# Flow-Level Dynamic Bandwidth Allocation in SDN-Enabled Edge Cloud using Heuristic Reinforcement Learning

| | |
|---|---|
| 著者 | Qadeer Arslan, Lee Myung J., Tsukamoto Kazuya |
| journal or publication title | 2021 8th International Conference on Future Internet of Things and Cloud (FiCloud) |
| page range | 1-10 |
| year | 2021-11-15 |
| URL | http://hdl.handle.net/10228/00008663 |

doi: https://doi.org/10.1109/FiCloud49777.2021.00009

# Flow-level Dynamic Bandwidth Allocation in SDN-enabled Edge Cloud using Heuristic Reinforcement Learning

Arslan Qadeer
*Department of Electrical Engineering*
*The City College of New York of CUNY*
*New York, USA*
*aqadeer000@citymail.cuny.edu*

Myung J. Lee
*Department of Electrical Engineering*
*The City College of New York of CUNY*
*New York, USA*
*mlee@ccny.cuny.edu*

Kazuya Tsukamoto
*Department of ECE*
*Kyutech*
*Japan*
*sukamoto@cse.kyutech.ac.jp*

***Abstract*—Edge Cloud (EC) is poised to brace massive machine type communication (mMTC) for 5G and IoT by providing compute and network resources at the edge. Yet, the EC being regionally domestic with a smaller scale, faces the challenges of bandwidth and computational throughput. Resource management techniques are considered necessary to achieve efficient resource allocation objectives. Software Defined Network (SDN) enabled EC architecture is emerging as a potential solution that enables dynamic bandwidth allocation and task scheduling for latency sensitive and diverse mobile applications in the EC environment. This study proposes a novel Heuristic Reinforcement Learning (HRL) based flow-level dynamic bandwidth allocation framework and validates it through end-to-end implementation using OpenFlow meter feature. OpenFlow meter provides granular control and allows demand-based flow management to meet the diverse QoS requirements germane to IoT traffics. The proposed framework is then evaluated by emulating an EC scenario based on real NSF COSMOS testbed topology at The City College of New York. A specific heuristic reinforcement learning with linear-annealing technique and a pruning principle are proposed and compared with the baseline approach. Our proposed strategy performs consistently in both Mininet and hardware OpenFlow switches based environments. The performance evaluation considers key metrics associated with real-time applications: throughput, end-to-end delay, packet loss rate, and overall system cost for bandwidth allocation. Furthermore, our proposed linear annealing method achieves faster convergence rate and better reward in terms of system cost, and the proposed pruning principle remarkably reduces control traffic in the network.**

***Keywords*-Edge Cloud, Software Defined Networking, Reinforcement Learning, Bandwidth Allocation, Resource Management, OpenFlow**

## I. Introduction

Next generation mobile (e.g. Augmented Reality (AR), Virtual Reality (VR)) and smart-city applications hold intensive resource hungry and real-time constraints [1]. Edge-cloud (EC) architecture is a stepping stone to meet the above real-time constraints by reducing the network latency and providing the compute, network and storage close to the user [2]. Edge clouds generally endure a limited amount of computational and network resources to target the local users [3]. A large amount of concurrent traffic can be anticipated and the infrastructure which connects the devices with EC become a bottleneck for the system. Thus, resource management for the user traffic from a wide range of applications at large scale with different QoS requirements is a challenge.

Resource management in a network is attainable with a number of different approaches [4], [6], [7]. These include multipath optimal switching [4], dynamic network orchestration [6] and cluster-based mechanisms to effectively allocate network resources [7]. The centralized Software Defined Networking (SDN) architecture can also be leveraged to control the resource usage when multiple heterogeneous users or applications strive for the resources in an EC based shared network. SDN enables programmability feature of forwarding devices to enhance the network performance via efficient resource provisioning and management [9]. Further, network resource management can be optimized through dynamic bandwidth allocation of the traffic flows. Bandwidth allocation problem to be addressed in this study implies allocating resources at the flow level. Resource allocation based on individual flows allows a fine-grained control for the traffic from any application and improve the overall network performance.

Many existing works have proposed different resource management techniques in SDN [4]–[14]. [5], [8], [11], [12], [14] adopt machine learning algorithms for bandwidth allocation, a game theoretic approach is applied for resource allocation in softwarized networks in [7], and [13] proposes a context-aware method to improve the content delivery QoS. [9] considers a special hybrid SDN scenario where a flow-level strategy with multi-path allocation is proposed to improve the network utility. An adaptive QoS algorithm is proposed in [10] which is based on Differentiated Services (DiffServ) and provides a fine-tuned control over per-flow bandwidth allocations. Aforementioned studies mainly use route optimization techniques to solve the problem of link congestion, inefficient bandwidth allocation and network delay. Demand-based dynamic bandwidth management for different kinds of traffic is not taken into account for better efficiency. Furthermore, considering the above mentioned parameters to emulate bandwidth allocation and test the feasibility in real EC based environment is still unexplored.

This study aims to develop a more general approach to bandwidth allocation and to validate the proposed technique via simulation of a realistic EC-based scenario. Thus, we propose a heuristic reinforcement learning based flow-level dynamic bandwidth allocation algorithm using OpenFlow meter [16]. The standard reinforcement learning is suitable for a small state space environment, however, to reduce training time for a large environment we add a heuristic function. An end-to-end bandwidth allocation framework has been deployed on real hardware OpenFlow switches [17] with the goal of minimizing overall system cost, packet loss rate, end-to-end delay, and improve network throughput with fast self-learning capability in a non-stationary environment. A simulation model is also setup using Mininet [18] to test scalability of the proposed model with respect to increase in the number of flows and switches. To the best of our knowledge, none of the existing works applied heuristic reinforcement learning to solve the bandwidth allocation problem using OpenFlow meter in SDN-enabled EC environment.

The remainder of this paper is organised as follows: System model for SDN-enabled EC system is highlighted in Section II. Section III describes our heuristic reinforcement leaning (HRL) based bandwidth allocation framework. Section IV presents the simulation scenarios and performance evaluation details, followed by the conclusion and future directions in Section V.

## II. System Model for SDN-enabled Edge Cloud System

### A. Testbed Description

The IRNC (International Research and education Network Connections) is a program by NSF (National Science Foundation) for *"high-performance network connectivity required by international science and engineering research and education collaborations"*. The NSF IRNC supported COSMIC (COSMOS Interconnecting Continents) testbed proposes to build a fully programmable network (from the optical/radio physical layer and above) and computational infrastructure (Edge Clouds). In the proposed COSMIC architecture, a data center near to the mobile users can serve as an EC and be used to offload service requests of real-time and latency sensitive applications. The data center in a remote LAB or other continents will be emulated as a core cloud, and the delay-tolerant services can be offloaded to the data center to achieve efficient resource sharing in a multi-domain network environment. As a first step in this paper, we consider COSMOS (Cloud Enhanced Open Software-Defined Mobile Wireless Testbed) [19] topology at The City College of New York (CCNY). COSMOS hosts an edge node at CCNY and further connects Columbia University in New York City at one end and Kyutech University in Japan at other end as shown in Fig. 1. COSMOS CCNY has three 5G

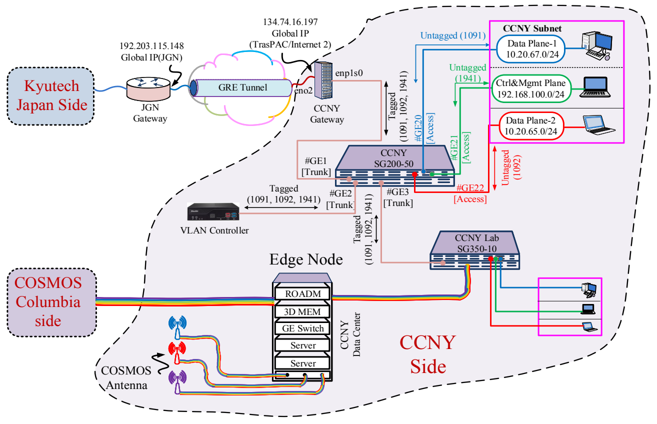

Figure 1: COSMOS CCNY Edge Cloud testbed Model. The detail about System Model is given in Section II-A.

antennas (Software Defined Radios (SDR)) and connects wireless (mobiles, IoTs, Street Cameras etc.) users to the EC at CCNY data center. There are other network nodes[1] as well which connect hosts from different research labs of CCNY to the EC. This testbed is instrumental in the research and development of low-latency infrastructure and efficient resource provisioning algorithms for next-generation mobile and IoT applications. The rest of this paper describes the end-to-end implementation of flow-level dynamic bandwidth allocation framework and validates it in the above stated real-world COSMOS testbed scenario.

### B. Traffic Model

In SDN, flows (forwarding rules) are generally installed based on MAC/IP addresses of the hosts. When a packet arrives on a switch, data-plane matches the existing flows, if a match is found then the packet is forwarded via an appropriate port. If a match is not found, the data-plane forwards the packet to the control-plane which installs a new flow based on the topology information and destination MAC/IP address in the packet. This conventional way of matching the rules cannot differentiate the various kinds of traffic coming from a same user or host. For example, a host can generate many kinds of traffic (e.g. video, voice, email, file sharing etc.) which can be categorised as time-sensitive or best-effort [20]. Installing a distinct flow for each type of traffic and allocating bandwidth based on the QoS requirement of the flow provides a granular control in management of the resources. Thus, we install flows and classify traffic based on the L4 (TCP/UDP) port numbers and assign weight $\omega$ to each flow depending upon the QoS requirement of that traffic. These weights can be assigned dynamically to the flows which is detailed in the following section.

[1]Currently, these nodes are legacy network switches, however, soon to be replaced with OpenFlow enabled advanced switches to support SDN experiments.

### C. Bandwidth Model

We formulate a general bandwidth allocation model that can be utilized in any SDN-enabled EC based system. As shown in Fig. 1, an EC system possesses $N$ number of network nodes (i.e. Switches) $\{1, 2, 3, ..., N\}$, $P$ number of ports $\{1, 2, 3, ..., P\}$ per node $n$ and $F$ number of flows $\{1, 2, 3, ..., F\}$ per port $p$. Each flow $f$ has a weight $\omega_f$ which signifies the quality preference as described in Section II-B. $B_p$ is the total bandwidth/capacity available on a port $p$ (e.g. $100 Mbps$). In order to allocate bandwidth to the flows we make use of OpenFlow meter feature. OpenFlow meters are associated with the flows and control the packet/data rate, thus, called $RateLimiter$ [16]. Data rate is assigned to the meter bands in the form of discrete units. The minimum allocated bandwidth is 1 unit ($1 Mbps$) and maximum is 10 units. Multiple flows with similar category (e.g. video) can be grouped and associated with one meter. However, for the sake of simplicity and easy management we create $M$ meters $\{1, 2, 3, ..., M\}$ equal to the number of Flows and associate each flow to a dedicated meter. We evaluate the bandwidth utility of the switch per port basis. The total usage of the port capacity at time $t$ is the sum of the occupied resources by all the flows and given as:

$$U_p(t) = \frac{\sum_{f=1}^{F} b_f^p}{B_p}, \tag{1}$$

where $b_f^p$ is the number of bandwidth units that is allocated to the flow $f$ on port $p$.

## III. HRL-Agent and Bandwidth Allocation Framework

In this section, we present a framework to solve the bandwidth allocation problem in the EC based system. As depicted in Fig. 2, SDN controller acts as a bridge between HRL-agent and data-plane. It has two responsibilities: 1) Collecting statistics from the data-plane and building state of the network; 2) Creating/updating meters according to the bandwidth allocation policy learned by the HRL agent. This process of information collection and meter creation forms a closed loop and called network control loop [11]. With the help of this network control loop, the HRL agent interacts with the network environment to learn optimal bandwidth allocation policies.

### A. Semi-Markov Decision Process (SMDP) Formulation

We first formulate the bandwidth allocation problem in the EC system into semi-Markov decision process (SMDP). We define state space and action space followed by our unique reward model. At each state, different resource allocation decisions yield different rewards. The goal is to maximize the long-term reward (i.e. minimize end-to-end delay, system cost, packet loss and improve throughput).

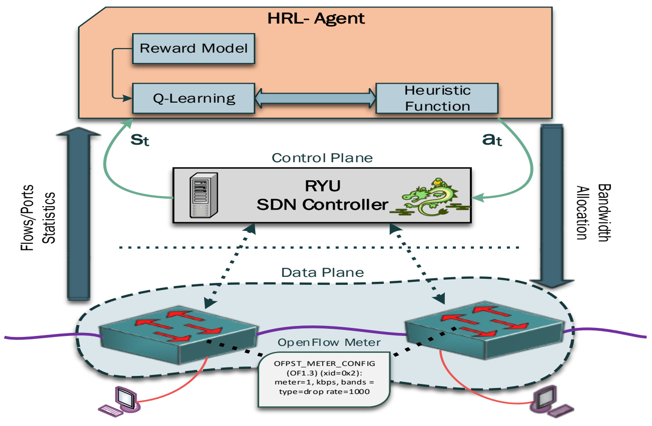

Figure 2: The structure of HRL-Agent and bandwidth allocation framework. The detail is described in Section III.

*1) State Space:* The state of the system at any time $t$ is the observation of traffic demands of flows and packet loss rates at a switch. Therefore, each distinct state is a sequence of observations of all the nodes in a network, $s_t = \{(R_f^p(t), L_f^p(t))_1, (R_f^p(t), L_f^p(t))_2, ..., (R_f^p(t), L_f^p(t))_N\}$. In each decision epoch, HRL agent learns optimal resource allocation strategies based on these sequences which leads to a very large but finite SMDP chain. $R_f^p$ and $L_f^p$ denote the traffic rate and packet loss rate of a flow $f$ at port $p$, respectively. Thus, the current average rate at time $t$ is calculated as the rate during the last $\tau$ till the current time $t$ and can be determined as:

$$R_f^p(t) = \frac{Rx_f^p(t) - Rx_f^p(t-\tau)}{t-\tau} \times 8, \tag{2}$$

where $Rx_f^p(t)$ is the total number of received bytes of a flow $f$ on port $p$ at time $t$, and we multiply by 8 to convert it in $bits/sec$. Eq. (2) gives us an average data rate of an interval $(t - \tau)$ which is fair to consider the current traffic status of a flow. Similarly, the packet loss rate is given as:

$$L_f^p(t) = \frac{\Delta Rx_{pkt}{}_f^p(t) - \Delta Tx_{pkt}{}_f^p(t)}{\Delta Rx_{pkt}{}_f^p(t)}, \tag{3}$$

where $\Delta Rx_{pkt}{}_f^p$ and $\Delta Tx_{pkt}{}_f^p$ represent the change in number of received and transmitted packets, respectively, of flow $f$ on port $p$ during $(t-\tau)$. As explained in later section, we try to keep this time interval as short as possible in order to replicate a real environment and measure precise readings.

*2) Action Space:* The action for a flow $f$ on port $p$, in any state, refers to all possible allocations in that state which is chosen according to the action selection policy ($random, greedy$ etc.). Therefore, the action set becomes, $A_{s_t} = \{b_f^p\}, (f \in [1, F], p \in [1, P])$, where $b$ represents a new allocation at time $t$. By this new allocation ($b$ units of bandwidth (Section II-C)) at time $t$, the state advances into the next state $s_{t+1}$.

*3) Reward Model:* The goal of HRL agent is to minimize the overall system cost by taking a sequence of actions in all the states. According to the definition of a real-valued reward function in [21], [22], in order to find an optimal resource allocation policy by taking action $a_t$ at state $s_t$, system advances into a new state $s_{t+1}$ and receives a reward $r_t$ from the environment, which is system cost for action $a_t$. In our model, the reward can be calculated as the sum of cost and penalty of resource procurement in the system.

$$cost_f = b_f \times C, \tag{4}$$

where $C$ is the system cost per unit of bandwidth and this can be set according to the given environment. Penalty for a flow is defined as the extra cost for imprecise resource allocations and given as:

$$penalty_f = \omega_f \times \frac{R_f}{b_f}, \tag{5}$$

where $\omega_f$ is the weight, $R_f$ is current rate and $b_f$ is new rate allocated to the flow $f$. The penalty equation helps the HRL agent to learn precise resource allocation policy by keeping the penalty ratio to minimum. Therefore, reward is given as:

$$r_t(s_t, a_t) = cost_f(s_t, a_t) + penalty_f(s_t, a_t). \tag{6}$$

We calculate reward for all the flows on all ports. This approach is worthwhile in a way that the QoS requirement (current rate and weight) in each flow may vary and calculating reward for every individual flow can better assist the HRL agent to derive an optimal resource allocation policy.

The optimization problem of bandwidth allocation, while considering the varying QoS requirements and constrained resources, is the maximization of the long-term reward i.e. minimization of the system cost in our case, and formulated as below:

$$maximize \sum_{t=1}^{T} r_t(s_t, a_t) \tag{7}$$

subject to:

$$U_p^n(t) \leq 1, \forall t \in T, \forall p \in P, \forall n \in N \tag{8}$$

$$1 \leq b_f \leq b_{f_{Max}}, \forall f \in F \tag{9}$$

$$L_f < \theta_f, \forall f \in F, \tag{10}$$

where constraint (8) describes that the usage of all ports on all switches does not exceed it's total available capacity ($B_p$) at any time. The constraint (9) guarantees bandwidth allocation between 1 unit and maximum allowed units for the flow $f$. Lastly, the constraint (10) ensures that the packet loss rate remains below the threshold $\theta_f$ which is further explained in the later section.

### B. Heuristic Reinforcement Learning Algorithm

Traffic patterns in an EC can change over time which makes it a completely non-stationary environment. So, we modify our training algorithm from the standard Q-Learning. The goal is to devise an algorithm which is capable of quickly adapting to the changes in the environment.

*1) Q-Learning:* Resource allocation policy is derived using Q-Learning which is an Off-Policy based machine-learning algorithm. Q-Learning agent can follow any policy ($\epsilon - greedy$, $\epsilon - soft$, $softmax$ etc.) to calculate the value function $Q(s, a)$ by utilizing the above given reward model (Eq. (6)). The optimum action-value function $Q^*(s, a)$ is the maximum expected achievable reward which follows the Bellman equation [21]. Therefore, the $Q$ value estimates are updated as follows:

$$\begin{aligned} Q_{t+1}(s_t, a_t) = (1-\alpha) Q_t(s_t, a_t) + \\ \alpha(r_t(s_t, a_t) + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1})), \end{aligned} \tag{11}$$

where $\alpha$ is the learning rate and $\gamma$ is the discount factor which is used to influence the impact of the future reward on the current action [22].

*2) Heuristic Function:* Resource allocation policy derivation is performed through trial-and-error interactions of the Q-Learning agent with the environment which is a very time consuming process. A heuristic function can be used to speed up the convergence rate. In order to improve agent's behavior, heuristic function selects suitable actions to steer the exploration of the state-action space in the direction of useful regions [24]. The heuristic function $H(s_t, a_t)$ is an action policy modifier which specifies the significance of performing an action $a_t$ at time $t$ when visiting state $s_t$. Domain information is extracted from the environment and a heuristic is composed from the extracted structural information to accelerate the learning process. How to incorporate the heuristic function with standard $\epsilon - greedy$ Q-Learning action value function is given below:

$$\pi(s_t) = \begin{cases} \arg\max_{a_t} [Q_t(s_t, a_t) + \xi(H_t(s_t, a_t))^\beta], & if\ q \leq \epsilon_t, \\ a_{random} & otherwise. \end{cases} \tag{12}$$

In above Eq., $\xi$ and $\beta$ are design parameters to control the impact of heuristic value function on action selection policy $\pi(s_t)$. $q$ is uniform random value from 0 to 1, $\epsilon_t$ is also between 0 and 1 which is the probability of exploration (random action) at time $t$ and $a_{random}$ is a uniform random action chosen from the possible actions in state $s_t$. According to [24], [25], when $\xi = \beta = 1$, the heuristic function update formula is given as:

$$H_{t+1}(s_t, a_t) = \begin{cases} \max_a Q_t(s_t, a) - Q_t(s_t, a_t) + \eta, & if\ a_t = \pi^H(s_t), \\ 0 & otherwise, \end{cases} \tag{13}$$

where $a_t$ is an optimal action influenced by the heuristic $\pi^H(s_t)$, and $\eta$ is a small positive value usually set to 1 [24], [25]. According to [24], a heuristic is derived in two stages, in the first stage, heuristic is built only once by extracting the domain information. In the following stage, function is not updated to overcome a bad heuristic and used as is in the action selection policy. On the other hand, [25] proposes a heuristic reinforcement learning based on the state backtracking technique which no longer requires stage one; nonetheless, it causes additional delay to update the action transfer probability function at the end of every training episode. Therefore, we adapt the former approach in order to save the overall training time.

*3) Linear Annealing:* Bianchi [24] used fixed exploration rate which is suitable only for small environments. However, for a large environment, we start exploration with highest probability and then linearly decay the exploration rate. This strategy gives sufficient trials to explore a large environment randomly at the beginning and then exploits the already known good policies more often. Further, randomness breaks the correlation of learning data and reduces the variance in the update [22]. The decay frame or the linear annealing frame $LAF(t)$ for $\epsilon_t$ is given below:

$$LAF(t) = \frac{\epsilon_S - \epsilon_F}{T} \times t, \tag{14}$$

where $T$ is maximum number of iterations (steps) in one training episode, and $\epsilon_S$ and $\epsilon_F$ represent the start $\epsilon$ and final $\epsilon$, respectively. Thereafter, $\epsilon_t$ can be given as follows:

$$\epsilon_t = \max(\epsilon_F, (\epsilon_S - LAF(t))) \tag{15}$$

The Eq. (15) describes that the exploration rate is high at the beginning and annealed with the time that comes to its final state at the end of first training episode. A novel study, human-level control through deep reinforcement learning [26], uses a fixed approach of linear annealing where $\epsilon$ is decayed in a fixed number of iterations. However, our proposed model gives the malleability to adjust annealing rate with respect to the size of the environment and the number of learning iterations.

*4) Training Process:* A recent study in SDN [12] periodically collects network status information which generates a lot of control traffic. We avoid this problem by sending flows/ports statistics requests to the switches one by one in a round-robin fashion. This is beneficial in two ways:

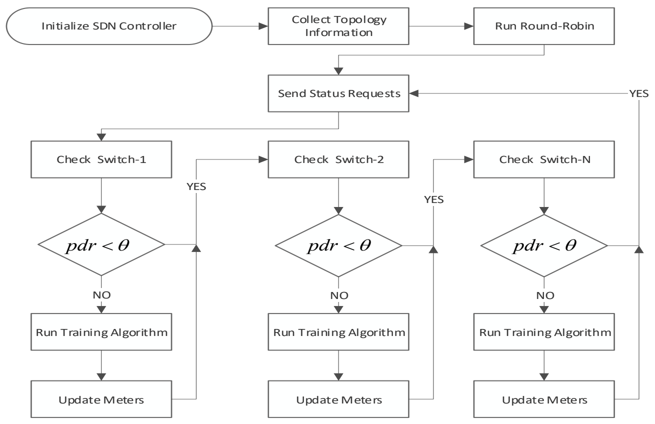

Figure 3: Workflow for Round-Robin based control algorithm with Pruning Principle. The detail is given in Sections III-B4 and III-B5.

1) It minimizes the control traffic in the network; 2) It is computationally light weight as the SDN controller has to process small amount of information at a time.

*5) Pruning Principle:* According to the round-robin based training process (Section III-B4), one switch is processed at a time, which causes delay in the convergence of whole topology. To overcome this situation, we introduce a pruning principle to avoid processing of all the switches every time. We define a packet loss rate threshold $\theta$. This threshold is different for all the flows which is inversely proportional to the weight ($\omega$) of the flow and given as:

$$\theta_f = \frac{1}{\omega_f} \times K, \tag{16}$$

where $K$ is constant for all the flows. This inverse relationship helps to have a different threshold based on the weights which also ensures the QoS requirements (Eq. (10)) for all the flows. The major contribution of our proposed pruning principle is the reduction of topology convergence time by a significant amount which eliminates the issue caused by round-robin training method.

A workflow for round-robin based training process and pruning principle together is depicted in Fig. 3. SDN controller is initialized, and topology information is collected which includes switches, ports and flows information. Thereafter, the round-robin process is started that sends flows/ports statistics request to each switch one-by-one. If the packet loss rate for all flows in a switch is below the defined threshold ($\theta_f$), then we prune that switch for retraining and proceed to the next one, otherwise we run our proposed training algorithm to determine optimal resource allocation policy and update the meters. Note that pruning principle further reduces the control traffic by avoiding futile meter updates in the data-plane.

The round-robin control algorithm with pruning principle is shown in Algorithm 1. The algorithm for Heuristic reinforcement learning with linear annealing to obtain optimal bandwidth allocation policy ($Q^*$) in SDN-enabled EC environment is summarized in Algorithm 2.

**Algorithm 1:** Round-Robin control Algorithm with Pruning Principle

```
Input  : Network Topology
1  Initialize SDN controller
2  Collect network topology information
3  while true do
4      for switch = 1 to N do
5          Send flows and ports statistics request to switch
6          for port = 1 to P do
7              for flow = 1 to F do
8                  if $pdr_f^p < \theta_f$ then
9                      $continue$;
10                 else
11                     Run Training algorithm and update
                         OpenFlow meters on switch
12                     Proceed to next switch
13                 end
14             end
15         end
16     end
17 end
```

**Algorithm 2:** Heuristic Reinforcement Learning (HRL)

```
Input  : Traffic flows with varying QoS demands
1  Initialize the environment
2  Initialize action-value function $Q$ arbitrarily
3  Initialize Heuristic function $H$
4  for episode = 1 to E do
5      Reset the environment to initial state
6      for $t$ = 1 to $T$ do
7          Calculate $\epsilon_t$ using Eq. (15)
8          Choose action $a_t$ using Eq. (12)
9          Execute action $a_t$, observe next state $s_{t+1}$ and receive
             reward $r_t$
10         Update the values of heuristic function using Eq. (13)
11         Update the values of $Q$ according to the update rule in
             Eq. (11)
12         $s_t = s_{t+1}$
13     end
14 end
Output: Optimal Bandwidth Allocation policy $Q^*$
```

## IV. Experimental Results

In this section, we analyze the performance of our proposed bandwidth allocation framework. We develop the system model (Traffic and Bandwidth model in Section (II)) and proposed HRL algorithm (Section (III)) in Python using RYU SDN controller 4.34. The meter feature is supported in OpenFlow version 1.3 and above [16] which is not supported by the POX controller. According to [27], RYU has overall better performance compared to the POX controller. Open Daylight, which is a Java based SDN controller, also supports OpenFlow version 1.3 [28]. However, it is easier to build machine learning environment in Python, thus, we choose RYU controller to implement our end-to-end framework. The proposed framework is then evaluated in both Mininet and hardware OpenFlow switches based environments which is explained in the following section.

### A. Experiment Setup

We validate the long-term stability, effectiveness and scalability of our proposed methods in two different environments. One is based on real hardware OpenFlow switches (Zodiac FX) and other is based on Mininet. Scalability testing of the algorithm is performed by creating topology in Mininet which uses openvswitch (v2.14.1 with OpenFlow 1.3 support) to emulate the switches. Whereas, stability and the long-term performance efficiency of our proposed strategy is evaluated in both Mininet and hardware switches based environments. Zodiac FX supports up to 212 flows and 8 meters per switch and the port capacity/bandwidth for these switches is $100Mbps$, therefore, we scale down the traffic accordingly. In both cases, the RYU SDN controller runs on Ubuntu 18.04 with 3.40 GHz Intel Core i7 processor and 8GB memory.

The list of hyperparameters and their corresponding default values that are used during the experiment are given in Table I. The two baselines that we use to compare with our proposed methods are described below:

- **HAQL [24]**: Existing heuristic accelerated Q-learning approach to accelerate the learning mechanism in Q-learning. This approach used fixed exploration rate of 10% to calculate the heuristic only once. This baseline is used to compare the convergence speed and learning efficiency with our proposed algorithm. The rest of the parameters ($\alpha, \beta, \gamma, \eta, \xi$) and reward function are kept the same for a fair comparison.
- **MMWFS Allocation [29]**: As explained in Section (II-B), our proposed algorithm prioritizes flows based on their weights. So, we choose *max-min weighted fair share* (MMWFS) bandwidth allocation algorithm for a fair comparison which considers resource sharing associated with the weights of different flows. This baseline algorithm is common to both conventional and advanced SDN networks and defined as follows:
    - *"Resources are allocated in the order of increasing demand, normalized by the weight"*
    - *"No source/flow gets a resource share larger than its demand/current rate"*
    - *"Sources/flows with unsatisfied demands get resource shares in proportion to their weights".*

  In our testing, MMWFS is used to compare the key performance metrics such as throughput, end-to-end delay, packet-loss rate and overall system cost with our proposed algorithm.

### B. Convergence

The envisioned deployment of COSMOS testbed connects Rutgers, Columbia, CCNY and New York University [19].

Table I: Hyperparameters and corresponding Default Values with Description

| Hyperparameter | Value | Description |
|---|---|---|
| $\alpha$ | 0.1 | The learning rate alpha, set between 0 and 1, used in the Q-learning update. |
| $\beta$ | 1 | Design parameter to control the influence of heuristic function. |
| $\gamma$ | 0.99 | Discount factor gamma, set between 0 and 1, used in the Q-learning update. |
| $\epsilon_S$ | 1 | Start value of exploration rate ($\epsilon - greedy$). |
| $\epsilon_F$ | 0.1 | Final value of exploration rate ($\epsilon - greedy$). |
| $\eta$ | 1 | A small positive value used in the heuristic function update. |
| $\xi$ | 1 | Design parameter to control the influence of heuristic function. |
| $B$ | 1000 | Total bandwidth in Mbps available at each switch port. |
| $C$ | 1 | Per unit of bandwidth cost in cents/hour. |
| $E$ | 10 | Number of episodes to train the HRL agent. |
| $T$ | 10000 | Number of learning steps or iterations per episode. |
| $K$ | $10^{-1}$ | A constant used in the calculation of packet loss rate threshold. |

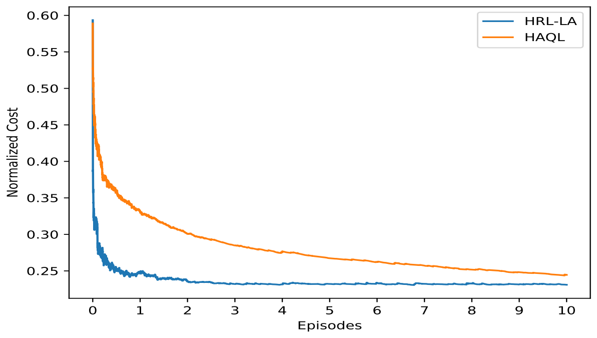

Figure 4: Convergence speed and Normalized cost comparison between HRL-LA and HAQL.

Each site is considered to have a similar topology as shown in Fig. 1. Therefore, to perform the scalability test, a topology is created using Mininet with 16 switches (4 switches at each site), 32 hosts, 47 links and 500 plus flows in one switch. Traffic is generated from various flows with varying data rates (using IPERF v3.1.3) and QoS demands so that the learning agent can derive an efficient resource allocation policy with better long-term reward. We compare the convergence speed of our proposed strategy HRL-LA (Heuristic Reinforcement learning with linear annealing) with the existing HAQL algorithm. As shown in Fig. 4, HRL-LA converged almost in one episode and obtained overall better reward (lower cost) than the HAQL. Initially, there are some oscillations in the case of HRL-LA which means the agent explores the environment. However, after the 1st episode when heuristic is applied, the divergence and oscillations are significantly reduced which demonstrates the higher efficiency of the learning procedure of our proposed strategy. Multiple experiments are carried out and the recorded average time that it takes to reach the convergence point for one switch is 9.6 seconds for HRL-LA and 138 seconds for HAQL. The improvement of this magnitude is due to our linear-annealing method as described in Section III-B3. This shows that our proposed method outperforms the existing approach by reducing the convergence time to nearly $1/15^{th}$ and achieving a better reward which can be practical for large environments.

## C. Performance Analysis

We miniaturize COSMOS CCNY testbed environment, as shown in Fig. 1, by using 4 real Zodiac FX OpenFlow switches (*Firmware* v0.86) [17]. A similar topology is emulated in Mininet as well to assess the long-term stability of our algorithm across different environments. During the experiment four categories of traffics with varying data rate and different QoS demands are generated that can be anticipated in an Edge-Cloud (EC) environment. RT-1 (real-time) is considered time critical and bandwidth hungry traffic. In general, this type of traffic is generated by Augmented and Virtual Reality (AR/VR) based devices, surveillance drones/cameras and Zoom meetings. The data rate for such traffic is set to vary in the range [6Mbps, 8Mbps]. RT-2 is also very time critical traffic but not bandwidth hungry. Smart-city sensors (temperature, waste management, smart parking sensors etc.) and other IoTs in an EC can be the source of RT-2 traffic, and the data rate is set to vary in the range [1Mbps, 2Mbps]. BE-1 (best-effort) is considered delay tolerant but bandwidth hungry traffic. Such traffic can be expected from the users downloading or uploading files, and data rate for BE-1 varies in the range [6Mbps, 8Mbps]. Lastly, BE-2 is also delay tolerant similar to BE-1, however, produces fewer amount of traffic in the range [1Mbps, 2Mbps] which can contain text, email, web etc. traffic. Real-time traffic has higher weight ($\omega$) values that defines the QoS demands and helps the algorithm to prioritize this traffic over best-effort.

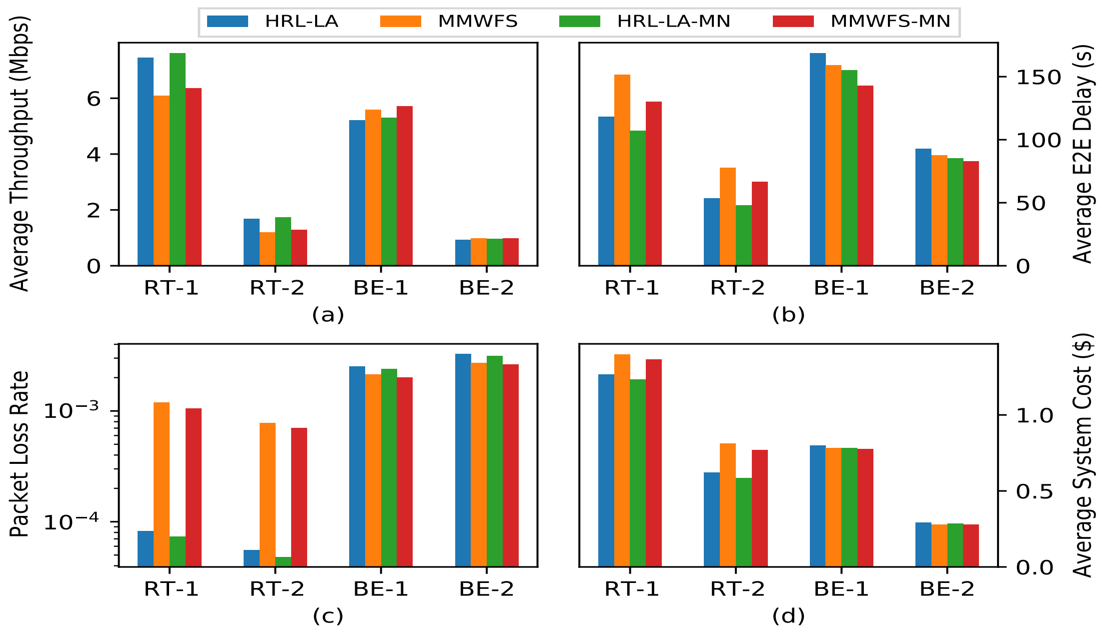

Figure 5: Performance comparisons of proposed strategy with *max-min weighted fair share* bandwidth allocation algorithm. Both evaluations are conducted on hardware switches and Mininet for real-time-1, real-time-2, best-effort-1 and best-effort-2 traffics. In view of (a) Average Throughput, (b) Average End-to-End Delay, (c) Packet Loss Rate and (d) Average System Cost; vertical axis of (c) is in log scale.

The experiment is performed for 5 hours (18000 seconds) in each environment (virtual and physical) and analyzed the long-term performance of our proposed strategy by comparing the throughput, end-to-end delay, packet loss rate and the overall system cost with *max-min weighted fair share* (MMWFS) bandwidth allocation approach [29]. In Fig. 5, HRL-LA-MN and MMWFS-MN refer to the results in Mininet based network. Our proposed algorithm is stable in both environments, however, the performance in virtual network (Mininet) is marginally better as compared to the physical network. This is because the virtual network is with in the same machine (Section IV-A) which induces lower deviations in the jitter, and other real-world factors are also neglected in the case of Mininet [30]. In the following, we discuss the performance of our proposed strategy and MMWFS in hardware switches based network only, which demonstrates a near real-world environment.

After the derivation of first bandwidth allocation policy for flows, only 6 additional training requests were triggered during the whole experiment. Whereas, in the case of MMWFS algorithm, 265 re-computations were recorded. This shows that our proposed algorithm can quickly adapt to the changes in a non-stationary environment and obtain a long-term resource allocation policy which can endure fluctuations in the traffic. After the end of every retraining or re-computation, meters are updated using control messages from control-plane to the data-plane. In effect of the significant less number of retrainings compared to MMWFS, the proposed strategy helps to minimize the meter updates, thus, reducing control traffic in the network to nearly $1/45^{th}$.

Fig. 5(a) shows average throughput which is measured with IPERF v3.1.3 during the experiment for different traffics. It is evident from the results that the resource allocation policy derived from HRL-LA prioritizes and allocates more bandwidth to real-time traffic at the expense of less bandwidth allocation to the best-effort traffic. Nonetheless, the overall network throughput is improved by 10.82% as compared to the MMWFS approach. This is due to the fact that MMWFS algorithm never allocates more than the demand (Section IV-A). Fig. 5(b) depicts the average end-to-end (E2E) delay comparison. In order to calculate E2E delay, 100MB of data is transferred for RT-1 and BE-1 traffic, and 10MB of data is transferred for RT-2 and BE-2. This experiment is repeated several times and average is measured. Although E2E delay is directly dependant on the allocated bandwidth, it is important to show the impact of network throughput on E2E delay. It can be clearly seen in Fig. 5(b) that MMWFS algorithm treats all kinds of traffic equally based on their weights and demands. Whereas, HRL-LA prefers the real-time traffic over best-effort and improves the overall E2E delay in the network by 9.67%.

Packet loss rate is an essential performance metric in the QoS requirements which is calculated using Eq. (3). As shown in Fig. 5(c), our proposed strategy maintains a packet loss rate for RT traffic under $10^{-4}$. Although HRL-LA compromises on the BE traffic, yet, keeps packet loss rate below the threshold level (Eq. (16)) on average to satisfy the QoS demands. When compared to MMWFS, a 13.17% less packet loss rate is observed in case of our proposed method. As retraining of the resource allocation policy directly depends on the packet loss rate (Fig. 3), the improvement in packet loss rate also contributes in the reduction of re-computations as discussed above. Lastly, average system cost for bandwidth allocation is calculated using Equations (4) and (5) by utilizing the observed average throughput (Fig. 5(a)) for all traffic types. The cost for allocating fewer amount of bandwidth is smaller (Eq. (4)); however, penalty equation (Eq. (5)) poses an additional amount if bandwidth is imprecisely allocated. In effect of this, the overall cost is increased for allocating less bandwidth to a high weighted flow. Therefore, RT traffic, despite having more bandwidth, induces less cost due to the low amount of penalty and best-effort traffic induces slightly more cost. The small difference in the cost of BE traffic is due to the fact that the weight values for BE traffic are smaller which does not cause high impact on the penalty. As shown in Fig. 5(d), there is an 8.93% improvement in the overall system cost for bandwidth allocation.

In summary, our proposed algorithm prioritizes real-time traffic over best-effort while satisfying the QoS demands by all kinds of traffic. HRL-LA improves the overall network throughput, E2E delay, packet loss rate and system cost on average at the expense of marginal training time which benefits in the long-run to significantly reduce control traffic in the network. Moreover, our proposed algorithm consistently outperforms MMWFS approach in both virtual (Mininet) and hardware switches based environments.

## V. Conclusion

In this study, we investigated the problem of network resource management for the IoTs and mobile users in an SDN enabled edge-cloud environment. We presented a novel dynamic bandwidth allocation framework by introducing HRL-LA, a heuristic reinforcement learning-based system with linear annealing technique. OpenFlow meter is utilized to achieve a fine-grained control for demand-based bandwidth allocation of individual flows. Our proposed strategy is fast in convergence speed, scalable and adaptive to the dynamics of a practical environment. To show the effectiveness of our strategy we developed the proposed framework in Python and conducted two experiments, one in the Mininet and other on the Zodiac FX hardware switches by replicating real COSMOS CCNY testbed environment. As compared to an existing algorithm, our proposed technique reduces convergence time by nearly $1/15^{th}$. When compared to a well-known *max-min weighted fair share* bandwidth allocation algorithm, HRL-LA improves the overall network throughput, end-to-end delay, packet loss rate and system cost by 10.82%, 9.67%, 13.67% and 8.93% on average, respectively, while satisfying the QoS demands as well. A pruning principle is also proposed to avoid undesired computations, which aids to reduce control traffic in the network to nearly $1/45^{th}$. Furthermore, our proposed strategy is stable and outperforms alternative approach in both virtual (Mininet) and physical (Zodiac FX) networks.

For future work, we plan to investigate for on-demand training procedures which will further reduce the topology convergence time. Additionally, investigation on multi-agent reinforcement learning will help to expand the current framework to several distributed edge-clouds to solve the resource management problem in NSF IRNC testbed based multi-domain network.

## Acknowledgment

This work is supported in part by NSF JUNO2 (*Japan-US Network Opportunity 2*) (Award No. 1818884) and NSF IRNC (Award No. 2029295).

## References

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, "Mobile edge computing—A key technology towards 5G," in ETSI (European Telecommunications Standards Institute), Sophia Antipolis, France, ISBN: 979-10-92620-08-5 no. 11, Sep. 2015.

[2] Z. Ning, X. Kong, F. Xia, W. Hou and X. Wang, "Green and Sustainable Cloud of Things: Enabling Collaborative Edge Computing," in IEEE Communications Magazine, vol. 57, no. 1, pp. 72-78, January 2019.

[3] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.

[4] M. A. Salahuddin, A. Al-Fuqaha and M. Guizani, "Software-Defined Networking for RSU Clouds in Support of the Internet of Vehicles," in IEEE Internet of Things Journal, vol. 2, no. 2, pp. 133-144, April 2015, doi: 10.1109/JIOT.2014.2368356.

[5] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung and H. Yin, "Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep Reinforcement Learning Approach," in IEEE Communications Magazine, vol. 55, no. 12, pp. 31-37, Dec. 2017, doi: 10.1109/MCOM.2017.1700246.

[6] A. Bentaleb, A. C. Begen, R. Zimmermann and S. Harous, "SDNHAS: An SDN-Enabled Architecture to Optimize QoE in HTTP Adaptive Streaming," in IEEE Transactions on Multimedia, vol. 19, no. 10, pp. 2136-2151, Oct. 2017, doi: 10.1109/TMM.2017.2733344.

[7] S. D'Oro, L. Galluccio, S. Palazzo and G. Schembra, "A Game Theoretic Approach for Distributed Resource Allocation and Orchestration of Softwarized Networks," in IEEE Journal on Selected Areas in Communications, vol. 35, no. 3, pp. 721-735, March 2017, doi: 10.1109/JSAC.2017.2672278.

[8] Stampa, G., Arias, M., Sanchez-Charles, D., Muntes-Mulero, V., and Cabellos, A., “A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization”, *arXiv:1709.07080v1*, 2017.

[9] X. Huang, T. Yuan and M. Ma, ”Utility-Optimized Flow-Level Bandwidth Allocation in Hybrid SDNs,” in IEEE Access, vol. 6, pp. 20279-20290, 2018, doi: 10.1109/ACCESS.2018.2820682.

[10] J. M. Boley, E. Jung and R. Kettimuthu, ”Adaptive QoS for data transfers using software-defined networking,” 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), Bangalore, 2016, pp. 1-6, doi: 10.1109/ANTS.2016.7947874.

[11] Y. Hu, Z. Li, J. Lan, J. Wu and L. Yao, ”EARS: Intelligence-driven experiential network architecture for automatic routing in software-defined networking,” in China Communications, vol. 17, no. 2, pp. 149-162, Feb. 2020, doi: 10.23919/JCC.2020.02.013.

[12] S. Kumar, G. Bansal and V. S. Shekhawat, ”A Machine Learning Approach for Traffic Flow Provisioning in Software Defined Networks,” 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 2020, pp. 602-607, doi: 10.1109/ICOIN48656.2020.9016529.

[13] R. Haw, M. G. Rabiul Alam and C. S. Hong, ”A context-aware content delivery framework for QoS in mobile cloud,” The 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, 2014, pp. 1-6, doi: 10.1109/APNOMS.2014.6996607.

[14] S. Lin, I. F. Akyildiz, P. Wang and M. Luo, ”QoS-Aware Adaptive Routing in Multi-layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach,” 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA, 2016, pp. 25-33, doi: 10.1109/SCC.2016.12.

[15] M. Rahouti, K. Xiong and Y. Xin, ”Secure Software-Defined Networking Communication Systems for Smart Cities: Current Status, Challenges, and Trends,” in IEEE Access, vol. 9, pp. 12083-12113, 2021, doi: 10.1109/ACCESS.2020.3047996.

[16] ”OpenFlow Switch Specification,” *Version 1.5.1 (Protocol version 0x06)*, ONF TS-025, March 2015.

[17] ”Zodiac FX,”, *Firmware Version 0.86* Northbound Networks, March 2017 [online] Available at: https://forums.northboundnetworks.com/ [Accessed Jan 19, 2021].

[18] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. ”A network in a laptop: rapid prototyping for software-defined networks.” In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX). Association for Computing Machinery, New York, NY, USA, Article 19, 1–6. DOI:https://doi.org/10.1145/1868447.1868466

[19] NSF, PAWR ”COSMOS: Cloud Enhanced Open Software Defined Mobile Wireless Testbed for City-Scale Deployment” https://cosmos-lab.org.

[20] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, ”An Architecture for Differentiated Services”, RFC 2475, DOI 10.17487/RFC2475, December 1998.

[21] Y. Liu, M. J. Lee and Y. Zheng, ”Adaptive Multi-Resource Allocation for Cloudlet-Based Mobile Cloud Computing System,” in IEEE Transactions on Mobile Computing, vol. 15, no. 10, pp. 2398-2410, 1 Oct. 2016.

[22] M. Cheng, J. Li and S. Nazarian, ”DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers,” 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), Jeju, 2018, pp. 129-134.

[23] R. S. Sutton and A. G. Barto, ”Reinforcement Learning: An Introduction,” The MIT Press, Cambridge, Massachusetts, sec. 6.5, 1998, [online] Available: http://incompleteideas.net/sutton/book/ebook/the-book.html

[24] Bianchi, R.A.C., Ribeiro, C.H.C. and Costa, A.H.R. Accelerating autonomous learning by using heuristic selection of actions. J Heuristics 14, 135–168 (2008).

[25] M. Fang, H. Li and X. Zhang, ”A Heuristic Reinforcement Learning Based on State Backtracking Method,” 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Macau, 2012, pp. 673-678.

[26] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).

[27] A. L. Stancu, S. Halunga, A. Vulpe, G. Suciu, O. Fratu and E. C. Popovici, ”A comparison between several Software Defined Networking controllers,” 2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS), Nis, 2015, pp. 223-226, doi: 10.1109/TELSKS.2015.7357774.

[28] M. Z. Shaikh and S. H. Darekar, ”Performance Analysis of Various Open Flow Controllers by Performing Scalability Experiment on Software Defined Networks,” 2018 3rd International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2018, pp. 783-787, doi: 10.1109/ICICT43934.2018.9034343.

[29] J. Chou and B. Lin, ”Optimal multi-path routing and bandwidth allocation under utility max-min fairness,” 2009 17th International Workshop on Quality of Service, Charleston, SC, 2009, pp. 1-9, doi: 10.1109/IWQoS.2009.5201396.

[30] J. M. Jimenez, O. Romero, A. Rego, J. Lloret, ”Analyzing the Performance of Software Defined Networks vs Real Networks,” International Journal on Advances in Networks and Services, vol 9 no 3 & 4, 2016, pp. 107-116.