2023 International Conference on Computer Communication and Informatics (ICCCI) | 979-8-3503-4821-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCI56745.2023.10128241

Maximising Efficiency of Network Resources in SDN through Optimal Packet Routing using Reinforcement Learning

Anush K Byakodi, Mohammad Zakee Hungund, Aman Khan N Athani, Khadar Wali B Afghan, Somashekar Patil, Narayan D. G.
School of Computer Science and Engineering
KLE Technological University
Hubballi, India

anushkbyakodi@gmail.com, zakeehungund54@gmail.com, amank2061@gmail.com, khaderaliafghan@gmail.com
Skpatil@kletech.ac.in, narayan_dg@bvb.edu

***Abstract*—Software-defined networking (SDN) has increased in popularity in recent years owing to its ability to centralize network device control. This is the cornerstone of data center networks, and as more IoT-enabled devices are added, it is expected to skyrocket. The employment of Machine Intelligence to handle and route the majority of traffic can enhance centralized SDN-based control, allowing for high and occasionally subpar network management. Owing to the development of Reinforcement Learning (RL) combined with Deep Learning, which allows autonomous flow management in SDN, intelligent routing has gained popularity. However, the existing SDN routing algorithm has poor link usage and is unable to update and change according to real-time network conditions, making it challenging to forecast and predict how communication networks will evolve in the future. Consequently, we present a routing optimization strategy based on Quality of Service (QoS) variables for SDN-based Data Center Networks using Deep Reinforcement Learning. We provide a reinforcement learning-based routing method for SDN that outperforms Dijkstra's shortest path in terms of the throughput and average latency. In the proposed method, we integrated SDN with RL, which takes input from the state in which the current network is and routes traffic to achieve the maximum QoS parameters. The parameters considered were the throughput and delay. The presented solution is a network-oriented, self-learning routing system named DeepQoSR, based on Advantage Actor-Critic Networks. We compared our proposed technique with traditional algorithms, such as the shortest path, and present our analysis.**

***Index Terms*—Software Defined Networks, Data Center Networks, Deep Reinforcement Learning, Machine Learning, Quality of Service (QoS), Advantage Actor-Critic.**

I. Introduction

Software-defined networks (SDN) [1] centralize network management by redefining networking fundamentals by separating critical operations such as network programming and control from traffic routing. In the OSI model, switches are found at lower levels and several central controllers share control over them. This creates a wide range of novel network programming paradigms, including the optimization of traditional methods and autonomous routing algorithms. One area that requires further research is machine intelligence.

Furthermore, because the controllers are centralised devices, they can include more processing power than a distributed strategy that controls network traffic with fixed and dedicated hardware devices such as routers and switches. This classic network is built on hardware network appliances, and is static.

Deep Learning is a subset of Machine Learning (ML) that excels in black-box optimization and, as a result, can outperform humans in tasks such as categorization and recognition. They can be employed to forecast and fit data or assist people in providing insights into the data.

Our work focuses on Reinforcement Learning, which is a technique that can be used in a variety of SDN situations. RL employs the principles of optimal control, dynamic programming, and value functions in the agent training process. Various SDN strategies for addressing SDN challenges were described in [2].

Data centers are a key component of modern internet services, providing the infrastructure that enables internet development. However, traditional network management solutions may struggle to scale effectively, leading to decreased efficiency. In contrast, centrally monitored networks, such as those using software-defined networking (SDN), can provide comprehensive control over all devices connected to a central controller. In addition to enabling programmability, SDN allows for a complete view of the network. Quality of service (QoS) parameters can be used to create self-healing, self-learning data center networks, which can also serve as learnable characteristics for machine learning applications. We used Mininet [6], a tool that provides easy access to control nodes and hosts within the network, and employed reinforcement learning to automate network management.

The main contributions of this study are as follows.

- We propose a novel solution for routing traffic in software-defined networks using Advantage Actor-Critic Networks.
- We compared the proposed solution with traditional routing algorithms based on packet loss and latency.
- In terms of throughput obtained and minimal time between the source and destination, our proposed method outperforms the existing technology.

The following is how the paper is set up: Section II discusses the various works carried out in the context of Reinforcement Learning based routing in SDNs with a primary focus on Deep Reinforcement Learning, as well as the study on SDN-based techniques and the relevance of Reinforcement Learning in network management. Section III describes the methodology and the proposed system design. The findings of the suggested research are presented in Section IV along with a comparison with the Shortest Path(SP) model. In Section V, the study results and prospective research are discussed.

II. Related Work

In this section, we discuss the SDN-based strategy and the importance of reinforcement learning in network management.

This study proposed a deep reinforcement learning-based approach for routing (DRL-R) [3]. To decrease latency, they described a technique for recombining numerous network resources with different metrics such as bandwidth and cache. Additionally, they offer a resource-combined state routing system. By allocating network resources for traffic as effectively as feasible, a DRL agent operating on an SDN controller communicates with the network in real time to conduct acceptable routing adaptively based on the network status.

Another deep reinforcement-learning-based routing optimization technique was proposed by [4]. By improving network performance (delay and throughput), the algorithm may reduce network maintenance and operating costs, as well as perform continuous black-box optimization.

We will use concepts taken from [5] where an actor–critic network is used, which provides a Quality of Service (QoS)-based routing optimization approach using Deep Reinforcement Learning for SDN-based Data Center Networks (DeepQoSR). In general, actor-critic algorithms are a type of RL algorithm that combine the ideas of the actor-based approach (where the agent learns a policy that maps states to actions) and the critic-based approach (where the agent learns a value function that estimates the expected reward for taking a particular action in a particular state). These algorithms are often used when the environment is complex or continuous, or when it is not possible to define a clear reward function. Actor-critic algorithms can be more efficient and easier to implement than some other types of RL algorithms, authors of the research paper chose to use the actor-critic variant of RL because they wanted to take advantage of its efficiency and flexibility. The authors constructed an efficient and robust matrix representing a given topology and its current state to achieve the same objective.

To construct intelligent QoS-aware routing systems, we used a more efficient deep reinforcement-learning technique. Advantage actor-critic is a variant of the actor-critic algorithm that is designed to address some of the limitations of the basic actor-critic approach. Like the basic actor-critic algorithm, A2C involves learning both a policy and a value function, but it uses a modified update rule for the value function that takes into account the advantage of taking a particular action in a particular state.

The advantage of an action is defined as the difference between the expected return from taking the action and the expected return from the best action that could be taken in the same state. By incorporating the advantage of each action into the value function update rule, A2C algorithms can learn more efficiently and converge to better solutions faster than some other types of RL algorithms. We compared the performance of our suggested strategy with that of existing, similar routing strategies to assess its effectiveness.

III. System Design

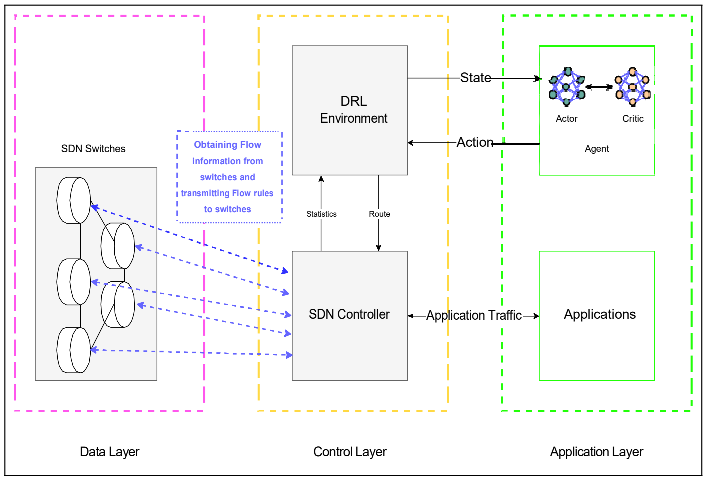

Fig. 1. High Level Design

A. Proposed System

The proposed system comprises three modules. Deep-RL (DRL) Environment, Deep-RL Agent, and Mininet Backend.

Figure 1 shows a high-level overview of the entire system with few specifics and is used to comprehend the system as a whole. It explains the project's many components and how they interact with one another.

The Deep-RL Environment, Deep-RL Agent, and the Computer Network created using Mininet are the three functionally independent parts that constitute our proposed system. These three are modular; hence, any Reinforcement Learning agent/algorithm can perform routing in any topology within its environment. The environment plays a crucial role in coordinating the interactions between the DRL agent and the SDN while providing room for customizability.

The System Model in Figure 1 depicts the entire proposed system. The information collected from the switches is transferred to the SDN Controller in the Control Layer, which also shares Application Traffic information from the Application Layer. The Learning Agent makes decisions and informs the SDN Controller of the upcoming flows to be added to virtual switches. The SDN Controller then transmits the data from the Data Layer to the DRL Environment. The Application Layer is divided into two parts to isolate DeepQoSR's decision-making modules from other applications.

*1) **Environment**:* The Environment is the reactive component of a Reinforcement Learning system that changes its state in response to an agent's activities. A network of datacenters maintains the entire system, which is primarily how the DeepQoSR system perceives the environment. Mathematical representations of the environmental status and action spaces are presented below.

State Space: The values that the agent experiences are based on the state space, which is sometimes referred to as the observation space and is utilized as the input of the neural network. In the case of DeepQoSR, the state space is described as a vector containing every connection statistic for a node. This vector can capture the intricacies communicated to our agent through congestion details and information. The Advantage Actor-Critic agent's decision-making abilities are driven by congestion information, which is represented by a variety of criteria. To construct the state vector, we employed the available bandwidth, packet loss, and latency across each link as QoS parameters. To represent the data layer connected to the controller, a graph G(V,E) is used, where E determines all edges linking the switches, and V is the set of switches.

$$E = \{l_1, l_2, \ldots, l_n\} \tag{1}$$

$$l_i = \{\text{packet loss, available bandwidth, latency}\}$$

We normalized the values of all QoS parameters to acquire consistent features across all the links. Following is the definition of the normalisation function:

$$l_i^{normalised} = \frac{\vec{l_i} - \min(\vec{l_i})}{\max(\vec{l_i}) - \min(\vec{l_i})} \tag{2}$$

Finally, the normalized values are summed to generate the S (State Space) vector.

$$S = \{l_1^{normalised}, l_2^{normalised}, \ldots, l_n^{normalised}\} \tag{3}$$

Action Space: The various activities that the neural agent can perform to identify the best path to the destination are outlined. The following is a definition of Action Space:

$$A = \{a_1, a_2, a_3, a_4 \ldots, a_n\} \quad \text{for n links} \tag{4}$$

A subset of A is a valid step to execute for each node V and is therefore mapped to the corresponding links, whereas the remainder are invalid. Each time a subpar link is selected, the agent must be punished.

Reward Function: The agents were trained using a trial-and-error method using reinforcement learning, which is a subset of machine learning. The objective of the agent is to increase the overall rewards or end rewards. Agents must develop decision-making skills in diverse states. It is important to understand that sometimes giving up immediate benefits is necessary to maximize the long-term benefits. The reward function is the stimulus that serves as a benchmark to ensure that the agent learns what is correct or incorrect. The agent can achieve the optimal policy feasible with the help of this reinforcement-based feedback. A learning episode's reward is always related to the agent's most recent activity.

The reward function principles that we introduced are based on the packet arriving at its destination. The total throughput, or bandwidth, is the amount of time it takes for a package to get from its point of origin to its final destination (delay).

To fully utilize the reward function and ensure that it is punished severely to take wrong actions and rewarded moderately to take the right action, an exponential-based reward function was introduced. The exponential part of the reward function ensures that, if the delay is larger than the accepted standards, a larger negative reward is given; otherwise, a moderate reward is given to achieve a good amount of delay. We utilize an Advantage Actor-Critical network that responds promptly to any feedback received. The reward function consists of two components. Our reward function consists of a routing outcome-based reward, R. Some QoS criteria are prioritized over others by user-configurable weights ϕ1 and ϕ2.

$$\text{Reward} = (\text{Bandwidth} - \phi 1 * \text{MaxBandWidth}) - e^{(Delay - \phi 2 \ * MaxDelay)} \tag{5}$$

*2) **Agent**:* The agent determines the optimal route for a certain location. We used Advantage Actor-Critical Networks to obtain the optimal policy and routing depending on the activity sequence of the agents. To create an optimal policy that is more efficient and calls for fewer learning steps, the Advantage Actor-Critic blends Q-learning and reinforcement learning. The Agent was a combination of two separate neural networks. The Actor is also known as the policy network, because it links the observation space to the action space. The Critic network calculates a Q value to determine the valuable/good of the actor's selected action. If we also intuitively decide how much better it is to perform a specific action under a certain condition than to take the average, generic action. This value is referred to as the advantage value. In each learning step, the actor parameters with policy gradients and the advantage value, as well as the critic parameter, were changed.

$$\nabla_\vartheta \mathbf{J}(\theta) \sim \sum_{t=0}^{T-1} \nabla_\vartheta \log \pi_\vartheta(a_t|s_t) A(s_t, a_t) \tag{6}$$

The Actor network's update equation is shown in 6, where θ represents the approximation function of the actor neural network, which is trained with a learning rate alpha with the goal of optimizing the policy ppi with the aid of a critique value estimated by the critic network [5].

*3) **Back-end**:* This module consists of a network environment and controller used for data transmission. The controller transforms the agent's route into a set of flows or rules to be applied to relevant virtual switches. For each connection and switch in the network, the controller provides environment updates. QoS settings were used to send updates.

Finally, the newly connected path is used for routing. The RL module is self-contained and only communicates with the environment through the controller and back-end. We succeeded in achieving modularity, which enables us to alter the routing strategy without altering the testbed at the core. Consequently, the Shortest Path algorithms can be executed quickly and separately.

B. Algorithms

By loosely coupling SDN and Machine Learning, the following section describes the crux of network management. Although the components function independently as black boxes, they work together to finally route the traffic optimally.

*1) **QoS Feature Vector Retrieval**:* The algorithm used in [5] demonstrated the necessary pseudocode for obtaining the observation state. The Link Map and a Switch Datapath id containing the details of connections connecting to the specified node/switch S are read. The packet loss values are balanced between 0 and 1, where 1 represents the greatest packet loss, indicating no connectivity, and 0 represents the least loss. The range of the available bandwidth is 0 to 1, with 0 and 1 denoting no connectivity and maximum data throughput, respectively. These numbers are connected and sent back as the node characteristics.

*2) **Routing of Traffic**:* After the agent calculates the path, the controller and the back-end module run the routing algorithm from [5] to create flows that direct the traffic. The flows change dynamically in response to an agent's actions. The state of the world network is reflected in the results of this algorithm. The agent learns to perform better with each iteration of training time on the network and with each iteration of the training period. This provides an improved routing option.No action was performed by the agent during the routing process.The agent then waits for the final result. The update rule is applied to the agent once again with the freshly acquired reward after the data have been successfully transferred. This signals the end of an agent's job and the conclusion of an episode.

*3) **Agent Training**:* The entire training procedure of the DeepQoSR system is described by Algorithm 1. It invokes other algorithms to obtain the QoS parameters, which are: estimate action, execute action, and route traffic. The bulk of the programmable parameters are controlled by this algorithm, which was created with Advantage Actor-Critic networks in mind. In the same environment, training and action estimation occur; in the background, the controller performs routing. This is similar to what occurs in real-world networks. Once the routing protocol is defined, the network architecture can work independently. In real time, the same method can be adjusted to accommodate customized data in this project's application. Once the agent is trained, it ignores the weight updates and transmissions. A forward pass through the network is sufficient only at this point to obtain the action.

Algorithm 1: Agent Training

```
1 Begin
2 Initialize Network Devices
3 Initialize Environment
4 Get the number of actions in action space
5 Initialize Advanatge Actor Critic Agent
6 Initialize reward_history
7 Set x _episodes
8 for n in range(x_episodes) do
9     Environment Reset
10    score ← 0
11    while episode_not_over do
12        obs ← state space
13        Calculate values and policy_dist from
            adv_actor_critic
14        Get action using policy_dist
15        Execute action in Environment
16        Get new_state, reward, episode_over
17        Agent.learn(obs,not episode_not_over,new_obs)
            score ← reward + state
18        if destination reached or max_steps reached
            then
19            Calculate Q-val from adv_actor_critic
20            Route traffic
21            Measure achieved bandwidth
22            break
23        else
24            Continue
25    Calculate Q-values
26    Calculate Advanatage values and losses
27    Optimize adv_actor_critic
28    Append score to reward_history
29 Turn off Network Devices
```

IV. Results

A. Experimental setup

We worked with Python (3.7), in which in order to set up the whole backend network, we made use of Mininet (2.3.0) [6] emulator. The flows in this environment were handled by switches in Mininet. The Ryu Controller [7] was used to keep track of the switches and communicate among them. The most common uses of the Ryu Controller are to obtain QoS settings and configure the flow-table entries. An open-source implementation of a distributed virtual multilayer switch is available under the name Open VSwitch [8]. The primary function of Open vSwitch is to offer a switching stack for use in hardware virtualization settings. In addition, it offers support for a wide variety of protocols and standards used by computer networks. The Gym framework from OpenAI (0.17.3) [9] allows for the creation of bespoke environments, specification of reward functions, and streamlining of agent training processes. It has definitions for state and action spaces, and is easy to connect with the backend. The object-oriented methodology was used by the well-known deep learning framework PyTorch (1.71) [10]. The Actor and Critic networks in PyTorch were designed with adjustable parameters to provide a high degree of customization. All of this was performed on an Ubuntu OS (20.04).

With these mentioned frameworks, we build the routing system - DeepQoSR.

We consider the topology shown in Figure 2 with a distinct number of hosts and switches. The topology had a maximum link bandwidth of 10 Mbps. There are a total of 32 hosts connected. Our design can assist data center networks in lowering their network latency as well as the number of hops, which ultimately improves network efficiency.

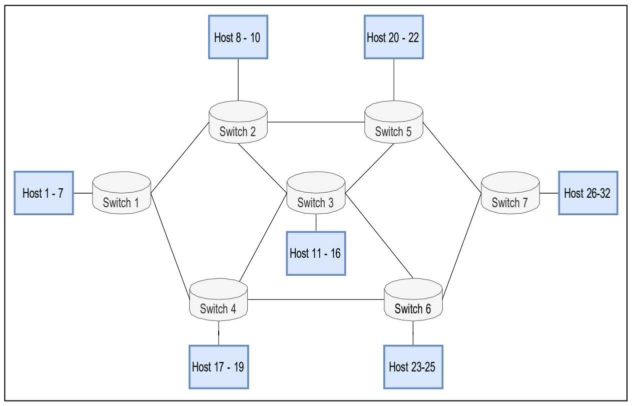

Fig. 2. Topology with Seven Switches

We trained the networks with a learning rate of α = 0.0003 for the actor network and β = 0.0003 for the critic network. The critic cannot have a lower learning rate because if the actor changes faster than the critic, then the estimated Q-value will not truly represent the value of the action, because that value is now based on past policies. Across both test beds, we used a discount factor of γ = 0.99. For the packet available bandwidth and latency, the reward weights of ϕ1 = 0.5 and ϕ2 = 0.4 are used. These parameters are obtained by testing and are placed into equation 5. Finally, Iperf [11] was used to perform routing and measure the bandwidth, and the incentive was assigned accordingly. Over 1000 episodes are used to train the neural network pair.

B. Results analysis

The analysis and inferences drawn from the data are discussed in this section. Two essential factors are used to evaluate the performance of the system: the achieved bandwidth and the average delay. In **??** we used Dijkstra's algorithm because we wanted to compare the performance of our proposed approach to a widely used and well-established routing protocol, and we wanted to evaluate the approach in terms of the improvement it provides over a simple baseline. We chose to use the shortest-path algorithm as a baseline because these algorithms are relatively simple to implement and can be run efficiently, allowing us to focus on evaluating the performance of their proposed approach rather than the underlying routing algorithm.

*1) **Learning Curves**:* Figure 3 shows the learning curves with the average rewards over the epochs. The learning graph was uniform, indicating that the model learned at a good rate. Nearing 1000 epochs, the average rewards become positive, indicating that the model performs well and achieves good bandwidth and less delay.

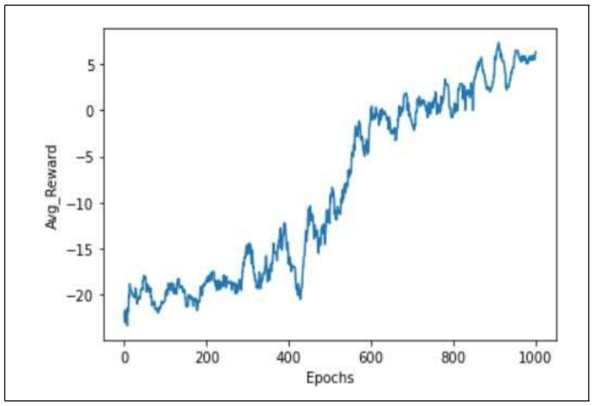

Fig. 3. Learning Curve Topology

*2) **Rewards vs Epochs**:* Shown in Figure 4. Negative rewards suggest that the package was not successfully delivered to the target destination. As the epoch count increases, the proportion of negative incentives decreases, and as the epoch count approaches 1000, almost all rewards are positive. This suggests that the packet was successfully transferred to the destination. It also demonstrates that we can achieve a good bandwidth and delay.

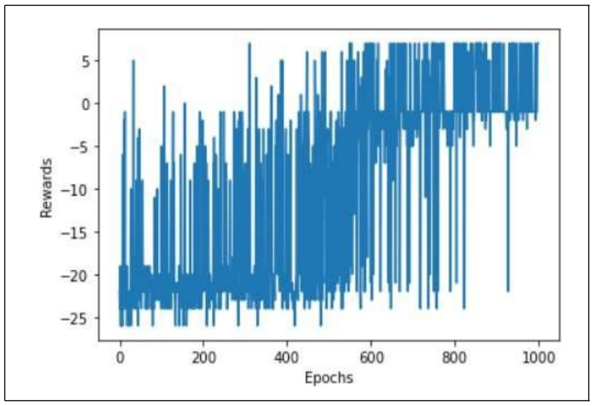

Fig. 4. Rewards vs Epochs

*3) **Bandwidth vs Epoch**:* Figure 5 representing Bandwidth against Epoch seems similar to the graphs of Rewards against Epoch because the reward function is dependent on Bandwidth and Delay. In the given graph, if the bandwidth reaches zero, it indicates that the packet cannot reach the destination. As the number of epochs reached 1000, there were hardly any bandwidth values reaching zero, indicating that we were able to reach the destination consistently. We achieved an average bandwidth of 4.74 Mbps.

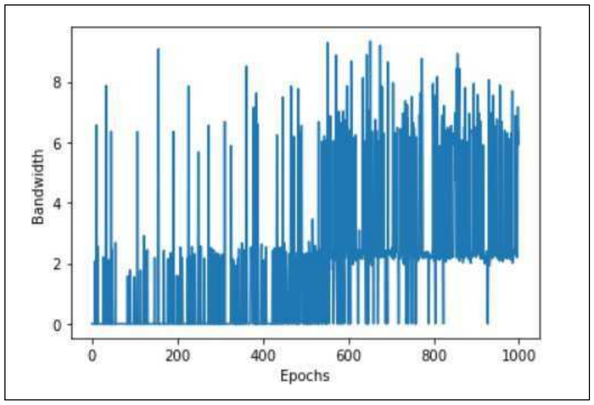

Fig. 5. Bandwidth vs Epochs

*4) **Comparing Results**:* Figure 6 and 7 compare the available techniques with the Advantage-Actor-Critic with respect to the delay and bandwidth, respectively. For comparison, we employed the Shortest Path algorithm as Dijkstra's algorithm based on two parameters. SP-Loss, which chooses the link with the least packet loss, and SP-Latency, which chooses the link with the lowest rate of latency. As shown in Figure 7 Advantage-Actor-Critic model was able to produce a higher bandwidth than the shortest-path algorithms based on both Loss and Latency. In Figure 6, the Advantage-Actor-Critic model was able to produce less latency than the shortest-path algorithms based on latency as well as loss. Overall, because our DeepQoSR algorithm is based on artificial neural networks, it can dynamically route traffic with the primary goal of increasing throughput while decreasing latency.

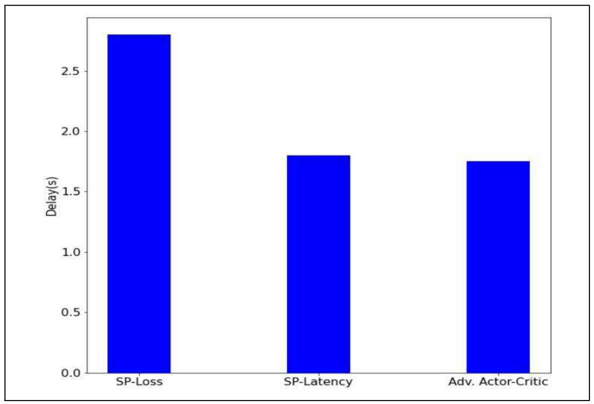

Fig. 6. Delay Comparison

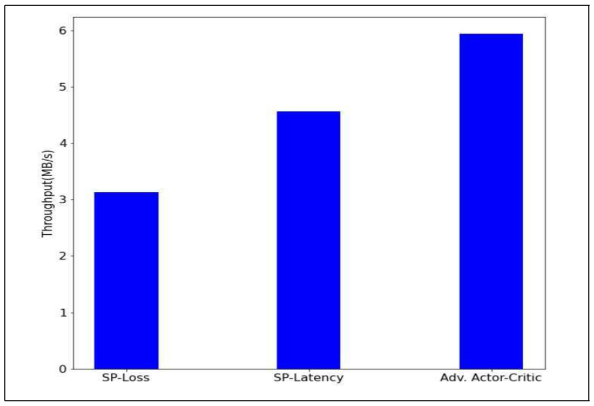

Fig. 7. Bandwidth Comparison

V. Conclusion

This method is based on the concept of reinforcement learning and uses Advantage-Actor-Critic networks. The currently employed method, as well as a variety of other routing techniques, have been subjected to in-depth analysis and comparison. Advantage-Actor-Critic outperformed the shortest path algorithms based on both latency and packet loss because it was able to obtain greater bandwidth while experiencing less delay. DeepQoSR, in the main, guarantees consistency across the topology, together with the least amount of latency and the highest possible throughput. These two technologies hold hope for an improved Internet and better global connection in the not-too-distant future.

References

[1] K. Benzekki, A. El Fergougui and A. Elbelrhiti Elalaoui, "Softwaredefined networking (SDN): A survey", Security Commun. Netw., vol. 9, no. 18, pp. 5803-5833, Dec. 2016.

[2] J. Xie et al., "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges," in IEEE Communications Surveys Tutorials, vol. 21, no. 1, pp. 393-430, Firstquarter 2019, doi: 10.1109/COMST.2018.2866942.

[3] Wai-xi Liu, Jun Cai, Qing Chun Chen, Yu Wang, DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks, Journal of Network and Computer Applications, Volume 177, 2021, 102865, ISSN 1084-8045

[4] Chunlei Xu, Weijin Zhuang, and Hong Zhang. 2020. A Deep-reinforcement Learning Approach for SDN Routing Optimization. In *Proceedings of the 4th International Conference on Computer Science and Application Engineering* (*CSAE 2020*). Association for Computing Machinery, New York, NY, USA, Article 26, 1–5. DOI:https://doi.org/10.1145/3424978.3425004

[5] A. A. Magadum, A. Ranjan and D. G. Narayan, "DeepQoSR: A Deep Reinforcement Learning based QoS-Aware Routing for Software Defined Data Center Networks," 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), 2021, pp. 1-7, doi: 10.1109/ICCCNT51525.2021.9579514.

[6] "Mininet". https://mininet.org/. Last Accessed November 2021

[7] "Ryu". http://ryu.readthedocs.io/en/latest/index.html/ Last Accessed November 2021

[8] "OpenvSwitch". https://www.openvswitch.org/ Last Accessed November 2021

[9] "OpenAI Gym". http://gym.openai.com/. Last Accessed November 2021

[10] "PyTorch". https://pytorch.org/. Last Accessed November 2021

[11] "iperf". https://iperf.fr/ Last Accessed November 2021

[12] R, N., & S.V, E. S. (2021). An Efficient Mining Approach for Handling Web Access Sequences. International Journal of Computer Communication and Informatics, 3(1), 15-25.