Server and Route Selection Optimization for Knowledge-defined Distributed Network Based on Gambling Theory and LSTM Neural Networks

Son Duong, Nguyen Tuan, Hoang Nam-Thang, Tong Van, Tran Hai-Anh, Nguyen Giang, Mellouk Abdelhamid, Tran Truong

Server and Route Selection Optimization for Knowledge-defined Distributed Network Based on Gambling Theory and LSTM Neural Networks

Son Duong*, Tuan Nguyen*, Nam-Thang Hoang*†, Van Tong†, Hai-Anh Tran†, Giang Nguyen†, Abdelhamid Mellouk‡, Truong Tran§

*Faculty of Information Technology, Hanoi University of Civil Engineering, Vietnam
{son167464, tuan1553564, thanghn}@huce.edu.vn

†School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam
{vantv, anhth, giangnl}@soict.hust.edu.vn

‡University of Paris-Est Creteil, LISSI, Tinc-NET, F-94400, Vitry-sur-Seine, France, mellouk@u-pec.fr

§School of Science, Engineering, and Technology, Penn State University, USA, truong.tran@psu.edu

***Abstract*—Server and route selection (SARS) optimization is a critical aspect of traffic engineering to allocate network resources to meet diverse service requirements effectively. Existing studies have primarily focused on finding profitable or optimal solutions for the SARS problem within current time steps, considering specific constraints. However, they often have failed to address the dynamic and uncertainty of future network states. To address this gap, this paper proposes an algorithm named GAL to optimize server costs and response time while accounting for future network dynamics. GAL combines a server selection inspired by the gambling theory and a network routing based on Long Short-Term Memory Networks (LSTM). The server selection method is formulated as a gambling problem and solved using the decision-making Tug-of-War (TOW) dynamic algorithm. The routing mechanism is optimized based on predictions of future network states made by LSTM neural networks, which excel in capturing long-term dependencies. We have implemented GAL through a distributed software-defined networking (SDN) system and obtained good evaluation results regarding average response time and server cost compared to benchmark methods. These results demonstrate that GAL can effectively tackle the SARS optimization problem by considering present constraints and future network dynamics. This study can advance traffic engineering and lays a foundation for more robust resource allocation strategies in dynamic network environments.**

***Index Terms*—SARS Optimization, Multi-armed Bandits, Software Defined Networking, Distributed SDN, LSTM Networks.**

I. Introduction

The optimization of Server and Route Selection (SARS) is an important problem in computer networks [1]. Server selection is the process of selecting a provisioning server from a set of servers that provide heterogeneous services, while routing determines how packets are delivered from the source to the destination server by selecting the path for packet transmission. An effective SARS scheme can shorten the users' waiting time, allocate traffic evenly into network components, and provide high-quality services. If the SARS scheme can optimize itself without any additional engineering effort, network operators may have more time for other responsibilities while still enhancing the significant benefits of automated network control [2]. Among other technologies, software-defined networking (SDN) can enable dynamic and automated network control better than traditional network management. SDN offers flexibility and programmability by separating the control plane of network devices from their data plane, allowing centralized management of those devices via automated scripts [2].

In SDN, the SARS problem has been studied extensively. Among these studies, Tran et al. [3] proposed a server selection method for a content delivery network (CDN) - a technology that replicates content from the original server to replica nodes closer to users). In that work, the server selection is based on a multi-armed bandit solution, which chooses a replica server regarding the idea that the longer a server has been running, the less likely it is to be selected. After a long enough time, through trial and error, an optimal replica server will eventually be exploited. In [4], Kaur et al. utilized multiple distributed servers with a load balancer acting as a gateway for distributing requests. The load balancer is based on a round-robin mechanism where each server is uniformly selected according to a circular queue. The method is based on several real-time metrics, such as server cost, hop count, and packet overhead in real-time. In overall, the works mentioned above mainly consider optimizing the SARS scheme based on the current state of the network. However, due to the stochastic and dynamic environment, the network metrics might constantly change with unexpected high loads, traffic bottlenecks, or deactivated links. Thus, it is necessary to design a solution that considers future network states and optimizes long-term network performance in an uncertain environment.

This paper proposes a hybridization of a gambling-inspired server selection method and an LSTM-based routing mechanism, namely Gambling and LSTM (GAL) algorithm, to address the SARS problem, while considering the future network' uncertainty and stochastic charac-

teristic. Specifically, GAL aims to target the server cost and server response time minimization in long-term network performance. There are two key stages in our method. First, we formulate the server selection as a multi-armed bandit (MAB) problem - a gambling problem of choosing a profitable machine from a set of machines that provide stochastic rewards to maximize cumulative reward in the long term. Then, we utilize a tug-of-war (TOW) dynamic algorithm that is inspired by the game tug-of-war, to solve the MAB problem. The reason for such a choice is because it is proved that the TOW algorithm showed superior performance in many real-world applications compared to other traditional MAB algorithms (e.g., ϵ-greedy, UCB1, etc.) [5]. Second, we develop a routing mechanism that is optimized based on the future network states predicted by an LSTM model.

This work makes the following key contributions:

- We formulate server selection as an MAB gambling problem and propose an algorithm to address it. Specifically, we introduce TOW with **a new reward function** that incorporates server cost and server response time. Additionally, we propose a novel $Q(.)$ function that effectively estimates the quality of servers.
- We introduce GAL, **a pioneering hybridization of a gambling-inspired server selection approach and an LSTM-based routing algorithm**. GAL uses two vital factors of server performance, namely response time and server cost, as well as three key metrics for links: link utilization, packet loss, and latency.
- We provide **a comprehensive performance analysis of GAL compared to different types of MAB methods**, including ϵ-Greedy, UCB1, and the original TOW algorithm. The analysis focuses on their efficiency in terms of response time and average server cost within a distributed and heterogeneous SDN network utilizing ONOS and RYU controllers.

These contributions can advance the field of server and route selection optimization. By framing server selection as a MAB gambling problem and introducing enhancements to the TOW algorithm, our approach offers a novel perspective on resource allocation. The GAL algorithm's hybridization of gambling-inspired server selection and LSTM-based routing incorporates critical factors and metrics, leading to efficient network resource utilization. Through a comprehensive evaluation, including comparisons with established methods, we demonstrate the superior performance of GAL in terms of response time and average server cost. This research provides valuable insights for designing robust resource allocation strategies in dynamic network environments, contributing to the continued progress of traffic engineering.

The remainder of this paper is structured as follows. Section II presents some related background. Section III describes the formulation of the problem and our proposal to tackle it. Section IV provides experimental scenarios and results. Finally, Section V shows the conclusions of this research.

II. Background

In this section, we provide a brief overview of the MAB gambling problem and various popular methods for solving it. Indeed, our proposed server selection method is inspired by the MAB techniques.

A. Multi-Armed Bandits Theory

The MAB problem is a classic reinforcement learning problem in which an agent has to choose between multiple actions (e.g., slot machines or the one-armed bandits) in order to maximize its cumulative rewards in the long-term [5] through a series of choices, where the beginning payout of each one-armed bandit is unknown. The MAB problem is modeled as a tuple $< A, T, R >$ where $A = \{a_1, a_2, ..., a_n\}$ is a set of actions, T is the total of rounds, and $R_{a_i}(t)$ is a scalar (reward) that an agent receives when selecting an action a_i at round t. Let denote $P_v \in P$ as one playing policy (a sequence of selected actions) after T rounds. The goal of an agent is to find an optimal policy that maximizes the sum of cumulative rewards:

$$P^* = \arg\max_{P_v \in P} \left\{ \sum_{t=1}^{T} R_{a_i}(t) \right\} \tag{1}$$

As the reward distribution of each action is unknown, an agent has to estimate the quality (goodness) of every action at a current round to select an optimal action:

$$Q_{a_i}(t) = \frac{1}{N_{a_i}} \sum_{u=1}^{t} R_{a_i}(u) \tag{2}$$

where $Q_{a_i}(t)$, referred as $Q(.)$ function, is the *expected reward* of an action a_i at time slot t, and N_{a_i} is the number of times an action a_i that has been previously selected. However, Equation (2) raises an *exploitation* versus *exploration* dilemma of whether an agent should keep choosing an action with the best value of the $Q(.)$ function (exploitation), or it should choose an action that might not optimize the $Q(.)$ function at the moment (exploration). By exploring other sub-optimal actions, an agent might learn more about their reward distribution that might be superior in the future [6].

B. Multi-Armed Bandits Methods

There are a number of methods that balance the *exploration* versus *exploitation* dilemma for solving the MAB problem. Among them, the most widely used methods are $\epsilon-$greedy, UCB1, and TOW algorithms.

$\epsilon-$Greedy. The $\epsilon-$greedy is the simplest algorithm based on the idea that an agent will exploit an action that maximizes the $Q(.)$ function at current time slot t (see Equation (2)) with a probability $1-\epsilon$, otherwise exploring other random actions.

$$a^*(t) = \begin{cases} \max_{a_i(t)} Q(.) & \text{with probability} 1-\epsilon, \\ \text{a random action } a_i(t) & \text{otherwise.} \end{cases} \tag{3}$$

UCB1. UCB1, or Upper Confidence Bound 1, balances the exploitation-exploration trade-off as it gathers more knowledge about different actions [6]. The intuition behind UCB is as

follows: 1) an action with a high value of *expected reward* should be more prioritized; 2) an action that has been selected frequently in previous rounds should be less exploited. Mathematically, this strategy is expressed as:

$$a^*(t) = \max_{a_i(t)} \left\{ Q(.) + \sqrt{\frac{2\ln(t)}{N_{a_i}}} \right\} \tag{4}$$

where t is a current round. In Equation (4), the left–hand side encourages exploitation: the higher the value of $Q(.)$, the greater the change of exploitation. On the other hand, the right-hand side promotes exploration: If an action a_i has not been selected for a long time, N_{a_i} decreases. Consequently, the right-hand side term increases and an action a_i will be more prioritized.

TOW. The TOW algorithm is inspired by the game tug-of-war. In the context of MAB problem, for simplicity, suppose that there are two teams (machines) $a_1, a_2 \in A$. Let denote the variable X_k, $k \in \{1, 2\}$ represents the displacement of the endpoint k of the rope from their initial position when a team a_k pulls the rope. If $X_1 > X_2$, then the team a_1 is the winner, or we can assume that the machine a_1 is chosen. Thus, the displacement X_1, or how much chance should we choose a machine a_1, is given as:

$$X_1(t) = Q_1(t-1) - Q_2(t-1) \tag{5}$$

$$Q_k(t) = Q_k(t-1) + R_k(t) \tag{6}$$

First, we choose a machine a_k with the highest value of $X_k(t), k \in \{1, 2\}$ (see Equation (5)). Second, we receive a reward $R_k(t)$ from such a machine a_k, and calculate its $Q_k(t)$ function from Equation (6). It is noteworthy that $Q_k(t)$ represents the quality (*expected reward*) of a machine a_k based on information on past experience accumulated until current time t, and current reward obtained. This $Q_k(t)$ function at time slot t will be an input for a new $X_k(t+1)$. In general, the TOW algorithm can be extended with many teams to solve the MAB with multiple slot machines [7].

III. Methodology

This section presents the SARS problem and the proposed GAL algorithm that is composed of a gambling-inspired server selection method and an LSTM-based routing mechanism.

A. SARS Optimization Problem

The server and route selection (SARS) problem is expressed with the use of a global weighted graph $G = (N, E)$ where $N = \{H, S, D\}$ is a set of nodes, and $E = \{(a, b), \forall a, b \in N, a \neq b\}$ is a set of links (edges) connecting all possible nodes of N. The set $H = \{h_1, h_2, ..., h_m\}$ is a set of hosts in the network, $S = \{s_1, s_2, ..., s_n\}$ is a set of servers, and $D = \{d_1, d_2, ..., d_v\}$ is a set of possible forwarding devices (switches in this case). Each link (a, b) is associated with a non-negative scalar $C_{a,b}$, called link cost, that measures the quality of this link. $C_{a,b}$ can be a representation of different metrics (e.g., latency, packet loss rate, etc.) on the link (a, b).

Let consider $P = \{h_o, d_1, d_2, ..., s_i\}$ is one possible route (path) from a host $h_o \in H$ to a chosen server $s_i \in S$. Our aim is to break the SARS problem into two successive subproblems: first selecting an optimal server, second, routing traffic data between hosts and servers. Thus, the following conditions should be taken into account:

Server selection optimization:

- Minimize the average server response time for every user's request. This ensures smooth and seamless user experiences during the long-term period [8].
- Minimize the average server cost. This guarantees allocating a large number of concurrent requests evenly into various servers, thus enhancing resource usage [8].

Routing optimization:

- Minimize the link cost values of all links over a selected route where each link cost value can be a function of different metrics. Reducing the cost means complying with certain quality of service (QoS) metrics and requirements (e.g., less hop-count, reduced latency, etc.) [9].
- Express the link cost $C_{a,b}$ as a combination of three QoS metrics. As mentioned in our study [10], many applications' performance mainly depends on QoS metrics (e.g., link utilization, packet loss, etc.)

B. GAL: Gambling and LSTM Algorithm for SARS

1) GAL: Server Selection

We define the problem of server selection as an MAB problem as follows. When a user (host) $h_o \in H$ requests a service, multiple servers can respond. The promising server s_i is selected as follows:

$$X_{s_i}(t) = Q_{s_i}(t-1) - \frac{1}{n-1} \sum_{s_i' \neq s_i}^{n} Q_{s_i'}(t-1) \tag{7}$$

$$s^*(t) = \max_{s_i \in S} X_{s_i}(t) \tag{8}$$

Equation (7) is an extended version of a TOW algorithm explained in Section II, for $n > 2$, where n is the number of servers, and $X_{s_i}(t)$ represents the *confident level* of choosing a server s_i. After choosing an optimal server s_i (see Equation (8)), the reward $R_{s_i}(t)$ is proposed as:

$$SC_{s_i}(t) = \frac{T_{s_i}(t)}{C_{s_i}} \tag{9}$$

$$R_{s_i}(t) = 1 - \frac{arctan(SC_{s_i}(t)) + arctan(RT_{s_i}(t))}{\pi} \tag{10}$$

Equation (9) shows that the server cost metric $SC_{s_i(t)}$ of a server s_i is calculated by dividing its current traffic $T_{s_i(t)}$ to the maximum capacity C_{s_i} (e.g., Mbit/s, etc.) of a link that is a direct connection between such a server and a switch (endpoint). Equation (10) indicates that the reward of a server s_i, $R_{s_i}(t)$ is measured by its server cost $SC_{s_i(t)}$, and server response time $RT_{s_i}(t)$. It is noteworthy that $arctan(SC_{s_i}), arctan(RT_{s_i}) > \frac{\pi}{2}$ $\forall SC_{s_i}, RT_{s_i} > 0$, thus the numerator $\in [0, \pi]$, consequently $R_{s_i}(t) \in [0, 1]$. The one minus operator is required since the GAL algorithm aims to maximize its reward.

After obtaining $R_{s_i}(t)$, the $Q(.)$ function is calculated as:

$$Q_{s_i}(t) = Q_{s_i}(t-1) + R_{s_i}(t) \tag{11}$$

Equation (11) shows that the quality (*expected reward*) of a server s_i is based on past experiences and current gained reward. However, as the server metrics change dynamically, a parameter referred to as the *forgetting ratio* $\gamma \in [0, 1]$ is proposed to reduce the impact of past experiences. Subsequently, Equation (11) is proposed to be rewritten as:

$$Q_{s_i}(t) = \begin{cases} \gamma Q_{s_i}(t-1) + R_{s_i}(t) & t \leq k, \\ \frac{\gamma \sum_{j=t-k}^{t} Q_{s_i}(j)}{k} + R_{s_i}(t) & \text{otherwise.} \end{cases} \tag{12}$$

where $k \in Z^+$ is the number of past experiences. When the γ is close to 1, $Q_{s_i}(t)$ are largely influenced by past experiences; for γ close to 0, $Q_{s_i}(t)$ is not significantly affected by past experiences. This $Q(.)$ function will then be an input for a new *confidence level* $X_{s_i}(t+1)$.

GAL utilizes a convergence test to decide whether the algorithm should be terminated or not:

$$CV(A) = \frac{\sigma(A)}{E(A)} = \frac{\frac{1}{T_r} * \sum_{u=t-T_r}^{t} (A(u) - \sum_{u=t-T_r}^{t} \frac{A(u)}{T_r})^2}{\sum_{u=t-T_r}^{t} \frac{A(u)}{T_r}} \tag{13}$$

where A is a list of RT or SC values stored from the $t - T_r$ to the t connection times, and T_r is the length of a sliding window. If $CV(A) < ThCV$ (threshold convergence value), then the convergence test is satisfied.

2) GAL: Routing Optimization

This section introduces the future link costs (network states) predictions and the process for finding the optimal path from a host to a selected server.

Let $\mathbf{x_{a,b}} = (x_1, x_2, ..., x_m)^T$ is a features vector for one specific link (a, b) in the network, an entry $x_i \in \mathbb{R}$ is a feature of the link, and $m \in \mathbb{Z}^+$ is the number of features. We constrain the link features to three QoS metrics ($m = 3$): link utilization (LU), packet loss rate (PL), and delay (DL). The link cost $C_{a,b}$ of a link (a, b) is calculated as:

$$C_{a,b} = (\mathbf{x_{a,b}})^T \boldsymbol{\theta} = \begin{pmatrix} x_{LU} & x_{PL} & x_{DL} \end{pmatrix} \begin{pmatrix} \theta_{LU} \\ \theta_{PL} \\ \theta_{DL} \end{pmatrix} \tag{14}$$

where $\boldsymbol{\theta} = (\theta_{LU}, \theta_{PL}, \theta_{DL})^T$ is a weight vector that represents the relationship of three QoS metrics: LU, PL, and DL.

The first objective of the GAL routing process is to predict the link cost value $C_{a,b}$ of any link $\mathbf{x_{a,b}}$ at time step H via a series of $H-1$ historical QoS feature-vectors of that link. We use an LSTM network model to perform the prediction based on one of our previous works as in [11]. Figure 1 provides a general view of the LSTM models with stacked LSTM layers and feed-forward neural networks (FFN).

The primary goal of GAL routing is to find the host-server path that has the minimum total link costs. After the link cost predictions phase, the link costs of all links at a future time step are predicted. Then GAL routing process will utilize the Dijkstra algorithm to determine the shortest route from a host to the selected server (which has been selected using the GAL Server Selection stage as in Section III-B1).

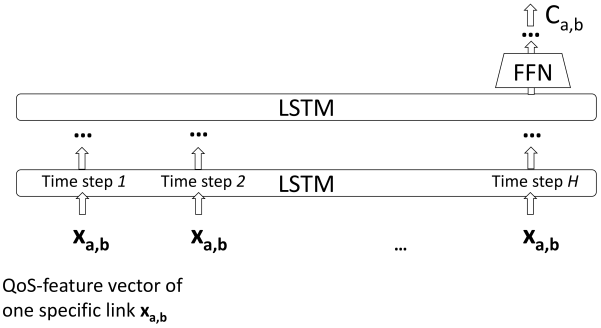

Fig. 1: Link cost prediction using stacked LSTM combined with multiple FFNs. The output of the last LSTM layer is then fed into multiple FNNs which perform prediction.

C. Intergration of the Proposed GAL Algorithm into Distributed Network

Figure 2 depicts the flowchart of the GAL algorithm and its integration into a knowledge-defined distributed network architecture (KDN), which includes three planes: Control, Data, and Knowledge planes. The KDN architecture, derived from the original of SDN architecture, is explained in our previous study [11]. The SARS problem is solved at the top of the Knowledge plane.

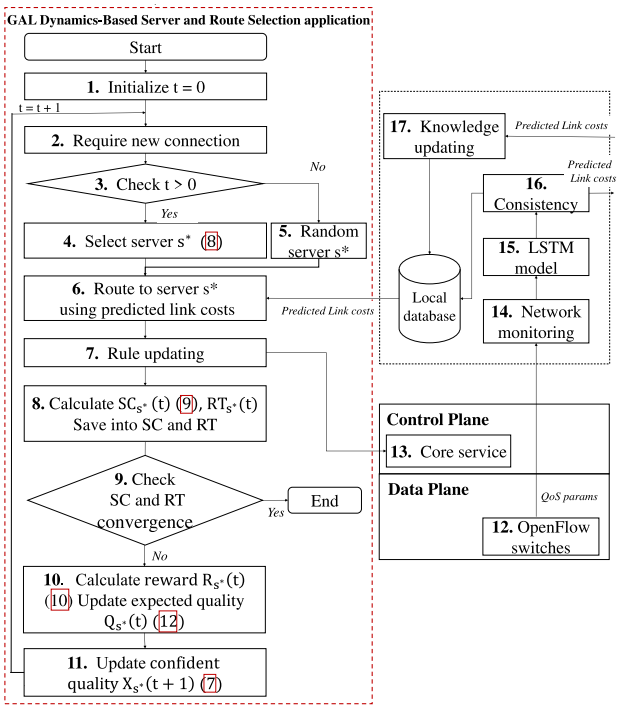

Fig. 2: Flowchart of the proposed GAL algorithm integrated into Knowledge-defined distributed network.

The red frame for our GAL algorithm for solving the SARS for every new request is described as follows.

- **1.** Assumption of the value $t = 0$.
- **2.** Receiving a service request with a set S of servers that can possibly provide the service.
- **3.** Verifying whether the time slot t is greater than 0 or not. If so, move to 4; otherwise, move to step 5.
- **4.** Server selection phase: choosing an optimal server with the highest $X_{s^*}(t)$ Equation (8), which has been calculated at $t - 1$.
- **5.** Server selection phase: randomly selecting a server at $t = 0$.
- **6.** Routing optimization phase: Finding a path from a host to the selected server s^* using the Dijkstra algorithm on the distributed graph with the weights stored in a local database (as shown in the black frame).
- **7.** Generating a set of actions (*forwarding rules*) and installing them into the Core Service in the Control Plane (module 13) for directing actual traffic data.
- **8.** Receiving *server cost* and *server response time* of a selected server s^*, at t.
- **9.** If the $CV(SC)$ and $CV(RT)$ based on Equation (13) in the sliding window length T_r is less than the threshold convergence value *ThCV*, then the GAL algorithm is considered to converge; otherwise, go to step 10.
- **10.** GAL updates the *reward* and *expected quality* functions of the selected server s^* according to Equation (10), (12) respectively.
- **11.** GAL calculates the value of *confident level* at the next time for Equation (7) and continues to select an optimal server and route for a new request.

In the black frame, the Networking monitoring (module 14) is responsible for capturing the metrics of the links from OpenFlow switches (module 12) in the Data plane, and these values are fed into an LSTM model (module 15) that is designed to generate predictive link weights of the graph. Furthermore, these predicted link costs are transmitted to the local database and broadcast to other SDN network domains to ensure information consistency between distributed domains in the Consistency module (module 16). At the same time, the Knowledge Updating module (module 17) is used to receive the predicted link costs from other domains before these costs are stored in the local database.

IV. Experimental Results

This section conducts simulations to validate the effectiveness of the proposed GAL algorithm in terms of server cost and server response time. We also compared our proposed GAL with other SARS benchmarks.

A. Experimental Setup

The SDN controllers are implemented using ONOS and RYU, two widely known SDN controller platforms. Mininet is used to create the network topology, involving a large number of switches and replica servers. The topology Viatel, taken from the Internet Topology Zoo database [12], is chosen to validate the scalability of the proposed GAL algorithm. Sharing common characteristics with a real-world network, Viatel is composed of 92 OpenFlow switches (abbreviated as 92N topology), and it acts as a bridge between many European countries (e.g., France, Italy, Belgium, etc.). The network configurations are summarized in Table I. Every link capacity is uniformly set to 800 Mbps, and the amount of traffic generation is set to be between 80-500 Mbps by a simple HTTP server tool. To verify the proposed GAL algorithm's ability to handle dynamic networks, the network configurations, including random deactivating and activating links, are modified stochastically.

TABLE I: Simulation parameters

Parameters	Values
Number of servers	[18, 30, 50]
Number of switches	92
Link delay (ms)	Random [25, 50, 75, 100]
Link packet loss (%)	Random [0.1, 1, 3, 5]
Link capacity (Mbps)	800
Request counts (per second)	Range [10, 200]
Traffic generation tool	Simple HTTP server
Traffic range (Mbps)	Random [80, 500]

B. Benchmarks

We compare our proposal GAL algorithm in terms of server cost and server response with three MAB variations (ϵ-greedy, UCB1, and original TOW) and LCP-WAC [11]. The ϵ-greedy, UCB1, and original TOW are the benchmarks for the server selection method, which are explained in Section II. For a fair evaluation, these benchmarks also use a Dijkstra algorithm for finding the shortest path between a host-server pair with the link weights predicted by an LSTM model.

The LCP-WAC benchmark (presented in our previous study [11]) utilizes the same LSTM-based Dijkstra algorithm; however, it identifies each possible shortest path for each possible server and chooses the server with the lowest server cost instead of choosing a server according to traditional MAB methods mentioned in Section II. Indeed, this LCP-WAC benchmark takes up a high computational cost.

C. Performance Analysis

In this experiment, we test the length of a sliding window T_r considered on different instances: 1200, 6000, 12000, 18000, and 24000 with 10, 50, 100, 150, and 200 requests per second respectively. Also, the threshold convergence values in server cost and response time are fixed at 0.02 and 0.005 respectively to determine system dwell time.

As illustrated in Figure 3, the proposed GAL algorithm considers real-time statistics about the past experience and the current experience simultaneously to optimize the procedure of allocating users' requests, where the forgetting ratio γ is 0.1, 0.5, 0.85, and 1.0. GAL (γ = 0.85) has better performance in the server cost and the server response time, achieving approximately 0.15ms and 55% respectively at 200 requests per second. Conversely, when γ is 0.1 or 1.0, it means that because the value of the reward function can either completely forget or consider past experiences, thus the proposed GAL algorithm is hardly adaptable to dynamic environments.

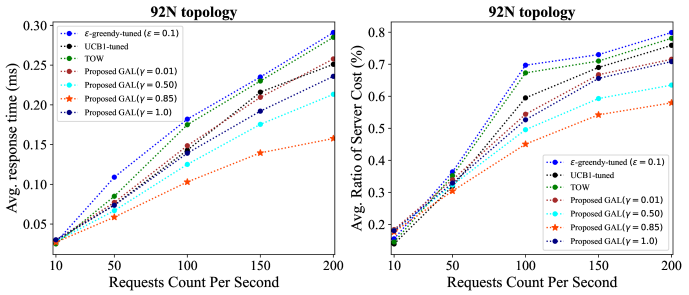

Fig. 3: Number of requests vs. server performance, where there are 92 nodes and 18 servers.

Figure 4 illustrates the relation between the number of servers and the RT and SC respectively. This simulation has 10 replica servers and 10 requests per second. The forgetting ratio γ is set to 0.85. The benchmark LCP-WAC's response time significantly increased to over 1.75 ms. This can be explained that as the number of servers increases, the benchmark PLC-WAC algorithm has to calculate all the shortest paths to every possible server before choosing the server with a minimum server cost, thus requiring more computational cost and processing time. However, the response time of MAB algorithms (ϵ-greedy, UCB1, TOW, and the proposed GAL) only changes slightly, since their server selection method is simply about choosing the server with the highest expected reward (as shown in Section II, and III). In contrast, when the number of servers increase from 18 to 50 replica servers, the LCP-WAC benchmark witnessed a remarkable decrease in server cost value, plummeting from around 20% to a mere 2%. Besides, the proposed GAL algorithm has better performance in server cost with the second-lowest ranking, regardless of the number of servers. The reason is that the proposed forgetting parameter can reduce the impact of past experiences, thus making the method more adaptable in a stochastic environment.

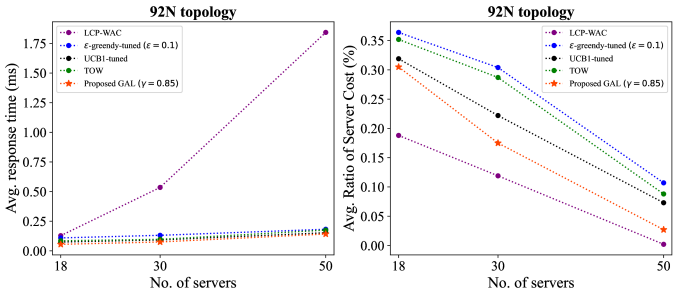

Fig. 4: Number of servers vs. server performance, where there are 92 nodes and 10 requests per second.

V. Conclusion

This paper has introduced GAL algorithm as a new approach to tackle the server and route selection problem utilizing the Gambling method and LSTM neural networks. The algorithm aims to minimize server cost and server response time over an extended time while considering the dynamic future of network states. GAL comprises two essential phases: server selection and routing optimization. The server selection phase is formulated as a gambling problem and addressed using a modified tug-of-war (TOW) algorithm inspired by the game of tug-of-war. This adaptation incorporates a new reward function that combines server cost and response time and a new approach to estimating the quality of servers using the $Q(.)$ function. The routing optimization phase leverages a Dijkstra algorithm with link weights predicted by an LSTM network. The integration of LSTM-based routing mechanisms enables GAL to effectively account for future network states and make informed routing decisions. To test the efficiency of the proposed GAL algorithm, we conducted extensive tests within a distributed multi-domain software-defined networking environment. The results demonstrate that GAL outperforms other state-of-the-art benchmarks, exhibiting superior average server response time and server cost. GAL can contribute to advancing the SARS optimization study. The hybrid approach of combining gambling-inspired server selection with LSTM-based routing enhances the effectiveness in addressing the long-term objectives while adapting to dynamic network conditions. These findings highlight the potential of GAL for improving resource allocation in complex network environments for more efficient and cost-effective network management strategies.

References

[1] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[2] F. Fitzek, F. Granelli, and P. Seeling, *Computing in Communication Networks: From Theory to Practice*. Academic Press, 2020.

[3] H.-A. Tran, S. Souihi, D. Tran, and A. Mellouk, "Mabrese: A new server selection method for smart sdn-based cdn architecture," *IEEE Communications Letters*, vol. 23, no. 6, pp. 1012–1015, 2019.

[4] 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)*. IEEE, 2015, pp. 2136–2139.

[5] S.-J. Kim, M. Aono, and E. Nameda, "Efficient decision-making by volume-conserving physical object," *New Journal of Physics*, vol. 17, no. 8, p. 083023, 2015.

[6] N. H. Nguyen, P. Le Nguyen, H. Dinh, T. H. Nguyen, and K. Nguyen, "Multi-agent multi-armed bandit learning for offloading delay minimization in v2x networks," in *2021 IEEE 19th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, 2021, pp. 47–55.

[7] J. Ma, T. Nagatsuma, S.-J. Kim, and M. Hasegawa, "A machine-learning-based channel assignment algorithm for iot," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2019, pp. 1–6.

[8] H. Zhong, Q. Lin, J. Cui, R. Shi, and L. Liu, "An efficient sdn load balancing scheme based on variance analysis for massive mobile users," *Mobile Information Systems*, vol. 2015, 2015.

[9] P. Sossalla, "Investigation of reinforcement learning strategies for routing in software-defined networks," 2019. [Online]. Available: https://cn.ifn.et.tu-dresden.de/wp-content/uploads/2022/11/DA_Sossalla_Peter.pdf

[10] L. Amour, V. Tong, S. Souihi, H. A. Tran, and A. Mellouk, "Quality estimation framework for encrypted traffic (q2et)," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[11] N.-T. Hoang, V. Tong, H. A. Tran, C. S. Duong, and T. L. T. Nguyen, "Lstm-based server and route selection in distributed and heterogeneous sdn network," *Journal of Computer Science and Cybernetics*, vol. 39, no. 1, p. 79–99, Mar. 2023. [Online]. Available: https://vjs.ac.vn/index.php/jcc/article/view/17591

[12] 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.