Journal of Computer Science and Cybernetics, V.39, N.1 (2023), 79–99
DOI no 10.15625/1813-9663/17591

LSTM-BASED SERVER AND ROUTE SELECTION IN DISTRIBUTED AND HETEROGENEOUS SDN NETWORK

NAM-THANG HOANG[1,2], VAN TONG[2], HAI-ANH TRAN[2,*],
CONG-SON DUONG[1], TRAN-LE-TUAN NGUYEN[1]

[1]*Faculty of Information Technology, Hanoi University of Civil Engineering (HUCE), Ha Noi, Viet Nam*
[2]*School of Information and Communication Technology (SOICT), Hanoi University of Science and Technology (HUST), Ha Noi, Viet Nam*

Abstract. Nowadays, Software-defined Networking (SDN) is increasingly being deployed in network systems and by network operators due to its benefits, such as automation leading to excellent reliability, more efficient network management, cost-savings, and faster scalability. The most common deployment architecture for SDN is a distributed system consisting of many independent domains, each controlled by an SDN controller. One of the well-known applications in SDN is server selection and routing. However, deploying server and route selection in distributed and heterogeneous SDN networks faces two issues. Firstly, the lack of global views of the entire system is due to the absence of standardized inter-communication between SDN domains for the distributed and heterogeneous SDN network. To address this issue, this paper utilizes our previous work, the open East-West interface SINA, to adaptively ensure network state consistency in a multi-domain SDN networks. Secondly, selecting the path for packet transmission based solely on the current network states of a local SDN domain is ineffective since the system cannot respond to unexpected changes in the network state. Predicting the link cost of the entire routing path is necessary. Therefore, this paper proposes an LSTM-based link cost prediction for the server and route selection mechanism in a distributed and heterogeneous SDN network. The experimental results demonstrate that our proposal improves link utilization, packet loss rate, response time, and overhead by up to 15%, 10%, 14%, and 25% respectively to benchmarks.

Keywords. SDN, Inter-SDN domain, LSTM, network state prediction, QoS, server and route selection algorithm.

1. INTRODUCTION

Since its appearance, the Software-defined Network (SDN) network model has shown its outstanding benefits, such as the ability to re-route networks on the fly, offering real-time visibility to the whole system, providing the real-time ability to automatically re-route or to stand-up new functions and routes without adding any new hardware. With these benefits, in recent times, many organizations have researched and deployed SDN on their

*Corresponding author.
E-mail addresses: thanghn@huce.edu.vn(N.-T.H.); duongcongson01gmail.com(C.-S.D); tuan158764@huce.edu.vn(T.-L.T.N.);

systems, such as Google, NTT (Japan), and Orange (France). The most common deployed architecture is the distributed architecture with multiple domains, each controlled by an SDN controller. One of the implementations that has attracted the most attention from network operators and the research community is server and route selection (SARS) in such *distributed* and *heterogeneous* network systems. The participation of many domains represents the *distribution*, and the *heterogeneity* is represented by the difference in the type of SDN controllers and the different policies in each domain. However, this implementation is facing two serious problems as follows.

Firstly, each domain has only its network state information. The SARS mechanism will only be effective if one domain has the network state information of others. The cause of this problem is the need for cross-domain communication. The fact is that there has yet to be an interface that makes it possible for different SDN controllers to communicate with each other. To solve this issue, this paper implements the Knowledge-Defined Network framework (KDN) [8] using our previous work, the East-West interface SINA [10], at the control plane, allowing the controllers to exchange network state information and guaranteeing the consistency of network state information across SDN domains.

Secondly, SARS is not efficient based solely on the current state of the network. A path chosen in the current state is not necessarily a good solution for future states. Traditional Open Shortest Path First (OSPF) often selects paths based on min hop counts. Therefore, many links can be under-utilized while others need to be more utilized, which can cause a reduction in the efficiency in using the network's resources, Quality of Service (QoS), and waste network resources. Akyildiz et al. [1] proposed a QoS-based framework to select servers and routes. However, this study only considered the current network state, which might need more information to optimize the long-term network performance. Consequently, many recent studies use ML algorithms to solve this problem [16]. For example, Sowmya Sanagavarapu et al. [20] introduced SDPredictNet with a neural network to predict future network states for routing. However, it is only implemented on small-scale intra-SDN domains. To overcome this second limitation and deploy the SDN network as a large-scale distributed network with multi-controllers, this article proposes an LSTM-based link cost prediction mechanism to enhance the performance and accuracy of the SARS solution in using our previous work, the East-West interface SINA [10].

Our contributions are summarized as follows:

- The KDN architecture is deployed with three planes: the data plane, the control plane, and the knowledge plane. In the control plane, our previous work, SINA [10], is deployed to guarantee the network state consistency in the distributed and heterogeneous inter-SDN domains network. SINA allows network state sharing between different SDN controllers in order to provide a global view of the network to distributed controllers in each domain. At the top of the knowledge plane, a SARS mechanism is implemented after having a global view of the network.
- Based on the KDN architecture, the link cost prediction mechanism is developed using Long Short-Term Memory Network (LSTM) to optimize our SARS. The objective of using LSTM is to discover the non-linear nature and uncertainty of traffic flows. It can take advantage of historical traffic parameters instead of considering only current network states. The predicted link cost consists of several network metrics: delay, packet loss, link overhead, and link utilization.

The outline of this paper is structured as follows. In Section 2, some related works are presented. In Section 3, the KDN architecture is briefly introduced. Section 4 is devoted to the SARS framework. In Section 5, the LSTM model is sketched and described. In Section 6, some experiments are conducted to verify the effectiveness of our link cost prediction and SARS methods. The conclusion and future work are discussed in Section 7.

2. RELATED WORK

Regarding studies using prediction-based routing mechanisms, Sowmya Sanagavarapu et al. [20] presented the SDPredictNet, a framework including LSTM and artificial neural network (ANN) to solve the route prediction problem. The aim of using LSTM and ANN is to help update the flow table for routing, with the delay being reduced after recognizing congested areas in the networks. In this SDPredictNet framework, the traffic features are captured by the LSTM model before these features are fed into the ANN model to recommend the path for future packets at the given instant. To illustrate, the authors use the iperf tool to generate data and obtain traffic parameters (jitter, packet loss rate, transferred data size, throughput). Given (n–1) packet parameters, the LSTM attempts to forecast the nth packet's parameters. Then, the ANN tries to map a sequence of predicted packets' parameters to the output switches with two values (0 or 1). The final path is determined by taking the sequence of switches labeled as 1. The results show that the RMSE metrics and accuracy of an SDPredictNet framework achieve a 0.07 score and 99.88%, respectively. Despite that, SDPredictNet is only implemented on a small-scale network with seven switches and eight hosts.

In addition to that, Long Jiang et al. [13] proposed a two-phase routing mechanism to predict link quality in an SDN-based ad hoc network. In the first phase, XGBoost - a library that uses an ensemble of decision trees to generate predictive models, is used for the leaf node score's optimization of each decision tree before forecasting the future link quality. In the second phase, a minimum cost tree method generates routes with the highest packet delivery ratio (PDR). The experimental results indicate that the proposed algorithm records an improvement in PDR, throughput, and delay compared to existing routing mechanisms.

Furthermore, Long Jiang et al. [12] focused on a routing optimization problem to satisfy the user's QoS in the SDN-based mobile ad hoc network. In the first stage, they use a wavelet neural network (WNN) to forecast the following timestamp link quality values. After obtaining the future link quality values, during the second stage, the routing problem is formulated as the binary knapsack problem, where the differential search (DS), a type of meta-heuristic optimization algorithm, is used to generate feasible solutions. The numerical results suggest that their proposal achieves high throughput and low packet loss rates with reduced delay.

Wu et al. [22] introduced AIER (Artificial Intelligence Enabled Routing), based on an ANN network, to predict traffic congestion before making route decisions. AIER is known to solve the inadequacy of learning capacity from precedent experiences to prevent inefficient route selection by the dynamic routing mechanism of the SDN controller. There are three primary phases in AIER: 1) AIER collects the training data, traffic flows in the network, and its label. The label can be either 1 (congestion) or 0 otherwise. For example, the transmitted data between host two and destination three is 55M which is labeled as 1 (congestion); 2) the

ANN model is trained before being deployed on the control plane; 3) the AIER can choose an appropriate path to avoid congestion using the Dijkstra algorithm. The simulation results demonstrate that the AIER framework significantly outperforms static and dynamic routing mechanisms regarding average throughput, packet delay, and packet loss. Nevertheless, the simulated topology does not have the characteristics of the real-world network, with only five switches and four hosts.

Abdelhadi Azzouni et al. [3] proposed NeuRoute, a routing framework based on a neural network to optimize the network throughput. NeuRoute aims to outperform the traditional routing algorithm in the SDN controller in terms of computational complexity; at every round, it uses the predicted link loads as input before performing the graph search to find the optimal paths. It has two main core blocks: the first block to predict the traffic matrix and the second block for traffic routing. The former is an LSTM which attempts to predict the traffic matrix, which is then used in the next step. The latter is a feed-forward network responsible for selecting optimal paths using the predicted traffic matrix from the previous step. The experiments reveal that NeuRoute's performance is well improved. However, the authors only use a small-scale topology with small databases comprising 10000 samples, which might suffer the overfitting problem.

Regarding the traffic matrix prediction problem, Jitendra Bhatia et al. [7] introduced the SDVN architecture (Software-defined Vehicular Networks) based on a neuron network to model the traffic flows efficiently in urban areas. The most prominent feature of the SDVN architecture is to use the LSTM network to capture and learn the non-linear nature of the traffic flow. This architecture integrates SDN into the cloud to optimize computational expenses and storage efficiencies. The architecture comprises two significant phases: detecting the congestion-sensitive spots and forecasting the traffic congestion trends. During the first phase, the K-means algorithm is utilized to detect the congestion spots. In the next phase, the LSTM network is utilized to learn the traffic flows behaviors of such areas before deployment for the future traffic density prediction of each spot. In the evaluation phase, it is shown that the LSTM achieves stunning results with 97% accuracy in traffic density prediction.

Moreover, Azzouni et al. [4] proposed the NeuTM framework using a deep neural network to predict the traffic matrix. Specifically, they take advantage of the LSTM network as it is suitable for learning from data and classifying time series with time intervals of unknown size. The traffic matrix is a two-dimensional matrix where its $(i,\ j)$ elements capture the amount of traffic transmitted from node i to node j in a network. As a result, network operators know how traffic flows through their network and can make wise decisions on network and resource management. In the NeuTM framework, Azzouni and his associate apply a sliding window technique for mapping the input-output pairs to feed the LSTM network. They deploy NeuTM in a GEANT topology, including 23 nodes and 38 links with a POX controller. The experimental results show that their model's accuracy and prediction measurements outperform traditional methods with better learning of non-linear patterns.

To conclude this section, all of the mentioned above studies utilized Machine Learning techniques to leverage their works with the improvement in throughput, delay, and better execution time; therefore, this motivates us to take advantage of the Machine Learning technique in terms of predicting future network states so that our route and server selection mechanism can quickly react to the upcoming network state changes.

3. IMPLEMENTATION OF KNOWLEDGE-DEFINED NETWORK

This section describes the implementation of the KDN architecture (Figure 1). KDN aims to make the distributed large-scale network system more autonomous and smarter with the ability to share knowledge across multi-domains. KDN contains three layers: the data plane, the control plane, and the knowledge plane. The communication between these layers is performed via Northbound and Southbound API.

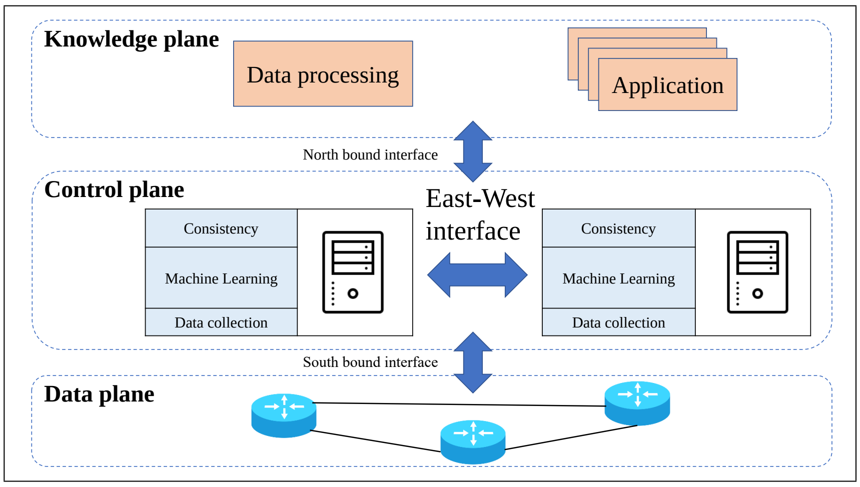

Figure 1: Knowledge-defined heterogeneous architecture

The *data plane* comprises OpenFlow-enable switches and other network devices to accomplish instructions from the control plane (e.g., forwarding, dropping packets, etc.).

The *control plane* includes multiple heterogeneous distributed controllers. In this paper, two widely known SDN controller platforms are deployed: ONOS [5] and RYU [2]. ONOS (Open Network Operating System), written by a JAVA programming language, is designed as a distributed system with high scalability. RYU is an open-source SDN controller written in Python. Due to its simplicity, it can be customized to implement various SDN applications. Due to its flexibility to support multiple protocols such as OpenFlow, Netconf, OF-config, RYU can be used in various types of networks and applications.

In our previous work, SINA [10] - a type of East-West interface, was proposed and standardized with a consistency mechanism to allow cross-domain communication and information consistency among heterogeneous controllers. The multiple distributed controllers can share the states with others through SINA after collecting their network states. However, when the communication ability in the distributed system improves significantly, problems regarding security and reliability may arise. In a multi-threading system like SDN, there is still a risk of data leakage when SDN controllers exchange their states. To overcome this limitation, each SDN controller should retain its data and exchange its knowledge. An ML model in each SDN domain produces the knowledge. As a result, this idea might prevent the raw data leakage problem. Then, the consistency mechanism can be used to guarantee the consistency of knowledge among multiple controllers with reasonable communication costs.

The East-West interface SINA is considered a communication bridge. It guarantees consistency with two main components: REST API modules and the listener. The latter is to listen to the local knowledge generated from a machine learning model before notifying them of other SDN domains. The former enables other SDN controllers to update the propagated knowledge based on the REST API platform.

In addition, the consistency mechanism is implemented to ensure that the knowledge of distributed SDNs is consistent. It includes two major parts: quorum replication and reinforcement-learning techniques. For example, let us denote N, N_r, and N_w as the total number of controllers, the number of reads and write operations, respectively. If there is new-found knowledge in one SDN domain, the local controller sends them to N_w other controllers. On the other hand, when a local controller performs the read operation, it gets knowledge from N_r other controllers. When $N_w+N_r > N$ is satisfied, this is called the strong consistency mechanism. This mechanism ensures that the knowledge of multiple controllers is consistent at the cost of more communication operations. Based on the reinforcement learning algorithm - a type of strategy that learns a set of actions (read and write operations) from getting feedback from the environment, our consistency mechanism can adaptively select N_w and N_r, ensuring $N_w + N_r \leq N$, and maintaining the balance between controllers communication overhead and the network state consistency.

The *knowledge layer* gets all distributed knowledge from multiple SDN domains into storage (or database) before being processed for various applications and services. In this paper, the SARS application is taken into consideration.

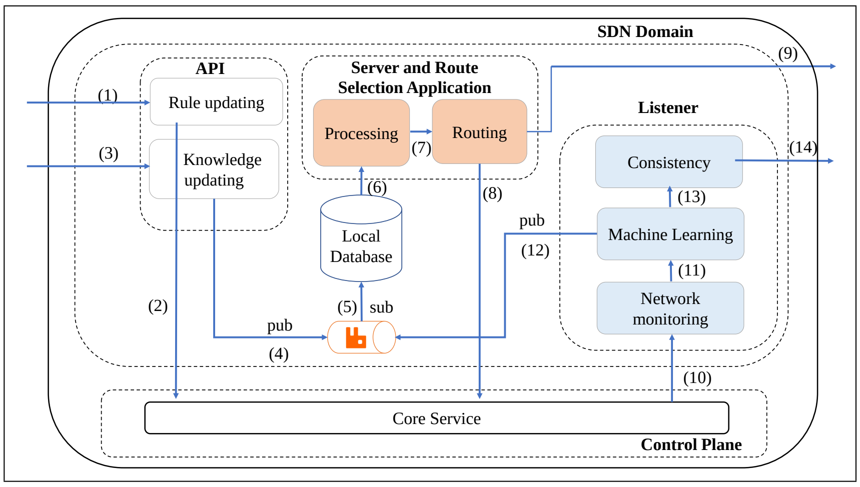

Figure 2: The data flow of the SARS application

Figure 2 demonstrates how data flows through various modules in the SARS application. Firstly, the forwarding rules from other SDN domains are sent to the rule updating module of the current domain before they are passed into the core service (flow 1, 2). At the same time, the current domain also received the network data (QoS metrics) from other domains via the knowledge updating module before these metrics are stored in the local database via a queue message (flow 3, 4, 5). The data can now be retrieved by the SARS application for analyzing the relationship of the traffic and making the server and routing selection (flow 6, 7). The resulting rules are then transferred into the core service and other SDN domains' API (flow 8, 9).

One note-worthy observation of Figure 2 is the presence of the listener component that includes the consistency, machine learning, and network monitoring modules. The network monitoring module captures the QoS metrics sent by the controller every 3 seconds (flow 10). The machine learning module then discovers these QoS parameters for finding valuable knowledge and features (flow 11). This knowledge can now be sent to the message queue (flow 12) or to the consistency module, where local knowledge is broadcasted to other SDN domains.

To conclude, this section has illustrated the knowledge-defined heterogeneous network architecture and the data flow of the SARS application. In the next section, the algorithm for selecting a route and server in the SARS application is explained in great detail.

4. SERVER AND ROUTE SELECTION FRAMEWORK

This section depicts the SARS application in the inter-SDN domains network, which is illustrated in Figure 3. In this network, let denote $N = \{n_i | i = 1, 2, ..., n\}$ as the set of domains, $C = \{c_i | i = 1, 2, ..., n\}$ as the set of controllers. When traffic and services are requested from hosts to servers, the network data are generated from these connections and collected by each domain controller. The QoS data from a sub-domain are then stored in the local database and analyzed before being broadcasted among $N - 1$ other domains. This is when the SARS application combines multiple knowledge from every individual domain to make the server and routing decisions that optimize the network performance.

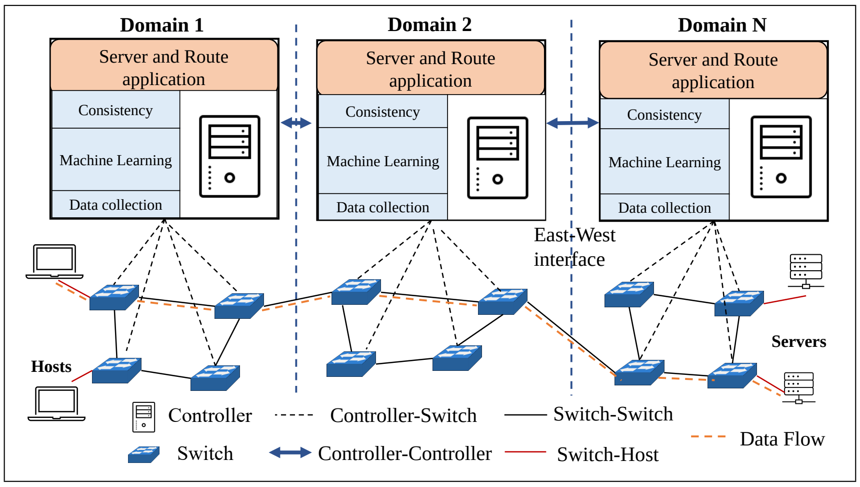

Figure 3: Inter-SDN domains network with KDN architecture

4.1. Problem formalization

The SARS problem in SDN inter-domain can be formalized as follows. The whole network topology is denoted as an undirected weighted graph $G = (V, E)$, where:

- $G = G_1 \cup G_2 \cup G_3 \cup ... \cup G_l$ with G_i is the graph of ith domain in the whole distributed network.
- $V = \{v_1, v_2, ..., v_n\}$ is the set of switches.

- $E = \{e_1, e_2, ..., e_m\}$ is the set of links with each link being assigned a link cost LC_i.
- $H = \{h_1, h_2, ..., h_u\}$ is the set of hosts.
- $S = \{s_1, s_2, ..., s_v\}$ is the set of servers with each server being assigned a server cost SC_i.

Mathematically given:

- **Input**: host $h_i \in H$, switch $v_i \in V$, link $e_j \in E$, $(i = 1, 2, ...n), j = (1, 2, ..., m)$
- **Output**: a server s and a path $p = \{(v_1, v_2), (v_2, v_3), ...(v_{p-1}, v_p)\}$ with $v_1 = h$ and $v_p = s$

The objective is to consider the trade-off between the server cost (SC) and the path cost (PC). Therefore, the problem can be stated as choosing a path p^* in all available paths to minimize the total cost function (TC), which is a combination of SC and PC

$$TC(SC, PC) = \alpha \cdot SC + \beta \cdot PC, \tag{1}$$

$$p^* = \min_p \{TC \mid p \in P_{s,h}\}, \tag{2}$$

where α and β are proportional coefficients and $P_{s,h}$ is all available paths from host s to server d. It is undeniable that link cost LCs plays an essential role in constructing the path cost function PC. To ensure flow requirements, the following constraints are taken into account

$$\sum_{k \in F} x_{ij} \beta_k \le b_{ij}, \tag{3}$$

$$\sum_{(i,j) \in E} x_{ij} LC_{ij} \le \sigma_k, \tag{4}$$

where F is the set of unidirectional flows, x_{ij} is a binary coefficient; equal to 1 if a path is going through the link (i, j) and equal to 0 otherwise, β_k is the bandwidth of flow k, b_{ij} is the available bandwidth between switch i and j, σ_k is the maximum requested cost of flow k, and LC_{ij} is the cost of the link between switch i and j. Besides,

$$x_{ij} = \begin{cases} 1 & \text{if flow } k \text{ uses link } (i, j) \\ 0 & \text{otherwise.} \end{cases}$$

Constraint (3) ensures that the bandwidth capacity for a link (i, j) is enough for every flow going through that link. Constraint (4) guarantees that the cost of all the selected links (i, j) does not surpass the maximum acceptable cost for the flow k.

4.2. Link cost prediction

This section presents the overview of the process of creating the LSTM-based link cost prediction model (LCP) as depicted in Figure 4. This process has four stages: data collection, preprocessing, construction, and model deployment.

In the first stage (data collection), the network traffic is obtained once every three seconds at the data layer. Depending on a specific application, a wide range of features of packets can be extracted, such as the number of bytes sent, the number of bytes received, the network protocol used, and others. In this paper, four selected features of the SARS problem are

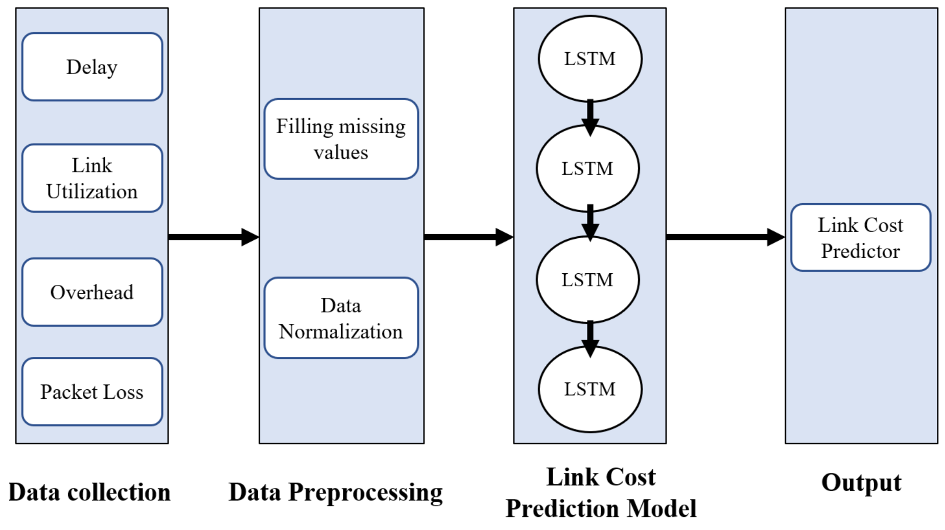

Figure 4: The process of building pretrained LSTM-based link cost prediction model

delay, link utilization, link overhead, and packet loss rate. These QoS metrics are crucial factors in estimating the network's overall performance and services. The collected original network traffic is then processed in the second stage (data preprocessing), which uses various techniques such as filling in missing values or data normalization. The mean interpolation method is used to fill in the missing values of the QoS data in this paper. Moreover, the data normalization operation is applied to map the QoS data range to the range [0, 1], which aims to reduce the impact of imbalanced data units (too large or too small). However, this transformation can be quite sensitive to outlier values.

In the third stage, the LCP model is constructed by the LSTM networks and fully connected neural networks (FNN). To be more specific, the transformed QoS features already obtained in the second stage is to flow through the multi-layer LSTMs before entering the multi-layer FNNs. The LSTMs will learn the long-term non-linear relationship between our target-link cost values and time-series QoS metrics. On the other hand, the FNNs apply various linear transformations to learn the matrix weights that can predict future link cost values. The predicted links' costs are then moved into a new stage: the SARS mechanism.

4.3. Server and route selection mechanism

This section depicts the SARS algorithm by the following ideas: The first is balancing the link traffic distribution, and the second is distributing the load equally among the servers. Therefore, the total cost includes the path and server costs.

To find the path cost, the LC values are predicted by our LCP models (as explained in Section 5) in multiple SDN domains. Then, the path cost is determined by the sum of all the LC values of the crossed links over the path p

$$PC(p) = \sum_{i \in p} LC_i. \tag{5}$$

To estimate the server cost (SC), use Eq. (6) where ST is the current server traffic (e.g., bytes/s, etc.) and SB is the server bandwidth capacity

$$SC(s) = \frac{ST}{SB}. \tag{6}$$

As mentioned in Subsection 4.1, the proposed total cost TC can be reached by taking a linear combination (weighted sum) of the SC and PC

$$TC(SC, PC) = \alpha \cdot SC + \beta \cdot PC. \tag{7}$$

It is worth mentioning that the sum of α and β equals one, and these coefficients represent the importance of SC and PC.

Algorithm 1 QoS-based SARS algorithm

```
Input: Graph $G = (V, E)$, set $N$ of SDN domains, set $C$ of controllers, set $DB$ of databases,
    set $M$ of LCP models
Output: Server $s \in S$ and Path $p = \{(v_1, v_2), (v_2, v_3), ...(v_{p-1}, v_p)\}$
 1: In every time period:
 2: for each SDN domain in $N$ domains do
 3:    Predict link cost values using the LCP model
 4:    Compute $SC$ values using (6)
 5: end for
 6: if (a new server connection is requested) then
 7:    Get predicted link costs from the set of local databases using a consistency mechanism
 8:    for each server $s \in S$ do
 9:       Compute $PC$ values using (5)
10:       Compute $TC$ value using (7)
11:    end for
12:    Choose path $p$, server $s$ that minimizes $TC$ value
13:    for each controller $c \in C$ do
14:       Insert forwarding rules to OpenFlow switches for the selected path
15:    end for
16: end if
```

The proposed SARS algorithm is depicted in Algorithm 1. The algorithm's first loop ranges over the SDN domains, which predict their links and server costs using the already pre-trained deep learning LCP model. It is important to note that each domain would have its own LCP model, which is explained in Subsection 5.3. When a new server connection is requested, the application pulls these predicted values from a local database with a global view of all knowledge among multiple SDNs using the ACM (as explained in Section 3). Then, the algorithm's second loop iterates over servers and finds each server's shortest path by calculating its path cost and total cost. Next, the path p and server s are chosen with minimum TC value. This is when traffic flows are installed to the switches in the selected path p. Next section will discuss how the LSTM is trained to predict link cost values.

5. LSTM-BASED LINK COST PREDICTION MECHANISM

The proposed LSTM-based link cost prediction mechanism is presented in this section. The objective is to predict link cost values for the SARS mechanism.

5.1. Data collection

This module is used to collect the network features generated from the traffic flows of communication between hosts and servers in the SDN inter-domains. Four features, which include delay (DL), link utilization (LU), packet loss rates (PLR) and link overhead (*LO*), are extracted in each SDN domain, thanks to the API PortStatistics [17]. Using the Round Robin algorithm [14], each server is selected following a circular queue.

Having collected data, we can consider a training set $D = \{(x_1, y_1), ..., (x_T, y_T)\}$ consisting of T inputs $x_t \in \mathbb{R}^4$ and corresponding link cost $y_t \in \mathbb{R}$, $t = 1, 2, ..., T$. Each x_t can be treated as a 4-dimensional feature vector of an edge e_{i-j} at time t. For example, an edge e_{i-j} can have a vector $x_1 = (0.5, 0.2, 0.3, 0.4)$, where its dimensions are DL, LU, PLR and LO. According to [18], LO can be calculated as in Eq. (8)

$$LO = \frac{(ByteSent + ByteReceived) - ByteThreshold}{ByteThreshold}. \tag{8}$$

The relationship between the feature vector x_t and the link cost y_t is defined as

$$\begin{aligned} y_t = x_{Delay\_t} \cdot \alpha_{Delay} &+ x_{LinkUtilization\_t} \cdot \alpha_{LinkUtilization} \\ &+ x_{PacketLossRate\_t} \cdot \alpha_{PacketLossRate} \\ &+ x_{LinkOverHead\_t} \cdot \alpha_{LinkOverHead}, \end{aligned} \tag{9}$$

where the sum of α_{Delay}, $\alpha_{LinkUtilization}$, $\alpha_{PacketLossRate}$, and $\alpha_{LinkOverHead}$ is equal to 1.

5.2. Data preprocessing

In this module, the QoS parameters are captured at the data link layer to build the real dataset within approximately 38 hours. Table 1 describes the number of samples in 18 SDN domains.

Firstly, the noise values generated by the monitoring tools (e.g., link layer discovery protocol) are removed from each of the 18 datasets. After that, the links' features are scaled between 0 and 1 [6] via the normalization technique (Eq. 10) to decrease the range of large input values. This technique increases the speed at which the model is trained by decreasing the time required to converge to the local minima [20]

$$\hat{X} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}. \tag{10}$$

5.3. LSTM-based link cost prediction

This section presents the detail of LSTM-based link cost prediction. Figure 5 depicts the two main phases of the LCP module: the training and testing phases. The LCP module takes collected features from the previous data collection and data preprocessing modules as input to the deep learning algorithm.

In Table 1, each row of the table represents a dataset in a specific domain. Each dataset in each domain is then split into three sets: the training, the validation, and the testing sets, with a percentage of 70%, 15%, and 15%, respectively. This means that during the training phase, 18 domains would train their own dataset independently in order to generate 18 LCP models for the testing phase. In the testing phases, if any links belong to a specific domain, their *LC* values would be predicted by the LCP model from that domain. The *LC* values

Table 1: The number of samples in each SDN domain

Ip of SDN domain	SDN controller platform	Number of samples
10.20.0.200	onos	182 990
10.20.0.206	onos	185 592
10.20.0.207	onos	141 275
10.20.0.208	onos	163 191
10.20.0.209	onos	146 592
10.20.0.211	onos	174 510
10.20.0.212	ryu	169 058
10.20.0.213	ryu	179 679
10.20.0.214	ryu	145 609
10.20.0.215	ryu	166 987
10.20.0.216	ryu	155 307
10.20.0.217	ryu	173 008
10.20.0.218	ryu	157 134
10.20.0.219	onos	148 007
10.20.0.220	onos	158 085
10.20.0.221	onos	155 963
10.20.0.222	onos	184 028
10.20.0.223	onos	152 734

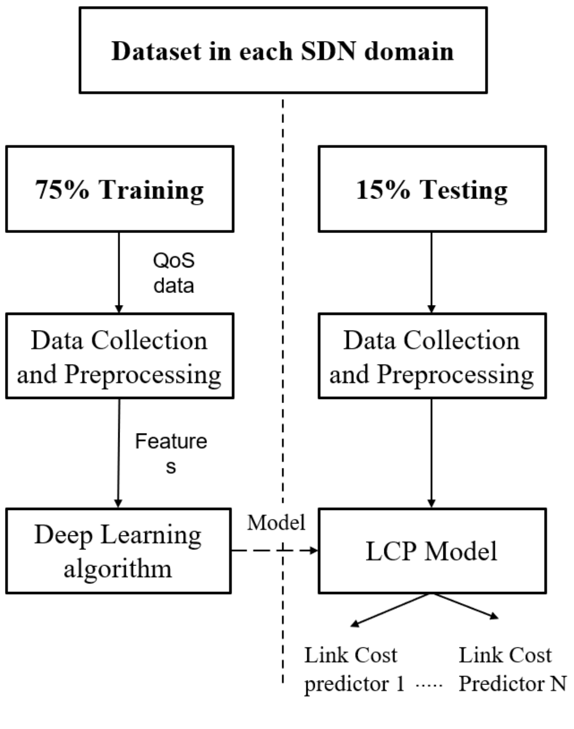

Figure 5: The deep learning-based link cost prediction method

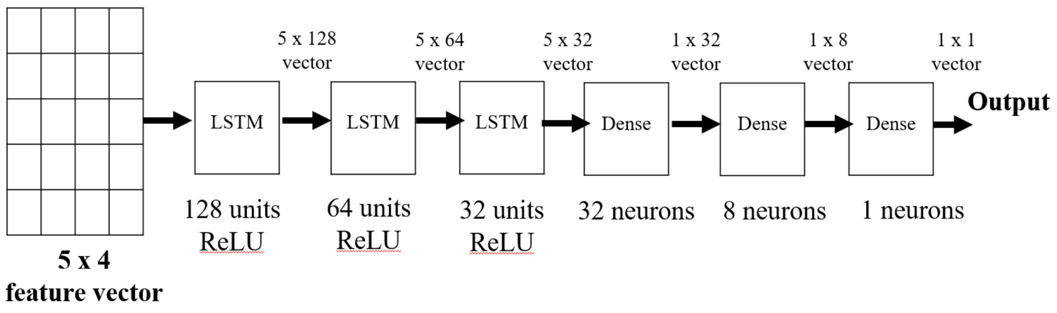

Figure 6: Architecture of the deep learning-based link cost prediction model

are predicted from the history of QoS parameters that include delay, link utilization, link overhead, and packet loss rate. Now, moving on to the architecture of an LCP model, it is described in Figure 6 and contains two essential layers:

LSTM layer: There are three LSTM layers in this architecture. The first one receives the input as a 5 × 4 feature vector where the first dimension is the number of *time_steps*, and the second dimension is the number of features (DL, LU, PLR, and OH). It is worth mentioning that the *time_steps* is selected after the overall performance of the LCP model is assessed. The output of this layer is a 5 × 128-dimensional vector (or *time_steps* × *LSTM_neurons* dimensional space). In the second LSTM layer, the 5 × 128-dimensional vector from the previous step is regarded as an input, generating a 5 × 64-dimensional vector. Likewise, the third LSTM layer gets the 5 × 64-dimensional vector and produces a 5 × 32-dimensional vector as an output.

ReLU activation function: Each output vector from each LSTM layer is passed into a ReLU activation function. The ReLU maps every element of a 5 × n-dimensional vector into new values as given by the following Eq. (11)

$$f(x) = \begin{cases} 1 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases} \tag{11}$$

Dense layer: There are three dense layers (or fully connected layers) with 32, 8, and 1 neurons. In summary, Table 2 succinctly describes the neural network-based link cost prediction model's dimension.

Table 2: Dimension of the neural network-based link cost prediction model

Layer	Input space	Output space	Activation
LSTM	5 × 4	5 × 128	ReLU
LSTM	5 × 128	5 × 64	ReLU
LSTM	5 × 64	5 × 32	ReLU
Dense	5 × 32	1 × 32	
Dense	1 × 32	1 × 8	
Dense	1 × 8	1 × 1	

In the testing phase, features are collected and pre-processed before being fed into the LCP model to predict the link costs.

6. EXPERIMENTAL RESULTS

6.1. Experimental setup

According to the experiments, a distributed system is implemented with one master and 18 slave machines. The master machine is a Dell PowerEdge R840 with 32 GB memory. The server and route applications are implemented on this machine to make the routing and server selection decisions. The remaining 18 slave machines include 8 RYU controllers and 10 ONOS controllers in charge of collecting network states, predicting LCs, and exchanging predicted values with one another. Mininet [11] is used to emulate the network topology. Such a large-scale topology is a Viatel Europe [15] containing 92 switches and 18 replica servers.

To verify the scalability of inter-SDN domains with KDN architecture, the large Viatel Europe topology is divided into 18 subdomains, where each of which is controlled by one SDN controller. To assess the performance of the LCP model, the topology is configured as a dynamic network. In this case, after a random time, denoted as RT, the state of links is changed based on a probability value. The network traffic generator used in this paper is the simple HTTP server [23], which generates files of different sizes [10, 60] MB.

Dynamic network	Parameter
Link	Random Delay [25, 50, 75, 100, 125]ms
Switch	Random Loss [0.1, 1, 2, 5]%
Bandwidth	100MB
Traffic generation tool	Simple HTTP server
Traffic range	[10, 60] MB

Table 3: Dynamic network configurations

6.2. Tuning hyperparameters of the link cost prediction model

This section depicts the process of tuning hyperparameters for our LCP model compared with other network architectures. As mentioned in Subsection 5.2, each SDN domain has its dataset comprising roughly 150 000 samples. The obtained dataset from these 18 domains can go up to more than 2.7 billion samples. With the explosion of the amount of data, the traditional centralized ML model might not be feasible as it requires intensively increasing computation time and enormous resources (e.g., high-performance GPU, expensive specialized hardware, etc.) [21]. To overcome this limitation, this paper takes advantage of the distributed training approach where each SDN domain trains its own data.

It has been demonstrated across numerous tasks that stacked LSTMs can outperform single-layer networks [19]. Therefore, this paper exploits this characteristic by stacking multiple LSTM, dense layers, and LSTM units. However, as the number of stacks increases, the training costs can rise quickly.

To verify the performance of the proposed LSTM network in link cost prediction, the mean squared error (MSE) is chosen in this phase Eq.(12)

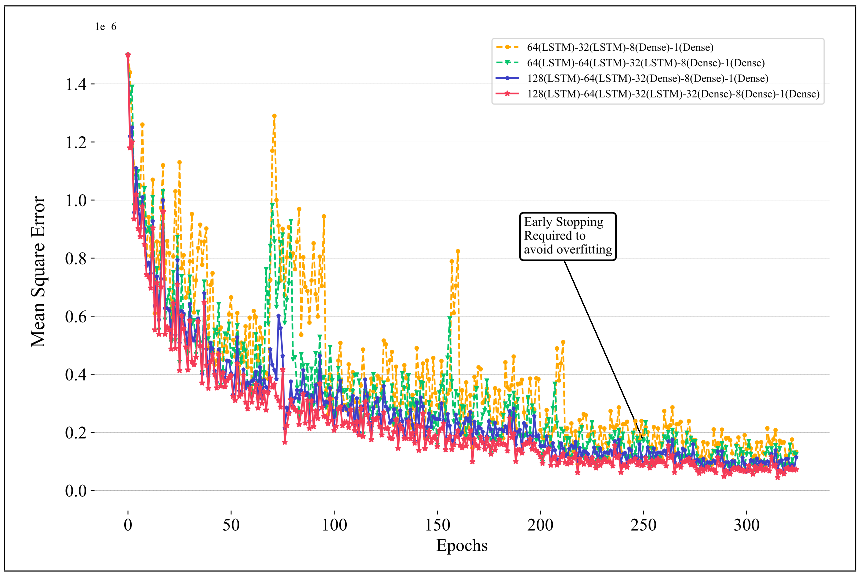

Figure 7: Effect of LSTM architecture on convergence in learning

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(\tilde{y_t} - y_t)^2. \tag{12}$$

Figure 7 illustrates the 18 SDN domains' average MSE loss with different LSTM architectures according to [7]. It can be observed that over the period shown, a dense neural network offers a better learning capability than a spare one. One note-worthy observation is that the MSE error of the "128(LSTM)-64(LSTM)-32(LSTM)-32(Dense)-8(Dense)-1(Dense)" neural network architecture used in all 18 domains was the lowest during most of the given epochs. After 150 epochs, it converged to a global minimum, achieving approximately $1.8 \cdot 10^{-7}$, which was much lower than other networks, which all achieved more than $2.4 \cdot 10^{-7}$. As a result, the early stopping technique is used to hinder it from overfitting.

6.3. Benchmarks

This paper proposes and re-implements some server and route selection techniques from [9] as benchmark algorithms in order to compare their performances with our proposal (Algorithm 1). These techniques are round robin with strong consistency (RR-WSC), QoS-based algorithm with strong consistency (QSARSA-WSC), link cost prediction with strong consistency (LCP-WSC), and QoS-based algorithm with adaptive consistency (QSARSA-WAC), all of which will be depicted in the following sections.

6.3.1. Benchmarks with strong consistency

RR-WSC: According to [9], the RR-WSC benchmark selects servers following a circular queue. Moreover, it is fixed with two parameters N_r and N_w as 1 and 18, respectively. As detailed in Section 3, this strong consistency mechanism guarantees information consistency across 18 domains at the expense of extra communication overhead.

QSARSA-WSC: This benchmark is based on a similar strong consistency mechanism to RR-WSC. The only difference is that QSARSA-WSC selects paths using Dijkstra's algorithm with real-time link cost values. These values can be estimated using Eq. (9).

LCP-WSC: In this case, LCP-WSC works on LSTM models that are already trained to predict the link cost values. However, it uses a strong consistency mechanism rather than an adaptive one.

6.3.2. Benchmark with adaptive consistency

QSARSA-WAC: This benchmark has the same path selecion mechanism with the second benchmark QSARSA-WSC. However, it takes advantage of an adaptive consistency mechanism with N_r and N_w being adaptively updated according to the state of a network. As a result, this mechanism helps the network become more consistent across multi-domains in QoS data but with less communication overhead.

6.4. Performance analysis

The experiments and numerical results are illustrated in this section to evaluate our proposal's performance against benchmarks.

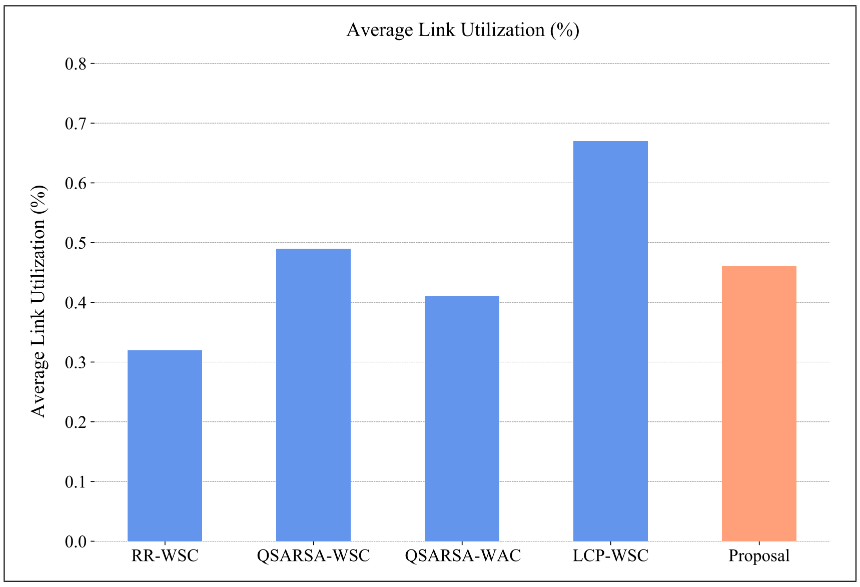

Figure 8: Comparison of Average Link Utilization between the proposal and benchmark methods

6.4.1. Comparing the QoS performance

According to Figure 8 and Figure 9, our proposal ranked in third place on the PLR and ALU charts. Regarding algorithms that use the adaptive consistency mechanism, our

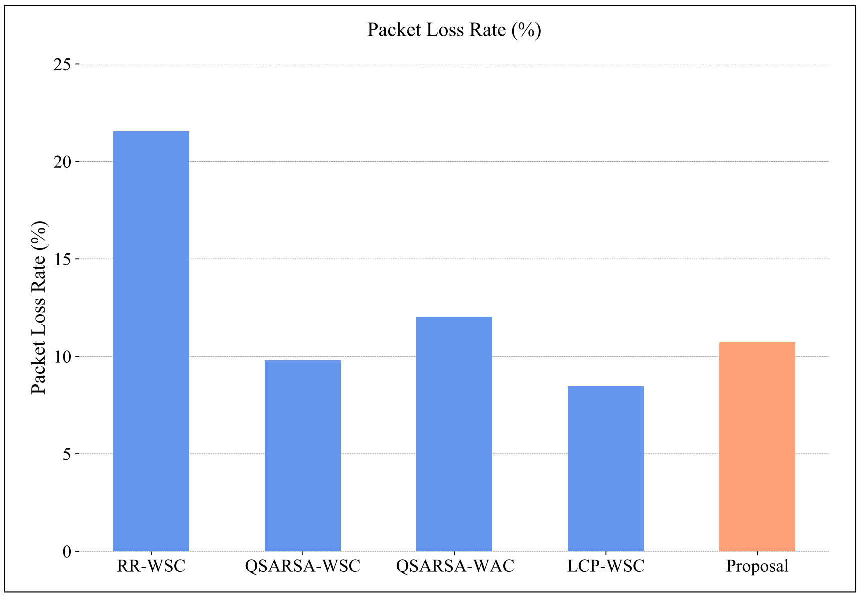

Figure 9: Comparison of Packet Loss Rate (%) between the proposal and benchmark methods

proposal achieved 11% of PLR and 46% of ALU. These results were slightly higher than those of the QSARSA-WAC algorithm which obtained around 9% of PLR and 43% of PLR.

The QSARSA-WSC and LCP-WSC algorithms that use the strong consistency mechanism, had a better QoS performance than other algorithms, with a PLR of each under 10% and ALU above 50%. It is striking that the RR-WSC is the only algorithm that obtained the highest PLR (22%) and the lowest ALU (31%). In summary, RR-WSC has proven ineffective with the worst case of PLR and ALU metrics because it only selects servers without considering the network state. Compared to the RR-WSC, the QSARSA-WSC and QSARSA-WAC calculate link cost values every ten time-steps, producing better QoS performance. However, they do not take full advantage of all the past knowledge and still require high processing time. By contrast, in using an LSTM model, our proposed algorithm exploits all the past knowledge.

6.4.2. Comparing the response time and overhead

In this phase, an SLA threshold is set up at approximately 170,000 ns. This threshold is an upper bound to estimate the servers' response time using different tested algorithms. From Figure 10, by the end of the period, it can be observed that our proposal achieved the lowest servers' response time (approximately 60,000 ns). The RR-WSC algorithm attained the highest response time (around 170,000 ns).

During the first phase (the first 5000s), our proposal and LCP-WSC algorithms, which exploited the LSTM model, obtained a better server response time than their counterparts. The proposal and LCP-WSC curves dropped below the SLA line, from 170,000 ns to just under 160,000 ns each. On the other hand, RR-WSC, QSARSA-WSC, and QSARSA-WAC were all higher than the SLA threshold (all above 170,000 ns).

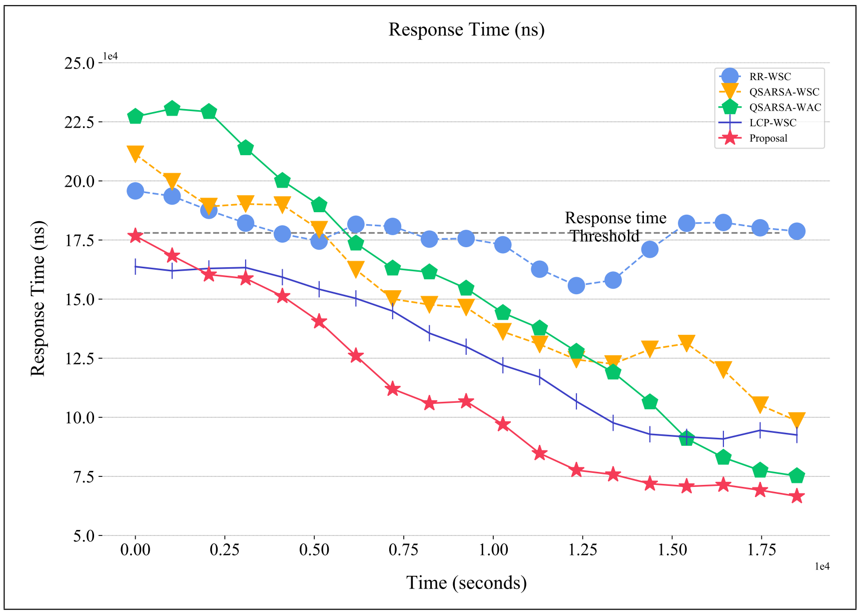

Figure 10: Comparison of Response Time between the proposal and benchmark methods

During the remaining phase (from 5000s onwards), the RR-WSC was the only algorithm to reach a plateau, varying 160,000 ns and 170,000 ns. At the same time, all other algorithms decreased significantly. Our proposal ended up at the bottom place of the graph.

In summary, for the SARS problem, the response time is the most important aspect in many online services (e.g., file sharing, online gaming, video streaming, etc.) [24]. Also, the overhead metric Eq. (8) is considered as the excessive network traffic factor on links. Therefore, Table 4 illustrates all our experiments' average response time and link overhead. In Table 4, the third and fifth columns compare the difference between our proposal and benchmarks regarding average response time and overhead metrics. Our proposal outperforms the LCP-WSC with the average response time and overhead at 14% and 25%, respectively.

Table 4: Comparison of average response time and average link overhead between proposal and benchmarks.

Benchmark	Response time (ns)	Improvement of response time	Overhead	Improvement of overhead
RR-WSC	177,282	38%	0.70	75%
QSARSA-WSC	150,737	27%	0.40	57%
QSARSA-WAC	152,897	28%	0.35	51%
LCP-WSC	128,000	14%	0.23	25%
Our proposal	**110,089**	-	**0.17**	-

7. CONCLUSIONS

This paper proposes an LSTM-based approach for selecting servers and routes in a distributed SDN network to enhance long-term network operation with high QoS. Our approach addresses the single point of failure issue in intra-SDN domains by introducing KDN. Additionally, our proposed server and route selection method exploits link cost values predicted by a Long Short-Term Memory network (LSTM) to collaborate and share predicted knowledge among multiple domains before selecting a server and route. The objective is to efficiently utilize links and prevent bottlenecks. Simulation results indicate that the LSTM predicts link cost parameters with a low MSE of approximately $1.8 \cdot 10^{-7}$. Furthermore, our proposal significantly improves link utilization, packet loss, response time, and overhead, by 15%, 10%, 14%, and 25%, respectively, compared to benchmarks.

Our work has two limitations. Firstly, there is a possibility of an imbalanced dataset situation, where the sample size in one domain can be significantly larger than in other domains (e.g., new domains), which might prevent the model from learning knowledge across multiple network domains. Additionally, our route and selection mechanism is not adaptive in complex environments because routing decisions cannot be updated if unexpected congestion occurs during the routing process.

However, these limitations present opportunities for future research. To address the first limitation, we plan to explore Transfer Learning (TL), which transfers already learned knowledge from one domain to other data-hungry domains. To deal with the second limitation, we aim to study various evolutionary algorithms (e.g., genetic, ant-colony techniques, etc.) with multi-intelligent agents to make our server and route selection mechanism more adaptive in stochastic environments.

ACKNOWLEDGMENT

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314

REFERENCES

[1] H. A. Akyıldız, I. Hokelek, M. Ileri, E. Saygun, and H. A. Cirpan, “Joint server and route selection in SDN networks,” in *2017 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, 2017, pp. 1–5.

[2] S. Asadollahi, B. Goswami, and M. Sameer, “Ryu controller’s scalability experiment on software defined networks,” in *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, 2018, pp. 1–5.

[3] A. Azzouni, R. Boutaba, and G. Pujolle, “Neuroute: Predictive dynamic routing for software-defined networks,” in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–6.

[4] A. Azzouni and G. Pujolle, “NeuTM: A neural network-based framework for traffic matrix prediction in SDN,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–5.

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 1–6.

[6] S. Bhanja and A. Das, "Impact of data normalization on deep neural network for time series forecasting," 2018. [Online]. Available: https://arxiv.org/abs/1812.05519

[7] J. Bhatia, R. Dave, H. Bhayani, S. Tanwar, and A. Nayyar, "SDN-based real-time urban traffic analysis in VANET environment," *Computer Communications*, vol. 149, pp. 162–175, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419308916

[8] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 3–10.

[9] N.-T. Hoang, C.-S. Duong, T.-L.-T. Nguyen, V. Tong, and H. A. Tran, "Knowledge-defined heterogeneous network: Use-case of qos-based server and route selection in large-scale network," in *Proceedings of the 11th International Symposium on Information and Communication Technology*, 2022, pp. 150–157.

[10] N.-T. Hoang, H.-N. Nguyen, H.-A. Tran, and S. Souihi, "A novel adaptive east-west interface for a heterogeneous and distributed sdn network," *Electronics*, vol. 11, no. 7, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/7/975

[11] R. Jawaharan, P. M. Mohan, T. Das, and M. Gurusamy, "Empirical evaluation of SDN controllers using mininet/wireshark and comparison with Cbench," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–2.

[12] L. Jiang, W. Xia, F. Yan, L. Shen, Y. Zhang, and Y. Gao, "QoS-aware Routing Optimization Algorithm using Differential Search in SDN-based MANETs," in *2021 IEEE Global Communications Conference (GLOBECOM)*, 2021, pp. 1–6.

[13] L. Jiang, W. Xia, Y. Zheng, F. Yan, L. Shen, Y. Zhang, and X. Yang, "QoS sensitive routing algorithm with link quality prediction in SDN-based Ad Hoc networks," in *2020 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2020, pp. 1188–1193.

[14] S. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Round-robin based load balancing in software defined networking," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2015, pp. 2136–2139.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[16] H. Li, Y. Govind, S. Mudgal, T. Rekatsinas, and A. Doan, "Deep learning for semantic matching: A survey," *Journal of Computer Science and Cybernetics*, vol. 37, no. 4, pp. 365–402, 2021.

[17] ONOS. (2021) Portstatistics api. [Online]. Available: https://api.onosproject.org/1.12.0/org/onosproject/net/device/PortStatistics.html

[18] S. Patil, "Load balancing approach for finding best path in SDN," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, pp. 612–616.

[19] A. Sahar and D. Han, "An LSTM-based indoor positioning method using Wi-Fi signals," in *Proceedings of the 2nd International Conference on Vision, Image and Signal Processing*, 2018, pp. 1–5.

[20] S. Sanagavarapu and S. Sridhar, “Sdpredictnet-a topology based sdn neural routing framework with traffic prediction analysis,” in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0264–0272.

[21] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A survey on distributed machine learning,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–33, 2020.

[22] Y.-J. Wu, P.-C. Hwang, W.-S. Hwang, and M.-H. Cheng, “Artificial intelligence enabled routing in software defined networking,” *Applied Sciences*, vol. 10, no. 18, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/18/6564

[23] Y. Xie, “Servr: A simple http server to serve static files or dynamic documents,” *R Package Version 0.4*, vol. 1, 2016.

[24] H. Zhong, Y. Fang, and J. Cui, “Reprint of LBBSRT: An efficient SDN load balancing scheme based on server response time,” *Future Generation Computer Systems*, vol. 80, pp. 409–416, 2018.

Received on October 16, 2022
Accepted on February 09, 2023