*electronics*

*Article*

# Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers

**Sangho Yeo, Ye Naing, Taeha Kim and Sangyoon Oh ***

Department of Artificial Intelligence, Ajou University, Suwon 16499, Korea; soboru963@ajou.ac.kr (S.Y.); yenaingmdy@ajou.ac.kr (Y.N.); kth@ajou.ac.kr (T.K.)
* Correspondence: syoh@ajou.ac.kr; Tel.: +82-031-219-2633

**Abstract:** Distributed controllers in software-defined networking (SDN) become a promising approach because of their scalable and reliable deployments in current SDN environments. Since the network traffic varies with time and space, a static mapping between switches and controllers causes uneven load distribution among controllers. Dynamic migration of switches methods can provide a balanced load distribution between SDN controllers. Recently, existing reinforcement learning (RL) methods for dynamic switch migration such as MARVEL are modeling the load balancing of each controller as linear optimization. Even if it is widely used for network flow modeling, this type of linear optimization is not well fitted to the real-world workload of SDN controllers because correlations between resource types are unexpectedly and continuously changed. Consequently, using the linear model for resource utilization makes it difficult to distinguish which resource types are currently overloaded. In addition, this yields a high time cost. In this paper, we propose a reinforcement learning-based switch and controller selection scheme for switch migration, switch-aware reinforcement learning load balancing (SAR-LB). SAR-LB uses the utilization ratio of various resource types in both controllers and switches as the inputs of the neural network. It also considers switches as RL agents to reduce the action space of learning, while it considers all cases of migrations. Our experimental results show that SAR-LB achieved better (close to the even) load distribution among SDN controllers because of the accurate decision-making of switch migration. The proposed scheme achieves better normalized standard deviation among distributed SDN controllers than existing schemes by up to 34%.

**Keywords:** distributed controllers; software-defined networking (SDN); load balancing; switch migration; reinforcement learning

check for updates

**Citation:** Yeo, S.; Naing, Y.; Kim, T.; Oh, S. Achieving Balanced Load Distribution with Reinforcement Learning-Based Switch Migration in Distributed SDN Controllers. *Electronics* **2021**, *10*, 162. https://doi.org/10.3390/electronics10020162

Received: 30 November 2020
Accepted: 11 January 2021
Published: 13 January 2021

## 1. Introduction

SDN provides powerful programmable network architecture and can be used to design a logical topology that defines the placement of a network entity (e.g., hardware equipment such as a router, switch, load balancer, firewall, and VPN) [1]. SDN separates network space into a control plane and a data plane. In the data plane, the switches transfer actual data between hosts, while the control plane defines the flow rule, which is the rule for routing data among switches [2]. Decoupling of the data plane and the control plane makes the efficient management of the network logic possible.

However, when incoming packets are massive, a single SDN controller [3] could become a performance bottleneck where all SDN switches are connected to a single SDN controller. In addition, the controller could be a single point of failure [4] since all packet processing must be stopped. To address these two major problems, distributed SDN controllers are proposed [5]. In distributed SDN controllers, each SDN controller manages its own domain network while SDN switches are statically mapped to the specific controller. Therefore, load imbalances between controllers could occur, and, owing to the static mapping between switches and controllers, uneven load distribution between controllers becomes significant.

To resolve this load imbalance problem, dynamic switch-to-controller mapping changes have been studied to achieve an evenly balanced load among distributed SDN controllers [6–12]. These approaches are majorly focused on load balancing of distributed SDN controllers with switch migration schemes, and they are based on the repetitive optimization methods with a greedy selection of switch-to-controller mapping for the migration. In addition, some approaches adapt the reinforcement learning (RL) based decision-making for switch-to-controller mapping changes [13–15]. Generally, RL-based decision-making for switch migration concerns training the model adaptively in various SDN environments. However, the conventional RL-based methods use tabular Q-learning, which is a well-known model-free RL method that is utilized for the exploration of the environment for avoiding local optima problems. Because its definition of the state is not able to represent a load of each controller as a continuous value due to the limited size of the Q table [14,15], it is hard to utilize the dynamic change of load among controllers. This limitation makes it difficult to work promptly with the changes of a load of each controller for decision-making of switch migration.

To resolve this problem, a state of the art RL-based method, MARVEL [13], adopts a neural network to approximate the value of actions in continuous state space, and it models load balancing among controllers as linear optimization that is widely used for modeling network flow. To model linear optimization, they define the utilization of each controller as a linear summation of utilization per resource type with a fixed weight. However, a recent study [16] shows that the utilization of each controller cannot be effectively modeled by linear optimization because correlations of utilization per resource type are changed when the rate of PACKET_IN is changed. In addition, when controllers use a high network bandwidth, CPU utilization could unexpectedly jitter even if the utilization of network bandwidth is constantly high [17]. As a result, the linear model of resource utilization makes it difficult to distinguish which resource type is currently overloaded. Furthermore, the simplified action space of MARVEL worsens the problem. It is because its action space only considers the limited number of switch migration cases.

Consequently, the conventional RL methods make a less-optimized decision for switch migration, and that yields an unevenly balanced load among controllers. Furthermore, the time cost of migrating switch to another domain is relatively high [9,18]. In this study, we propose a switch-aware reinforcement learning load balancing (SAR-LB) scheme to resolve the problem in linear optimization for load balancing. We consider the utilization ratio of various resource types as an input state. In addition, to consider all possible switch migration cases effectively with small action sizes, we define a switch as an RL agent in SAR-LB. As we define a switch as an RL agent, the action size of our RL model is limited to the number of possible target controllers while enabling us to consider all possible switch migration cases in the working phase. As a consequence, SAR-LB selects the target controller more accurately.

We evaluated our proposed scheme in a simulated SDN environment. The results show that our proposed scheme achieves better normalized standard deviation among distributed controllers of up to 34% than other schemes. The contributions of this study are as follows:

- We enhance the switch migration decision-making scheme by considering various types of resource utilization of controllers and a switch to migrate as an input state of the RL model to cope with changed correlation among resource types;
- We efficiently consider all possible cases of migration with relatively small action sizes (i.e., the number of controllers) by modeling the switches as RL agents in the proposed scheme;
- We conduct the performance evaluation of our proposed scheme and other competing schemes on stochastic and various environment settings. We evaluate SAR-LB in two network conditions (i.e., with/without bursty network traffic) and five cases with different numbers of switches and controllers. The results show that our scheme performs better when the number of switches is much larger than the number of

controllers, and this confirms that our SAR-LB is well suited to the common data center where the controller manages more than dozens of switches [19–22].

The rest of the paper is organized as follow. We describe the background of our study in Section 2. In Section 3, we present the related works. We describe the details of our proposed scheme and present the experiment setups and the evaluation results in Sections 4 and 5, respectively. The discussion is presented in Section 6, and we conclude our paper in Section 7.

## 2. Background

In this section, we describe the background of the switch migration for the load balancing in distributed SDN controllers. Then, we introduce the RL used in the switch migration domain of the load balancing in distributed SDN controllers.

### *2.1. Switch Migration in Load Balancing of Distributed SDN Controllers*

In distributed SDN controllers, a switch can be controlled by multiple SDN controllers. However, only one controller can manage the switch to process the PACKET_IN requests and installation of flow rules. The relationship between switch and controller is specified according to the southbound interface protocol.

For example, OpenFlow [2] provides the multiple connections of SDN controllers to switches with different types of roles: (i) "master", (ii) "equal", and (iii) "slave". An SDN switch can be controlled by multiple controllers with "equal" and "slave" roles. The difference is that SDN controllers with "slave" roles have read-only access to the SDN switches. However, controllers with the "equal" role receive all asynchronous messages and have full access to manage these messages send from the SDN switches. The "master" role is similar to the "equal" role; however, the difference is that there is only one "master" for an SDN switch to manage asynchronous messages and other controllers become "slave" for the switch. Controllers with different types of roles can provide fault-tolerance to the SDN network. Additionally, even load distribution among the distributed SDN controllers can be achieved by migrating the switches from overutilized to an underutilized controller with role changes between controllers. As a result, PACKET_IN requests from the switches can be controlled by multiple controllers.

### *2.2. Reinforcement Learning*

RL is a type of learning concerned with what to do in dynamic situations of the environment [23]. Unlike other learning categories (e.g., supervised learning, unsupervised learning), the learning of RL is processed by interacting with the environment. This interaction is made by the basic elements of RL framing: the agent (i.e., it receives a reward from the environment and sends back the action to the environment), the state (i.e., it represents the current situation of the environment), and the reward (i.e., it is the feedback from the environment to change the state). RL can deal with complex problems. For example, RL is used as a tool for training AI models in complex systems such as healthcare, robotics, and network automation [24].

Figure 1 presents an overview of deep Q networks (DQN). DQN [25] is the most well-known deep reinforcement learning method, and it has two characteristics. The first characteristic is the replay buffer. By using the replay buffer, DQN can store and utilize previous experiences in the current training steps. The second is the target Q network. The target Q network is a supportive network for the Q network and is slowly trained by the Q network, while the Q network is trained directly via samples from the replay buffer. By setting the target Q network, the Q network can be trained more stably due to the conservative change of the target Q network.

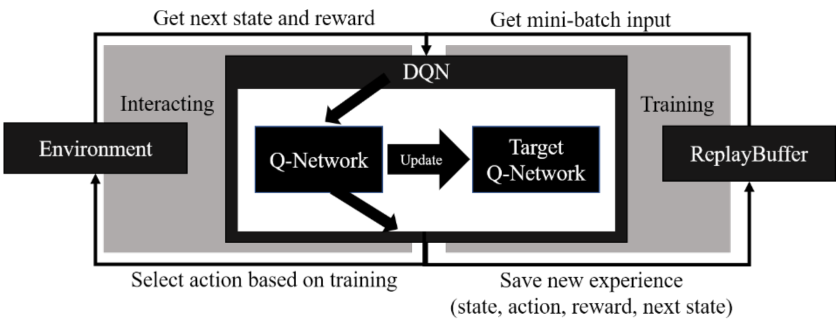
Get next state and reward
Get mini-batch input
DQN
Interacting
Training
Environment
Q-Network
Update
Target Q-Network
ReplayBuffer
Select action based on training
Save new experience (state, action, reward, next state)

**Figure 1.** Overview of deep Q-networks (DQN).

Algorithm 1 describes the whole process of DQN. In line 4, the DQN agent selects an action by the $\epsilon$-decay policy. In the $\epsilon$-decay policy, the action is selected randomly or by the Q network, and the probability of selecting the random action or the action by the neural network is determined by the value of $\epsilon$. After selecting the action, the experience, which consists of state, action, next state, reward, and done (i.e., Boolean value for defining the end of the episode), is saved to the replay buffer. The training process of DQN is defined in lines 11–19. Especially, the loss function of DQN is an important part of the training. To define the loss function, $y_x$ is defined by the combination of reward and the maximum target Q value of the next state. This $y_x$ is utilized as the labeling value of supervised learning. Thus, the mean square error of the value of Q network and $y_x$ become the loss function of DQN.

**Algorithm 1** DQN

```
Input: ReplayBuffer: data store for saving experience (i.e., state, reward, next state, action, done)
Q: the neural network which is directly trained for the approximate value of actions in each state,
target Q: the neural network which is trained through the Q neural network for stable training,
EPISODE_NUM: total number of episodes, STEP_NUM: total step numbers in each episode
1.  $time\_step = 0$
2.  for i in range (EPISODE_NUM):
3.    for j in range (STEP_NUM):
4.      Select $action$ by $\epsilon$-decay policy
5.      Send $action$ to the environment and get $state_{time\_step+1}$ and $reward$ by $action$
6.      if (j == STEP_NUM-1):
7.        $done$ = True
8.      else:
9.        $done$ = False
10.     Store experience ($,action,state_{time\_step+1},reward,done$) in ReplayBuffer
11.     if learning start < $time\_step$:
12.       Sample random experiences ($state_x$, $action_x$, $state_{x+1}$, $reward_x$, $done$) as
13.       if ($done$)
14.         $y_x = reward_x$
15.       else;
16.         $y_x = reward_x + \gamma \max_{a'} target\ Q(state_{x+1},\ action')$
17.       Optimize a Q network by loss $(y_x – Q(state_x,\ action_x))^2$.
18.       if target Q model update frequency % $time\_step$ == 0:
19.         Apply weight parameter of Q to target Q
20.   $time\_step++$
```

## 3. Related Works

Many researchers have studied various solutions for load balancing the distributed SDN controllers. This section discusses the existing studies based on conventional optimization and RL-based decision-making for switch migration.

### *3.1. Switch Migration Schemes for Load Balancing in SDN Environment*

Switch migration schemes in the previous studies focused on PACKET_IN processing, resource utilization, and switch group migration for migration efficiency. In the studies of [6–8], the decisions of switch migration schemes are considered according to the number of PACKET_IN requests to the distributed SDN controllers. In [6], the authors designed a migration algorithm with the distributed hopping algorithm (DHA). Their proposed algorithm randomly selected a switch from the overutilized controller and broadcast the coming migration activity to the neighbor controllers in each iteration. Therefore, communication overheads increase along with the synchronization of the states for global network view after migration. Al-Tam et al. [7] proposed a heuristic algorithm with "shift" and "swap" moves for switch migration. However, their proposed approach caused an increase in the decision-making time for migration. This was because their proposed "shift" and "swap" moves iteratively occur until the optimal switch-to-controller pair is decided for even load distribution among SDN controllers. Jie et al. [8] also proposed a switch migration scheme based on the response time of the PACKET_IN request. The response time depends on the network condition of the southbound communication. For example, the link failures between the switch and controller may affect the delay in the response time.

In [9,10], the authors also proposed a switch migration schemes focusing on maximizing the resource usages of distributed SDN controllers. In their approaches, the overutilized controller selects a switch with less load and at the farthest distance to eliminate the overutilization status of the controllers. In Cheng et al. [9], each underutilized controller played the game independently until there was only one controller (i.e., targeted controller), indicating that the controller migrated. The decision-making for switch migration is repetitively occurring until the optimal targeted controller is selected for the migration. Sahoo et al. [10] also proposed a framework called ESMLB. Their proposed framework uses the preliminary iterative calculation for every pair of switch-to-controller prior to decision-making. Therefore, the computation overhead of the preliminary calculation may affect the response time of the load balancing along with the scalability in the distributed SDN controller's environment.

Switch migration decision with switch groups' migration has been studied in [11] and [12]. Therein, switch group migration from an overutilized to an underutilized controller was iteratively performed. Switch group migration increased the migration cost. In Cello et al. [11], the authors also proposed a decision algorithm based on the graph partitioning technique. In their approach, a switch migration decision was conducted among the partitions in the short distance between the switches and the controller. Therefore, their approach was suitable for small networks. Wu et al. [12] designed a switch migration-based load balancing scheme by utilizing a non-cooperative game with iterative migration of switches. In their approach, switch migration activities occurred greedily in each round of the game. Moreover, switch group migration increases the extra overhead in the migration of switches.

### *3.2. Reinforcement Learning-Based Load Balancing in SDN Environment*

The load balancing issue in distributed SDN controllers has been addressed by the integration of artificial intelligence (AI) in recent studies. RL is a widely used AI method in the SDN environment. Particularly, it is used for the load balancing through an optimal policy, which is trained by the RL agent. In recent studies, RL-based research in the SDN environment can be divided into two categories: (i) use of RL in load balancing of the data plane, and (ii) use of RL in switch migration for load balancing of the control plane.

In the case of RL used in the load balancing of the data plane, Mu et al. [26] detect elephant flows in advance to reduce traffic between the controllers and switches. They employed two types of RL algorithms (i.e., traditional reinforcement learning and deep reinforcement learning algorithms) to better control overhead up to 60% and 14% improvement in table-hit ratio. In [27], Wu et al. proposed a deep Q-network (DQN) empowered dynamic flow data-driven approach for controller placement problem (D4CPP) that takes full consideration in the flow fluctuating, latency reducing, and load balancing. The results of their simulation show that D4CPP outperforms the traditional scheme by 13% in latency and 50% in load balance in the SDN environment with dynamic flow fluctuating. Tosounidis et al. [28] concentrate on dynamic load balancing with predictive methods using CNN and Q-learning. These methods try to predict the traffic and the state of controllers and networks in order to maximize the availability of the network. Their proposed method compares the performance of their approach with traditional methods only, such as round-robin, weighted round-robin, without the comparison with other RL-based solutions.

In the case of RL used in load balancing of the control plane, the switch migration-based load balancing of a distributed SDN controllers has been addressed with a multi-agent RL called MARVEL [13]. MARVEL trained the agents with the controller load generated by the Poisson process, revealing the overutilization of the controller caused by the distributed SDN controllers in real-world SDN controller load. MARVEL simplified the switch migration action by defining the action space as three cases (i.e., STAY, IMPORT, and EXPORT). However, simplified action space does not consider many other possible switch migration cases in current steps. Li et al. [14] also proposed RL-based switch migration decision-making for SDN controller load balancing. Their proposed approach is focused on the selection of an underutilized controller with minimal hop counts between the switch and the controller. This limits the selection space of underutilized controllers and cannot provide the scalability of the distributed SDN controller architecture. Min et al. [15] addressed the switch migration problem of distributed SDN controllers with a Q-learning algorithm. In their paper, the authors described the analysis of the proposed algorithm. However, there is no comparison with other switch migration algorithms in the evaluation of their algorithm.

## 4. Design of SAR-LB

Our proposed switch-aware reinforcement learning load balancing (SAR-LB) is a reinforcement learning-based switch and controller selection scheme for switch migration. It takes the utilization ratio of various resource types as input state of RL agent and defines a switch as an RL agent to limit the number of action size while considering all possible cases of switch migration. We present the design of SAR-LB as three-part: (i) the definition of RL communication channel, (ii) training and working phase, and (iii) DNN model for DQN agent.

To describe SAR-LB in detail, it would be effective to compare our proposed schemes, SAR-LB, with existing RL approaches in switch migration. Table 1 shows the comparison between the conventional RL based switch migration methods and our SAR-LB. Both MARVEL and SAR-LB use the DNN model to approximate the value of actions in the current SDN environment status, while other methods in [14,15] use tabular Q learning in the computation of the Q value. If the methods use the tabular Q learning, the only limited number of states can be defined as states because of the limited space of the Q table. In the Q table, a row index represents the controller selected as the initial controller, and a column index represents the selected switch migration cases. When the action is selected by the Q table, it is passively updated by reward. Thus, it is difficult to adapt to a dynamic variation of loads.

To adapt promptly to a dynamic variation of the load from controllers, a load of each controller should be used as an input state of a neural network. Or a machine-learning method should be used to approximate the value of actions in each load information of controllers. MARVEL and SAR-LB are taken these considerations into their methods. However, there are differences between MARVEL and SAR-LB. Unlike SAR-LB, MARVEL is based on multi-agent reinforcement learning (MARL), which is a distributed version of RL. However, MARVEL does not consider the utilization of resources in each resource type; it models resource utilization of each controller as a linear optimization instead. This yields a less-optimized switch migration since the utilization of each resource type has not been consistently correlated. For example, the memory and CPU utilization of the controller is exponentially increased as the number of PACKET_IN message increases [16].

Thus, it is almost impossible that the linear model with a constant weight of a resource type models the dynamically changing correlation among the various resource types. Moreover, when the network bandwidth of a controller is highly used (i.e., the utilization of network bandwidth is high), CPU utilization of a controller could unexpectedly jitter, while the utilization of network bandwidth is still constant [17]. Thus, the reason for overutilization is not clarified if the utilization per resource type is not defined. Because of this, it is better to model the load of the controller as the utilization ratios of each resource type. Furthermore, all possible switch migration cases should be considered for accurate decision-making for switch migration.

**Table 1.** Comparison of existing reinforcement learning (RL)-based switch migration methods with switch-aware reinforcement learning load balancing (SAR-LB).

| Type | MARVEL | Li et al. [17] | Min et al. [17] | SAR-LB (Ours) |
|---|---|---|---|---|
| Utilized multi-agent model | Yes | No | No | No |
| State definition | Coarse-grained resource utilization | Index of initial controller | Index of initial controller | Fine-grained resource utilization |
| Utilized DNN | Yes | No (tabular Q learning) | No (tabular Q learning) | Yes |

## *4.1. The Definition of RL Communication Channel for Switch Migration*

SAR-LB uses the resource utilization of various resource types as inputs to the training phase and can provide more precise action by considering more switch migration cases. We constructed the RL framing for our study, as shown in Figure 2. To compose the framing of RL, it is necessary to define two entities and three communication channels. The two entities are defined as agent and environment. An agent is defined as a candidate Open vSwitch [29] for migration, and the environment is defined as an SDN environment with targeted controllers to migrate the candidate switch. The three communication channels (i.e., state, reward, and action) are important factors in the context of the learning process of the agent. Because state and action space define the input and output space for the RL agent. Moreover, the reward is an important term because the reward is used when the objective function of each agent is calculated.

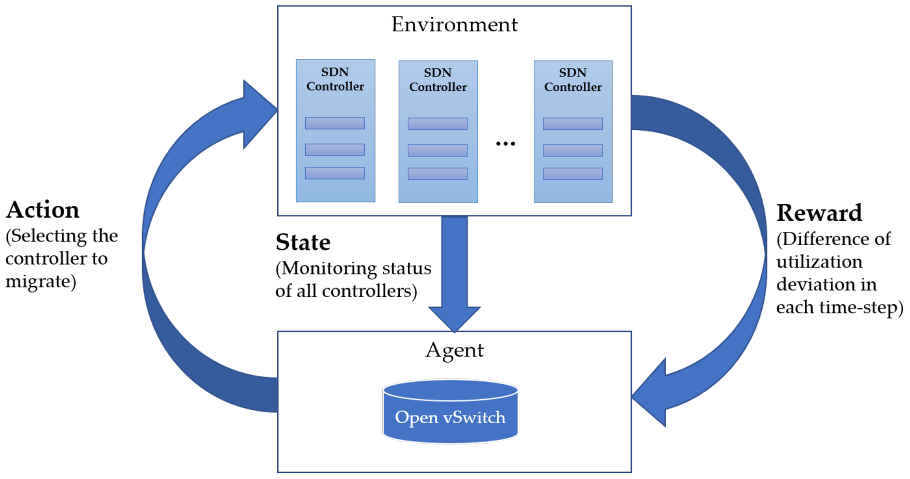
Environment
SDN Controller
SDN Controller
...
SDN Controller
Action
(Selecting the controller to migrate)
State
(Monitoring status of all controllers)
Reward
(Difference of utilization deviation in each time-step)
Agent
Open vSwitch

**Figure 2.** Reinforcement learning model in our proposed scheme.

We defined switches as RL agents, and the action is defined as selecting the controller for migration (i.e., selecting the target controller). Furthermore, the action size is defined by the number of controllers that a switch can select. Because we use the neural network to evaluate the value of actions in each input state, the output of the neural network is defined as the value of controllers in the current SDN environment status. Thus, the target controller for the switch is selected by the index of the output element, which has a maximum value.

Figure 3 defines the state for the RL model. The state is linear feature vectors of resource utilization of controllers and a resource utilization ratio of selected switch to migrate. In Figure 3, we represent linear feature vectors in a table format for a better understanding of state definition. Each row defines each controller's resource utilization per resource type. This resource utilization of the controller is defined as five tuples; $C_i.ID$ is an identifier number to classify each controller, and zero (i.e., 0) is a Boolean flag indicating the current information is about a controller. Moreover, the following three $C_i.Util$ define the controller's resource utilization rate for each resource type (e.g., CPU, RAM, and network bandwidth). This information for the controller merges with the information of the remaining controllers, which are defined as the same formats that we described in this paragraph.

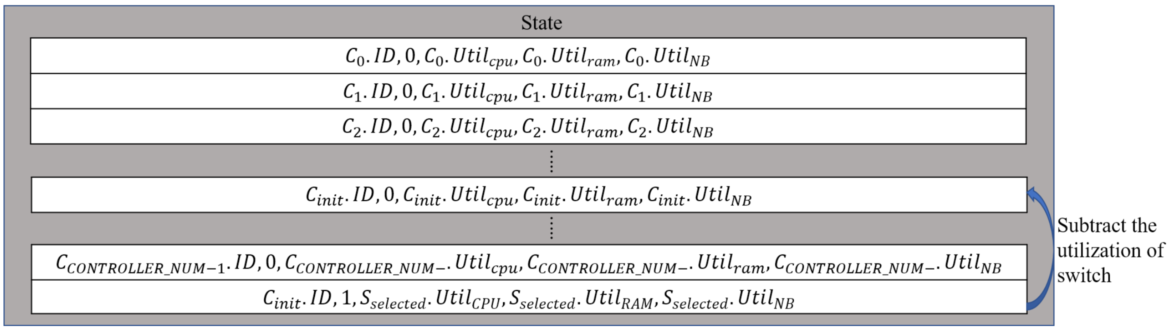
State
$C_0.ID, 0, C_0.Util_{cpu}, C_0.Util_{ram}, C_0.Util_{NB}$
$C_1.ID, 0, C_1.Util_{cpu}, C_1.Util_{ram}, C_1.Util_{NB}$
$C_2.ID, 0, C_2.Util_{cpu}, C_2.Util_{ram}, C_2.Util_{NB}$
$C_{init}.ID, 0, C_{init}.Util_{cpu}, C_{init}.Util_{ram}, C_{init}.Util_{NB}$
$C_{CONTROLLER\_NUM-1}.ID, 0, C_{CONTROLLER\_NUM-}.Util_{cpu}, C_{CONTROLLER\_NUM-}.Util_{ram}, C_{CONTROLLER\_NUM-}.Util_{NB}$
$C_{init}.ID, 1, S_{selected}.Util_{CPU}, S_{selected}.Util_{RAM}, S_{selected}.Util_{NB}$
Subtract the utilization of switch

**Figure 3.** State definition of SAR-LB.

After the controller information is described, additional information on the selected switch to migrate is added to the feature vector, as shown in the last row of the table in Figure 3. The information on the switch is similar to the description of the controller information. However, the ID of the controller in the switch information represents the ID of the initial controller where the switch is located. In this context, the Boolean flag is one (i.e., 1), which shows the information is about a switch, and the following three $S_{selected}.Util$ represents the resource utilization ratio in the initial controller by the selected switch for migration.

Switch migration can be defined as a process of selecting a target controller wherein the load of the selected switch will be added. In our definition of switch migration, the initial controller also needs to be considered as a candidate controller along with other controllers in the learning environment. In this context, as shown in the blue arrow of Figure 3, a load of the selected switch should be removed from a load of the initial controller, $C_i$, for a fair comparison of other controllers because the target controller selection is based on the comparison of loads among controllers in the environment.

To reflect the difference between the resource utilization of the initial controller and the target controller in the reward, the reward is calculated as the variation of the square difference of resource utilization per resource type between the previous step and the current time-step. It is calculated as described in Equation (1).

$$Reward = Util\_Difference_{t-1}(init_{controller}, dest_{controller}) - Util\_Difference_t(init_{controller}, dest_{controller}) \quad (1)$$

where, $Util\_Difference_t(init_{controller}, dest_{controller}) = (init_{controllerCPU_t} - dest_{controllerCPU_t})^2$
$+(init_{controllerRAM_t} - dest_{controllerRAM_t})^2$
$+(init_{controllerNB_t} - dest_{controllerNB_t})^2$

### 4.2. Training and Working Phases of SAR-LB

For load balancing of distributed SDN controllers, we propose switch migration decision algorithms with training and working phases in Algorithms 2 and 3, respectively. Algorithm 2 describes our proposed switch migration scheme using the DQN agent [25]. The number of repetitions is set in line 1 of Algorithm 2. After initializing the SDN environment, the initial state value is created according to the information of controllers and target switch to migrate (lines 2 to 4 in Algorithm 3). The communication and action execution process between the DQN agent and the SDN environment is performed from Lines 6 to 12. Lines 13 to 14 define the learning process of the DQN agent.

**Algorithm 2** Training phase of SAR-LB

```
Input: ReplayBuffer: the memory space for saving previous experiences
EPISODE_NUM: the number of episodes, STEP_NUM: the number of steps,
CONTROLLER_NUM: the number of controllers in the SDN environment,
SWITCH_NUM: the number of switches in the SDN environment,
Output: switch migration decision
1.  for i in range (EPISODE_NUM):
2.      Initialize SDN environment and get initial state $state_0$ with CONTROLLER_NUM and SWITCH_NUM
3.      Select migration target switch, $S_0$ randomly in the overutilized controller
4.      Append information of migration target switch on initial state $state_0$ as shown in equation (1)
5.      for j in range (STEP_NUM):
6.          Select destination controller (i.e., action), $C_j$ by DQN agent with epsilon greedy policy
7.          Migrate target switch, $S_j$ based on selected controller, $C_j$
8.          Get next state, $state_{j+1}$ and reward, $reward_j$ from SDN environment
9.          Save experience ($state_j$, $controller_j$, $reward_j$) on ReplayBuffer
10.         Select next target switch for migration, $switch_{j+1}$ randomly in the overutilized controller
11.         Append information for $S_{j+1}$ on next_state $state_{j+1}$ as shown in Equation (1)
12.         Next state, $state_{j+1}$ assigns to current state, $state_j$
13.         Sampling experience from ReplayBuffer
14.         Train DQN agent by sampled experience//which is equal to line 11–19 in algorithm 1
15.     end for
16. end for
```

In Algorithm 2, we proposed a training phase for switch migration through reinforcement learning. However, the training phase is hard to utilize directly due to its randomness switch selection. To effectively utilize the trained model, the working phase for switch migration is described in Algorithm 3.

In Algorithm 3, the controller with the highest resource utilization rate is selected through line 2 to 5. In line 7 to 10, the information of controllers with the resource utilization rate of each switch, $C_{max}.S_j$, in the controller, $C_{max}$, is taken as the input of the neural network. After this, actions and rewards for each switch in the controller, $C_{max}$, are collected. In lines 11 to 12, switch migration is performed through the action with the highest reward.

### 4.3. DNN Model for DQN Agent

The DQN agent was configured through two types of layers, which are the LSTM [30] layer and the fully connected (FC) layer to recognize the change in the incoming load. Long short-term memory (LSTM) is a type of recurrent neural network (RNN) [30]. An RNN is a type of neural network that is designed to operate sequential information between layers. Moreover, second, the FC layer is simple, feed-forward neural networks. FC layers commonly form the last few layers in the network. Thus, the input to the fully connected layer is the output from the final pooling or convolutional layer or LSTM layer. In the combination of LSTM and FC, the number of input features in the FC layer is the number of hidden units in LSTM.

As shown in Figure 4, history length, which defines the number of consecutive experiences when sampling experiences in the DQN replay buffer, is defined as the LSTM sequence length. Thereafter, the result value of the LSTM layer is connected to the fully connected layer so that the behavior considering the continuous experience can be determined.

**Algorithm 3** Working phase of SAR-LB

```
Input: $C_i$: Controller $i$,
$C_i.CPU$,$C_i.RAM$,$C_i.NB$ : CPU utilization, RAM utilization, and network bandwidth utilization of $C_i$
$C_i.utilization$ : total resource utilization rate,
$C_i.S_j$ : switch $j$ from $C_i$,
CONTROLLER_NUM: the number of controllers in SDN environment,
Output: Load balanced controllers
1.  Set list $List\_Utilization_{controller}$ for $C_i.utilization$ of all controllers
2.  for i in range (CONTROLLER_NUM):
3.      Append each utilization of controller in $C_i.utilization$ in $List\_Controller_{utilization}$
4.  end for

5.  Find $C_{max}$ in $List\_Utilization_{controller}$ which has maximum resource utilization
6.  Set list $List\_Reward_{switch}$ for rewards of switches in $C_{max}$,
7.  for $C_{max}.S_j$ in $C_{max}$:
8.      Get reward, $C_{max}.S_j.reward$ by selected action, $C_{max}.S_j.action$ of the agent for switch,
$C_{max}.S_j$ based on current input state
9.      Append reward $C_i.S_j.reward$ in $List\_Reward_{switch}$
10. end for
11. Find action (i.e., destination controller), $action_{max}$which has maximum reward,
12. Migrate target switch which have maximum reward action to target controller, $action_{max}$
```

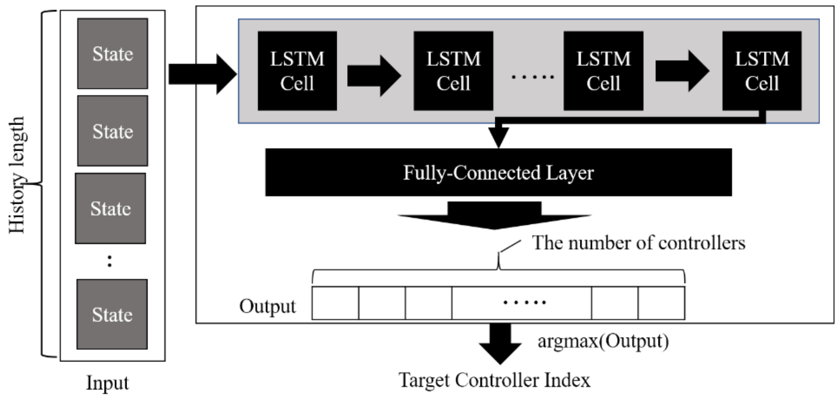

**Figure 4.** DNN model definition for DQN agent.

## 5. Evaluation and Results

### *5.1. Experimental Environment*

To evaluate our proposed scheme, SAR-LB, we defined a simulation environment using an SDN environment based on Python, and we used the host machine with Intel Xeon Skylake (Gold 6230)/2.10 GHz (20-cores) processor, 384 GB RAM and V100 GPU as shown in Table 2. Each switch from the simulated SDN environment creates a workload through a Poisson distribution [31] for each episode. Additionally, the loads for each resource type (i.e., CPU, RAM, network bandwidth) are also generated differently for each environment creation, which enables our proposed scheme to learn in a dynamic load environment. The DQN-agent based on PyTorch [32] performed the learning by configuring the hyperparameters, as shown in Table 3.

**Table 2.** Hardware configuration.

| Processor | Memory | Graphics Card |
|---|---|---|
| Intel Xeon Skylake (Gold 6230)/ 2.10 GHz (20-cores) processor | 384 GB | V100 GPU |

**Table 3.** Hyperparameter setting.

| Hyper Parameters | Values |
|---|---|
| Minibatch size | 64 |
| Replay buffer size | 1,000,000 frames |
| Learning start (i.e., time-steps to start learning) | 3000 time-steps |
| Learning frequency (i.e., time-steps intervals to train RL agent) | 4 time-steps |
| History length | 32 frames |
| Target Q model update frequency | 10,000 time-steps |
| Learning rate | 0.0001 |
| Optimizer | RMSProp [33] |
| The number of episodes | 10,000 |
| The number of time-steps per episode | 300 |

### 5.2. *Competing Schemes to Evaluate with SAR-LB*

To evaluate our proposed scheme, we set and tested the following schemes as comparisons. For evaluating our schemes statistically, we trained SAR-LB and MARVEL on a stochastic SDN network environment. This stochastic environment means that the network environment at each episode generates different network overhead randomly. Furthermore, we averaged our evaluation results on four different network simulations, which are generated by setting different random seeds. Our SAR-LB is compared with other approaches as follows:

Round robin: In a round-robin (RR) scheme, the switch with the highest load in the controller is selected from the overutilized controller, and the targeted controller for migration is selected in a round-robin fashion.

ElastiCon [34]: ElastiCon is well-known research in dynamic switch migration. For balancing load between controllers, their scheme compares every case of switch migration and selects the best migration case based on a standard deviation of utilization among controllers. For a fair comparison with MARVEL and SAR-LB, we have modified their algorithm to find the best migration case using utilization difference between an initial controller and targeted controller as shown in Equation (1), which is used as a reward function in both MARVEL and SAR-LB.

MARVEL [13]: In MARVEL, controllers run multiple agents, and the RL model takes controller resource utilization as input and decides the optimal switch-to-controller mapping for the migration. However, we did not compare our schemes with [14,15] because we consider MARVEL as state-of-the-art schemes that utilize the RL for switch migration than previous studies, and the study of [14,15] is difficult to handle the dynamic changes of load in switches.

SAR-LB: In our proposed scheme, we collect actions (i.e., selected target controller by DQN) of switches for the controller with the highest load and then migrate a switch that has a maximum reward to the target controller. Compared to MARVEL, our scheme utilizes the novel input state definition for representing the utilization ratio per resource type. In addition, action space for our schemes can consider every possible target controller in which this switch can be migrated.

### 5.3. *Evaluation Results and Analysis*

In our simulation, we conducted the experiments within 300 time-steps in five scenarios: three cases that increase only the number of switches and two cases that increase the number of switches and controllers both. In addition, we conducted the evaluations in these five scenarios on two cases of network conditions: (i) network condition without bursty traffic and (ii) network condition with bursty traffic. The average workload of network conditions with/without bursty traffic is represented in Figure 5. The workload in each controller is dynamically changed along with the time. Therefore, uneven load distribution is shown in Figure 5. Figure 5b shows the average workload distribution of bursty traffic that we utilized in the evaluations. In this figure, the network overhead of each controller is sharply increased unexpectedly.

#### 5.3.1. Comparison of Load Balancing Rate

We applied our proposed switch migration decision-making scheme, SAR-LB, to all controllers in five scenarios with two network conditions, as explained above. The load balancing rate before and after applying our scheme is shown in Figures 6 and 7. In figures, the $x$-axis shows the time-steps, and the $y$-axis shows the standard deviation of workload distribution. As shown in Figures 6 and 7, the load balancing of the controller after applying SAR-LB ("red" line in Figures 6 and 7 has more even load distribution among controllers in contrast to the load distribution pattern without load balancing. Furthermore, we also applied alternative schemes in the testbed (i.e., RR, ElatiCon, and MARVEL) to evaluate the performance of our proposed scheme, and the result of the workload distribution in these schemes is shown in Figures 6 and 7 with different colors, respectively. Overall, the standard deviation of load distribution with our approach is better than other competing schemes when the case with four controllers and 80 switches.

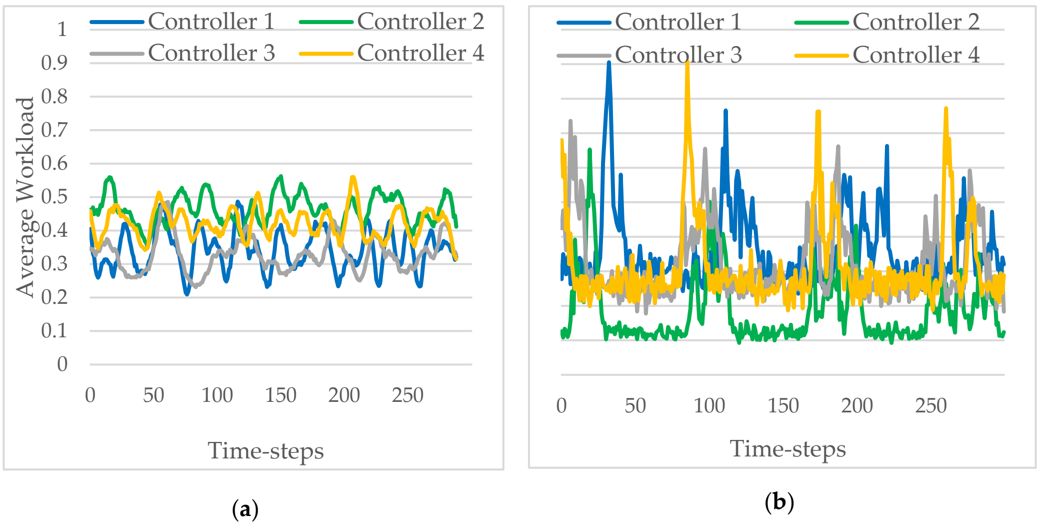

**Figure 5.** Average workload distribution of four controllers and 20 switches: (**a**) general network traffic; (**b**) bursty network traffic.

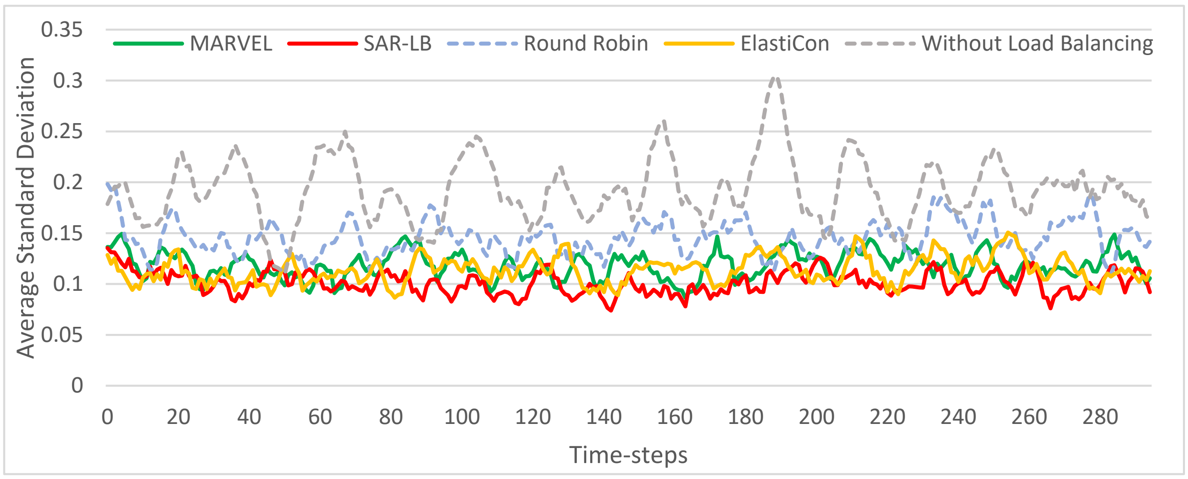

**Figure 6.** Comparison of workload distribution after load balancing in four controllers with 80 switches without bursty network traffic.

Tables 4 and 5 show the average standard deviation of workload load distribution between SAR-LB and competing schemes in the simulation environments in five scenarios, respectively, in two network conditions. The standard deviation of RR also resulted in higher than the average standard deviation without load balancing in both scenarios. In the environment with four controllers and 20 switches, testbed without bursty network traffic, MARVEL, ElastiCon, and SAR-LB achieved 112%, 121%, and 126% better load balancing than without load balancing schemes, respectively. This trend is clearer in the case where only the number of switches is increased. However, the load balancing performance of our proposed scheme, SAR-LB, is getting worse when both the number of switches and controllers are increased. As a result, SAR-LB shows up to 20% better normalized standard deviation than ElastiCon and outperforms other schemes in four of five evaluation scenarios. However, it shows 26–24% less normalized standard deviation with the scenario with 16 controllers and 80 switches compared to ElastiCon and MARVEL.

The evaluation result with bursty network traffic is shown in Table 5. This table shows similar results as Table 4. However, the difference in standard deviation becomes larger as the number of switches or controllers are changed. Therefore, SAR-LB shows a 1% better normalized standard deviation than ElastiCon in the case of four controllers and 20 switches, and this difference becomes much larger than the network condition without bursty traffic. As a result, SAR-LB shows up to 34% better normalized standard deviation than ElastiCon.

As a result, SAR-LB outperforms MARVEL and ElastiCon when the number of controllers is less than eight, and this trend is much stronger when the number of switches is increased. However, SAR-LB shows less efficient load balancing performance than the other schemes when the number of controllers is increased. This is due to the increased dimension of input state and action space of our scheme when the number of controllers is increased, and it makes it difficult for RL agents in our scheme to learn the current states of the SDN environment and selecting an action.

**Table 4.** Average standard deviation of workload distribution without bursty network traffic (normalized standard deviation compared to the case without load balancing in parentheses).

| Schemes | 4 CTRLs, 20 Switches | 4 CTRLs, 40 Switches | 4 CTRLs, 80 Switches | 8 CTRLs, 40 Switches | 16 CTRLs, 80 Switches |
|---|---|---|---|---|---|
| W/O Load Balancing | 0.0811 (100%) | 0.1210 (100%) | 0.1918 (100%) | 0.0970 (100%) | 0.0960 (100%) |
| ElastiCon | 0.0669 (121%) | 0.0857 (141%) | 0.1136 (169%) | 0.0687 (141%) | 0.0729 (132%) |
| Round Robin | 0.0991 (82%) | 0.1175 (103%) | 0.1455 (132%) | 0.0948 (102%) | 0.0942 (102%) |
| SAR-LB | 0.0645 (126%) | 0.0781 (155%) | 0.1014 (189%) | 0.0670 (145%) | 0.0906 (106%) |
| MARVEL | 0.0724 (112%) | 0.0900 (134%) | 0.1183 (162%) | 0.0732 (132%) | 0.0740 (130%) |

**Table 5.** Average standard deviation of workload distribution with bursty network traffic (normalized standard deviation compared to the case without load balancing in parentheses).

| Schemes | 4 CTRLs, 20 Switches | 4 CTRLs, 40 Switches | 4 CTRLs, 80 Switches | 8 CTRLs, 40 Switches | 16 CTRLs, 80 Switches |
|---|---|---|---|---|---|
| W/O Load Balancing | 0.1162(100%) | 0.1809(100%) | 0.2818(100%) | 0.1126(100%) | 0.1209(100%) |
| ElastiCon | 0.0717(162%) | 0.0917(197%) | 0.1364(207%) | 0.0751(150%) | 0.0850(142%) |
| Round Robin | 0.1028(113%) | 0.1313(138%) | 0.1682(168%) | 0.1014(111%) | 0.1103(110%) |
| SAR-LB | 0.0712(163%) | 0.0852(212%) | 0.1170(241%) | 0.0740(152%) | 0.1157(104%) |
| MARVEL | 0.0781(149%) | 0.1027(176%) | 0.1588(177%) | 0.0796(141%) | 0.0873(138%) |

#### 5.3.2. Comparison of Decision Time per Time-Step

Table 6 describes the comparison of decision-making time per time-step, which describes the results of the experiments over five scenarios. SAR-LB shows 53–104% decision-making time when it compares to MARVEL. This result is made by the optimization of the input pipeline, which merges the input states of every switch in a controller as one input state. Furthermore, SAR-LB and MARVEL show slower decision time than ElastiCon and RR. This is due to the inference overhead of MARVEL and SAR-LB which utilize the DNN model. However, the inference overhead is relatively small when it compares to total switch migration time, as shown in [9,18]. Thus, we assumed that the inference overhead is acceptable to select a more efficient switch migration case.

Although SAR-LB shows a similar and cheaper time cost than MARVEL in Table 6, our algorithm is difficult to distribute among controllers. This problem is the main disadvantage of our algorithm. Therefore, we will discuss this problem with the scalability problem of our scheme in Section 6.

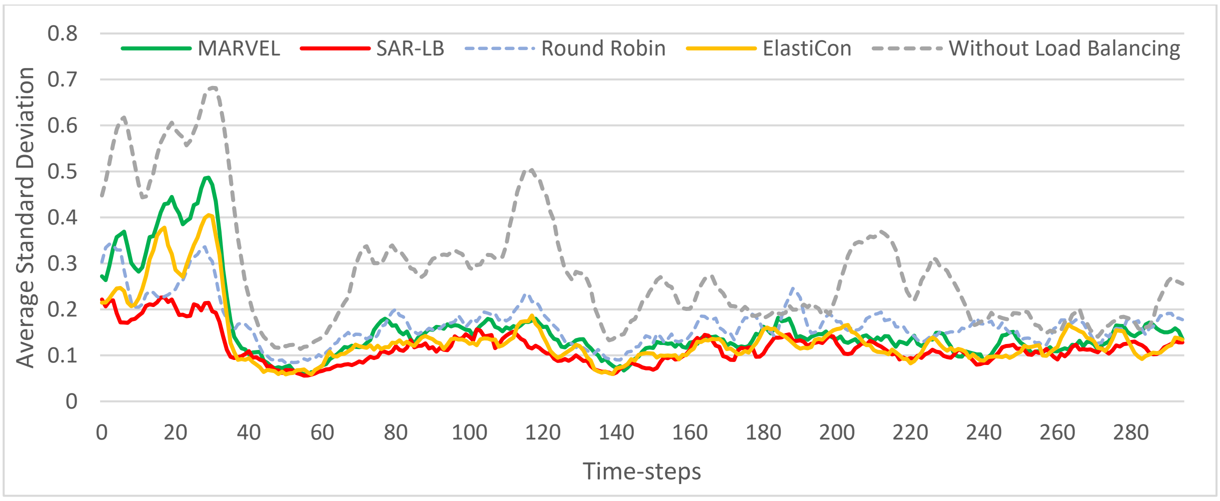

**Figure 7.** Comparison of workload distribution after load balancing in four controllers and 80 switches with bursty network traffic.

**Table 6.** Decision-making time per time-step (normalized standard deviation compared to MARVEL is shown in parentheses).

| **Schemes** | **4 CTRLs, 20 Switches** | **4 CTRLs, 40 Switches** | **4 CTRLs, 80 Switches** | **8 CTRLs, 40 Switches** | **16 CTRLs, 80 Switches** |
|---|---|---|---|---|---|
| W/O load balancing | 0.00212 (34%) | 0.00447 (52%) | 0.00889 (65%) | 0.00439 (35%) | 0.00882 (36%) |
| ElastiCon | 0.00277 (44%) | 0.00535 (62%) | 0.01054 (76%) | 0.00562 (45%) | 0.01234 (50%) |
| Round Robin | 0.00214 (34%) | 0.00422 (49%) | 0.00839 (61%) | 0.00433 (35%) | 0.00857 (35%) |
| SAR-LB | 0.00517 (82%) | 0.00817 (94%) | 0.01426 (104%) | 0.00773 (62%) | 0.01312 (53%) |
| MARVEL | 0.00630 (100%) | 0.00865 (100%) | 0.01377 (100%) | 0.01238 (100%) | 0.02478 (100%) |

## 6. Discussion

Our proposed scheme, SAR-LB, shows that it improves up to 34% higher normalized standard deviation among controllers than other competing schemes. The improvement is mainly because of our effective design of communication channels, training and working phases, and the DQN model. In addition, the performance of SAR-LB gets better as the number of switches is increased. Thus, based on these experiment results, we claim that SAR-LB can be used in a common SDN environment in data center networks because each controller in the data center manages dozens of switches generally [19–22].

However, SAR-LB still has room to be improved in its scalability. We suggest transforming the feature vectors into a fixed size image format. Since an input size is independent of the numbers of controllers and switches, and we are able to stack layers deep on the neural network if we use a convolution layer, this transformation can be taken as an advantage. As a consequence, this transformation process will increase scalability as well as producing more accurate decision-making.

Parallelizing our working phase algorithm into multiple controllers to enhance the scalability is hard to achieve. Thus, we suggest the data parallelism of distributed deep learning. It can be achieved by distributing the inference overhead of the working phase algorithm onto multiple controllers and gathering the results of inference by utilizing a collective communication library (e.g., MPI [35]). The other way to reduce the inference overhead is to set the threshold for inference, i.e., the inference process can be performed only for switches that occupy a large resource usage of the controller with the utilization threshold.

## 7. Conclusions

In this paper, we presented our design of switch migration scheme with RL-based switch migration decision-making, SAR-LB, which achieves more evenly balanced load distribution between distributed SDN controllers than other schemes. Unlike the existing RL approaches, we consider the resource utilization ratio of various resource types in state information as well as making a switch as an RL agent to consider all switch migration cases efficiently. As a consequence, our proposed scheme provides better load distribution because of its accurate decision-making for switch migration.

We evaluated the performance of SAR-LB in the simulated testbeds with the five scenarios. In each scenario, we vary the number of controllers or switches. The results show that our proposed SAR-LB performs better than competing schemes in load distribution in four of five scenarios, and the improvement of load distribution between distributed SDN controllers is up to 34% more than existing schemes.

**Author Contributions:** Conceptualization, S.Y. and S.O.; methodology, S.Y. and S.O.; software, S.Y. and Y.N.; validation, S.Y., Y.N., T.K. and S.O.; writing—original draft preparation, S.Y., Y.N., T.K. and S.O.; writing—review and editing, S.O. and S.Y.; visualization, Y.N. and S.Y.; supervision, S.O.; project administration, S.O.; funding acquisition, S.O. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2020-2018-0-01431) supervised by the IITP (Institute for Information & Communications Technology Promotion) and Basic Science Research Program Through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07043858).

**Data Availability Statement:** The data presented in this study are openly available in FigShare at 10.6084/m9.figshare.13562441.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
2. Shin, M.K.; Nam, K.H.; Kim, H.J. Software-defined networking (SDN): A reference architecture and open APIs. In Proceedings of the IEEE 2012 International Conference on ICT Convergence (ICTC), Jeju Island, Korea, 15–17 October 2012; pp. 360–361.
3. Open Flow Protocol 1.3. Available online: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf (accessed on 30 November 2020).
4. Pashkov, V.; Shalimov, A.; Smeliansky, R. Controller failover for SDN enterprise networks. In Proceedings of the IEEE 2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), Moscow, Russia, 28–29 October 2014; pp. 1–6.
5. Hu, T.; Guo, Z.; Yi, P.; Baker, T.; Lan, J. Multi-Controller Based Software-Defined Networking: A Suvey. *IEEE Access* **2018**, *6*, 15980–15996. [CrossRef]
6. Ye, X.; Cheng, G.; Luo, X. Maximizing SDN control resource utilization via switch migration. *Comput. Netw.* **2017**, *126*, 69–80. [CrossRef]
7. Al-Tam, F.; Correia, N. On Load Balancing via Switch Migration in Software-Defined Networking. *IEEE Access* **2019**, *7*, 95998–96010. [CrossRef]
8. Cui, J.; Lu, Q.; Zhong, H.; Tian, M.; Liu, L. A Load-Balancing Mechanism for Distributed SDN Control Plane Using Response Time. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1197–1206. [CrossRef]
9. Cheng, G.; Chen, H.; Hu, H.; Lan, J. Dynamic switch migration towards a scalable SDN control plane. *Int. J. Commun. Syst.* **2016**, *29*, 1482–1499. [CrossRef]
10. Sahoo, K.S.; Puthal, D.; Tiwary, M.; Usman, M.; Sahoo, B.; Wen, Z.; Sahoo, B.P.S.; Ranjan, R. ESMLB: Efficient Switch Migration-Based Load Balancing for Multicontroller SDN in IoT. *IEEE Internet Things J.* **2020**, *7*, 5852–5860. [CrossRef]
11. Cello, M.; Xu, Y.; Walid, A.; Wilfong, G.; Chao H., J.; Marchese, M. BalCon: A Distributed Elastic SDN Control via Efficient Switch Migration. In Proceedings of the 2017 IEEE International Conference on Cloud Engineering (IC2E), Vancouver, BC, Canada, 4–7 April 2017; pp. 40–50. [CrossRef]
12. Guowei, W.; Jinlei, W.; Obaidat, M.; Yao, L.; Hsiao, K.F. Dynamic switch migration with noncooperative game towards control plane scalability in SDN. *Int. J. Commun. Syst.* **2019**, *32*, e3927.
13. Sun, P.; Guo, Z.; Wang, G.; Lan, J.; Hu, Y. MARVEL: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning. *Comput. Netw.* **2020**, *177*, 107230. [CrossRef]

14. Li, Z.; Zhou, Z.; Gao, J.; Qin, Y. SDN Controller Load Balancing Based on Reinforcement Learning. In Proceedings of the 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 23–25 November 2018. [CrossRef]
15. Zhu, M.; Hua, Q.; Zhao, J. Dynamic switch migration with Q-learning towards scalable SDN control plane. In Proceedings of the 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China, 11–13 October 2017. [CrossRef]
16. Zhu, X.; Chang, C.; Xi, Q.; Zuo, Z. Attribute-Guard: Attribute-Based Flow Access Control Framework in Software-Defined Networking. *Secur. Commun. Netw.* **2020**, *2020*, 6302739. [CrossRef]
17. Imran, M.; Durad, M.H.; Khan, F.A.; Derhab, A. Reducing the effects of DoS attacks in software defined networks using parallel flow installation. *Human Cent. Comput. Inf. Sci.* **2019**, *9*, 16. [CrossRef]
18. Wang, C.A.; Hu, B.; Chen, S.; Li, D.; Liu, B. A switch migration-based decision-making scheme for balancing load in SDN. *IEEE Access* **2017**, *5*, 4537–4544. [CrossRef]
19. Wang, T.; Liu, F.; Xu, H. An efficient online algorithm for dynamic SDN controller assignment in data center networks. *IEEE/ACM Trans. Netw.* **2017**, *25*, 2788–2801. [CrossRef]
20. Wang, T.; Liu, F.; Guo, J.; Xu, H. Dynamic SDN controller assignment in data center networks: Stable matching with transfers. In Proceedings of the IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
21. Bogdanski, B. Optimized Routing for Fat-Tree Topologies. Ph.D. Thesis, Department of Informatics Faculty of Mathematics and Natural Sciences University of Oslo, Oslo, Norway, January 2014.
22. Liu, W.; Wang, Y.; Zhang, J.; Liao, H.; Liang, Z.; Liu, X. AAMcon: An adaptively distributed SDN controller in data center networks. *Front. Comput. Sci.* **2020**, *14*, 146–161. [CrossRef]
23. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
24. Rafique, D.; Velasco, L. Machine learning for network automation: Overview, architecture, and applications [Invited Tutorial]. *IEEE/OSA J. Opt. Commun. Netw.* **2018**, *10*, D126–D143. [CrossRef]
25. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
26. Mu, T.; Al-Fuqaha, A.; Shuaib, K.; Sallabi, F.M.; Qadir, J. SDN Flow Entry Management Using Reinforcement Learning. *ACM Trans. Auton. Adapt. Syst.* **2018**, *13*, 1–23. [CrossRef]
27. Wu, Y.; Zhou, S.; Wei, Y.; Leng, S. Deep Reinforcement Learning for Controller Placement in Software Defined Network. In Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, 6–9 July 2020.
28. Tosounidis, V.; Pavlidis, G.; Sakellaiou, I. Deep Q-Learning for Load Balancing Traffic in SDN Networks. In Proceedings of the SETN: Hellenic Conference on Artificial Intelligence, Athens, Greece, 2–4 September 2020.
29. Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Amidon, K. The design and implementation of open vswitch. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI' 15), Oakland, CA, USA, 13 January 2015; pp. 117–130.
30. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv* **2014**, arXiv:1412.3555.
31. Xu, Z.; Tang, J.; Meng, J.; Zhang, W.; Wang, Y.; Liu, C.H.; Yang, D. Experience-driven networking: A deep reinforcement learning based approach. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 15–19 April 2018; pp. 1871–1879.
32. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Desmaison, A. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
33. Ruder, S. An overview of gradient descent optimization algorithms. *arXiv* **2016**, arXiv:1609.04747.
34. Dixit, A.; Hao, F.; Mukherjee, S.; Lakshman, T.V.; Kompella, R.R. ElastiCon; an elastic distributed SDN controller. In Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Marina del Rey, CA, USA, 20–21 October 2014.
35. Gropp, W.; Gropp, W.D.; Lusk, E.; Skjellum, A.; Lusk, A.D.F.E.E. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*; MIT press: Cambridge, MA, USA, 1999; Volume 1.