# Preemptive SDN Load Balancing with Machine Learning for Delay Sensitive Applications

Abderrahime Filali, *Student Member, IEEE,* Zoubeir Mlika, *Member, IEEE,* Soumaya Cherkaoui, and Abdellatif Kobbane, *Senior Members, IEEE*

***Abstract*—SDN is a key-enabler to achieve scalability in 5G and Multi-access Edge Computing networks. To balance the load between distributed SDN controllers, the migration of the data plane components has been proposed. Different from most previous works which use reactive mechanisms, we propose to preemptively balance the load in the SDN control plane to support network flows that require low latency communications. First, we forecast the load of SDN controllers to prevent load imbalances and schedule data plane migrations in advance. Second, we optimize the migration operations to achieve better load balancing under delay constraints. Specifically, in the first step, we construct two prediction models based on Auto Regressive Integrated Moving Average (ARIMA) and Long Short-Term Memory (LSTM) approaches to forecast SDN controllers' load. Then, we conduct a comparative study between these two models and calculate their accuracies and forecast errors. The results show that, in long-term predictions, the accuracy of LSTM model outperforms that of ARIMA by 55% in terms of prediction errors. In the second step, to select which data plane components to migrate and where the migration should happen under delay constraints, we formulate the problem as a non-linear binary program, prove its NP-completeness and propose a reinforcement learning algorithm to solve it. The simulations show that the proposed algorithm performs close to optimal and outperforms recent benchmark algorithms from the literature.**

***Index Terms*—Software Defined Networking, Multi-Access Edge Computing, Machine Learning, Reinforcement Learning, Predictions, Load Balancing, Migration.**

## I. Introduction

A distributed architecture of the SDN control plane is the appropriate solution to overcome the present issues of the centralized architecture, especially in large scale networks [1]. Specifically, a distributed architecture provides high scalability and higher processing capacity for mobile applications at the control plane, while avoiding traffic congestion, high delays and single point of failure (i.e. bottleneck) issues [2]. In a distributed control plane architecture, see Fig. 1, the network is horizontally partitioned into several disjoint areas called control domains, where each control domain is managed by a single controller. This distributed architecture is currently coupled with several standards of existing and emerging technologies, notably Network Function Virtualization (NFV), Service Function Chaining (SFC) and Multi-Access Edge Computing (MEC) [3]. Moreover, it is a key component in the 5G network [4]. Being part of all these technologies, the distributed SDN architecture has, indeed, the ability to sustainably ensure the performance required by the target applications in the data plane, notably delay-sensitive applications [5]–[7].

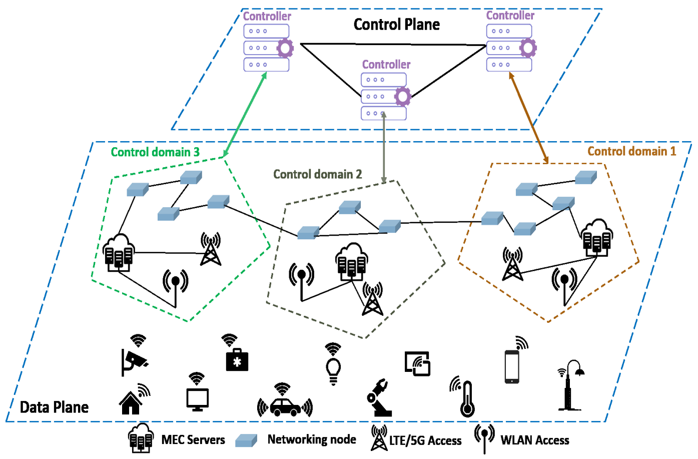


Fig. 1: A distributed SDN Mobile Networks Architecture

However, such an architecture gives rise to load distribution problems which can seriously affect the scalability of the control plane and decrease the exploitation of its resources (e.g. the controllers' processing capacity) [8]. In large-scale networks, where the number of applications is huge and traffic fluctuations are permanent, each controller should process and respond, as fast as possible, to thousands of requests received from the data plane components. Accordingly, this dynamic network traffic can lead to a load imbalance among controllers, i.e. some controllers will be overloaded while others will be underloaded. When a controller is overloaded, its response time to data plane components requests increases. Therefore, load balancing between SDN controllers is necessary to minimize latency and to efficiently exploit the control plane resources.

The data plane component migration is an efficient solution that is widely used to balance the load between controllers [9]. Indeed, when the load on controllers becomes disparate, some data plane components are migrated from overloaded controllers to underloaded controllers, which can fairly balance the load between controllers. A migration operation of a data plane component consists in changing its control domain through a four-phase mechanism [10]. Although the data plane component migration is a key-enabler to achieve load balancing in the control plane, it presents several challenges such as where the migration should happen (i.e. determining the overloaded and the underloaded controllers) and which data plane component should be migrated. To overcome these challenges, a large variety of algorithms and schemes have been proposed in recent years [11]–[21]. However, we identify two important research gaps in these works. First, all of them propose reactive mechanisms for data plane component migration. A reactive mechanism triggers a migration operation after detecting a controller overload, which makes it congested for a certain period of time. Consequently, the response time of the overloaded controller increases during this period. Second, the cost of a migration operation has not been defined in terms of delay, which can decrease the migration efficiency.

In this paper, we aim to preemptively balance the load between controllers for delay sensitive applications on the edge. Therefore, in order to fill the aforementioned research gaps, we: (1) propose a long-term prediction model that forecasts the controllers' load; and (2) solve the load balancing of the distributed SDN architecture by optimizing the trade-off between load balancing and migration operation costs that are based on the response times of SDN controllers. Long-term predictions allow to detect a load unbalanced in the control plane and, thus, react proactively by scheduling migration operations in advance. This proactive mechanism not only prevents controllers from being overloaded, but also allows a careful selection of which data plane components should be migrated and where the migration should happen. Consequently, we formulate the SDN load balancing problem, called SDN Load Balancing for Delay Sensitive Applications (LBDSA), as an optimization program where the objective is to minimize, through migration operations, a load balancing factor combined with a migration operation cost. Due to this proactive mechanism as well as the design of LBDSA model, the delay sensitive application requirements are respected.

To summarize, the novelty of this work lies in two main parts. The first part is a comparative study between Autoregressive Integrated Moving Average (ARIMA) [22]—a traditional stochastic model—and a machine learning prediction model—Long Short-Term Memory (LSTM) [23]. In the second part, we define the LBDSA problem and propose a reinforcement learning algorithm to solve it. The key contributions of this work are summarized as follows:

- We build and evaluate two prediction models, one based on ARIMA and the other on LSTM.
- We provide a performance analysis comparing the accuracy of the short and long term SDN controller load predictions of the ARIMA and LSTM models.
- We model LBDSA as a non-linear binary program and study its NP-completeness.
- Due to the NP-completeness result, obtaining an optimal solution to LBDSA is computationally expensive. Thus, we propose a reinforcement learning algorithm, called 2WSLS, that is based on the well-known Win-Stay-Lose-Shift (WSLS) learning policy [24], [25].
- We evaluate the performance of 2WSLS against the optimal solution and two state-of-art SDN load balancing algorithms [17] and [11]. We show that 2WSLS is close to optimal and outperforms the benchmark algorithms.

The rest of the paper is organized as follows. Section II discusses recent works on load balancing in the SDN control plane based on data plane components migration operations. Section III introduces the ARIMA and LSTM prediction models. Sections IV constructs the ARIMA and LSTM models used for long-term predictions and evaluates the obtained models. Section V compares the performance of the constructed models in terms of accuracy for short-term and long-term predictions. Section VI and Section VII present, respectively, the system model for the SDN load balancing optimization and the mathematical programming formulation of LBDSA and its NP-completeness. Section VIII presents the proposed learning algorithm, 2WSLS, and explain its operations. Section IX highlights the performances of 2WSLS and discusses the obtained results. Finally, section X concludes the paper.

## II. Related Work

The works presented in this section can be divided into those that: (i) do not consider any costs related to data plane component migration operations [11]–[16] and (ii) consider data plane component migration costs [17]–[20].

In [11], the authors propose a load balancing mechanism based on the real response time of controllers. They also use this response time to define an appropriate threshold to decide whether a controller is overloaded or not. Overloaded controllers simultaneously migrate the heaviest switches to underloaded controllers. However, the proposed mechanism can increase the control traffic overhead since migration operations are performed simultaneously. The authors of [12] use a load diversity factor, which is the ratio of their loads, to find overloaded controllers and underloaded controllers. When the diversity is caused by an overloaded controller, some switches should be migrated from its domain of control, and when it is caused by an underloaded controller, all switch under its control should be migrated. To solve this switch migration problem, they use a non-cooperative game in which the players are the immigration controllers. In [13], a modified version of the Hungarian algorithm is used to assign switches to controllers. In the assignment process, the proposed algorithm considers the round-trip time between the switches and the controllers as well as the current load of the controllers. The work in [14] solves the switch-controller assignment problem in a two-phase manner. First, based on the flow paths, a greedy set coverage algorithm is used to form a minimum number of control domains that contain the most switches on the paths. The obtained assignment is used in the second phase by a coalition game strategy to improve the load balancing between the controllers. In this phase, switches can change their coalitions to achieve a Nash equilibrium. However, the proposed approach results in a higher number of migration operations. To solve the dynamic controller-switch assignment problem in a long-term horizon, authors in [15] decompose it into a series of one time-slot assignment problems using the Randomized Fixed Horizon Control (RFHC) framework. In each time slot, given the request rate of each switch, a two-phase algorithm is used to assign switches to controllers. In the first phase, the assignment problem is modeled as many-to-one matching game. In the second phase, the obtained result from the first phase is used as an input for a coalition game to achieve a Nash equilibrium. To maximize the efficiency of the flow setup between controllers and switches, the authors in [16] define a long-term optimization problem for controller placement and controller-switch assignment. The placement of controllers is planned considering eventual future switch migration operations. After the controllers are placed, switch migration operations can be performed by choosing switches with the highest load from overloaded controllers. To sum up, the works in [11]–[16] do not consider any migration cost to evaluate the effectiveness of migration operations.

The authors of [17] deal with the switch migration problem by performing two types of switch movements. They propose a heuristic algorithm that uses shift and swap moves to migrate switches from overloaded controllers to underloaded controllers. A shift move is the classical migration operation of a switch while a swap move is when two switches exchange their master controllers through migration operations. The cost of migrating a switch depends on the latency between (i) the controllers involved in the migration operation and (ii) the migrated switch and the controllers involved in the migration operation. The swap move is very useful for load balancing between controllers when the shift move is impossible. However, performing swap moves hugely increase the migration cost in terms of time and control plane overhead. The authors of [18] define a load diversity factor to divide controllers into overloaded controllers and underloaded controllers. The load diversity factor between two controllers is the ratio of their loads and when this ratio exceeds a predetermined threshold, switch migration is performed. The overloaded controller chooses to migrate the switch which has a large latency to it and consumes less resources. A destination controller is selected to maximize the migration efficiency which is defined as the ratio of load balancing variation to migration cost. Similarly, the authors of [19] use the same load diversity factor but they consider the synchronization and routing overheads when calculating the load of a controller. In [20], a multi-criteria decision method called Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) is used to choose the target controller. These criteria are based on resource consumption and hop distance. Also, they use a probabilistic model in which the switch with a low resource consumption is selected for migration. The cost of a migration operation is mainly defined, in [18]–[20], by the load added to the control traffic overhead without considering the time required for this migration operation, which can decrease the migration efficiency.

## III. SDN Controller Load Prediction Models

In this section, we define the SDN controller load considered in this work and the basic concepts of time series. Then, we review the essential mathematical background on how the predictions are made using ARIMA and LSTM models.

### A. SDN controller load

The load of an SDN controller at time $t$, denoted by $L(t)$, can be defined as the sum of the requests received from (i) the managed data plane components, (ii) other SDN controllers, or (iii) any network entity which can communicate with this controller. In this work, we consider the requests sent by the data plane components as the most important load of an SDN controller [26], in particular, the $Packet\_In$ messages of the OpenFlow protocol.

To predict the load of each SDN controller in a distributed control plane architecture, we assume that there exists a root controller that has a global view of the network [27]. Indeed, the root controller is connected to the other controllers and does not manage any data plane component. In such an architecture, all controllers update the root controller of their control domain state. Thus, the root controller knows all information about each controller, including their load history. Based on the load history of the SDN controllers, the root controller can make load predictions for each controller.

### B. Time series and predictions

Time series is an ordered sequence of measured data points (i.e. observations) over a period of time, usually at regular time intervals. The importance of the order lies in a possible existence of dependencies between the observations, thus, changing the order could change the meaning of the data. In our case, we are working on a single variable, which is the SDN controller load $L(t)$. Therefore, the studied time series is termed univariate. Moreover, the controller load is measured every second, which makes it a discrete time series. The major benefit of analyzing the load of SDN controllers as a time series is to predict its future values. Make predictions is about forecasting the future with no error or as little error as possible. For this reason, a suitable prediction model must be trained by using the time series data. Also, the parameters of the model should be fitted to extract patterns from the data. Depending on the studied problem, predictions can be performed for short-term or long-term prospects. For the purpose of our study, which is the prediction of any load imbalance in the control plane to schedule data plane component migration operations in advance, the long-term forecast is the appropriate model.

### C. Multi-step load prediction

Our objective through load prediction is to detect any load imbalance in the control plane. Hence, it is possible to schedule migration operations for data plane components in advance. To perform effective migration operations, several tasks need to be executed such as choosing the data plane components to migrate and their new control domains. For this reason, long-term prediction is the appropriate model that allows a careful selection of which data plane components need to migrate and where the migration should happen. Long-term prediction refers to perform multi-step forecasting of SDN controller load. In this work, we used Multiple Input and Multiple Output (MIMO) strategy to predict multi-step of the SDN controller load. MIMO strategy uses the same past observations to predict the entire forecast sequence in one shot manner with the same fitted model. Let $L(t+w)$ be the predicted SDN controller load in step $w$ (i.e. at time $t+w$), the prediction of the sequence $\{L(t+1), L(t+2), ..., L(t+w)\}$ can be defined by $F(L(t), ..., L(t-p+1))$, where $F$ is vector-valued function and $p$ is the number of observations.

### *D. Load prediction using ARIMA and LSTM*

In order to investigate the effectiveness of time series models to make predictions with higher accuracy and lower forecast errors, our study aims to compare a traditional stochastic and a machine learning models, namely ARIMA and LSTM, respectively. The main reasons that led us to choose ARIMA as a representative of stochastic forecasting models are the non-stationary property of our dataset and that ARIMA is considered to be the most general model among linear models. As a representative of the machine learning models, we preferred LSTM for many reasons, namely our data can be nonlinear, it is dynamic and can comprise high autocorrelations across different periods of time. Also, LSTM can preserve and train the features of data for a longer period of time. Moreover, ARIMA and LSTM have been widely exploited in different forecasting domains [28], [29].

*1) Load prediction using ARIMA:*

In an Autoregressive Integrated Moving Average (ARIMA) model, the future values of a variable are supposed to be a linear combination of several past values (i.e. observations) and random errors. In time series analysis and prediction applications, the ARIMA model is the general model of the Autoregressive (AR) model, the Moving Average (MA) model, and the combination of AR and MA (ARMA) models. Indeed, an autoregressive model of order $p$, denoted by AR($p$), is based on the idea that the current value $L(t)$ of the time series at time $t$, i.e. the SDN controller load in our case, can be expressed as a linear combination of $p$ past values, with a random error $E(t)$. Rather than using past observations of the forecast variable, a moving average model of order $q$, denoted by MA($q$), uses the previous errors as predictors for future outcomes. ARMA($p, q$) model combines the two previous models as follows : $L(t) = E(t) + \sum_{i=1}^{p} \phi_i B^i L(t) + \sum_{j=1}^{q} \theta_j B^j E(t)$, where $\phi_i$ and $\theta_j$ are the parameters of the model and $E(t)$ is a random error with zero mean and constant variance. $B^i * \ L(t) = L(t-i)$ is the backshift operator.

The ARMA model can only be used for stationary time series. However, in many situations, the statistical properties (e.g. mean, variance) of a time series change over time, making it non-stationary. For this reason, the ARMA model has been generalized by the ARIMA model to integrate the case of non-stationary time series. A non-stationary time series becomes unpredictable and cannot be modeled or predicted. Thus, working with stationary time series is easier because they can be analyzed and forecasted. In an ARIMA $(p, d, q)$ model, a non-stationary time series can be transformed to a stationary series through the differentiation process. The ARIMA $(p, d, q)$ model is formulated as $\big(1 - \sum_{i=1}^{p} \phi_i B^i\big)(1 - B)^d L(t) = \big(1 + \sum_{j=1}^{q} \theta_j B^j\big) E(t)$, where $d$ is the degree of differencing and $(1 - B)^d L(t) = L(t) - L(t-d)$.

*2) Load prediction using LSTM:*

As a type of Recurrent Neural Network (RNN), the LSTM network has a powerful ability to remember information from earlier stages in order to make predictions over an extended horizon. In fact, the main idea behind these networks is to give the model visibility about the previous stages while generating predictions for the current input. Accordingly, previous states need to be remembered, which allows the network to learn and exploit the sequential observations when forecasting next steps. The LSTM network has the particularity to selectively remember pattern for long sequences. An LSTM network is a set of memory blocks called cells which are responsible for filtering, eliminating and adding information through three gates, namely the input gate, the forget gate and the output gate. The forget gate is used to decide whether the information from the previous cell should be thrown. The decision of which part should be memorized from the current input and hidden state is made by the input gate. The output gate acts as a filter to determine what will be conducted out of the cell.

## IV. Construction of Prediction Models

This section introduces the source of the dataset used in this work and how it is prepared before being used by prediction models. Next, we describe the process of constructing the ARIMA and LSTM models and evaluate the obtained models.

### *A. Simulation Setup*

All simulations and experiments, namely dataset generation, ARIMA and LSTM modeling, training process and predictions, were performed on a laptop with an Intel Core i7-8750H processor, 16 GB of RAM and NVIDIA GeForce GTX 1070 graphic card. The software environment used in this work includes Keras version 2.2.4 with TensorFlow 1.14.0 backend.

### *B. Dataset Description and Preparation*

The dataset used in this work is the SDN controller load, in other words, the sum of the number of requests sent by the data plane components to the controller. In order to have this type of data, we created an SDN network using the Mininet emulator. We used RYU [30] as an SDN controller and the OpenFlow protocol version 1.3 as a southbound interface between the data plane components and the SDN controller. As data plane components, we used virtual switches that support the OpenFlow protocol. We built a data plane layer that contains 38 virtual switches. The virtual switches are randomly connected, and each switch is connected to 3 hosts. To avoid network architecture dependency, different data plane topologies are created by regularly and randomly changing the connection between switches throughout the generation period of the dataset. In order to have dynamic traffic as in a real network, we used Iperf to generate UDP and TCP traffic, the D-ITG generator to produce VoIP and DNS traffic and Wget to generate HTTP traffic. Also, the generated VOIP and DNS traffics follow an exponential and uniform distribution, respectively. These setups enable the dynamicity in the generated traffic of the constructed network. All these traffics are generated by randomly selecting hosts. The traffic generators are executed for 6 hours while capturing the exchanged messages between the SDN controller and the virtual switches. Then, the captured traffic was filtered to keep only the requests received by the SDN controller (e.g. $Packet\_In$ messages).

Dataset preparation is crucial to achieve better forecasting performance. First, we dealt with noisy and inconsistent data

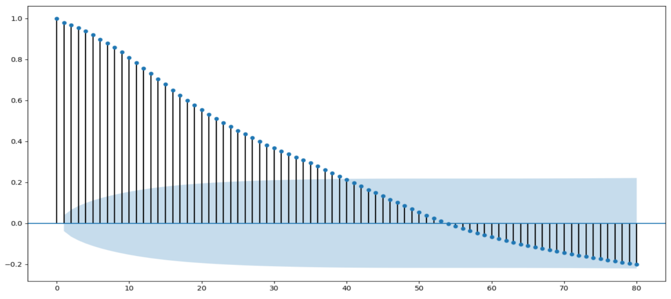


Fig. 2: Autocorrelation function of the original series

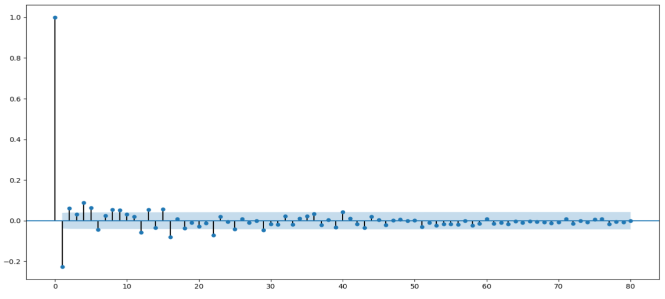


Fig. 3: Autocorrelation function of 1st differencing

by replacing outliers with the average of the sequence data. Since the processing capacity of an SDN controller is defined by the number of packets that it can handle per second, the obtained traffic is sampled every second. Thus, our data is transformed into time series with a time step size equal to 1 second. Each value in the time series represents an observation of the SDN controller load. Then, the data is divided into two partitions, train and test subsets, while maintaining the temporal order of observations. Training data is used to fit the model by adjusting its parameters. Test data is used to evaluate the obtained model after the training phase and is not used in learning phase. Our data is large enough, thus, using 80% for training and 20% for test keeps both subsets highly representative.

## *C. ARIMA Modeling*

An ARIMA model, denoted by ARIMA$(p, d, q)$, is composed of integrated I$(d)$, autoregressive AR$(p)$ and moving average MA$(q)$ components, where $d$ is the number of differencing required to make the time series stationary; $p$ is the number of lag observations included in the model; and $q$ is the number of lagged forecast errors in the prediction equation. In order to build an effective forecasting model, the three parameters $d$, $p$ and $q$ must be carefully determined.

*1) Differencing Order:*

The purpose of differencing a time series is to make it stationary (i.e. the properties of the time series doesn't depend on the time). In order to study the stationarity and define the right degree of differencing, we analyzed the AutoCorrelation Function (ACF). A time series is said to be stationary if the ACF plot quickly reaches zero, while the ACF of a non-stationary time series slowly decreases to zero. In Fig. 2, we observed that the ACF of the original time series has positive autocorrelations for a large number of lags (more than 10 lags), thus, the time series needs to be differentiated. Also, if the autocorrelation of lag 1 is too negative, less than -0.5, the time series is over-differentiated. Therefore, looking at the ACF plot of the 1st differencing, see Fig. 3, the lag 1 autocorrelation is greater than -0.5, which indicates that the time series is not over-differentiated. Accordingly, the optimal degree of differencing for this time series is $d = 1$. To remove all traces of autocorrelations in the residuals, autoregressive and moving average terms should be added.

*2) Autoregressive and moving average order:*

The next step in building the ARIMA model for our time series is the identification of the autoregressive degree $p$ and the moving average degree $q$. ARIMA model selection requires fitting multiple models on the prepared dataset (i.e. estimating the performance of several models) and then choosing the best one of them. A model is considered the best if its performance on the training dataset is good and its complexity is low. We can use the Partial AutoCorrelation Function (PACF) and the ACF to approximately figure out how many autoregressive and moving average terms, respectively, are required to remove any autocorrelation in the stationary time series. However, inspecting the PACF and ACF plots may not lead us to identify the optimal AR and MA degrees. In order to obtain an optimal ARIMA model we employed *Akaike Information Criterium* (AIC) method to select AR and MA degrees. AIC is an effective way to choose a model that has a good fit but few parameters (i.e. $p$ and $q$ in our case) and is given by $AIC = 2K - 2ln\,(L)$, where $K$ is the number of parameters in the model (e.g. $p$, $q$ degrees) and $L$ is the likelihood of the data. Since we determined before the degree of differencing, $d = 1$, we varied the values of $p$ and $q$ to find the smallest AIC result. The best model is ARIMA (4,1,3) that corresponds to an AIC equals to 28997.957.

*3) Evaluation of the obtained ARIMA model:*

All steps previously followed to build an adequate ARIMA model with the time series, namely the identification of the differentiation degree, the AR degree and the MA degree, aim to build an effective and parsimonious model. In order to evaluate the obtained model, i.e. ARIMA (4,1,3), we examined the residuals plot to ensure there are no patterns. Fig. 4 and Fig. 5 illustrate the residual errors plot and the density plot, respectively. The residual errors seem to be stationary, i.e. fluctuation around a mean of zero, and the density plot shows a Gaussian distribution with mean zero. Fig. 6 depicts the ACF plot of the residual errors in which we can observe that all autocorrelations are within the threshold limits, i.e. no autocorrelations between the residuals. Therefore, the fitted ARIMA(4,1,3) model is excellent.

## *D. LSTM Network Modeling*

To select an LSTM network that can make predictions with high accuracy, several hyperparameters, such as the number of hidden layers, the number of neurons of each layer, the loss function, the optimizer and the number of previous observations, need to be tuned. The hyperparameter tuning process

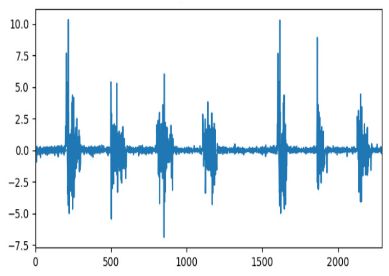


Fig. 4: Residual errors of the fitted ARIMA(4,1,3) model.

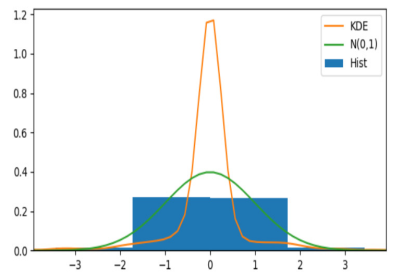


Fig. 5: Residual errors density of the fitted ARIMA(4,1,3) model.

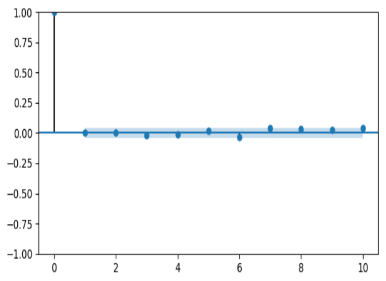


Fig. 6: Autocorrelation function of residual errors.

refers to find the best combination of hyperparameters to obtain a better performance. Before tuning the hyperparameters of the LSTM network, we first differentiated the time series to make it stationary. According to the construction process of the ARIMA model, performing first order differencing is sufficient to make the time series stationary. Then, we transformed the time series values to be on a scaled between 0 and 1. Data scaling is important because it avoids large input values that could slow down the leaning and convergence of the LSTM network. To provide an unbiased evaluation of the fitted model on the training dataset, we used the validation dataset. The model consults the validation dataset to validate its performance and not to learn from it.

In this work we performed the random search method to find the best solution for the built LSTM model. Based on this method, random combinations of the hyperparameters are used to train the LSTM model. The utilization of random search method allows to explicitly control the used parameters and the number of attempted combinations. After training and tuning multiple combinations, the most important hyperparameters of the retained LSTM model are summarized in Table II.

| **Hyperparameter** | **Value** |
|---|---|
| Number of hidden layers | 3 |
| Number of neurons in each layer | 200 |
| Activation function in all layers | Relu |
| Optimizer | Adam |
| Loss function | Mean Squared Error |
| Number of previous observations | 200 |

TABLE I: Retained Hyperparameters for LSTM network.

## V. Prediction Performance Evaluation

To compare the performance of ARIMA and LSTM in terms of short-term and long-term prediction, we present, in this section, the metrics used to evaluate the studied models. Then, we show the prediction results of each model and compare their performance in SDN controller load prediction.

### *A. Evaluation Metrics*

To assess the performance of the two prediction models, i.e. ARIMA and LSTM, in terms of accuracy, we used the Root Mean Square Error (RMSE), the Coefficient of Determination $R^2$, the Mean Absolute Error (MAE) and the Mean Absolute Percentage Error (MAPE) as evaluation metrics. For each step $w$ (i.e. at time $t+w$). In RMSE formula the errors are squared before they are averaged, which penalizes large errors. $R^2$ the square of the correlation between the actual and predicted values. MAE is the average of all absolute errors of predictions and it is a linear score, which means that all the individual differences are weighted equally in the average. MAPE measures the accuracy of a model as a percentage.

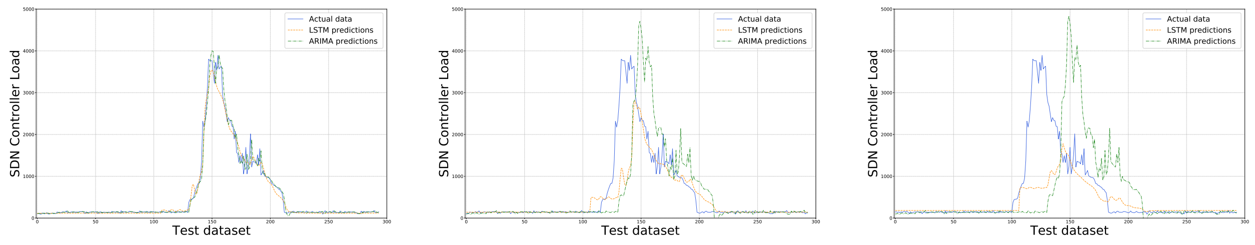
| **Metrics** | **Forecasting step** | **ARIMA** | **LSTM** |
|---|---|---|---|
| MAE | t+1 | 64.04 | 77.82 |
| | t+15 | 353.94 | 204.74 |
| | t+30 | 573.71 | 296.95 |
| MAPE(%) | t+1 | 12.49 | 17.40 |
| | t+15 | 50.02 | 32.16 |
| | t+30 | 99.99 | 45.32 |

TABLE II: MAE and MAPE scores of ARIMA and LSTM.

### *B. Prediction Results*

Using ARIMA and LSTM models, we performed multi-step predictions. The experiment involved forecasting the controller load from 1 to 30 seconds into the future. As mentioned before, we used the 20% of the dataset (i.e. dataset of tests) to compare ARIMA and LSTM in terms of prediction accuracy.

Fig. 7a, 7b and 7c illustrate the prediction results of SDN controller load for step 1, 15 and 30, respectively. From figure 7a, we observe that both models achieved good forecast performance for short-term predictions t+1. In fact, ARIMA and LSTM predictions for step t+1 closely follow the actual (i.e. real values) SDN controller load, displaying similar patterns. On the one hand, ARIMA prediction results slightly outperform those of LSTM in step t+1 with 4.91% and 13.78 of difference in terms of MAPE and MAE, respectively. On the other hand, from Table III, LSTM model has the minimum score in terms of MAE and MAPE for step t+15 and t+30. We can note that LSTM model can follow some peaks of SDN controller load fluctuations, Fig. 7b and Fig. 7c. The percentage error (i.e. MAPE) of the ARIMA model predictions in step 30 reached 99.99% while the MAPE of the LSTM model stopped at 45%. With this difference of 55%, the performance of LSTM is much better in long-term predictions than the ARIMA model. In addition, ARIMA predictions



(a) $t+1$. (b) $t+15$. (c) $t+30$.

Fig. 7: SDN controller load prediction results of ARIMA and LSTM models for different steps.

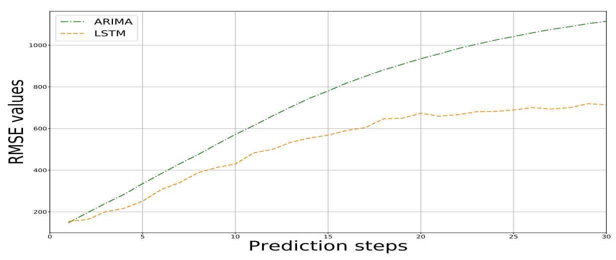


Fig. 8: RMSE of ARIMA and LSTM from t+1 to t+30.

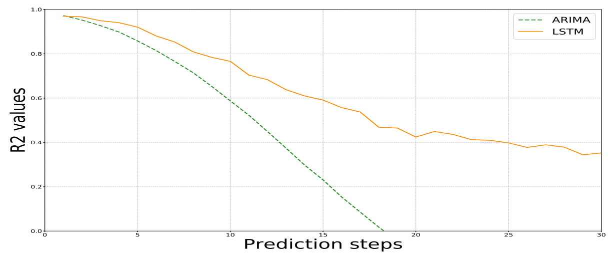


Fig. 9: $R^2$ of ARIMA and LSTM from t+1 to t+30.

results for the three steps, namely t+1, t+15 and t+30, have roughly the same outline, which is explained by the fact that an ARIMA model try always to follow the mean value.

Furthermore, to compare the accuracy of the ARIMA and LSTM models, the RMSE score and the coefficient of determination $R^2$ were plotted from step t+1 to step t+30. Fig. 8 shows that the RMSE score increases proportionally with the prediction steps. In fact, the errors in the prediction results of the two models, i.e. ARIMA and LSTM, become high where the forecasts are made for farther stages in the future. However, the LSTM model has better results in long-term predictions than ARIMA with a difference of 400.31 in terms of RMSE score for t+30, while ARIMA is still lightly good in short-term predictions. Similarly, we plotted and analyzed the $R^2$ scores in Fig. 9. $R^2$ varies between 0 and 1, i.e. between a weak prediction results when $R^2$ is close to 0 and a good prediction results when $R^2$ is close to 1. From Fig. 9, we can see that the $R^2$ scores of both models decrease tend to 0 as the prediction step increases. However, the $R^2$ of ARIMA model quickly reaches 0, starting from step 18, while the $R^2$ of LSTM achieves 0.35 at step t+30.

Using the predefined evaluation metrics, we compared the accuracy of ARIMA and LSTM to predict, in short-term and long-term, the load of SDN controllers. When a load imbalance is predicted in a distributed SDN control plane, we need to judiciously choose which data plane component to migrate and where the migration should happen to guarantee an efficient migration operation. Thus, the next step is to properly design a migration-based model to balance the load between controllers while considering a migration cost.

## VI. System Model

### A. Controller Response Time

We consider a distributed SDN control plane architecture composed of a finite set of SDN controllers $C = \{c_1, c_2, c_3, ..., c_n\}$, with $|C| = n$. Each controller $c_k$ has a limited processing capacity $\alpha_k$ defined as the possible number of requests that can be processed per time unit,and we denote by $\alpha = \{\alpha_1, \alpha_2, \alpha_3, ..., \alpha_n\}$ the set of capacities of the controllers. The data plane contains a finite set of components $S = \{s_1, s_2, s_3, ..., s_m\}$, with $|S| = m$. In a distributed control plane architecture, each controller manages a sub-set of data plane components. Accordingly, we define the association between controllers and data plane components as a binary matrix $X_{n\times m}$, where $x_{ki} = 1$ if and only if $c_k$ manages the data plane component $s_i$. As explained before, the load of an SDN controller is the sum of the *Packet_In* messages sent by the data plane components. Therefore, the load of the $k^{th}$ controller can be calculated as $L_k = \sum_{i=1}^{m} x_{ki} * \ l_i$, where $l_i$ is the number of requests sent by $s_i$.

In order to calculate the response time of a controller, we consider the following assumptions: (i) the arrival process of the number of requests follows a Poisson distribution (i.e. *Packet_In* messages) whereas the inter-arrival times follows an exponential distribution; (ii) the processing times (i.e. service times) of the requests, within an SDN controller, are independent of each other, independent of the arrival process and obey an exponential distribution. Accordingly, the SDN controller can be modeled as an M/M/1 queuing system. By applying Little's law, we calculate the response time of controller $c_k$ as follows:

$$T_{r_k} = \frac{1}{\alpha_k - L_k}. \tag{1}$$

To meet the requirements of delay sensitive applications, the response time of controllers should be adequate with the latency required by these applications. Thus, the response time of each controller must not exceed a threshold $T_{th}$.

From Eq. 1 we notice that the response time $T_{r_k}$ of controller $c_k$ is proportional to its load $L_k$. Thus, performing load balancing in the control plane maintains fairness between controllers in terms of response time. To measure how fairly the load is distributed among controllers, we define the load balancing factor *LB* as follows:

$$LB = \sum_{k=1}^{n} \sum_{k'>k}^{n} \left| T_{r_k} - T_{r_{k'}} \right| . \tag{2}$$

In Eq. 2, the lower the difference between the response times, the greater is the load balancing in the control plane.

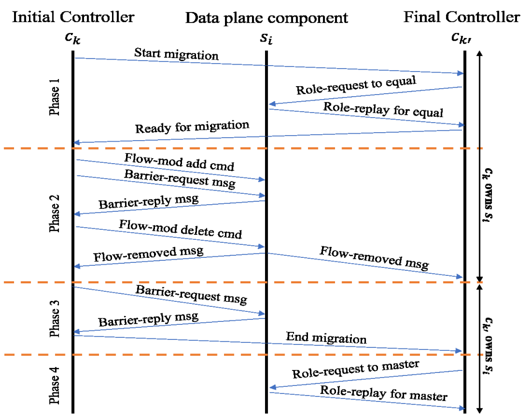


Fig. 10: Messages exchanged between the data plane component, the initial controller and the final controller during a migration operation [10].

## B. Migration Protocol

In a migration operation, when a data plane component $s_i$ migrates from controller $c_k$ to controller $c_{k'}$, the former is called the initial controller and the letter is called the final controller. In order to ensure a normal operation of $s_i$, $c_k$ and $c_{k'}$ during a migration operation, a migration protocol [10] should be implemented, which includes four phases, as shown in Fig. 10: (i) change role of the final controller to Equal, (ii) insert and remove a dummy flow, (iii) flush pending requests with a barrier, and (iv) change role of the final controller to Master. The first phase consists in changing the role of the final controller from Slave to Equal. For this reason, the final controller should respond to two received messages, the first one is sent by the initial controller (i.e. Start migration) and the second one is an asynchronous message sent by the migrated data plane component (i.e. Role-reply message for Equal). In the end of this phase, the final controller sends to the initial controller a message indicating that it is ready for the migration operation. Consequently, in the second phase, the initial controller sends two successive messages (i.e. a Flow-mod add command and a Barrier request message) to the migrated data plane component as a response to the last message from the final controller. Next, in response to the Barrier-replay message, the initial controller removes the flow table entry added by the Flow-mod command. In the third phase, the initial controller sends another Barrier message to the migrated data plane component as a consecutive command upon receiving the Flow-Removed message in the second phase. This message is used by the controller to ensure that all previous requests are processed by the data plane component before detaching from its management as a Master controller. Then, as a response to the Barrier-replay message, the initial controller sends a message to the final controller indicating the end of the migration. Finally, in the fourth phase, the final controller changes its role from Equal to Master by sending a Role-request message to the migrated data plane component.

By analyzing the behavior of the initial and the final controllers in the four-phase migration protocol, we can notice that the initial controller should react four times after receiving four messages and the final controller should react three times after receiving three messages. The four responses of the initial controller are: (i) Flow-mod add command and Barrier request message, (ii) Flow-mod delete command, (iii) Barrier request message, and (iv) End-migration.The three responses of the final controller are: (i) Ready for migration, (ii) Role-request message to Equal, and (iii) Role-request message to Master.

## C. Migration Cost

We denote the initial association matrix by $X_{n\times m}^{initial}$ and the final association matrix, after performing migration operations, by $X_{n\times m}^{final}$. The logical XOR operation between $X_{n\times m}^{initial}$ and $X_{n\times m}^{final}$ defines the binary migration matrix $M_{n\times m}$, i.e. $M_{n\times m} = X_{n\times m}^{inital} \oplus X_{n\times m}^{final}$, where $m_{ki} = 1$ if and only if $c_k$ is either an initial or a final controller of $s_i$.

To present the controllers involved in migration operations as initial controllers, we define the binary matrix $M_{n\times m}^{initial} = M_{n\times m} \bullet X_{n\times m}^{inital}$, where $\bullet$ is the AND operator and $m_{ki}^{initial} = 1$ if and only if $c_k$ is the initial controller of $s_i$. Similarly, we define $M_{n\times m}^{final} = M_{n\times m} \bullet X_{n\times m}^{final}$ to present the controllers involved in migration as final controllers, where $m_{ki}^{final} = 1$ if and only if $c_k$ is the final controller of $s_i$.

Using the OpenFlow-based migration mechanism, the migration operation of a data plane component is performed by exchanging a set of messages, as shown in Fig. 10, between this component, the initial controller and the final controller. Therefore, we define the migration cost of a data plane component based on the time required to exchange all of these messages. This needed time for the accomplishment of a migration operation depends on the transmission delay of these messages, the response time of the initial controller, the response time of the final controller and the response time of the data plane component. Since the response time of a controller is usually much greater than the transmission delay and the response time of the data plane component [31] [32], we assume that the cost of a migration operation depends only on the response time of the initial and final controller. Hence, the cost of migrating $s_i$ can be defined as follows:

$$T_{m_i} = \sum_{k=1}^{n} 4 * T_{r_k} * m_{ki}^{initial} + \sum_{k'=1}^{n} 3 * T_{r_{k'}} * m_{k'i}^{final} . \tag{3}$$

In Eq. 3, the first sum calculates the response time of the initial controller required to migrate $s_i$ and the second sum calculates the needed response time of the final controller. Since the initial controller should respond four times after receiving four messages and the final controller should respond three times after receiving three messages (see Section VI-B), we multiply the response time of the initial controller by four and that of the final controller by three. In addition, we define the total migration cost $T_m$ as follows:

$$T_m = \sum_{i=1}^{m} T_{m_i} \tag{4}$$

Performing many migration operations increases the migration cost, which affects the performance of controllers. Since we are studying the SDN load balancing problem for delay-sensitive applications on the edge [33], a significant migration cost heavily impacts the services offered by these applications. Accordingly, the objective of LBSDA is to minimize the *LB* factor through migration operations, but not at the expense of the migration cost. To optimally solve the LBDSA problem, we formulate it as an optimization problem in the next section.

## VII. Problem Formulation and NP-hardness

### A. Problem Formulation

In order to optimize the load balancing and the migration cost, we transform the two objectives, $LB$ and $T_m$, into a weighted sum by introducing a weight factor $\omega \in [0,1]$. Consequently, LBDSA can be formulated as follows:

$$\underset{X, M}{\text{minimize}} \quad \omega \times LB + (1-\omega) \times T_m \tag{5a}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ki} = 1, \forall i \in \{1,2,3,...,m\}, \tag{5b}$$

$$L_k < \alpha_k, \forall k \in \{1,2,3,...,n\}, \tag{5c}$$

$$T_{r_k} \leq T_{th}, \forall k \in \{1,2,3,...,n\}, \tag{5d}$$

$$x_{ki}, m_{ki}, m_{ki}^{initial}, m_{ki}^{final} \in \{0,1\}. \tag{5e}$$

Equation (5a) formulates the weighted summation of the two objectives. Constraints (5b) ensure that each data plane component must be assigned to exactly one controller. Constraints (5c) guarantee that the load of each controller should not exceed its processing capacity. Constraints (5d) state that the response time of each controller cannot exceed a defined threshold $T_{th}$ related to latency required by delay sensitive applications. Constraint (5e) lists the optimization variables.

Due to the non-linearity of Eq. 1, LBDSA is a non-linear binary programing problem.

### B. NP-hardness

We denote the LBDSA problem (Eq. 5) here by $P$. To prove that $P$ is NP-complete we (i) consider a decision version of the problem, (ii) show that $P$ belongs to the non-deterministic polynomial time class (NP), and (iii) reduce, on a polynomial-time, PARTITION problem [34] to $P$. In [35], the reduction from PARTITION problem has been used to demonstrate the NP-completeness of a similar load balancing problem in the SDN control plane. Since our problem is different from the one in [35], we cannot argue that our problem is NP-hard. A reduction from the problem of [35] to our problem is probably possible but it is not evident as both problems have different objectives and constraints. To present a simple proof, we use partition to prove the NP-hardness of our problem.

*Definition 1: PARTITION problem*
The PARTITION problem is a decision problem and is defined as follows: given a set $S$ of integers, the task is to find if $S$ can be divided into two subsets $S_1$ and $S_2$ such that the sum of all elements in $S_1$ equals the sum of all elements in $S_2$, i.e. $S_1 \cup S_2 = S$ and $\sum_{s_1 \in S_1} s_1 = \sum_{s_2 \in S_2} s_2$.

*Theorem 1: P is NP-complete*

*Proof:* We prove the theorem by restriction, i.e. we prove that a restricted version of $P$ is NP-complete. First, we prove that $P$ is in NP. This is true since the knowledge of an association matrix $X_{n \times m}$ between data plane components and controllers makes it is possible to verify, in polynomial time, that the processing capacity ($\alpha$) of the controllers has not been violated and the response time ($T_r$) of the controllers does not exceed the defined threshold ($T_{th}$).

PARTITION problem can be stated alternatively as follows: given a finite set $S$ and each element $s \in S$ present a load value $load(s) \in \mathbb{N}$, is there a subset $S' \subseteq S$ such that $\sum_{s \in S'} load(s) = \sum_{s \in S \setminus S'} load(s)$?

The restricted version of $P$ is constructed as follows. Let $C = \{c_1, c_2\}$, $\omega = 1$ and $\alpha_1 = \alpha_2$. Each $s_i$ represents an element $s \in S$ and has a number of requests (i.e. load) $l_i = load(s_i)$. Then, given a YES solution $S_1$ and $S_2$ for PARTITION problem, we can create an association matrix $X_{n \times m}$ as a solution to $P$ as follows, $x_{1i} = 1$ for all $i \in S_1$ and $x_{2i} = 1$ for all $i \in S_2$. We obtain $\sum_{i=1}^{m} x_{1i} \times l_i = \sum_{i=1}^{m} x_{2i} \times l_i$, so $L_1 = L_2$. Since, we choose $\alpha_1 = \alpha_2$, then $T_{r_1} = T_{r_2}$. Thus, we have a solution $x_{ki} \in \{0,1\}, \forall i \in \{1,2,...,m\}$ and $\forall\ k \in \{1,2\}$ with $LB = 0$. The reverse part can be done similarly. Since PARTITION problem is NP-complete, the reduction is done in polynomial-time and $P$ is in NP, this proves that $P$ is NP-complete which proves the theorem.

We showed that LBDSA is NP-complete, and thus it is very challenging to solve optimally, specifically in large-scale networks. To overcome this issue, we invoke machine learning tools for performing the association task between data plane components and SDN controllers in a computationally efficient manner. In particular, we use Reinforcement Learning (RL) to efficiently obtain the association solution to LBDSA. In the next section, we describe our proposed RL algorithm.

## VIII. Reinforcement Learning Algorithm

In RL, an agent interacts with its environment and its aim is to make suboptimal decisions based on its previous experience. The major challenge the agent is facing is to find a tradeoff between exploitation, the desire to make the best decision given current information, and exploration, the desire to try a new decision which may lead to better results. While interacting with its environment, the agent observes the state of the environment and performs an action using a certain policy. According to the performed action, the state of the environment changes and the agent receives a reward or penalty. The agent would like to maximize its accumulated reward (or minimize its accumulated penalty) in the long run. Every time the agent observes the state of the environment, performs an action it learns from this experience and refines its policy. This process is repeated until a suboptimal decision is found. The details are given in the sequel.

## *A. Preliminary Definitions*

First, we assume that the root controller includes a RL agent that knows the necessary network information about this architecture, namely the control domain of each controller, the load of controllers and the response time of controllers. In other words, we assume that the root controller represents the RL agent which will perform the learning process.

For LBDSA, we define the state set, action set and rewards as follows.

*States*. The observed state by the RL agent is given by the data plane components managed by each controller, the load of controllers, the response time of controllers (Eq. 1) and the achieved load balancing level *LB*.

*Actions*. An action represents the decision to choose a controller $c_k \in C$ as the final controller for a data plane component $s_i \in S$. To define the global action of the RL agent, first we define the set of actions available to migrate $s_i$ as $C$, i.e. the set of controllers. We denote the action for data plane component $s_i$ by $a_i \in \{c_1, c_2, ..., c_n\}$. If $a_i$ is the initial controller for data plane component $s_i$, it means that this data plane component will not be migrated.

The RL agent simultaneously chooses an action for each data plane component $s_i$. Therefore, we define the global action, denoted by $a$, of the RL agent and it's represented as a vector of size $|S| = m$, where $a \triangleq [a_1, a_2, ..., a_m]$. Note that a global action vector $a$ is equivalent to an association matrix $X_{(n \times m)}$ because each element $a_i$ of vector $a$ corresponds to the master controller of the data plane component $s_i$. A global action $a$ is ***feasible*** if it meets the constraints (7b), (7c) and (7d) of the LBDSA problem.

*Rewards*. After performing the actions for all data plane components and obtaining the global action, the RL agent receives a set of rewards denoted by $\{\mathcal{R}(a_i, a_{-i})\}_{i \in \{1,2,...m\}}$. Each reward $\mathcal{R}(a_i, a_{-i})$ depends both on the chosen action $a_i$ for data plane component $s_i$ and on the global action $a$. We define $\mathcal{R}(a_i, a_{-i})$ to belong to $\{\delta_1, ..., \delta_4\}$ where $\delta_1$, $\delta_2$, $\delta_3$ and $\delta_4$ are any real numbers such that $\delta_1 > \delta_2 > \delta_3 > \delta_4$. For $j \in \{1, 2, 3, 4\}$, the reward $\mathcal{R}(a_i, a_{-i})$ is set to $\delta_j$ if condition $\zeta_j$ is true. The conditions $\zeta_j$ are given by:

- $\zeta_1$: the global action $a$ is feasible, the data plane component $s_i$ has not been migrated and the *LB* factor has decreased.
- $\zeta_2$: the global action $a$ is feasible, the data plane component $s_i$ has been migrated and the *LB* factor has decreased.
- $\zeta_3$: the global action $a$ is feasible, the data plane component $s_i$ has been migrated and the *LB* factor has increased.
- $\zeta_4$: the global action $a$ is not feasible.

Next, we define how the agent responds to each action to find a suboptimal policy. Our approach is based on the well-known learning rule Win-Stay-Lose-Shift Algorithm [24].

## *B. The Two-Win-Stay-Lose-Shift Algorithm (2WSLS)*

In order to solve LBDSA based on the RL framework, we developed 2WSLS for two-win-stay-lose shift algorithm which is derived from the win-stay-lose-shift learning strategy. In the original WSLS algorithm, an action is either a winning action or a losing one while in our algorithm, we consider two types of winning actions, hence the name 2WSLS. Given an initial association between controllers and data plane component, 2WSLS aims to solve LBDSA by finding a suboptimal tradeoff between the load balancing and the migration cost while respecting the constraints (7b), (7c) and (7d). In the following, we describe the first iteration of the algorithm, the learning process and the termination process.

### *1) First iteration:*

The RL agent simulates 2WSLS independently for each data plane component. Every data plane component chooses a random action $a_i$ from its action space $C$. Once every data plane component has chosen an action, the RL agent formulates the global action $a = [a_1, a_2, ..., a_m]$ and calculates the load and the response time of each controller. If the global action is feasible, the agent calculates the rewards according to the constraints $\zeta_1$, $\zeta_2$, $\zeta_3$ and $\zeta_4$.

According to the computed rewards and the formulated global action, the RL agent can have information about the reliability of certain actions chosen for the data plane components. Indeed, when a global action is feasible the agent can find out if: (i) the non-migration of $s_i$ can increase the *LB* factor, i.e. $\mathcal{R}(a_i, a_{-i}) = \delta_1$ and (ii) the migration of $s_i$ can decrease or increase the *LB* factor, i.e. $\mathcal{R}(a_i, a_{-i}) = \delta_2$ or $\delta_3$, respectively. Whereas, when the global action is not feasible, the agent cannot know whether the migration or non-migration of a specific data plane component is beneficial. The acquired information should be efficiently used by the agent to choose a better action in the future. Therefore, we define for each $s_i \in S$ a probability vector $\pi_i = [\pi_i[a_1], \pi_i[a_2], ..., \pi_i[a_n]]$ of size $n = |C|$. Each element $\pi_i[a_k]$ for all $k \in \{1, 2, ..., n\}$ corresponds to the probability of playing action $a_i = c_k$ by RL agent for the data plane component $s_i$. Part of the 2WSLS pseudo-code executed by the RL agent for data plane component $s_i$ is presented in Algorithm 1.

### *2) Learning Process:*

Once the RL agent computes the rewards and associates them to the data plane components, it updates the probability vectors of the data plane components as shown in Algorithm 1. These updates should efficiently reflect the results that the RL agent received to improve the learning process. In WSLS, there are two types of actions: (i) a winning action if the received reward is higher and (ii) a losing action if the received reward is not good. In the case of a winning action, the RL agent continues to play this action in future iterations while in the case of a losing action, the RL agent should try another action. The WSLS original learning algorithm has been shown to be commonly used in binary rewards choice problems. However,

**Algorithm 1** 2WSLS for data plane component $s_i$

**Input:** $\mathcal{A}$, $\tau_1$, $\tau_2$, $\epsilon$, $\pi_i$
**Output:** action $a_i$

```
1:  if $\mathcal{R}(a_i, a) = \delta_1$ then
2:      $\pi_i[a_i] \leftarrow \pi_i[a_i] + \tau_1(1 - \pi_i[a_i])$
3:      for $a_j \in A$ do
4:          if $a_j \neq a_i$ then
5:              $\pi_i[a_j] \leftarrow \pi_i[a_j] - \tau_1\pi_i[a_j]$
6:  else if $\mathcal{R}(a_i, a) = \delta_2$ then
7:      $\pi_i[a_i] \leftarrow \pi_i[a_i] + \tau_2(1 - \pi_i[a_i])$
8:      for $a_j \in A$ do
9:          if $a_j \neq a_i$ then
10:             $\pi_i[a_j] \leftarrow \pi_i[a_j] - \tau_2\pi_i[a_j]$
11: else if $\mathcal{R}(a_i, a) = \delta_3$ then
12:     $\pi_i[a_i] \leftarrow \pi_i[a_i] - \epsilon$
13:     $\pi_i[c_{initial}] \leftarrow \pi_i[c_{initial}] + \epsilon$
14: return $a_i$
```

the modeling must be modified and improved for problems where the RL agent receives a non-binary reward after each iteration which is the case in LBDSA.

Since in 2WSLS there is a tradeoff design problem, a single winning action does not model the multi-objective correctly. Thus, we define two winning actions and one losing action. The first winning action results in the $\delta_1$ reward and the second winning action results in the $\delta_2$ reward. We consider that the first winning action is better than the second one ($\delta_1 > \delta_2$) because we aim to minimize the LB factor as well as the migration cost. Thus, the reward of $\delta_1$ that corresponds to a lower LB factor and no migration should have the highest reward. In the second winning action, the *LB* factor decreases but a cost incurs which corresponds to the migration cost of a data plane component. Therefore, to increase the chance to converge to the two winning actions at the end of the learning process, the probabilities of playing these actions must be increased, but with a preference for the first winning action. Accordingly, for $s_i \in S$, the probabilities corresponding to the two winning actions are updated as follows:

$$\pi_i[a_i] = \begin{cases} \pi_i[a_i] + \tau_1(1 - \pi_i[a_i]), & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_1, \\ \pi_i[a_i] + \tau_2(1 - \pi_i[a_i]), & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_2, \end{cases} \quad (6)$$

where $\tau_1$ and $\tau_2$ represent the winning increment factors such that $\tau_1 > \tau_2$ because the first winning action is better than the second one. In order to keep the sum of the probabilities in $\pi_i$ equal to one, all the probabilities $\pi_i[a_j]$ such that $a_j \neq a_i$ must be decreased by the same factor as follows:

$$\pi_i[a_j] = \begin{cases} \pi_i[a_j] - \tau_1\pi_i[a_j], & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_1, \\ \pi_i[a_j] - \tau_2\pi_i[a_j], & \text{if } \mathcal{R}(a_i, a_{-i}) = \delta_2. \end{cases} \quad (7)$$

When the reward is equal to $\delta_3$, we consider the action chosen as a losing one because it leads not only to increase the *LB* factor but also to migrate a data plane component which increases the migration cost. Therefore, the probability of choosing this action is reduced and the probability of not migrating the associated data plane component is increased, i.e. the probability of choosing the initial controller. In this way, we teach the RL agent that it is better not to migrate at all. Hence, the probability vector is updated as follows:

$$\pi_i[a_i] = \pi_i[a_i] - \epsilon, \quad (8)$$

$$\pi_i[c_{initial}] = \pi_i[c_{initial}] + \epsilon, \quad (9)$$

where $\epsilon$ represents the losing decrement factor and $c_{initial}$ is the initial controller of the data plane component $s_i$.

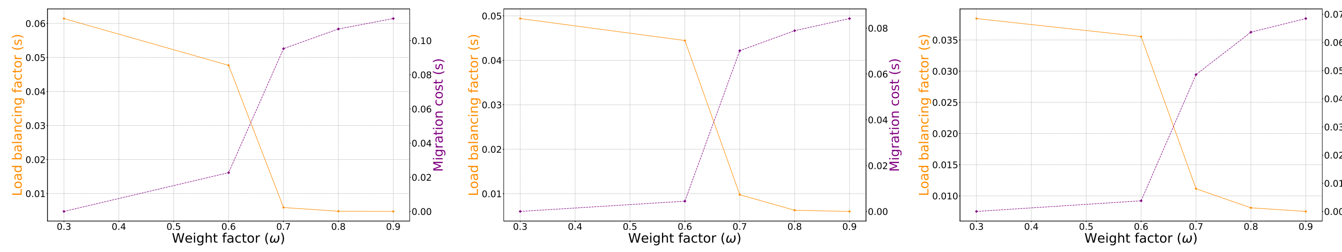
Finally, when the reward is equal to $\delta_4$, the global action is not feasible. In this case, we cannot find out whether the migration or non-migration of a specific data plane component is beneficial to minimize the *LB* factor and the migration cost. Therefore, the RL agent does not perform any probability update. In other words, the RL agent must favor the exploration in order to find out better solutions.

*3) 2WSLS termination:*

The RL agent terminates the execution of the 2WSLS algorithm when the number of iterations reaches a predefined threshold or the probability vectors converge. Then, the agent chooses the last feasible global action $a$ which represents the final association matrix $X^{final}_{n\times m}$.

## IX. Simulation Results

We conduct the simulations using three topologies with a different number of data plane components (switches), namely 20, 25 and 30 components. For all these topologies, the number of controllers is set to 5 and for the sake of simplicity, all controllers have the same capacity $\alpha$. We assume that the number of *Packet_In* messages sent by the data plane



(a) $m = 20$. (b) $m = 25$. (c) $m = 30$.

Fig. 11: The impact of the weight factor ($\omega$) on load balancing factor and migration cost for different data plane topologies.

components is random. To provoke a load imbalance state in the control plane, we apply great stress on at least one controller by generating a large number of *Packet_In* messages. The optimization problem (Eq. 5) is modeled in AMPL [36] and solved using the Knitro solver to obtain a global optimal solution OPT. For all simulations, we performed 100 independent random realizations which are then averaged out. Table IV summarizes the key parameters of the simulations.

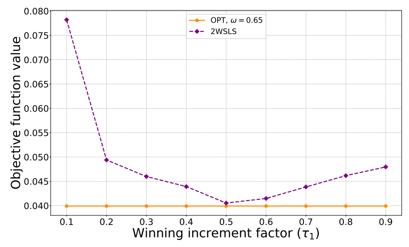
| Name | Description | Value |
|---|---|---|
| $l_i$ | Number of *Packet_in messages* sent by $s_i$ | [5-500] |
| $\alpha$ | Capacity of controllers | 2000 |
| $T_{th}$ | Controller response time threshold | 0.005 s |
| $R$ | Number of iterations of 2WSLS | 500 |
| $R_1$ | Number of realizations of 2WSLS | 100 |

TABLE III: Simulation parameters

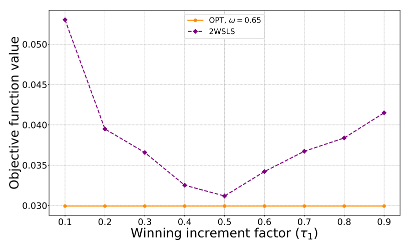
To benchmark our 2WSLS algorithm, we implemented two algorithms from the literature called MCBLB for migration competency-based load balancing [17] and SMCLBRT for SDN multiple controller load-balancing strategy based on response time [11]. In order to balance the load in the SDN control plane, MCBLB uses shift and swap moves to migrate switches between controllers. The shift move is a simple migration operation of a switch from an overloaded controller to an underloaded controller. However, in some cases, shift moves are not always possible. For instance, when the only solution to offload a controller is to migrate a heavy switch, but this switch may overload the final controller. In this case, swapping two switches can be beneficial for the load balancing. In SMCLBRT, the response time of controllers is used to figure out if a controller is overloaded or not. Indeed, based on an appropriate threshold, the controllers are divided into overloaded controllers and underloaded controllers. Then, the heavy switches are migrated simultaneously from overloaded controllers to less-loaded ones. We adapted this algorithm by using our definition of the controller response time (Eq. 1) and the controller response time threshold defined in Table IV. To fairly benchmark the performance of 2WSLS against the two algorithms in [11] and [17], the parameters of our simulation are chosen similar to those used in [11] and [17].

We evaluate the performance of 2WSLS in two stages. First, we analyze the parameters of the algorithm, namely the winning increment factors $\tau_1$ and $\tau_2$, the losing decrement factor $\epsilon$ and the number of iterations, denoted by $R$. Then, we compare 2WSLS to the optimal solution, denoted by OPT, MCBLB and SMCLBRT.

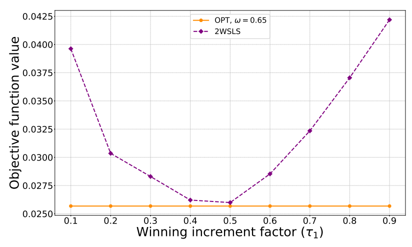
Before discussing the performance of 2WSLS, we start by analyzing the results obtained by solving the optimization problem (Eq. 5) through AMPL modeling and using the Knitro solver. Fig. 11a, 11b and 11c show the $LB$ factor (Eq. 2) and migration cost $T_m$ (Eq. 4) values for the three topologies of 20, 25 and 30 data plane components, respectively. For each topology, we plot the $L$B and $T_m$ values for different tuning tradeoff weights $w \in [0, 1]$. As expected, the results show that the more the policy is $LB$-oriented ($w > 0.5$), the more we sacrifice on migration cost $T_m$ (i.e. $T_m$ increases) and vice versa. This value is almost the same for the three topologies $m = 20$, $m = 25$ and $m = 30$. From these figures, we can see that Pareto solutions to the optimization problem (Eq. 5) can be found around $w = 0.65$. Therefore, in the rest of the simulations, $w = 0.65$ is chosen as an efficient and no dominated solution to the LBDSA problem.



(a) $m = 20$.

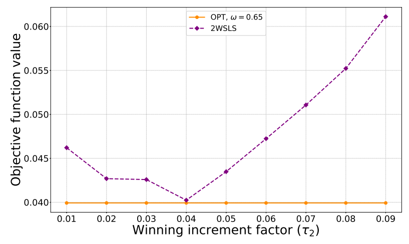


(b) $m = 25$.

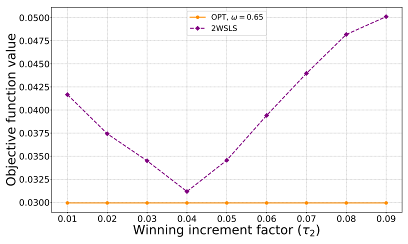


(c) $m = 30$.

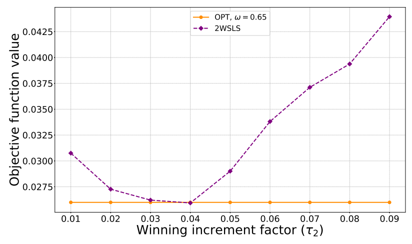
Fig. 12: The impact of the winning increment factor ($\tau_1$) on 2WSLS for different data plane topologies.



(a) $m = 20$.



(b) $m = 25$.



(c) $m = 30$.

Fig. 13: The impact of the winning increment factor ($\tau_2$) on 2WSLS for different data plane topologies.

## A. Parameters of 2WSLS

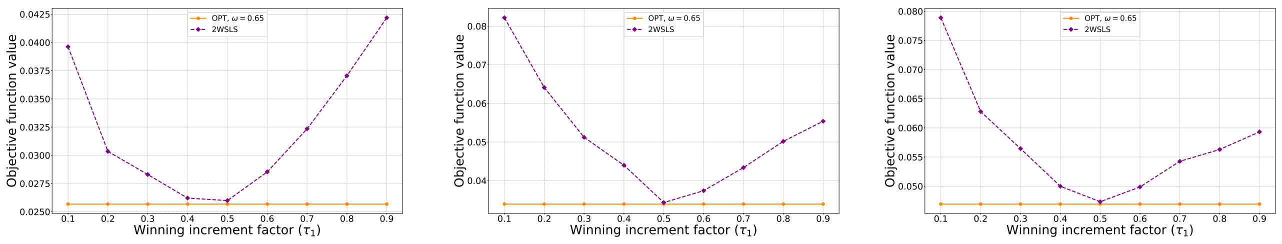
The performance of the 2WSLS algorithm depends on the choice of its parameters, namely the winning increment factors $\tau_1$ and $\tau_2$, the losing decrement factor $\epsilon$ and the number of iterations $R$. Accordingly, they should be chosen carefully to obtain good results. Fig. 12, 13, 14, 15 and 16 illustrate the impact of these parameters on the objective function (Eq. 5).

In order to figure out an optimal value of the winning increment factor $\tau_1$, we compare the performance of 2WSLS to that of OPT for different values of $\tau_1$. In other words, given a value of $\tau_1$, how much 2WSLS can minimize the objective function in Eq. 5 compared to the optimal solution. Fig. 12a, 12b and 12c show that there exists an optimal value of $\tau_1$ at which the performance of 2WSLS is close to the OPT value, given by $\tau_1 = 0.5$. This optimal value is almost the same for the three topologies $m = 20$, $m = 25$ and $m = 30$ which is a very good and an important observation that implies that our proposed learning algorithm is scalable. We notice, for all these topologies, that small values of $\tau_1$ prevent the RL agent from converging to the winning actions and, thus, the performance of 2WSLS decreases. On the other hand, when the value of $\tau_1$ is high, the probability of choosing the associated winning action will also be high. Therefore, the RL agent will not explore new actions, which may lead to better performances, but it will quickly choose few actions by simply exploiting the current information. Similarly, Fig. 13a, 13b and 13c show, for the three topologies, that there is an optimal value of the winning increment factor $\tau_2$, given by $\tau_2 = 0.04$, at which the performance of 2WSLS is close to OPT. Again, Fig. 13 shows the scalability of 2WSLS.

To further verify the scalability of our proposed 2WSLS algorithm, we compare its performance to that of OPT for different control plane topologies, namely $n = 5$, $n = 7$ and $n = 9$. In other words, we show, via simulations, that the optimal values of $\tau_1$ and $\tau_2$ are almost the same for $n = 5$, $n = 7$ and $n = 9$. To verify the 2WSLS algorithm scalability under a dense network topology, the number of data plane components is set to 30 for all these control plane topologies. Fig. 14a, 14b and 14c illustrate that the performance of our proposed algorithm is close to OPT performance at $\tau_1 = 0.5$ for $n = 5$, $n = 7$ and $n = 9$, respectively. Fig 15a, 15b and 15c show, for the three control plane topologies, the existence of an optimal value of $\tau_2$, given by $\tau_2 = 0.04$, at which 2WSLS has close-to-optimal performance. These results conform with those presented in Fig.12 and Fig.13 and demonstrate the scalability of our 2WSLS algorithm.

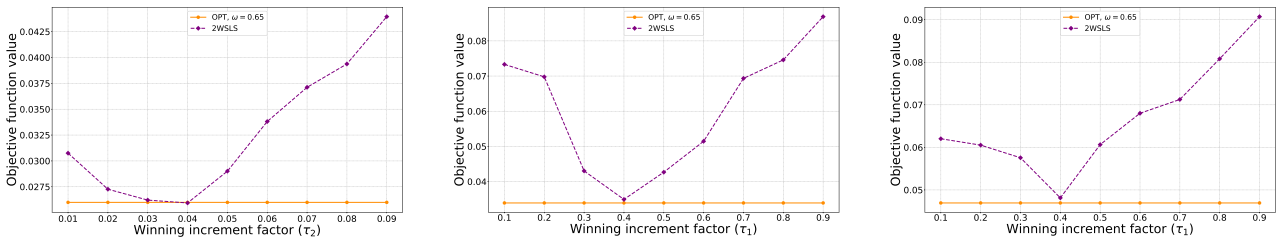
Fig. 16 illustrates the impact of the number of iterations $R$ on 2WSLS for different values of $\tau_1$. We notice that the performance of the optimal value of $\tau_1$, i.e. $\tau_1 = 0.5$, outperforms the performances of the other values. These results confirm those obtained in Fig. 12 and 14. Indeed, higher values of $\tau_1$ improve the performance of 2WSLS faster while they result in low performance for a large number of iterations $R$. For instance, the performance comparison between $\tau_1 = 0.5$ and $\tau_1 = 0.7$ shows that, with $\tau_1 = 0.7$, 2WSLS is able to reach an objective value equal to 0.046 in just 20 iterations while to reach roughly the same value, 2WSLS needs 30 iterations with $\tau_1 = 0.5$. However, $\tau_1 = 0.5$ gives better performances than $\tau_1 = 0.7$ for a large number of iterations. In the case of small values of $\tau_1$, i.e. $\tau_1 < 0.5$, we observe that 2WSLS does not converge towards good performances even with a large number of iterations, e.g. $R = 500$.

The value of $\delta_1$, $\delta_2$, $\delta_3$, and $\delta_4$ are not very important. In other words, any real number values should work fine



(a) $n = 5$. (b) $n = 7$. (c) $n = 9$.

Fig. 14: The impact of the winning increment factor ($\tau_1$) on 2WSLS for different control plane topologies.



(a) $n = 5$. (b) $n = 7$. (c) $n = 9$.

Fig. 15: The impact of the winning increment factor ($\tau_2$) on 2WSLS for different control plane topologies.

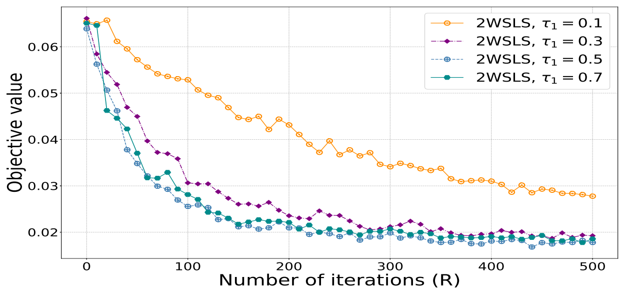


Fig. 16: The impact of the number of iterations on 2WSLS.

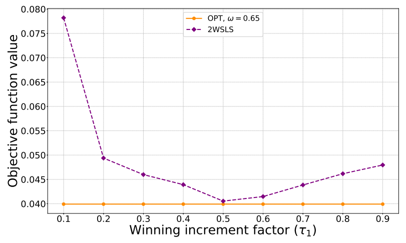
for the proposed 2WSLS algorithm subject to the constraints that $\delta_1 > \delta_2 > \delta_3 > \delta_4$. To show that the exact values of $\delta_1$, $\delta_2$, $\delta_3$, and $\delta_4$ are not so important, we compare the performance of our proposed RL algorithm to that of OPT algorithm for different chosen random values of $\delta_1$, $\delta_2$, $\delta_3$, and $\delta_4$. Precisely, we show, via simulations, that 2WSLS converges to the same optimal values of the winning increment factors $\tau_1$ and $\tau_2$. For this purpose, we define the vector $\Delta = (\delta_1, \delta_2, \delta_3, \delta_4)$ and we test three different configurations of $\Delta$. These configurations are given by $\Delta^1 = (2, 1, 0, -4)$, $\Delta^2 = (7, 5, 3, 1)$, and $\Delta^3 = (100, 70, 40, 10)$. Fig. 17a corresponding to $\Delta^1$, Fig. 17b corresponding to $\Delta^2$, and Fig. 17c corresponding to $\Delta^3$, show the performance of the proposed RL algorithm with comparison to OPT algorithm when varying the winning increment factor $\tau_1$. The three figures show the existence of an optimal value of $\tau_1$, given by $\tau_1 = 0.5$, at which the performance of our proposed algorithm is close to optimal. Similarly, Fig. 18a corresponding to $\Delta^1$, Fig. 18b corresponding to $\Delta^2$, and Fig. 18c corresponding to $\Delta^3$, show the performance of the proposed RL algorithm with comparison to OPT algorithm when varying the winning increment factor $\tau_2$. The three figures show the existence of an optimal value of $\tau_2$, given by $\tau_2 = 0.04$, at which the performance of our proposed algorithm is close to the optimal one. These results comply with those presented in Fig. 12, 13, 14 and 15 for the previously chosen configuration $(2, 1, 0, -4)$ and demonstrate that the values of the deltas are not very important unless $\delta_1 > \delta_2 > \delta_3 > \delta_4$. This, indeed, shows that the 2WSLS is flexible, robust and scalable.

## B. Performance of 2WSLS

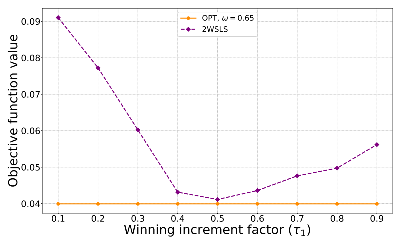
To verify the effectiveness of 2WSLS in solving the LBDSA problem, we compare its performance against OPT, MCBLB and SMCLBRT algorithms in terms of load balancing ($LB$) and migration cost ($T_m$) separately.

Fig. 19 presents the comparison of 2WSLS, OPT, MCBLB and SMCLBRT in terms of load balancing ($LB$) for the three topologies $m = 20$, $m = 25$ and $m = 30$. Based on these results, we make the following observations: (1) It is clear that the performance of 2WSLS is close to that of OPT for the different values of $m$; (2) The performance gap between 2WSLS and OPT is almost the same for the different topologies, which illustrates its scalability; (3) 2WSLS outperforms MCBLB and SMCLBRT. Indeed, 2WSLS minimize the load balancing factor better than MCBLB since the latter compares the degree of load balancing achieved by a shift or a swap move against only the non-execution of this move. Also, in the definition of the cost function associated to a shift move or a swap move, MCBLB includes a factor which encourages migrating switches with low latency to final controllers. Therefore, this factor can poorly offload overloaded controllers, which has an impact on the load balancing. In SMCLBRT, the load balancing between the controllers is not really considered since determining whether a controller is overloaded or not depends only on a predefined threshold. Moreover, SMCLBRT simply selects the switch with the maximum load from the overloaded controller's domain then migrates it to an underloaded one.

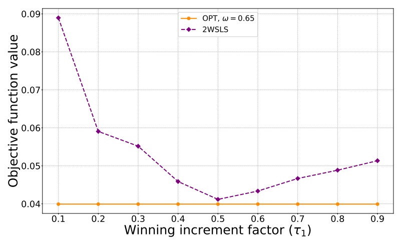
In Fig. 20, we compare the performance of 2WSLS, OPT, MCBLB and SMCLBRT in terms of migration cost ($T_m$) for the three topologies $m = 20$, $m = 25$ and $m = 30$. Note that the migration cost results, presented in Fig. 20, are obtained for the same realizations as the load balancing results in Fig. 19. We can observe that the migration cost of 2WSLS is the closest one to that of OPT for all $m$ values. Furthermore, for the three topologies, we notice that the gap between the 2WSLS and OPT migration costs remains constant which illustrates its scalability. We can once again confirm that 2WSLS gives better performance compared to MCBLB and SMCLBRT. MCBLB is outperformed by 2WSLS because it uses swap moves when shift moves are not possible. In other words, swap moves generate more migration operations than shift moves which increases the migration cost. As for SMCLBRT, we can see that it has the worst performance among all algorithms in terms of migration cost. Indeed, SMCLBRT uses



(a) Configuration $\Delta^1$.

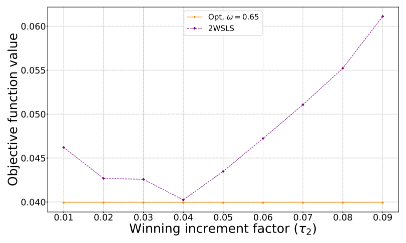


(b) Configuration $\Delta^2$.

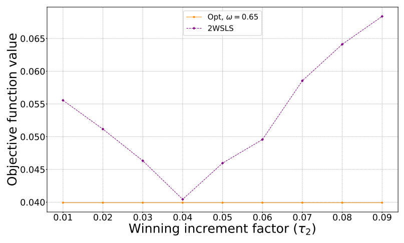


(c) Configuration $\Delta^3$.

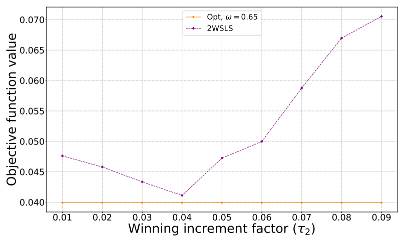
Fig. 17: The impact of the winning increment factor ($\tau_1$) on 2WSLS for different reward configurations.



(a) Configuration $\Delta^1$.



(b) Configuration $\Delta^2$.



(c) Configuration $\Delta^3$.

Fig. 18: The impact of the winning increment factor ($\tau_2$) on 2WSLS for different reward configurations.

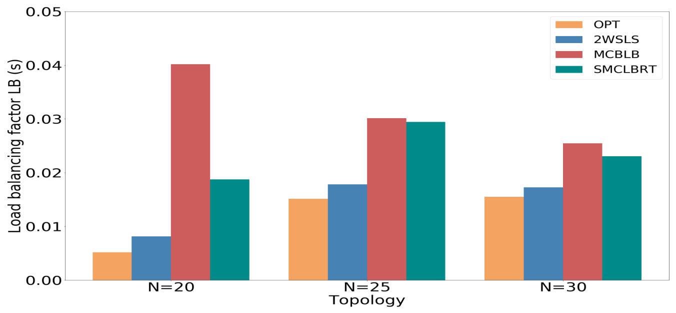


Fig. 19: The performance of 2WSLS in terms of load balancing (LB) compared to OPT, MCBLB and SMCLBRT.

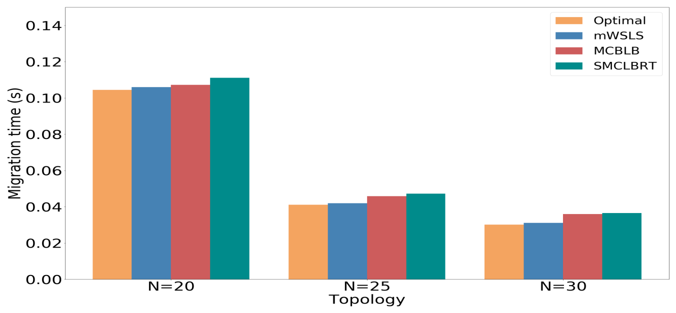


Fig. 20: The performance of 2WSLS in terms of migration cost ($T_m$) compared to OPT, MCBLB and SMCLBRT.

a threshold to determine the overloaded controllers and when this threshold is low which is the case in this paper, i.e. to meet the requirements of delay sensitive applications, the number of overloaded controllers is higher. Hence, the number of migrated switches from overloaded controller increases which increases the migration cost.

## X. Conclusion

In this paper, we tackled the load balancing problem in distributed SDN architecture using machine learning. Reactive load balancing mechanisms make the overloaded SDN controllers suffer from high response time until their offloading happens through migration operations. Furthermore, the time required to migrate a data plane component is not considered in most proposed algorithms. Thus, we propose to preemptively balance the load in the SDN control plane while minimizing the migration time such that the latency requirements of delay sensitive applications are satisfied. For this purpose, first, we built Auto Regressive Integrated Moving Average (ARIMA) and Long Short-Term Memory (LSTM) models to predict the load of SDN controllers and compared their performance in terms of accuracy in short-term and long-term predictions. Comparison results showed that ARIMA lightly outperforms LSTM in short-term predictions while LSTM widely surpasses ARIMA in long-term predictions. Long-term predictions allow to preemptively figure out if the load in the control plane will be unbalanced and, consequently, schedule migration operations in advance. Second, to select which data plane component should be migrated and where the migration should happen, we formulate the load balancing for delay sensitive applications (LBDSA) problem as a non-linear binary program and prove its NP-completeness. To solve this problem, we proposed a reinforcement learning algorithm, called 2WSLS, which is inspired by the well-known learning rule of win-stay-lose-shift. Finally, we benchmark 2WSLS against two algorithms from the literature. Simulation results demonstrated that 2WSLS has close to optimal performance and outperforms the benchmark algorithms in terms of load balancing and migration cost.

## References

[1] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, “Multi-controller Based Software-Defined Networking: A Survey,” *IEEE Access*, vol. 6, pp. 15 980–15 996, 2018.
[2] M. Karakus and A. Durresi, “A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN),” *Comput. Netw.*, vol. 112, pp. 279–293, 2017.
[3] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, “Multi-access edge computing: A survey,” *IEEE Access*, vol. 8, pp. 197 017–197 046, 2020.
[4] C. N. Tadros, M. R. M. Rizk, and B. M. Mokhtar, “Software Defined Network-Based Management for Enhanced 5G Network Services,” *IEEE Access*, vol. 8, pp. 53 997–54 008, 2020.
[5] C. Lin, G. Han, X. Qi, M. Guizani, and L. Shu, “A Distributed Mobile Fog Computing Scheme for Mobile Delay-Sensitive Applications in SDN-Enabled Vehicular Networks,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5481–5493, 2020.
[6] M. Azizian, S. Cherkaoui, and A. S. Hafid, “Vehicle software updates distribution with sdn and cloud computing,” *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 74–79, 2017.
[7] M. Azizian, S. Cherkaoui, and A. Hafid, “An optimized flow allocation in vehicular cloud,” *IEEE Access*, vol. 4, pp. 6766–6779, 2016.
[8] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, “A survey on software defined networking with multiple controllers,” *Journal of Network and Computer Applications*, vol. 103, pp. 101–118, 2018.

[9] A. A. Neghabi, N. Jafari Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," *IEEE Access*, vol. 6, pp. 14 159–14 178, 2018.

[10] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *Proc. IEEE/ACM Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oct. 2014, pp. 17–27.

[11] 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, 2018.

[12] G. Wu, J. Wang, M. S. Obaidat, L. Yao, and K.-F. Hsiao, "Dynamic switch migration with noncooperative game towards control plane scalability in SDN," *Int. J. Commun. Syst.*, vol. 32, no. 7, p. e3927, 2019.

[13] A. El Kamel and H. Youssef, "Improving Switch-to-Controller Assignment with Load Balancing in Multi-controller Software Defined WAN (SD-WAN)," *J. Netw. Syst. Manage.*, 2020.

[14] Z. Li, Y. Hu, T. Hu, and P. Wei, "Dynamic SDN Controller Association Mechanism Based on Flow Characteristics," *IEEE Access*, vol. 7, pp. 92 661–92 671, 2019.

[15] T. Wang, F. Liu, and H. Xu, "An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks," *IEEE/ACM Trans. Netw.*, vol. 25, pp. 2788–2801, Oct. 2017.

[16] N. Correia and F. AL-Tam, "Flow Setup Aware Controller Placement in Distributed Software-Defined Networking," *IEEE Syst. J.*, pp. 1–4, 2019.

[17] F. Al-Tam and N. Correia, "On Load Balancing via Switch Migration in Software-Defined Networking," *IEEE Access*, vol. 7, pp. 95 998–96 010, 2019.

[18] R. Chaudhary and N. Kumar, "LOADS: Load Optimization and Anomaly Detection Scheme for Software-Defined Networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 12, pp. 12 329–12 344, 2019.

[19] 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.

[20] K. S. Sahoo, D. Puthal, M. Tiwary, M. Usman, B. Sahoo, Z. Wen, B. P. S. Sahoo, and R. Ranjan, "ESMLB: Efficient Switch Migration-based Load Balancing for Multi-Controller SDN in IoT," *IEEE Internet Things J.*, pp. 1–1, 2019.

[21] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "SDN Controller Assignment and Load Balancing with Minimum Quota of Processing Capacity," in *Proc. IEEE ICC*, May. 2018, pp. 1–6.

[22] G. M. Jenkins, "Autoregressive–Integrated Moving Average (ARIMA) Models," in *Wiley StatsRef: Statistics Reference Online*. American Cancer Society, 2014.

[23] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] M. Nowak and K. Sigmund, "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game," *Nature*, vol. 364, no. 6432, pp. 56–58, 1993.

[25] Z. Mlika, E. Driouch, and W. Ajib, "A fully distributed algorithm for user-base station association in hetnets," *Computer Communications*, vol. 105, pp. 66 – 78, 2017.

[26] Y. Zhou, K. Zheng, W. Ni, and R. P. Liu, "Elastic Switch Migration for Control Plane Load Balancing in SDN," *IEEE Access*, vol. 6, pp. 3909–3919, 2018.

[27] M. A. S. Santos, B. A. A. Nunes, K. Obraczka, T. Turletti, B. T. de Oliveira, and C. B. Margi, "Decentralizing SDN's control plane," in *Proc. IEEE LCN*, Sep. 2014, pp. 402–405.

[28] A. Filali, S. Cherkaoui, and A. Kobbane, "Prediction-based switch migration scheduling for sdn load balancing," in *Proc. IEEE ICC*, 2019, pp. 1–6.

[29] A. Taïk and S. Cherkaoui, "Electrical load forecasting using edge computing and federated learning," in *Proc. IEEE ICC*, 2020, pp. 1–6.

[30] R. p. team, *RYU SDN Framework - English Edition*. RYU project team, Feb. 2014.

[31] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "SDN Controllers: Benchmarking Performance Evaluation," arXiv:1902.04491, 2019.

[32] D. Y. Huang, K. Yocum, and A. C. Snoeren, "High-fidelity switch models for software-defined network emulation," in *Proc. ACM HotSDN*, Aug. 2013, pp. 43–48.

[33] A. Abouaomar, S. Cherkaoui, A. Kobbane, and O. A. Dambri, "A resources representation for resource allocation in fog computing networks," in *Proc. IEEE GLOBECOM*, 2019, pp. 1–6.

[34] G. J. Woeginger and Z. Yu, "On the equal-subset-sum problem," *Information Processing Letters*, vol. 42, no. 6, pp. 299–302, 1992.

[35] X. Gao, L. Kong, W. Li, W. Liang, Y. Chen, and G. Chen, "Traffic Load Balancing Schemes for Devolved Controllers in Mega Data Centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, pp. 572–585, Feb. 2017.

[36] R. Fourer, D. M. Gay, and B. W. Kernighan, "A Modeling Language for Mathematical Programming," *Management Science*, vol. 36, no. 5, pp. 519–554, 1990.

**Abderrahime Filali** (Student Member, IEEE) received the B.Eng. degree in telecommunication and network engineering from the Ecole Nationale des Sciences Appliquée d'Oujda, Morocco in 2016. He is currently pursuing the Ph.D. degree in computer engineering at Department of Electrical and Computer Engineering of Université de Sherbrooke, Canada. He served as a reviewer for several international conferences and journals. His research interests include software-defined networks, network function virtualization, and algorithm design for next-generation networks.

**Zoubeir Mlika** (Member, IEEE) received the B.Eng. degree from the Higher School of Communication of Tunis, Tunisia, in 2011, and the M.Sc. and Ph.D. degrees from the University of Quebec at Montreal (UQAM), Canada, respectively in 2014 and 2019. From 2011 to 2012, he was an intern at UQAM, where he was working on relay selection algorithms in cognitive radio networks. He was a visiting research intern with the department of Computer Science at University of Moncton, Canada, between March 2018 and August 2018. From September 2019, he joined the Department of Electrical and Computer Engineering in University of Sherbrooke, Canada as Postdoctoral Fellow. His current research interests include vehicular communications, resource allocation, design and analysis of algorithms and computational complexity.

**Dr. Soumaya Cherkaoui** (Senior Member, IEEE) is a Full Professor at Department of Electrical and Computer Engineering of Université de Sherbrooke, Canada which she joined as a faculty member in 1999. Her research and teaching interests are in wireless networks. Particularly, she works on next generation networks Edge computing/Network Intelligence, and communication networks. Since 2005, she has been the Director of INTERLAB, a research group which conducts research funded both by government and industry. Before joining U. Sherbrooke, she worked for industry as a project leader on projects targeted at the Aerospace Industry. Her work resulted in technology transfer to companies and to patented technology. Pr. Cherkaoui has published over 200 research papers in reputed journals and conferences. She has been on the editorial board of several journals including IEEE JSAC, IEEE Network, and IEEE Systems. Her work was awarded with recognitions including a best paper award at IEEE ICC in 2017. She is currently an IEEE ComSoc Distinguished Lecturer. She is a Professional Engineer in Canada and has been designated Chair of the IEEE ComSoc IoT-Ad hoc and Sensor Networks Technical Committee in 2020.

**Dr. Abdellatif Kobbane** (Senior Member, IEEE) is currently Full Professor at the Ecole Nationale Suprieure d'Informatique et d'Analyse des Systemes (ENSIAS), Mohammed V University in Rabat, Morocco since 2009. He received his PhD degree in computer science from the Mohammed V-Agdal University (Morocco) and the University of Avignon (France) in September 2008. He received his research MS degree in computer science, Telecommunication and Multimedia from the Mohammed V-Agdal University (Morocco) in 2003. Doctor Kobbane is Adjunct Professor at L2TI laboratory, Paris 13 University, France. His research interests lie with the field of wireless networking, performance evaluation using advanced technique in game theory and MDP in wireless mobile network : IoT, SDN and NFV, 5G networks, resources management in wireless mobile networks, cognitive radio, Mobile computing, Mobile Social networks, Caching and backhaul problem, Beyond 5G and Future networks.