quanze Liu, Yong Liu, Qian Meng, and Tianyi Yu

***Abstract*—The software-defined Internet of Things (SD-IoT) utilizes the centralized control and programmability of software-defined networking (SDN) to enhance network performance optimization and efficient resource utilization in Internet of Things. As the network scale expands, the multiple-controller architecture becomes crucial for ensuring reliability and scalability in SD-IoT. However, the dynamic changes in traffic patterns often lead to imbalanced loads among controllers. Existing solutions primarily focus on switch migration schemes, but traditional schemes primarily rely on real-time data to assess controller loads, which fails to predict future controller loads and leads to unnecessary switch migrations. Meanwhile, existing schemes frequently encounter the challenge of overloading the target controller, leading to reduced migration efficiency. Furthermore, conventional schemes tend to overlook the issue of isolated nodes that arise from switch migrations, thereby compromising network reliability and security. To address these challenges, we propose bidirectional switch migration based on load prediction (BSM-LP), which utilizes an ATT-GRU model to accurately predict controller loads based on historical load data, thereby preventing unnecessary switch migrations. Moreover, we introduce a bidirectional switch migration algorithm that enhances migration efficiency while avoiding overloading the target controller. Additionally, we present an algorithm for identifying and integrating isolated nodes to reduce their occurrence. Finally, we validate the effectiveness of BSM-LP, and the experimental results demonstrate that it reduces the load imbalance rate by an average of 22.3% and the response time by 30.5% compared to existing schemes.**

***Index Terms*—Internet of Things (IoT), load balancing, load prediction, software-defined networking (SDN), switch migration.**

I. Introduction

THE Internet of Things (IoT) constitutes an intelligent network that possesses the ability to exchange information among objects without direct human involvement. These objects encompass a variety of devices, including sensors, actuators, computers, and other intelligent entities. Through the integration of advanced technologies, such as 5G communications, cloud computing, edge computing, and sensor systems, the IoT enables diverse functionalities and finds extensive applications in domains, such as healthcare, community services, transportation, and environmental monitoring [1]. However, the expanding scale of the IoT poses challenges for traditional network technologies. The complexity associated with managing the Internet hampers the dynamic deployment of new services, thus impeding the development of IoT.

To overcome these challenges, software-defined networking (SDN) has emerged as a potential solution [2], [3], [4]. SDN transforms traditional networks into programmable networks by separating the network control plane from the data plane and being operated by one or more SDN controllers. By integrating SDN with the IoT, a distributed software-defined IoT (SD-IoT) architecture with high controllability and flexibility has been proposed [5]. The general architecture of SD-IoT is illustrated in Fig. 1, which consists of the application, network, and perception layers. The application layer utilizes a specialized application programming interface (API) known as the northbound API to establish communication with the SDN controller. It leverages the information gathered at the network layer to provide customers with application-specific services. In the architecture, the SDN switch employs a flow table for packet forwarding, which can be interconnected and match a broader range of fields. When a packet arrives at the switch, it undergoes a comparison process with the entries in the flow table. If a match is found, the packet's forwarding actions associated with the entry are executed. Otherwise, the switch forwards the packet to the SDN controller for the forwarding decision via the southbound API. The SDN controller analyzes the received packet and determines the appropriate action. If necessary, it creates a new flow entry in the switch's flow table to define how similar packets should be handled in the future. At the bottom of the architecture lies the perception layer, which comprises diverse devices. Among these devices, sensors play a crucial role in various domains by enabling data perception and collection. Additionally, wireless access points (APs) like WiFi and WiMAX are employed to facilitate the seamless transmission of messages from IoT devices to the SDN switch through wireless channels.

Manuscript received 20 December 2023; revised 15 March 2024; accepted 10 April 2024. Date of publication 19 April 2024; date of current version 26 June 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62302136; in part by the Zhejiang Provincial Natural Science Foundation of China under Grant LQ22F010001; and in part by the Starting Research Funds from the Hangzhou Normal University under Grant 4115C50221204137 and Grant 4085C50220204093. *(Corresponding author: Yong Liu.)*

Quanze Liu, Yong Liu, and Tianyi Yu are with the School of Information Science and Technology, Hangzhou Normal University, Hangzhou 311121, China (e-mail: 2022112011053@stu.hznu.edu.cn; yongliu@hznu.edu.cn; 2022112011011@stu.hznu.edu.cn).

Qian Meng is with the School of Mathematics, Hangzhou Normal University, Hangzhou 311121, China (e-mail: mq@hznu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2024.3391300

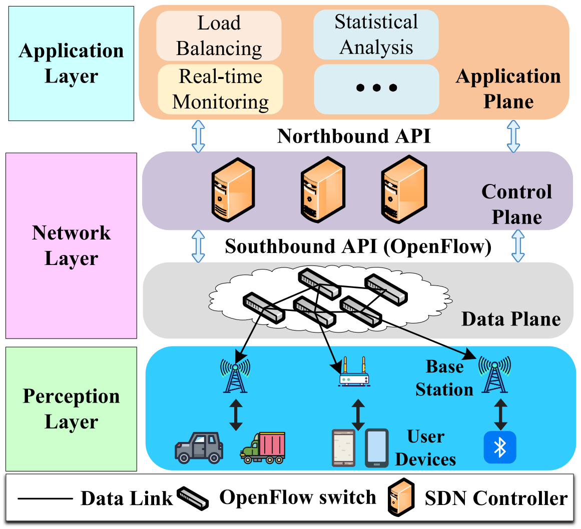

Fig. 1. SD-IoT architecture.

In the distributed SD-IoT architecture, the control plane consists of multiple physically distributed but logically centralized controllers. Compared to a single-controller architecture, a multiple-controller architecture can fulfill the network requirements of large-scale deployments, high performance, high flexibility, and robust support for stable network operations and efficient interoperability. However, the control plane often faces the challenge of load imbalances due to continuous changes in network traffic, which can not only result in wasted resources, degraded performance, and network instability but also hamper the overall interoperability of the IoT systems. Therefore, to address this challenge, researchers have proposed dynamic switch migration schemes. Its main idea is to dynamically adjust the mapping between switches and controllers in real time, with the aim of achieving load balancing at the granularity of switches. However, conventional schemes fail to fully exploit the controller load information. Additionally, relying solely on real-time load information makes it challenging to accurately predict future controller loads, leading to unnecessary switch migrations. Furthermore, existing schemes may overload the target controller after migration, failing to effectively alleviate the load imbalance and resulting in decreased load balancing performance. Moreover, traditional schemes overlook the issue of isolated nodes caused by switch migrations. This results in cross-domain communication between the migrated switch and other switch within the same control domain, which compromises network reliability and security.

To address the aforementioned issues, we propose the bidirectional switch migration based on load prediction (BSM-LP) scheme. This scheme utilizes an ATT-GRU model for predicting controller loads and preventing unnecessary switch migrations. Furthermore, a bidirectional switch migration strategy is employed to mitigate the overload issue of the target controller. Additionally, to enhance network reliability and security, we propose an isolated node integration algorithm to identify and integrate isolated nodes in the network.

In summary, the main contributions of this article are as follows.

1) In contrast to conventional schemes that depend on real-time data collection for controller load measurement, we present the controller load prediction algorithm based on the ATT-GRU model. By utilizing the temporal dependencies in historical load data, this algorithm accurately predicts the future load of controllers, thereby preventing unnecessary switch migrations.
2) Differing from the traditional unidirectional migration schemes, we propose a novel bidirectional switch migration algorithm to address the issue of overloading the target controller. Based on the improved grey wolf optimizer, this algorithm enables bidirectional switch migration among multiple controllers. It not only prevents target controller overload but also enhances migration efficiency.
3) To tackle the problem of isolated nodes caused by switch migration, we present an isolated node integration algorithm that identifies isolated nodes in the network and performs integration operations, thus improving the load balancing performance.
4) The proposed scheme has been evaluated on the Mininet platform with Ryu as the network controller, and the experimental results validate the effectiveness of the scheme. Compared to the existing schemes, it reduces the load imbalance rate by an average of 22.3% and the response time by 30.5%.

The remainder of this article is structured as follows. Section II provides an overview of the related work and the motivation behind this research. Section III describes the BSM-LP scheme in detail. The experimental evaluation is conducted in Section IV. Finally, this article is concluded in Section V.

II. Motivation and Related Work

A. Motivation

1) Problem of Overloading the Target Controller: In SD-IoT, the fluctuation of network traffic originating from IoT devices can lead to local network overload, resulting in an imbalance in load distribution among multiple controllers. Traditional switch migration approaches rely solely on unidirectional migration to achieve load balancing. However, in certain cases, migrating switches from an overloaded controller to an underloaded controller does not achieve load balancing and may even overload the target controller. Therefore, considering various types of migration approaches is necessary to achieve load balancing in the control plane. To further illustrate this issue, we introduce a specific example, as shown in Fig. 2. In this example, we assume a total capacity of 100% for each controller, with an overload threshold set at 90%. In Fig. 2(a), controller C_1 becomes overloaded while controllers C_2 and C_3 have sufficient remaining capacity to achieve load balancing. During unidirectional migration, due to the granularity of switches, migrating either switch S_1 or S_2 would lead to new overloaded controllers, as shown in Fig. 2(b). However, by employing bidirectional migration

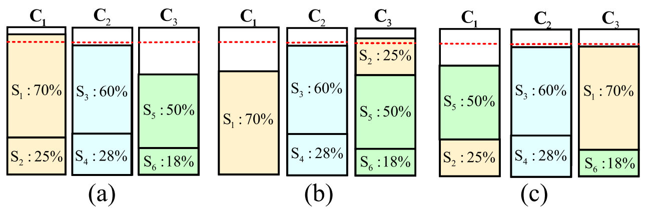

Fig. 2. Example of bidirectional migration. (a) Load is unbalanced. (b) Unidirectional migration. (c) Bidirectional migration.

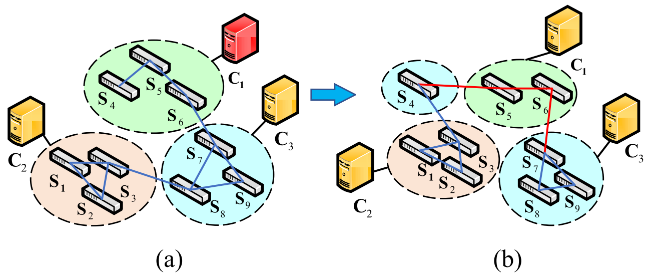

Fig. 3. Problem of isolated nodes. (a) Before switch migration. (b) After switch migration.

and migrating switch S_1 to C_3 and switch S_5 to C_1, load balancing can be successfully achieved in Fig. 2(c), ensuring that the load of all three controllers remains below the overload threshold.

2) Problem of Isolated Nodes: In conventional schemes, once a load imbalance is detected, switches are migrated from the domain network of the overloaded controller to the domain network of the underloaded controller to achieve load balancing. However, most of these schemes overlook the issue of isolated nodes arising from switch migration. When a network has an isolated node that needs to communicate with switches in the same control domain, it must traverse the domain network managed by other controllers. This results in a transformation of the original intradomain communication into interdomain communication. In Fig. 3, when controller C_1 becomes overloaded, switch S_4 is migrated to the target controller C_3 to achieve load balancing. However, when switch S_4 communicates with switches S_7–S_9, which are also managed by the same controller, it must send requests to both its master controllers C_3 and C_1 during interdomain communication. This places an additional burden on both controllers, resulting in performance degradation within the control plane. Moreover, since interdomain communication requires passing through other controllers, it also impacts network security and reliability.

B. Related Work

The centralized control and programmability of SDN enable efficient management of large-scale networks and integration with advanced technologies [6]. In work [7], researchers provide a novel SDN-based control and management framework for IoT that enables simultaneous traffic balancing among IoT servers and meets the QoS requirements of various services. Salehnia et al. [8] utilized a combination of WOA and AO algorithms to address the scheduling problem of IoT device requests in the functional cloud (FC), and they proved that the scheduling algorithm performs better with SDN.

The conventional single-controller architecture encounters challenges in complexity and flexibility when accommodating large-scale networks. Additionally, the single-controller architecture is susceptible to single points of failure and performance bottlenecks. Therefore, the concept of a multiple-controller architecture has been proposed and researched. This architecture addresses the performance deficiencies of the control plane while improving the scalability and reliability of the network. The controller placement problem was initially introduced by Heller et al. [9], who also proposed a scheme to determine the number and placement of controllers by constraining the transmission delay between the controller and the switch. Subsequently, researchers have proposed different schemes for controller placement. Saeed and Ullah [10] presented the CMOPHA scheme, which considers the path reliability for switch-to-controller and controller-to-controller communication and uses the NSGA-II algorithm to compute the optimum placement of controllers. Lange et al. [11] presented a framework for pareto-based optimal controller placement that offers optimal placements with respect to various performance metrics. Additionally, the framework provides both an exhaustive method with higher accuracy and a heuristic method with faster computational speed, catering to different network scales. Zhang et al. [12] formulated a multiobjective optimization controller placement problem and use the adaptive bacterial foraging optimization algorithm with redefined computation rules to achieve high network reliability, load balance among controllers, and low latency between controllers and switches. However, the aforementioned schemes for controller placement maintain a static mapping between controllers and switches, which is inadequate to adapt to the dynamic fluctuations in IoT traffic, leading to imbalanced loads among multiple controllers and a decrease in the performance of the control plane.

To tackle the challenge of load imbalance among multiple controllers, Dixit et al. [13] introduced the concept of switch migration strategy. The main idea of this strategy is to dynamically adjust the mapping between switches and controllers, allowing for dynamic load adjustment among multiple controllers and achieving load balancing. However, this strategy does not take into account the remaining capacity of the target controller, which can lead to the problem of target controller overload. In work [14], Cui et al. proposed a multicontroller load balancing strategy based on response time. They utilized the relationship between controller load and response time to select an appropriate response time threshold for detecting overloaded controllers. However, this approach also neglects the issue of target controller overload. Mokhtar et al. [15] proposed a switch migration scheme based on multilevel thresholds. This scheme divides the controller load into multiple levels. When the load level of a controller differs from that of other controllers, switch migration is triggered. The thresholds are dynamically adjusted to achieve continuous load balancing among distributed controllers. In work [16], Lai et al. proposed the time-sharing switch migration (TSSM) scheme. By allowing two controllers to share a switch's load sequentially in the same period, this scheme achieves load balancing in the control plane with finer-grained switch migration. In work [17], Li et al. proposed a switch migration strategy based on dynamic thresholds. This strategy dynamically adjusts the overload threshold of controllers based on real time changes in network load, enabling a balanced distribution of controller load. However, this strategy can only handle one overloaded controller at a time, resulting in lower efficiency in switch migration. Sun et al. [18] proposed a switch migration scheme based on multiagent reinforcement learning to generate switch migration actions. This scheme enables faster load balancing and improves network performance. In work [19], Sridevi and Saifulla proposed a metaheuristic approach for achieving load balancing in the control plane. Based on the artificial bee colony optimization algorithm, this approach aims to select the most suitable migration operation from multiple candidate solutions to balance the load in the long run and maintain an even distribution of the load.

In the aforementioned research, the measurement of controller load is achieved through real-time state collection. However, this approach does not effectively leverage historical load data and lacks accurate prediction of controller load. Due to the randomness and uncertainty of traffic in IoT, an overloaded controller in the current period may recover to a normal state in the next period. Depending solely on real-time load information for switch migration decisions, without considering future controller load trends, can lead to unnecessary switch migrations. Furthermore, most existing schemes adopt unidirectional switch migration, which can easily overload the target controller, thereby diminishing the efficiency of switch migration. Additionally, current schemes often overlook the issue of isolated nodes caused by switch migration, which further impacts the load balancing performance. To address these issues, we propose the BSM-LP scheme. In this scheme, we employ an ATT-GRU model to predict controller load in real time, effectively improving the accuracy of load measurement and preventing unnecessary switch migrations. Moreover, we introduce the multi-to-multi bidirectional switch migration strategy, which allows for the simultaneous handling of multiple overloaded controllers and resolves the issue of target controller overload. Furthermore, we propose an isolated node integration algorithm aimed at identifying and integrating isolated nodes to reduce their occurrence.

III. Design of BSM-LP

In this article, we assume that the SDN network consists of a set of switches $\hat{S} = \{S_1, S_2, S_3, \ldots, S_{N_S}\}$ and a set of controllers $\hat{C} = \{C_1, C_2, C_3, \ldots, C_{N_C}\}$, where N_C and N_S represent the total number of controllers and switches, respectively. Each controller $C_i \in \hat{C}$ manages a corresponding domain, where the set of switches in the domain is denoted by $\hat{S}_{C_i}$ and satisfies $\bigcup_{C_i \in \hat{C}} \hat{S}_{C_i} = \hat{S}$. In addition, for any two controllers C_i and C_j in the network, we stipulate that $\hat{S}_{C_i} \cap \hat{S}_{C_j} = \emptyset$, which indicates that the sets of switches managed by different controllers are distinct from each other. The OpenFlow protocol enables a switch to establish connections with multiple controllers. From the switch's perspective, three

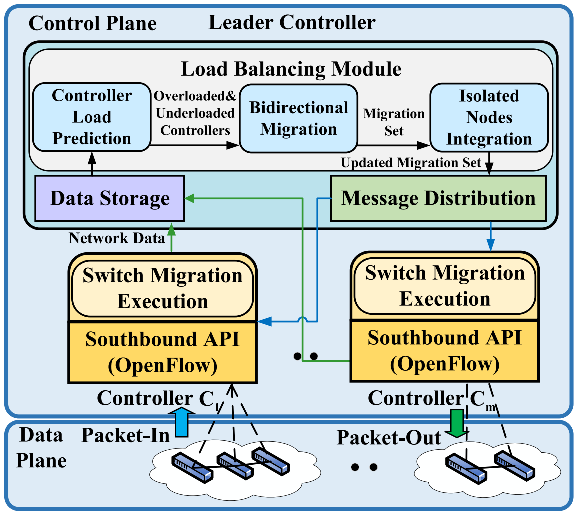

Fig. 4. Overview architecture of BSM-LP.

distinct controller roles are defined: 1) master; 2) slave; and 3) equal. The master controller is responsible for processing Packet-In messages, while the slave and equal controllers primarily serve as backup controllers [20]. Moreover, the protocol specifies that each switch can connect to multiple equal or slave controllers, but only one master controller. Therefore, for a switch, the controller with the master role can be considered the logically unique controller connected to the switch, as shown below

$$\sum_{i=1}^{N_C} r_{ki} = 1 \tag{1}$$

where r_{ki} represents the connection between controller C_i and switch S_k. If there is a connection between C_i and S_k, it is denoted as 1; otherwise, it is denoted as 0.

BSM-LP adopts a hierarchical architecture, as shown in Fig. 4. The leader controller consists of three parts: 1) the load balancing module; 2) the data storage component; and 3) the message distribution component. The load balancing module comprises three submodules: 1) the controller load prediction module; 2) the bidirectional migration module; and 3) the isolated node integration module. The data storage component is responsible for storing and maintaining network information from the lower-level controllers, while the message distribution component informs the involved controllers about the migration decisions. The controllers at the lower level have two primary functions: 1) executing switch migration and 2) handling various network requests from the SDN switches. Initially, the leader controller monitors the load status of each controller continuously to detect load imbalances. If a load imbalance is detected, it proceeds by dividing the underloaded and overloaded controllers. It then applies the multi-to-multi bidirectional migration algorithm to achieve controller load balancing. Subsequently, it executes the isolated node identification and integration operation to reduce the occurrence of isolated nodes. Moreover, while providing a detailed description of BSM-LP, we have made the following realistic assumptions.

Assumption 1: Each switch has only one master controller, while the remaining controllers can only run in an equal or slave mode.

Assumption 2: In a given time interval, all the controllers cannot be overloaded simultaneously.

A. Load Prediction

Traditional approaches typically employ periodic data collection to evaluate the load on the controllers. However, in SD-IoT, the load on controllers tends to fluctuate with variations in network traffic, and the currently overloaded controller may return to a normal load state in the subsequent period. Therefore, relying solely on real-time data for controller load measurement can result in misclassification of overloaded controllers and trigger unnecessary switch migrations. Therefore, we propose a controller load prediction algorithm based on the ATT-GRU model, which leverages historical data to predict the controller load and accurately classify overloaded controllers. Existing research indicates that the controller load primarily originates from the processing of Packet-In messages [21]. Hence, we use the total number of Packet-In messages processed by C_i within a period t to represent the load of controller C_i, denoted as $L(C_i, t)$. It can be calculated using the following equation:

$$L(C_i, t) = \sum_{m=1}^{N_s} L(S_m, t) \times r_{m,i} \tag{2}$$

where $L(S_m, t)$ represents the total number of Packet-In messages generated by switch S_m within period t. We denote Capacity_{C_i} as the total capacity of controller C_i, which represents the maximum number of Packet-In messages that controller C_i can handle in a period. Additionally, we use δ_i to represent the overload threshold of the controller, satisfying $\text{Capacity}_{C_i} > \delta_i$. When the current controller load exceeds the predefined overload threshold, we also predict the controller load for the next period using the ATT-GRU model. Only when the predicted load also exceeds the threshold will the subsequent switch migration decisions be triggered.

In this article, we use the improved gated recurrent unit (GRU) model based on the attention mechanism for load prediction. As an enhancement of recurrent neural networks (RNNs), GRU [22] accurately captures temporal dependencies using gate mechanisms and resolves the problem of vanishing and exploding gradients in RNNs. In addition, compared to the long short-term memory (LSTM) [23] model, the GRU model has a simpler structure, requires fewer training parameters, and exhibits higher training efficiency. Therefore, we utilize the GRU model to capture temporal dependencies. The model structure is shown in Fig. 5, and the calculation process for prediction is shown as follows:

$$u_t = \sigma\big(W_u \times [X_t, h_{t-1}] + b_u\big) \tag{3}$$
$$z_t = \sigma\big(W_z \times [X_t, h_{t-1}] + b_z\big) \tag{4}$$
$$c_t = \tanh\big(W_c \times [X_t, (z_t \times h_{t-1})] + b_c\big) \tag{5}$$
$$h_t = u_t \times h_{t-1} + (1 - u_t) \times c_t \tag{6}$$

where h_{t-1} denotes the output at period $t-1$ and h_t denotes the output at period t. u_t and z_t represent the update gate and

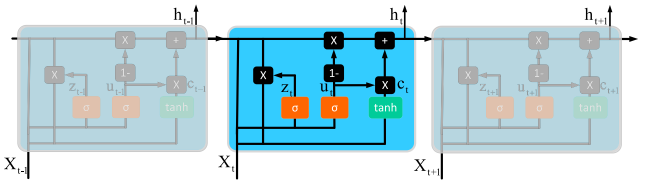

Fig. 5. Architecture of the GRU model.

reset gate at period t. c_t is the content of the memory cell at the period t. Based on this calculation process, GRU effectively captures the current load information while retaining the historical load trends.

With the sequential processing of time steps, the GRU model can capture temporal dependencies effectively. However, relying solely on temporal order is insufficient to accurately distinguish the importance of different time points. Therefore, a mechanism that learns global correlations is needed to enhance model performance. The attention model is realized on the basis of the encoder–decoder model and was initially developed for neural machine translation tasks. Nowadays, attention models have been widely applied in various domains, such as image caption generation [24] and recommendation system [25]. With the rapid development of such models, different types of attention mechanisms have been proposed, including soft attention and hard attention [26], global and local attention [27], and self-attention [28]. In our work, we adopt the soft attention mechanism to learn the importance of load information at each time step and calculate a context vector to express the global variation trends of the load state.

Suppose that we introduce a time series $x_i (i = 1, 2, \ldots, n)$, where n represents the length of the time series. The design process for the soft attention model is described as follows. First, the hidden states $h_i (i = 1, 2, \ldots, n)$ at different time steps are calculated using RNNs (and their variants) and represented as $H = (h_1, h_2, \ldots, h_n)$. Second, a scoring function to calculate the score/weight for each hidden state is designed. Then, an attention function to compute a context vector C_t is also designed to describe global load variation information. Finally, the final output result is obtained by using the context vector. Specifically, the features h_i at each time step were utilized as input when computing the weight for each hidden state, and the corresponding outputs can be obtained through two hidden layers. A Softmax normalized index function defined in (8) is used to determine the weights of each characteristic (α_i). The weights and biases of the first layer are denoted by $w_{(1)}$ and $b_{(1)}$, and the weights and biases of the second layer are denoted by $w_{(2)}$ and $b_{(2)}$, respectively

$$e_i = w_{(2)}\big(w_{(1)}H + b_{(1)}\big) + b_{(2)} \tag{7}$$
$$\alpha_i = \frac{\exp(e_i)}{\sum_{k=1}^{n} \exp(e_k)}. \tag{8}$$

Finally, we designed the attention function, and the calculation process of the context vector C_t that covers global load

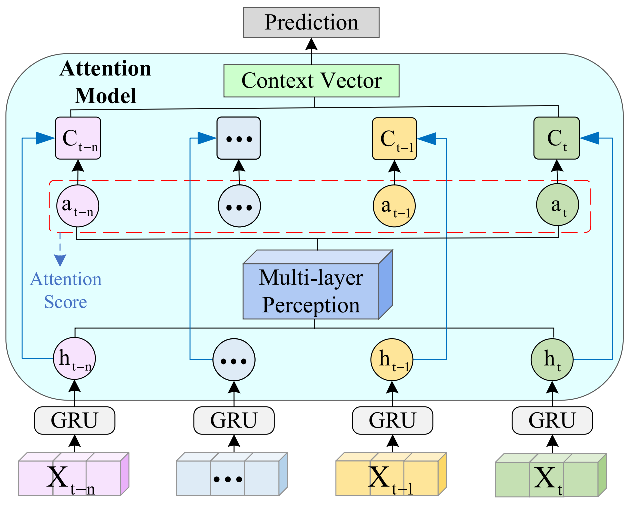

Fig. 6. Architecture of the ATT-GRU model.

variation information is shown below

$$C_t = \sum_{i=1}^{n} \alpha_i \times h_i. \tag{9}$$

The structure of the ATT-GRU model is illustrated in Fig. 6. By inputting n historical time series load data into the ATT-GRU model, the hidden states h covering the temporal characteristics are obtained: $\{h_{t-n}, \ldots, h_{t-1}, h_t\}$. Subsequently, the hidden states are fed into the attention model to determine a context vector that captures the global load variation information. Specifically, a multilayer perceptron structure and a Softmax function are used to calculate the weights for each hidden state h: $\{a_{t-n}, \ldots, a_{t-1}, a_t\}$. The context vector, representing the weighted sum of the hidden states, is then computed to capture the global load variation information. Finally, the prediction results are obtained through a fully connected layer. Furthermore, in model training, we aim to minimize errors between the real and predicted number of Packet-In messages in the SD-IoT network. Therefore, the objective function of ATT-GRU is shown below

$$\text{loss} = \left\| Y_t - \hat{Y}_t \right\| + \lambda L_{\text{reg}} \tag{10}$$

where the first term is to minimize the real and predicted number of Packet-In messages. Y and $\hat{Y}$ denote the real and predicted numbers of Packet-In messages sent from different switches at period t, respectively. In the second term, L_{reg} is a normalization term, and λ denotes a hyperparameter.

The specific execution process of the controller load prediction algorithm is illustrated in Algorithm 1. First, the load data is periodically obtained, and based on the set of switches $\hat{S}_{C_i}$ managed by each controller, the load of each controller is calculated using (2), as shown from steps 3 and 4. Then, the controller load is compared to the overload threshold. If the load is greater than or equal to the threshold, the algorithm retrieves the real-time prediction results from the ATT-GRU model and calculates the controller load for the next time period, as shown from steps 5 to 7. If $L(C_i, t+1) \geq \sigma_i$, it indicates that the controller will still be in an overloaded

Algorithm 1 Load Prediction

```
Input: Controller set $\hat{C}$, Switch set $\hat{S}$, Feature matrix $X_t$;
Output: Overloaded controller set $\hat{C}_O$, Underloaded controller set $\hat{C}_U$;
1:  Initialization: $\hat{C}_O \leftarrow \emptyset$, $\hat{C}_U \leftarrow \emptyset$;
2:  for $C_i \in \hat{C}$ do
3:      Fetch $\hat{S}_{C_i}$;
4:      Calculate $L(C_i, t)$ according to Eq. (2);
5:      if $L(C_i, t) \geq \sigma_i$ then
6:          $X_{t+1} \leftarrow ATT\text{-}GRU(X_t)$;
7:          Calculate $L(C_i, t+1)$ according to Eq. (2);
8:          if $L(C_i, t+1) \geq \sigma_i$ then
9:              $\hat{C}_O \leftarrow C_i$;
10:         else
11:             $\hat{C}_U \leftarrow C_i$;
12:         end if
13:     else
14:         $\hat{C}_U \leftarrow C_i$;
15:     end if
16: end for
17: return $\hat{C}_O$, $\hat{C}_U$.
```

state in the next time period. Hence, it is classified as an overloaded controller and added to the overloaded controller set $\hat{C}_O$. Otherwise, the controller is added to the underloaded controller set $\hat{C}_U$.

B. Multi-to-Multi Bidirectional Migration

When a load imbalance is detected, it is imperative to promptly make the corresponding load balancing decisions. Existing schemes rely on migrating switches from overloaded controllers to underloaded controllers to achieve load balancing. However, the traditional unidirectional migration approach can overload the target controller due to granularity issues in switch migration. Therefore, we propose a bidirectional switch migration strategy. Based on the improved grey wolf algorithm, this strategy can not only enable the simultaneous handling of multiple overloaded controllers but also facilitate bidirectional migration between overloaded and underloaded controllers, which enhances both flexibility and efficiency.

1) Overview of Grey Wolf Optimizer: Grey wolf optimization (GWO) is a population-based evolutionary algorithm inspired by the social hierarchy and hunting mechanisms of grey wolves in nature. It offers advantages, such as simplicity in operation, minimal parameter settings, and fast convergence speed [29]. Grey wolves are social animals that typically form packs of a dozen individuals and establish a strict pyramid-like hierarchical structure. In a wolf pack, the alpha wolf is the leader positioned at the top of the hierarchy. It is responsible for hunting and decision making, with other wolves required to follow its commands. The beta wolf occupies the second layer and assists the alpha wolf in decision making. It has control over the remainder of the pack and provides feedback based on information from other wolves. The delta wolf is in the third layer and executes the decisions made by the alpha and beta wolves. It holds a higher position than the omega wolf. The omega wolf, at the bottom layer, assists in hunting prey and supports the hunting action.

To mathematically model the hunting behavior of grey wolves, we start by randomly generating a set of grey wolves within the search space. Next, we estimate the prey's position using the positions of the alpha, beta, and delta wolves. The remaining wolves calculate their distances to the alpha, beta, and delta wolves and move closer to the prey, gradually surrounding it. Finally, the wolves capture the prey. The specific modeling steps are as follows, and we represent the positions of the prey and the grey wolves in the t^{th} iteration using position vectors $\vec{X}_p^t$ and $\vec{X}^t$, respectively. The description of the encircling mechanism in the $(t+1)$th iteration is as follows:

$$\vec{X}^{t+1} = \vec{X}_p^t + \vec{A}\cdot\vec{D} \tag{11}$$

$$\vec{D} = \left|\vec{C}\cdot\vec{X}_p^t - \vec{X}^t\right| \tag{12}$$

$$\vec{A} = 2a\cdot\vec{r_1} - a \tag{13}$$

$$\vec{C} = 2\cdot\vec{r_2} \tag{14}$$

where t represents the current iteration number, $\vec{A}$ and $\vec{D}$ are coefficient vectors, and $\vec{r_1}$ and $\vec{r_2}$ are randomly generated vectors within the range [0, 1]. a represents the exploration rate, which linearly decreases from 2 to 0 during the iteration process. Grey wolves possess the ability to identify the position of their prey and engage in encircling behavior. The alpha wolf typically leads the hunting process, occasionally assisted by the beta and delta wolves. However, in an abstract search space, the exact location of the optimal solution (i.e., the prey) cannot be determined. To simulate the hunting behavior of grey wolves, we assume that the alpha, beta, and delta wolves possess superior tracking abilities. Thus, we keep track of the three best solutions obtained and require other search agents (including the omega wolf) to update their positions based on the positions of the three best solutions, denoted as $\vec{X}_\alpha$, $\vec{X}_\beta$, and $\vec{X}_\delta$. The specific description of this approach is as follows:

$$\vec{D}_\alpha = \left|\vec{C}_1\cdot\vec{X}_\alpha - \vec{X}^t\right| \tag{15}$$

$$\vec{D}_\beta = \left|\vec{C}_2\cdot\vec{X}_\beta - \vec{X}^t\right| \tag{16}$$

$$\vec{D}_\delta = \left|\vec{C}_3\cdot\vec{X}_\delta - \vec{X}^t\right| \tag{17}$$

$$\vec{X}_1 = \vec{X}_\alpha + \vec{A}_1\cdot\vec{D}_\alpha \tag{18}$$

$$\vec{X}_2 = \vec{X}_\beta + \vec{A}_2\cdot\vec{D}_\beta \tag{19}$$

$$\vec{X}_3 = \vec{X}_\delta + \vec{A}_3\cdot\vec{D}_\delta \tag{20}$$

$$\vec{X}^{t+1} = \frac{\vec{X}_1+\vec{X}_2+\vec{X}_3}{3} \tag{21}$$

where $\vec{A}_1$, $\vec{A}_2$, and $\vec{A}_3$ are similar to $\vec{A}$, $\vec{C}_1$, $\vec{C}_2$, and $\vec{C}_3$ are similar to $\vec{C}$, as defined in (13) and (14).

2) Improved Grey Wolf Optimizer: The standard GWO algorithm is specifically designed for solving continuous optimization problems. However, the switch migration problem is a problem of combinatorial optimization, making the direct application of the standard GWO algorithm unsuitable. Therefore, we have proposed improvements to the standard GWO algorithm. By incorporating a hybrid integer encoding method and a novel wolf position update method, we adapt the algorithm to tackle the bidirectional switch migration problem. Next, we present the employed encoding method and the fitness evaluation approach. Finally, we present the enhanced mechanism for wolf position update and conclude with the pseudocode for the proposed algorithm.

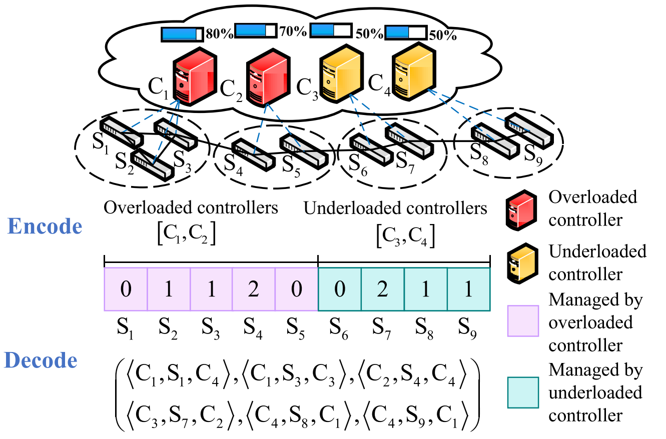
80%
70%
50%
50%
C_1
C_2
C_3
C_4
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
S_9
Encode
Overloaded controllers
$[C_1,C_2]$
Underloaded controllers
$[C_3,C_4]$
0 1 1 2 0 0 2 1 1
Decode
$\langle C_1,S_1,C_4\rangle,\langle C_1,S_3,C_3\rangle,\langle C_2,S_4,C_4\rangle$
$\langle C_3,S_7,C_2\rangle,\langle C_4,S_8,C_1\rangle,\langle C_4,S_9,C_1\rangle$
Overloaded controller
Underloaded controller
Managed by overloaded controller
Managed by underloaded controller

Fig. 7. Example of the hybrid integer encoding method.

a) Solution encoding and fitness evaluation: In the context of the bidirectional switch migration problem, this article adopts a two-segment hybrid coding method to encode each solution. Our objective is to achieve load balancing through bidirectional switch migration between overloaded and underloaded controllers. Therefore, each switch under the overloaded and underloaded controllers is associated with an integer number, and the solution is encoded as an integer string to represent the position of a wolf in the pack. Fig. 7 presents an example of the encoding method. The example consists of four controllers, with two controllers being overloaded and two being underloaded. A possible solution can be encoded as an integer string: {0, 1, 1, 2, 0, 0, 2, 1, 1}. Each integer on the left portion of the string represents a switch managed by an overloaded controller, and each switch can take on three values: 0, 1, or 2. 0 indicates that the current connection should be preserved, 1 indicates migration to the first underloaded controller C_3, and 2 indicates migration to the second underloaded controller C_4. Each integer on the right portion of the string represents a switch managed by underloaded controllers, with values of 0, 1, or 2. 0 indicates no migration, while 1 and 2 correspond to the positions of the overloaded controller set. Furthermore, to evaluate the quality of the current solution, a fitness function is designed, where a smaller function value indicates a better solution, and the function is defined as follows:

$$F(X^t) = P(X^t)\times\left(w_1\times\frac{mc(X^t)}{N_s} + w_2\times \mathrm{LIR}(X^t)\right) \tag{22}$$

where w_1 and w_2 are the weight coefficients of migration cost and load imbalance rate, with the sum of weights equaling 1. $P(X^t)$ represents the penalty function, which takes a value of 1 when the load of all controllers is below the overload threshold and a value of 10 otherwise. $mc(X^t)$ represents the migration cost, denoted by the number of migrated switches.

LIR represents the load imbalance rate, which is used to measure the degree of load imbalance among controllers and is defined as follows:

$$\mathrm{LIR} = \frac{\sum_{i=1}^{M} \left|\mathrm{Load}_{C_i} - \bar{L}\right|}{M \cdot \bar{L}} \tag{23}$$

where M represents the total number of underloaded and overloaded controllers, and $\bar{L}$ represents the average load of the controllers.

b) Population initialization and position update: In this approach, the reverse learning strategy with elite retention [30] is designed to initialize the population. The optimization capability and final solution quality of population-based intelligence optimization algorithms are directly influenced by the quality of the initial population. By enhancing the diversity of the initial population, the algorithm's optimization ability can be strengthened. However, the basic GWO algorithm utilizes a random strategy for population initialization, which fails to ensure diversity and consequently impacts its performance. Therefore, we employ a reverse learning strategy with elite retention for population initialization. Specifically, the primary steps of this strategy are as follows.

1) Randomly initialize the positions of N grey wolf individuals X within the search space as the initial population P_1, where N represents the number of individuals in the population.
2) Generate the corresponding reverse individual X' for each grey wolf individual X in P_1 and add to the reverse population P_2.
3) Merge the initial population P_1 and the reverse population P_2. Sort the merged $2 \times N$ grey wolf individuals based on their fitness function values. Retain the top N individuals with the best fitness as the final initial population P.

To adapt to the switch migration problem, the solution update strategy is also modified. By combining the encoding strategy mentioned above, we have introduced crossover (CR) and mutation (MU) operations from genetic algorithm [31] to update the positions of grey wolf individuals. Specifically, for each grey wolf individual, we select alpha, beta, and delta wolves with equal probabilities for crossover and mutation operations, which are shown as follows:

$$X^{t+1} = \begin{cases} CR\&MU\left[X^t, X_\alpha\right], & 0 \le r < \frac{1}{3} \\ CR\&MU\left[X^t, X_\beta\right], & \frac{1}{3} \le r < \frac{2}{3} \\ CR\&MU\left[X^t, X_\delta\right], & \frac{2}{3} \le r < 1. \end{cases} \tag{24}$$

First, we randomly select a wolf from alpha, beta, and delta with equal probabilities. Then, a crossover point is randomly selected for each solution, and the crossover operation is performed between the two solutions at the selected point. After the crossover, the mutation operation is carried out using the random resetting method. This method randomly modifies several elements in the solution by replacing them with permissible values within the specified range.

The specific execution process of the bidirectional switch migration algorithm is shown in Algorithm 2. The inputs to this algorithm include the set of overloaded and underloaded controllers obtained from Algorithm 1. Along with the set of switches, these two sets serve as the basis for encoding. The algorithm also requires parameters, including the maximum number of iterations and the population size. The first step is the initialization of the population, which involves employing the reverse learning strategy with elite retention as outlined from steps 2 to 5. After initialization, the three best individuals, X_α, X_β, and X_δ, are identified based on their fitness function value, as described in step 6. Subsequently, we proceed with the iterative search for the optimal solution, where the population individuals update their positions using the (24). After the position update, the fitness of each gray wolf individual is recalculated, and three best solutions are updated accordingly, as described from steps 7 to 14. At the maximum number of iterations, we assess if the optimal solution achieves load balancing in step 15. When there is a controller with a load exceeding the threshold, the function returns 0; otherwise, it returns 1. Finally, the algorithm returns the optimal solution, and the binary variable *LB*, which represents whether the optimal solution achieves load balancing or not.

Algorithm 2 Multi-to-Multi Bidirectional Switch Migration

```
Input: Overloaded controller set $\hat{C}_O$, Underloaded controller
    set $\hat{C}_U$, Switch set $\hat{S}$, Pupolation size $N$, Max number of
    iteration $Max\_Iter$;
Output: Optimal solution $X_\alpha$, Load balancing flag $LB$;
 1: Initialization: $n = 0, iter = 0, P_1 = \emptyset, P_2 = \emptyset, X_\alpha =$
    $\emptyset, X_\beta = \emptyset, X_\delta = \emptyset$;
 2: Randomly initialize the grey wolf population $P_1$;
 3: Generate the reversed population $P_2$ based on $P_1$;
 4: $P' = sort(P_1 + P_2)$;
 5: Select the top $N$ solutions from $P'$ and construct popula-
    tion $P$;
 6: Select the best three solutions as $X_\alpha, X_\beta, X_\delta$;
 7: while $iter < Max\_Iter$ do
 8:     for $X^t \in P$ do
 9:         Update $X^t$ using Eq. (24);
10:     end for
11:     Calculate the fitness of all solutions;
12:     Update $X_\alpha, X_\beta, X_\delta$;
13:     $iter = iter + 1$;
14: end while
15: $LB = balanced(X_\alpha)$;
16: return $X_\alpha, LB$.
```

C. Isolated Nodes Integration

Switch migration for control plane load balancing may lead to the presence of isolated nodes in the network. As shown in Fig. 8, when controller C_1 becomes overloaded, migrating switch S_4 to controller C_3 can achieve load balancing, but S_4 becomes an isolated node. When S_4 communicates with switches S_7–S_9 in the same control domain, it needs to pass through the domain network managed by controller C_1, resulting in cross-domain communication, as shown in Fig. 8(a). To address the issue of isolated nodes caused by switch migration, we propose the isolated node integration algorithm. Once the bidirectional migration process is finished,

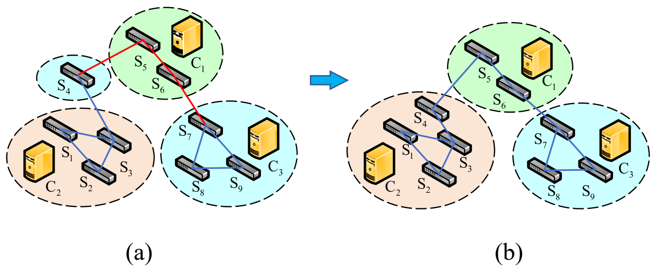

Fig. 8. Example of isolated node integration. (a) Before isolated node integration. (b) After isolated node integration.

the proposed algorithm proceeds to identify isolated nodes within the network. Subsequently, it initiates switch migration to transfer the isolated switches to the master controllers of adjacent nodes, effectively reducing their occurrence. As shown in Fig. 8(b), we perform integration for the isolated node S_4 and migrate it to the domain network controlled by C_2. This ensures the connectivity of all switches in the same control domain. In this case, we abstract the network topology as an undirected graph, denoted as $G = (\hat{S}, E)$, where $\hat{S}$ represents the set of switches, and each switch in the set also represents a node in the network. E represents the set of edges, representing the links between switches. Each switch S_k has its corresponding master controller, denoted as $C(S_k)$. A switch S_k is identified as an isolated node if its master controller $C(S_k)$ differs from the master controllers of all its adjacent switches, denoted as Neighbor(S_k). The mathematical representation of this definition is as follows:

$$\sum_{S_m \in \text{Neighbor}(S_k)} C(S_m) \cap C(S_k) = \emptyset. \tag{25}$$

During the process of node integration, in order to minimize the impact on controller load balancing, we first prioritize selecting the isolated node with the smallest sending volume of Packet-In messages for integration. Moreover, when selecting the appropriate controller for the isolated node, we prioritize the master controllers of its adjacent nodes based on their remaining capacity. Additionally, to avoid the controller overload, we define the following constraint:

$$L(S_k, t+1) + L(C_T, t+1) < \sigma_T \tag{26}$$

where σ_T represents the overload threshold of controller C_T, if the remaining capacity of controller C_T is insufficient, we refrain from executing the integration operation to prevent the occurrence of load imbalances.

The overall process of the isolated node integration algorithm is presented in Algorithm 3. The algorithm accepts two inputs: 1) the network topology G and 2) the migration set MS, obtained from decoding the optimal solution of bidirectional migration. First, we traverse all switch nodes in the network and identify all isolated nodes based on the mapping relationship between the switches and the controllers. Subsequently, we sort these isolated nodes in ascending order based on the number of Packet-In messages sent from each node and then select the first isolated node S_k for integration, as described from steps 11 and 12. To select the target controller for integration, we sort the controllers and choose the one with the lightest load, denoted as C_m. Next, we evaluate whether the integration operation would cause controller load imbalance. If C_m has enough remaining capacity, we generate a migration triplet and update the migration set MS, as shown from steps 13 to 17.

Algorithm 3 Isolated Nodes Integration

```
Input: Network topology $G$, Migration set $MS$;
Output: Migration set $MS$;
 1: Initialization: $iso\_list = \emptyset$, $con\_list = \emptyset$, $\hat{S} \leftarrow G$;
 2: for $S_i \in \hat{S}$ do
 3:     $C_i = master(S_i)$;
 4:     for $S_j \in neighbors(S_i)$ do
 5:         $con\_list[i].add(master(s_j))$;
 6:     end for
 7:     if $C_i \notin con\_list[i]$ then
 8:         $iso\_list.add(S_i)$;
 9:     end if
10: end for
11: Sort($iso\_list$);
12: Pick the first switch $S_k$ from $iso\_list$;
13: Sort($con\_list[k]$);
14: Pick the first controller $C_m$ from $con\_list[k]$;
15: if $L(S_k, t+1) + L(C_m, t+1) < \sigma_m$ then
16:     MS.update($\langle C_k, S_k, C_m \rangle$);
17: end if
18: return $MC$.
```

D. Switch Migration Execution

The migration execution algorithm coordinates the load prediction module, bidirectional migration module, and isolated node integration module. When detecting a load imbalance among multiple controllers, the algorithm utilizes the load prediction module to identify overloaded and underloaded controllers. It calls the multi-to-multi bidirectional migration algorithm to search for an optimal solution, which is then decoded to generate a migration set. The algorithm subsequently employs the isolated node integration algorithm to reduce the occurrence of isolated nodes and updates the migration set accordingly. Finally, switch migration operations are performed based on the updated migration set.

The specific procedure of the switch migration execution algorithm is depicted in Algorithm 4. The algorithm takes inputs, including the set of overloaded controllers $\hat{C}_O$, the set of underloaded controllers $\hat{C}_U$, the set of switches $\hat{S}$, and the network topology G. Switch migration will be carried out only when $\hat{C}_O$ is nonempty, and this condition is checked in step 2. If so, the bidirectional migration algorithm is called to search for the optimal solution within a limited number of iterations. To determine the feasibility of this solution, the return value LB is used to evaluate whether the current solution can achieve load balancing. If LB equals 1, it indicates that the solution can achieve load balancing, and the optimal solution is decoded to obtain the migration set MS. If LB equals 0, the solution is abandoned. Subsequently, the isolated node

Algorithm 4 Switch Migration Execution

```
Input: Overloaded controller set $\hat{C}_O$, Underloaded controller set $\hat{C}_U$, Switch set $\hat{S}$, Pupolation size $N$, Max number of iteration $Max\_Iter$, Network topology $G$;
Output: Migration set $MS$;
1: Initialization: $MS = \emptyset$, $opt\_sol = null$;
2: if $\hat{C}_O \neq \emptyset$ then
3:     $opt\_sol, LB \leftarrow$ Algo.2($\hat{C}_O, \hat{C}_U, \hat{S}, N, Max\_Iter$);
4:     if $LB == 1$ then
5:         $MS \leftarrow decode(opt\_sol)$;
6:     end if
7:     $MS \leftarrow$ Algo.3($G, MS$);
8: end if
9: return $MS$.
```

TABLE I
TOPOLOGY FEATURES

Feature \ Topology	Abilene	Noel	Janos-us	Pioro
Number of Links	15	25	42	89
Number of Nodes	12	19	26	40

integration algorithm is called to identify and make integration decisions for isolated nodes. Finally, the migration set *MS* is updated and returned.

IV. PERFORMANCE EVALUATION

The BSM-LP is implemented using the Ryu controller [32] and the Mininet simulation platform [33]. Ryu is an open-source SDN/OpenFlow controller written in Python that supports the OpenFlow v1.3 protocol. Mininet is a lightweight process virtualization simulation tool that allows users to quickly build large-scale SDN prototype systems on resource-constrained devices. In this experiment, our experimental device is powered by an AMD Ryzen 5 5600H CPU with Radeon graphics, and it has a memory capacity of 16.0 GB. Additionally, in this experiment, we deploy four controllers in the control plane, and each controller has a capacity (i.e., Capacity_{C_i}) of 2500 Packet-In per second, and the threshold δ_i is set to 0.6 Capacity_{C_i} [16], which means if a controller has to handle more than 1500 Packet-In per second, it will be denoted as the overloaded controller. In addition, we selected four topologies with increasing scales [34], [35] for experimentation to validate the adaptability of BSM-LP under different topology. The characteristics of these topologies are shown in Table I.

A. Performance Metrics and Comparison Schemes

In the experiment, we selected four metrics for performance evaluation.

1) *Response Time:* The average response time of the controller significantly increases when there is a load imbalance. As response time also reflects the quality of network services, we adopt it as an evaluation metric.
2) *Load Imbalance Rate:* The load imbalance rate is the most effective metric for measuring switch migration performance. A lower load imbalance rate indicates a more evenly distributed load and better load balancing performance.
3) *Migration Cost:* Switch migration can disrupt network services and affect network stability. Thus, migration cost is considered as an evaluation metric.
4) *Number of Occurrences of Overload (Below, It Is Called NOO for Short):* Improper switch migration can lead to an increase in NOO, leading to frequent switch migration operations and reduced efficiency. Therefore, NOO is considered as a metric.

Furthermore, we have implemented the following five schemes on Ryu for comparative evaluation.

1) *Static Mapping Between Controller and Switch (SMCS):* The mapping relationship between switches and controllers remains static, and no switch migration mechanism is employed in the experiment.
2) *Condition-Aware Switch Migration (CASM) [36]:* This scheme prioritizes the controller with the heaviest load and transfers the switch with the highest rate of Packet-In messages to the controller with the lightest load.
3) *TSSM [16]:* This scheme allows two controllers to share a switch's load sequentially in the same period and achieves load balancing with finer-grained switch migration.
4) *Switches Group-Based Load Balancing (SGLB) [37]:* This scheme selects a group of switches managed by overloaded controllers for migration while considering the remaining capacity of the target controller.
5) *Optimum Greedy-Based Switch Migration (OptiGSM) [38]:* The scheme dynamically partitions overloaded controllers based on average load and migrates switches to controllers with lighter loads using an optimum greedy strategy.

B. Load Prediction Model Evaluation

As a supervised learning model, ATT-GRU relies on a substantial amount of training data to learn temporal features before making predictions. In this study, we adopted the methodology outlined in [39] to obtain the required training and testing data sets, which involves generating network traffic by replaying the pcap file and conducting data collection. Using the Mininet tool, we simulated four network topologies: 1) Abilene; 2) Noel; 3) Janos-us; and 4) Pioro. In these topologies, each switch is connected to a virtual host, and MAC and IP addresses are assigned accordingly. For our experiment, we use the pcap files from the MAWI Working Group [40] in Japan, which is dedicated to network measurement and analysis. These pcap files contain anonymized global Internet traffic data, providing valuable insights into the performance, characteristics, and evolution of the Internet. Specifically, we selected a specific time interval (January 1, 2022, from 14:00 to 14:15) [41] and employed the tcpreplay tool [42] to replay the captured traffic. During traffic replay, we collect the number of Packet-In messages sent from each switch to the controller at fixed time intervals. This data is then processed and saved as CSV files, which serve as the training and testing data sets. We normalize the input data to the interval [0, 1], and the training set comprises 80% of the data, while the remaining 20% is allocated for testing. Moreover, to evaluate the prediction performance of the model, we select the following four evaluation metrics to measure the error between the real load information and the predicted values:

1) To quantify the prediction errors, Root Mean-Squared Error (RMSE) is calculated as follows:

$$\mathrm{RMSE} = \sqrt{\frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N}\left(y_i^j - \hat{y}_i^j\right)^2}. \tag{27}$$

2) To estimate the mean deviation, Mean Absolute Error (MAE) is calculated as follows:

$$\mathrm{MAE} = \frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N}\left|y_i^j - \hat{y}_i^j\right|. \tag{28}$$

3) To assess the effectiveness of the predictions, Accuracy is calculated below:

$$\mathrm{Accuracy} = 1 - \frac{\left\|Y - \hat{Y}\right\|_F}{\|Y\|_F}. \tag{29}$$

4) To express the prediction error as a percentage, Mean Absolute Percentage Error (MAPE) is calculated below:

$$\mathrm{MAPE} = \frac{1}{MN}\sum_{j=1}^{M}\sum_{i=1}^{N}\left|\frac{y_i^j - \hat{y}_i^j}{y_i^j}\right| \tag{30}$$

where y_i^j and $\hat{y}_i^j$ are the real and predicted number of Packet-In messages sent from switch i, respectively. N is the number of switches in the network and M is the number of temporal samples. Y and $\hat{Y}$ represent the sets of y_i^j and $\hat{y}_i^j$, respectively.

In addition, we compared the performance of our model with the following three methods.

1) *GRU Model [22]:* As a variant of RNN, GRU's gating mechanisms excel at capturing long-term dependencies and yielding great predictive results, making it a suitable baseline model for comparison.
2) *Support Vector Regression Model (SVR) [43]:* Being a traditional method, it effectively captures nonlinear patterns in the data through kernel functions and achieves accurate predictions, making it a selected baseline method.
3) *Temporal Graph Convolutional Network (TGCN) [44]:* As a hybrid model, it uses the spatiotemporal correlation within historical data to achieve accurate predictions. Thus, it is chosen as one of the baseline methods.

We performed traffic replay and generated corresponding data sets in different network topologies. Table II displays the comparison results of the ATT-GRU model with other models in terms of prediction error and accuracy on different data sets. The results indicate that the ATT-GRU model achieves better prediction performance under different evaluation metrics. Specifically, the ATT-GRU model achieves an average reduction of approximately 41.2%, 45.3%, and 48.5% in RMSE, MAE, and MAPE, respectively, compared to the three baseline methods, while also improving accuracy by an average of approximately 7.8%.

TABLE II
PREDICTION RESULTS OF THE ATT-GRU MODEL AND OTHER BASELINE METHODS

Topo	Metric	Methods			
		ATT-GRU	GRU	SVR	TGCN
Abilene	*RMSE*	**136.3856**	136.4012	279.2049	295.0306
	MAE	**75.6897**	77.4945	145.7801	173.6784
	MAPE	**0.1233**	0.1485	0.2741	0.3392
	Accuracy	**0.9152**	0.9142	0.8266	0.8167
Noel	*RMSE*	**64.7632**	80.9855	104.7128	211.1877
	MAE	**35.1553**	35.4923	49.4124	119.9271
	MAPE	**0.1271**	0.1469	0.1728	0.5055
	Accuracy	**0.9368**	0.9358	0.9183	0.8352
Janos-us	*RMSE*	**58.0282**	59.4471	70.6229	182.6338
	MAE	29.0333	**27.9151**	37.0988	102.5442
	MAPE	**0.1155**	0.11751	0.1404	0.4799
	Accuracy	**0.9361**	0.9352	0.9222	0.799
Pioro	*RMSE*	**39.0142**	41.5013	43.7939	185.2619
	MAE	**17.8977**	33.3678	19.5556	89.8495
	MAPE	0.2162	0.2522	**0.2115**	0.8434
	Accuracy	**0.9349**	0.919	0.9269	0.6909

C. Performance Evaluation

In the evaluation, we conduct the effectiveness test and the performance test separately. First, we recorded load changes of each controller under the BSM-LP and SMCS schemes during load imbalances to validate the effectiveness of our scheme. To further demonstrate the load balancing performance of the BSM-LP scheme, we conducted comparative tests under various topologies. In these tests, we simulated real-world network traffic by replaying the pcap files mentioned above. By default, the tcpreplay tool replays the traffic at the original recorded speed [45]. However, the tcpreplay tool allows users to adjust the replay speed using the multiplier option [46]. By adjusting the multiplier values, we can control the replaying speed to simulate various traffic load conditions without exceeding the controller's capacity. In our tests, we select four different multiplier values to simulate different traffic load conditions and evaluate the load balancing performance of each scheme under different network states. Finally, we also conducted the experiments to validate the scalability of the BSM-LP scheme.

1) Effectiveness of BSM-LP: To validate the effectiveness, we conducted a simulation test on the Abilene topology for 300 s with a sampling interval of 5 s. We compared the BSM-LP scheme with the SMCS scheme. In the SMCS scheme, we increased the load on controller C_4 starting from the 40th second, which significantly surpassed the load on the other three controllers and exceeded the threshold, as shown in Fig. 9(a). Due to the static controller-switch connections in SMCS, the load imbalance persisted throughout the experiment. In contrast, Fig. 9(b) shows the dynamic controller load adaptation in the BSM-LP scheme. As the load on controller C_4 increases, our proposed scheme efficiently detects the overloaded controller and implements switch migration strategies to offload a portion of the load from controller C_4 to the underloaded controller C_3. Consequently, the load on

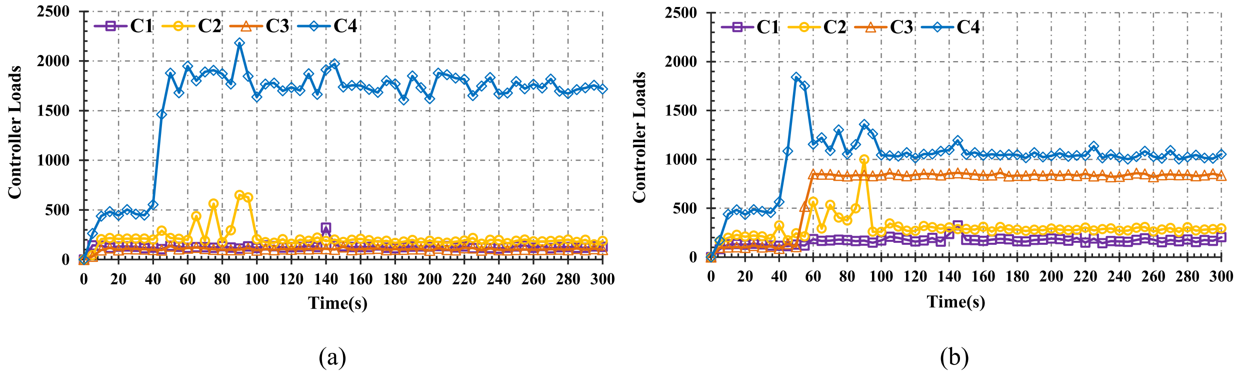

Fig. 9. (a) Load distribution under SMCS scheme. (b) Load distribution under BSM-LP scheme.

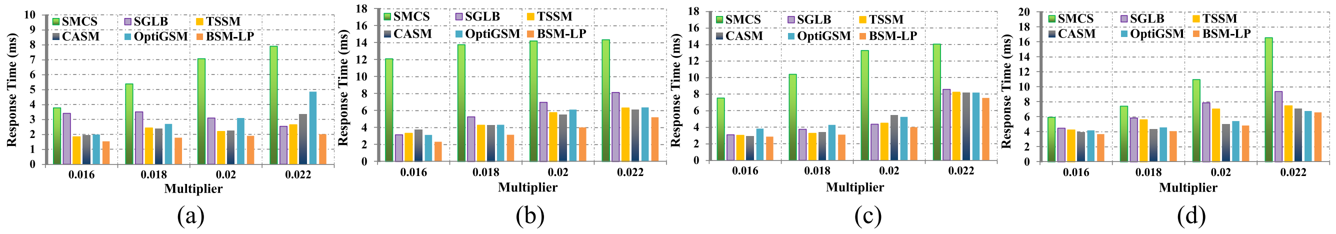

Fig. 10. Comparison of the response time under different topologies. (a) Abilene. (b) Noel. (c) Janos-us. (d) Pioro.

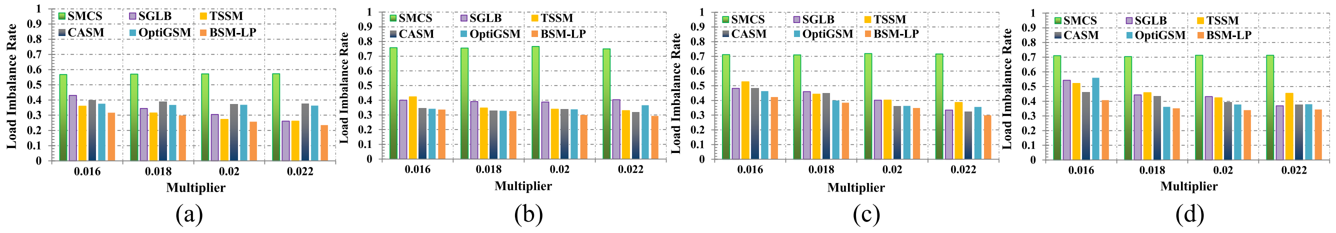

Fig. 11. Comparison of the load imbalance rate under different topologies. (a) Abilene. (b) Noel. (c) Janos-us. (d) Pioro.

controller C_4 remains below the overload threshold, achieving a balanced load distribution.

2) Response Time: The response time for processing a single Packet-In message is defined as the time interval between the arrival of the Packet-In message at the controller and the arrival of the corresponding generated Packet-Out message at the switch. As illustrated in Fig. 10, the SMCS scheme, which lacks load balancing strategies, exhibits an increase in average response time as the replay speed increases. However, the SGLB, TSSM, CASM, OptiGSM, and BSM-LP schemes adopt switch migration strategies to ensure load balancing. As a result, the average response time is reduced by 46.6%, 54.8%, 56.4%, 52.1%, and 63.4%, respectively, compared to the SMCS scheme. In addition, the BSM-LP scheme utilizes a bidirectional migration strategy based on load prediction, resulting in a more balanced load distribution among multiple controllers. Therefore, compared to the SGLB, TSSM, CASM, and OptiGSM schemes, the BSM-LP scheme achieves an average response time reduction of 29.3%, 18.6%, 17.2%, and 23.8%, respectively.

3) Load Imbalance Rate: The load imbalance rate for different schemes under various network topologies is depicted in Fig. 11. The load imbalance rate is defined based on (23), where a smaller value indicates more even load distribution among the controllers. Across different network topologies, the SMCS scheme, relying on static mapping, exhibits the highest load imbalance rate. In contrast, the SGLB, TSSM, CASM, OptiGSM, and BSM-LP schemes, employing switch migration strategies, achieve an average reduction in load imbalance rate of 41.8%, 43.1%, 43.3%, 43.9%, and 52.1%, respectively. Notably, the BSM-LP scheme incorporates the ATT-GRU model for load prediction, enhancing the accuracy of switch selection in bidirectional migration decision making, thereby maintaining the load imbalance rate below 0.5. Consequently, compared to the SGLB, TSSM, CASM, and OptiGSM schemes, which utilize traditional unidirectional migration strategies, the load imbalance rate of BSM-LP is further reduced by 17.1%, 15%, 14.2%, and 13.5%, respectively, resulting in superior load balancing effects.

4) Occurrences of Overload: In the experiment, we record the total number of controller overload occurrences (i.e., NOO) for different schemes. The controller in the SMCS scheme, as depicted in Fig. 12, has the highest NOO among all the schemes as it cannot adjust the mapping relationship. Compared to the SMCS scheme, the BSM-LP, CASM, SGLB, OptiGSM, and TSSM schemes achieve remarkable reductions

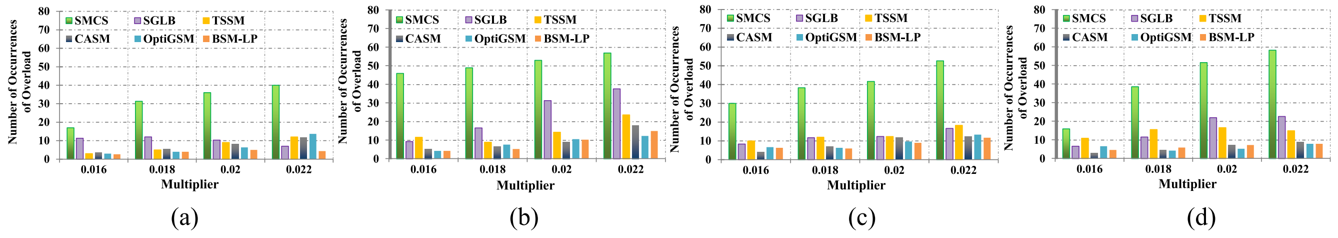

Fig. 12. Comparison of the NOO under different topologies. (a) Abilene. (b) Noel. (c) Janos-us. (d) Pioro.

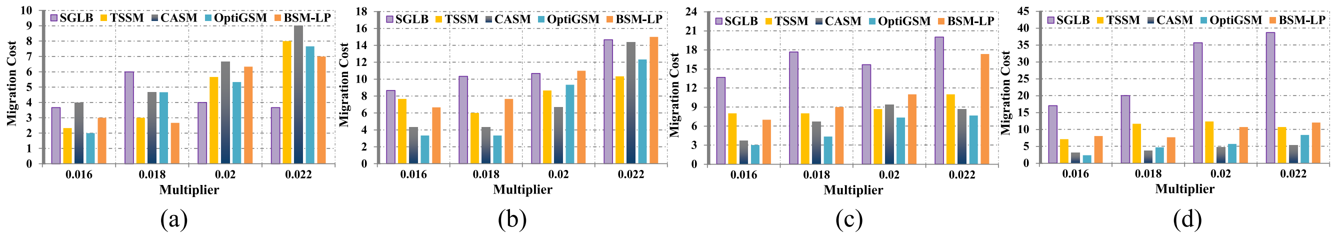

Fig. 13. Comparison of the migration cost under different topologies. (a) Abilene. (b) Noel. (c) Janos-us. (d) Pioro.

in NOO by approximately 83%, 77.9%, 62.3%, 80.4%, and 67%, respectively. In addition, the BSM-LP scheme leverages load prediction to assess the controller's overload state when the real-time load exceeds the threshold, thereby avoiding unnecessary switch migration operations. Under the four topologies, the BSM-LP scheme reduces the NOO by approximately 51.3% and 42.5% compared to the SGLB and TSSM schemes, respectively. Furthermore, in the Abilene topology, the BSM-LP scheme achieves an average NOO reduction of approximately 38.8% and 25.1% compared to the CASM and OptiGSM schemes.

5) Migration Cost: The test results of migration cost under different topologies are shown in Fig. 13. We define the migration cost as the aggregate number of times to migrate switches among different controllers. Since the SMCS scheme does not perform switch migration, it is not included in the comparison. The CASM scheme aims to alleviate the load of an overloaded controller by selecting the switch with the highest Packet-In message sending volume for migration, which can easily overload the target controller. The SGLB scheme uses a group-based switch migration strategy, which involves multiple switches in each migration, resulting in higher migration cost. The OptiGSM scheme achieves lower migration cost in various topologies by employing an optimum greedy strategy for selecting switches to migrate. In contrast, the BSM-LP scheme takes into account both migration cost and load imbalance rate during bidirectional migration decisions. Therefore, in the Janos and Pioro topologies, the BSM-LP scheme achieves an average reduction in migration cost of 49.3% compared to the SGLB scheme. In the Abilene topology, the BSM-LP scheme reduces migration cost by an average of 23.8% compared to CASM. However, due to the utilization of a bidirectional migration strategy and the requirement to integrate isolated nodes, the BSM-LP scheme exhibits a higher migration cost than the other four schemes in certain test scenarios.

6) Scalability: Fig. 14 shows the scalability of the BSM-LP scheme with a varying number of IoT devices and

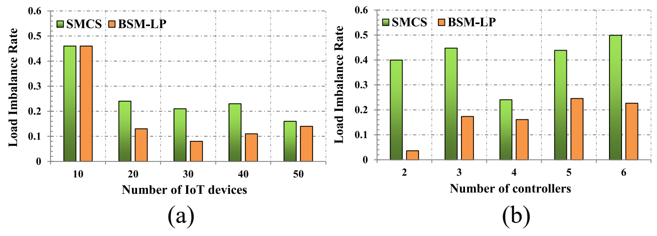

Fig. 14. (a) Load imbalance rate with varying number of IoT devices. (b) Load imbalance rate with varying number of controllers.

controllers under the Abilene topology. Fig. 14(a) shows that as the number of IoT devices increases, the BSM-LP scheme achieves improved load balancing performance. It reduces the average load imbalance rate by approximately 34.5% compared to the SMCS scheme. From Fig. 14(b), it is evident that as the number of controllers increases, the load imbalance rate also increases under the SMCS scheme. In contrast, the BSM-LP scheme is capable of maintaining the load imbalance rate below 0.3. Specifically, the BSM-LP scheme achieves an average reduction of approximately 56.7% in the load imbalance rate compared to the SMCS scheme.

V. Conclusion

In this article, we propose the BSM-LP scheme to address the problem of load imbalances among multiple controllers in SD-IoT. In this scheme, we use the ATT-GRU model to predict future controller loads, thereby avoiding unnecessary switch migrations. Meanwhile, we introduce a bidirectional migration algorithm that effectively improves migration efficiency and resolves the problem of overloading the target controller. Additionally, we present an isolated node integration algorithm to reduce the occurrence of isolated nodes. Finally, the experimental results validate the effectiveness of the scheme, and compared to existing schemes, it achieves an average reduction of 22.3% in load imbalance rate and an average decrease of 30.5% in response time. Although the proposed BSM-LP scheme improves load balancing performance among multiple controllers, further optimization is necessary. Specifically, the limitations of this scheme include the potential for higher migration cost in certain scenarios and the inability to dynamically adjust the number of controllers deployed in the control plane based on changes in the total load. Therefore, in our future research, we will focus on reducing switch migration cost and developing intelligent solutions to dynamically adjust controller numbers in the control plane for improved adaptation to network load changes.

References

[1] L. Zhao, J. Wang, J. Liu, and N. Kato, "Optimal edge resource allocation in IoT-based smart cities," *IEEE Netw.*, vol. 33, no. 2, pp. 30–35, Mar./Apr. 2019.

[2] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, "An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5141–5154, Dec. 2018.

[3] A. Montazerolghaem, "Software-defined Internet of Multimedia Things: Energy-efficient and load-balanced resource management," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 2432–2442, Feb. 2022.

[4] A. Montazerolghaem, "Efficient resource allocation for multimedia streaming in software-defined Internet of Vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 14718–14731, Dec. 2023.

[5] S. Zafar, Z. Lv, N. H. Zaydi, M. Ibrar, and X. Hu, "DSMLB: Dynamic switch-migration based load balancing for software-defined IoT network," *Comput. Netw.*, vol. 214, Sep. 2022, Art. no. 109145.

[6] A. H. Alhilali and A. Montazerolghaem, "Artificial intelligence based load balancing in SDN: A comprehensive survey," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100814.

[7] A. Montazerolghaem and M. H. Yaghmaee, "Load-balanced and QoS-aware software-defined Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3323–3337, Apr. 2020.

[8] T. Salehnia, A. Montazerolghaem, S. Mirjalili, M. R. Khayyambashi, and L. Abualigah, "SDN-based optimal task scheduling method in fog-IoT network using combination of AO and WOA," in *Handbook of Whale Optimization Algorithm*. London, U.K.: Elsevier, 2024, pp. 109–128.

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

[10] K. Saeed and M. O. Ullah, "Toward reliable controller placements in software-defined network using constrained multi-objective optimization technique," *IEEE Access*, vol. 10, pp. 129865–129883, 2022.

[11] S. Lange et al., "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[12] B. Zhang, X. Wang, and M. Huang, "Multi-objective optimization controller placement problem in Internet-oriented software defined network," *Comput. Commun.*, vol. 123, pp. 24–35, Jun. 2018.

[13] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "ElastiCon: An elastic distributed SDN controller," in *Proc. 10th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, 2014, pp. 17–28.

[14] J. Cui, Q. Lu, H. Zhong, M. Tian, and L. Liu, "A load-balancing mechanism for distributed SDN control plane using response time," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1197–1206, Dec. 2018.

[15] H. Mokhtar, X. Di, Y. Zhou, A. Hassan, Z. Ma, and S. Musa, "Multiple-level threshold load balancing in distributed SDN controllers," *Comput. Netw.*, vol. 198, 2021, Art. no. 108369.

[16] W.-K. Lai, Y.-C. Wang, Y.-C. Chen, and Z.-T. Tsai, "TSSM: Time-sharing switch migration to balance loads of distributed SDN controllers," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1585–1597, Jun. 2022.

[17] S. Li, Z. Xin, X. Xu, and Z. Zhang, "Load balancing algorithm of SDN controller based on dynamic threshold," in *Proc. 3rd Asia–Pacific Conf. Commun. Technol. Comput. Sci. (ACCTCS)*, 2023, pp. 517–520.

[18] P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, "MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning," *Comput. Netw.*, vol. 177, Aug. 2020, Art. no. 107230.

[19] K. Sridevi and M. A. Saifulla, "LBABC: Distributed controller load balancing using artificial bee colony optimization in an SDN," *Peer-to-Peer Netw. Appl.*, vol. 16, pp. 947–957, Feb. 2023.

[20] T. Hu, J. Lan, J. Zhang, and W. Zhao, "EASM: Efficiency-aware switch migration for balancing controller loads in software-defined networking," *Peer-to-Peer Netw. Appl.*, vol. 12, no. 2, pp. 452–464, 2019.

[21] G. Cheng, H. Chen, Z. Wang, and S. Chen, "Dha: Distributed decisions on the switch migration toward a scalable SDN control plane," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, 2015, pp. 1–9.

[22] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, *arXiv:1409.1259*.

[23] L. S.-T. Memory, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 2010.

[24] K. Xu et al. "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2048–2057.

[25] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T.-S. Chua, "Attentional factorization machines: Learning the weight of feature interactions via attention networks," 2017, *arXiv:1708.04617*.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[27] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," 2015, *arXiv:1508.04025*.

[28] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–15.

[29] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[30] Z. Zhu, X. Li, H. Chen, X. Zhou, and W. Deng, "An effective and robust genetic algorithm with hybrid multi-strategy and mechanism for airport gate allocation," *Inf. Sci.*, vol. 654, Jan. 2024, Art. no. 119892.

[31] S. Mirjalili, "Genetic algorithm," in *Evolutionary Algorithms and Neural Networks: Theory and Applications*. Cham, Switzerland: Springer Nat., 2019, pp. 43–55.

[32] F. Tomonori, "Introduction to Ryu SDN framework," presented at Open Networking Summit, 2013, pp. 1–14.

[33] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *Proc. Int. Conf. Commun. Comput. Syst. (ICCCS)*, 2014, pp. 139–142.

[34] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[35] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—Survivable network design library," *Netw. Int. J.*, vol. 55, no. 3, pp. 276–286, 2010.

[36] P.-T. Tivig, E. Borcoci, and M. Vochin, "Dynamic offloading the SDN control plane in large area networks by condition-aware migration of forwarding devices," in *Proc. 25th Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, 2022, pp. 205–210.

[37] Y. Zhou, Y. Wang, J. Yu, J. Ba, and S. Zhang, "Load balancing for multiple controllers in SDN based on switches group," in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, 2017, pp. 227–230.

[38] U. Prajapati, B. C. Chatterjee, and A. Banerjee, "OptiGSM: Greedy-based load balancing with minimum switch migrations in software-defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 2, pp. 2200–2210, Apr. 2024.

[39] D.-H. Le, H.-A. Tran, S. Souihi, and A. Mellouk, "An AI-based traffic matrix prediction solution for software-defined network," in *Proc. IEEE Int. Conf. Commun.*, 2021, pp. 1–6.

[40] K. Cho, K. Mitsuya, and A. Kato, "Traffic data repository at the WIDE project," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2000, pp. 1–9.

[41] "Traffic Trace Info." 2022. [Online]. Available: https://mawi.wide.ad.jp/mawi/samplepoint-F/2022/202201011400.html

[42] A. Turner. "Tcpreplay: PCAP editing and replay tools for^* nix." 2005. [Online]. Available: http://tcpreplay. synfin. net/

[43] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, pp. 199–222, Aug. 2004.

[44] L. Zhao et al., "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020.

[45] E. Brands, "Flow-based monitoring of GTP traffic in cellular networks," M.S. thesis, Dept. Electr. Eng. Mathe. Comput. Sci., Univ. Twente, Enschede, The Netherlands, 2012.

[46] A. Papadogiannakis, G. Vasiliadis, D. Antoniades, M. Polychronakis, and E. P. Markatos, "Improving the performance of passive network monitoring applications with memory locality enhancements," *Comput. Commun.*, vol. 35, no. 1, pp. 129–140, 2012.