A QoS Optimization Technique with Deep Reinforcement Learning in SDN-Based IoT

Mohammadreza Moslehi[1*], Hossein Ebrahimpour-Komleh[2], Salman Goli-Bidgoli[3], Reza Taji[4]
1- Department of Computer Engineering, University of Kashan, Kashan, Iran.
Email: moslehi@acecr.ac.ir (Corresponding author)
2- Department of Computer Engineering, University of Kashan, Kashan, Iran.
Email: ebrahimpour.kashanu@gmail.com
3- Department of Computer Engineering, University of Kashan, Kashan, Iran.
Email: salmangoli@gmail.com
4- Independent Researcher in the Field of AI and Neural Networking.
Email: ibdrezataji@gmail.com

Received: March 2021 Revised: May 2021 Accepted: July 2021

ABSTRACT:
In recent years, exponential growth of communication devices in Internet of Things (IoT) has become an emerging technology which facilitates heterogeneous devices to connect with each other in heterogeneous networks. This communication requires different level of Quality-of-Service (QoS) and policies depending on the device type and location. To provide a specific level of QoS, we can utilize emerging new technological concepts in IoT infrastructure, Software-Defined Network (SDN) and, machine learning algorithms. We use deep reinforcement learning in the process of resource management and allocation in control plane. We present an algorithm that aims to optimize resource allocation. Simulation results show that the proposed algorithm improved network performances in terms of QoS parameters, including delay and throughput compared to Random and Round Robin methods. Compared to similar methods, the performance of the proposed method is also as good as the fuzzy and predictive methods.

KEYWORDS: Internet of Things, Software-Defined Networking (SDN), Deep Reinforcement Learning, QoS.

1. INTRODUCTION

Consequence of the rapid growth in new technologies including IoT, Smart city, and Smart home would result in significant increase in number of connected things and devices such as smartphones and higher demands for more network resources. Cisco's annual internet report white paper predicts a growth of 14.7 billion connected devices and 5.7 billion for total number of global mobile subscribers by 2023 [1]. The results are a rapid growth on the resource demands and an increase in network traffic, delay, and congestion. These also pose significant challenges such as complexity of network management and the lack of scalabilities. Smart city applications such as energy, security, and e-health have stringent requirements with expected QoS. In traditional network technologies, network devices such as switches and routers work on their own data plane and cannot meet those challenges. To cope with these issues, we need flexible and new networking technologies to adapt to dynamics needs. Software-Defined Networking or SDN [2] allows programming for configuration and modification of different network devices. With such technology, we are capable to cope with today's network requirements in a more timely manner and efficient way. Open Networking Foundation or ONF characterized SDN technology as modern and emerging architecture which is perfect for new applications with a delay-sensitive, high-bandwidth, and dynamic nature. Some of SDN-based IoT advantages can be the dynamic manageability, cost-efficiency, and adaptability [3]. The network control plane is separated from the data plane or forwarding functions in SDN architecture. So, we can make the network control directly programmable. With the SDN, the underlying infrastructure is abstracted from applications and network services. With the dynamic network programming ability of SDN, we can centrally control and manage the entire network. With a centralized SDN controller, it is possible to perform dynamic network optimization operations and management. One of the requirements of IoT environments is network management automation which is possible via the programmability of SDN [4]. Other challenges in the networking domain is resource

Paper type: Research paper
DOI: https://doi.org/10.52547/mjee.15.3.105
How to cite this paper: M. R. Moslehi, H. Ebrahimpour-Komleh, S. Goli-Bidgoli and R. Taji, **"A QoS Optimization Technique with Deep Reinforcement Learning in SDN-Based IoT"**, *Majlesi Journal of Electrical Engineering*, Vol. 15, No. 3, pp. 105-113, 2021.

allocation. Deep reinforcement learning has been successfully utilized to solve this issue.
So, our main contributions in this paper are:

- Investigating the importance of machine learning algorithms, especially deep reinforcement learning, in resource allocation process and improving the QoS of SDN-based IoT.
- Proposing a deep reinforcement learning algorithm for solving resource allocation between clients and servers in an SDN-based IoT.

Our simulations and evaluations results show that the proposed solution improves QoS in terms of delay and throughput compared to Random and Round Robin methods, and also show that the performance of the proposed method is near the fuzzy and predictive method in [5]. The remainder of the paper is organized as follows: Section 2 reviews some related works. In section 3, a brief overview of DRL is explained. Section 4 proposes the DRL model. We analyze the results in Section 5, and the paper concluded in section 6.

2. RELATED WORKS

Several techniques are used to improve QoS in network resource allocation policies. These techniques are significantly related to the domain of network environments. One of the broad research domains is QoS in the IoT. IoT has a rapid growth of its connected devices. IoT applications need some levels of QoS such as bandwidth, delay, and reliability.

several works have been done regarding IoT and QoS issues by using networking emerging technologies such as SDN [5]–[10]. In traditional networks, hardware devices manage connections between sources and destinations based on the static and inflexible parameters. Recently SDN-based IoT has received remarkable attention in field of QoS. Authors in [11] have been used SDN technology to overcome latency and security issues. They propose a deep-reinforcement-learning-based quality-of-service (QoS)-aware secure routing protocol (DQSP) that can extract knowledge from past traffic demands and optimize the routing policy dynamically with guaranteeing QoS. Authors in [12] focus on QoS differentiation by exploiting a multi-topology routing feature in RPL. They propose different objective functions to ensure the QoS differentiation at the network level by virtualizing the physical network into several RPL instances. In [13], the authors use an SDN-based IoT infrastructure based on the simulated annealing algorithm and propose a QoS routing algorithm for that system.

In [14], a Reliable and Dynamic Routing Technique (RaDRT) is proposed that regulates traffic flows routing in an Edge-MANET environment by adopting the SDN approach. Their solution is based on SDN monitoring and management of networks while considering QoS requirements for running applications. Using SDN technology in smart city networks, a big data analysis approach is suggested in [6]. Their system includes Data Collection, Data Management, and Application levels. These elements are further connected via two intermediate levels working on SDN principles. Authors in [15] design a deep learning application awareness on SDN technology to improve QoS.

3. REINFORCEMENT LEARNING AND SDN

Reinforcement Learning, or RL, is one of the machine learning techniques in which an *agent* acts in the environment to maximize *reward*. In the RL, the agent learns by trials and errors. Due to the good results of using RL in various fields of application, this technique has received more attention. In general, machine learning refers to algorithms that can infer a mathematical model using large amounts of data. In most applications, problem-solving is formulated as an optimization problem, in which the algorithm minimizes the loss function, which is equal to the difference between the predicted result and the actual value. In the popular algorithm of machine learning, supervised learning, the algorithm takes a training set (a subset of the available complete data) and decreases the gradient until the loss function is reduced to an acceptable value or all the data is sampled. The goal of a machine learning algorithm is learning data distribution that accurately categorizes input data and prevents overfitting, that is, learning a mathematical model that processes only the dataset of the algorithm. RL can be described by the conditional domain of behavioral psychology. RL mimics the agent conditioned to adapt to an unknown environment. Instead of minimizing the loss function, the agent improves his behavior by receiving a reward for the performance of his actions. It does this by *stepping* into the environment. At each stage, the agent observes the current states, performs an action, and receives a numerical reward.

The agent chooses an action, and when action is done, the state of the environment changes. The desirability of this change is sent by a scalar reward to the agent. During the learning, the agent tries to maximize the cumulated reward [16]. The trial and error learning of the agent is done with several learning algorithms [17]. *Q-learning* is one of the learning algorithms used to learn the RL agent, this is a *temporal difference* or *TD learning* algorithm [18].

In each state, TD algorithms gradually build information for the best actions to take. This is done by the *policy*. The policy is a strategy that applies to the learning agent to decide the next action based on the current state. In other words, policy maps states of the environment to actions to be taken when in those states [16]. *Q-values Q (s, a*) are values that show the

desirability of each action *a*, in the state, *s*. In step *i* a reward $R(s_i, a_i)$ will be returned by the environment. The *Q*-value $Q(s_i, a_i)$ is related to the state-action pair (s_i, a_i) and is an approximation of the expected long-term reward for that pair. All the *Q*-values are stored in the *Q* table and based on these values; the agent chooses the next action by an *ε-greedy* policy.

By adopting the *ε-greedy* policy, an action with the highest *Q*-value will be selected by the agent with a probability of ε; otherwise, a random action with the probability of $1-\varepsilon$ will be chosen. Then the reward of $R(s_i, a_i)$ is learned, and the state will shift to s_{i+1}, and also *Q*-values stored on the table with the temporal difference way will be updated. *Q*-Values continuously updates until the agent takes the best action by using these values [18].

3.1. *Deep Q-learning*

Deep reinforcement learning or DRL as one of the RL methods has been widely proposed in recent years for challenges such as decision making in large state space [19]. *Q-network* is the agent that applies a neural network to represent the *Q* function. $Q(s_i, a_i; \theta)$ is a notation of a *Q*-network that θ is weights for the neural network. To approximate real *Q* value, *Q-network* is trained by updating θ. Neural networks are applied in *Q-Learning* because of great flexibility, but they may cause instability [20].

Deep *Q*-network benefits from a deep neural network. It is proven that *Deep Q-networks* have greater efficiency and more strong learning [20]. In addition to convert an ordinary *Q*-network into a *deep Q-network*, the *deep Q-network* utilized *Experience Replay* to store experience tuple $e_i=((s_j, a_j, R_j(s_j, a_j), s_{j+1}))$ at each step *i* into *replay memory* $D_i = \{ e_1, \ldots e_i)\}$ [21]. *Q-learning* training directly uses successive samples but to train weights of the deep network, randomly sample batches from the experience pool. Random sample batches from the experience pool cause the deep network to learn from various past experiences and keep the network from only focusing on what it is immediately doing. To generate the target *Q* values, the *deep Q-network* is adopting a second network called target network.

The loss for each action in the training process is computed. Using one network will result in falling into feedback loops between the target and expected values for both estimates of *Q* values and target *Q* values. Therefore, the weights of the target network are set and regularly adjusted to stabilize the training.

3.2. SDN and DRL

SDN architecture as an evolution of network technology has emerged and aimed to create networks with more flexibility and better management with lower complexity. The main idea of SDN is decoupling the control plane from the data plane. SDN enables better programmability, agility, and flexibility in the network and allows to manage network centrally that keeps a global view of the network [3]

RL and DRL or deep reinforcement learning as two powerful machine learning technics widely apply in emerging communication networks technologies such as SDN [17]. In the RL, an agent acts in an environment to maximize reward. The RL agent acts based on collected information by the centralized SDN controller. The RL agent also learns based on the feedback of the network states and continually updates its policy. Centralized network management with an SDN controller has the following advantages:

- The controller can disseminate optimal global information in the form of actions and policies throughout the network.
- The agent can act as an interface between the IoT and smart city network and the RL algorithm.
- There is less delay in communicating with the central controller system.

4. RESOURCE ALLOCATION USING DRL IN SDN-BASED IOT

4.1 Architecture

The controllers and OpenFlow switches in SDN send the traffic without desirable QoS required for many applications in the domain of IoT and smart city networks. The traffic flows are assigned with the minimum value of the rate. As indicated in the proposed architecture for smart city networks (Fig. 1), several user end devices and application request data.

In the architecture defined in Fig. 1, the number of *M* end-user devices can request the data from the MEC server. When the user wants to access content, or the application needs to process the data, the content is responded from the desired CDN (server) at the edge of the network, which reduces latency. Network nodes are all equipped with internal cache and also have processing capabilities. The SDN controller manages these nodes. These nodes can be virtualized to multiple virtual subnetworks using a hypervisor.

Users can connect to network components such as *BSs*, *APs*, *RSUs* and MEC servers, and content caches. Each user device also accesses one of these CDNs (BS, AP, RSU) based on the defined QoS and SLA requirements.

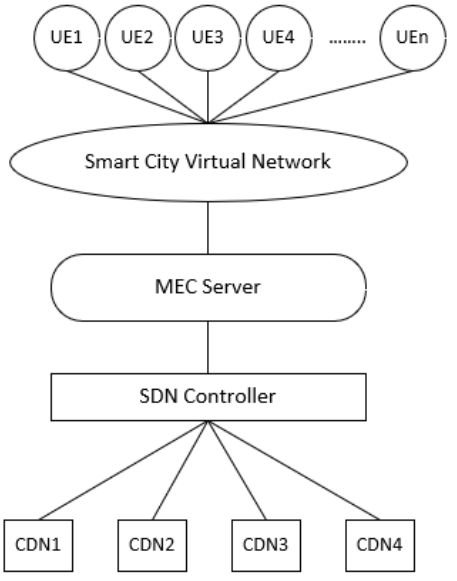

Fig. 1. Proposed architecture.

The SDN controller selects from the available CDNs (servers) according to the QoS parameters and the server's cost that has the lowest cost and the highest QoS. The centralized SDN controller is responsible for collecting states with a global view of the network. These states are obtained from periodic queries from each of the MEC hosts, CDNs, and servers to maintain a global view.

4.2. Reinforcement learning (RL) framework

One of the main pillars of the RL problem is how the environment is represented, or in other words, what the algorithm observes. This observation can be done in different ways. Switch statistics, application flows, and additional information can be used to generate observations. Ideally, the central controller should have complete information about all traffic flows to infer traffic correlations. This may not be possible for a variety of reasons, including security issues or hardware limitations. The statistics of network switches are collected and stored in a *p*f* traffic matrix. This matrix models the assumed network as *p* ports and *f* network features. A query with a switch can get various information such as *active flows*, the number of *lost packets*, *RTT*, and *port usage*.

The obtained states are sent to the agent of the reinforcement learning algorithm. This agent can be an agent of the DQN algorithm. The optimal policy for setting the right resource for a particular user device is determined from the algorithm feedback. After getting the action, a notification is sent to the user's device stating which of the available CDNs it can connect to.

In each RL algorithm, a *goal* is defined, and the agent is trained by *trial and error* to achieve that goal. One of the most essential parts of an RL algorithm is the definition of the reward function. We add to the reward if we receive feedback and evidence of the action being acceptable or desirable, and for actions that are not good, a punishment is subtracted from the reward. The cumulated of the rewards obtained is the goal of the RL algorithm. The objective of the RL agent is to maximize the reward, which it does by learning an optimal set of actions.

All system-specific information is abstracted away from the agent, providing flexibility in obtaining data and modeling it. *OpenAIGym* is an RL standard evaluation platform that aims to expedite the reproduction and evaluation of the performance of an algorithm in different areas.

4.3. Problem formulation

In this problem, we are faced with discrete actions (i.e., choosing between the servers) and continuous and infinite state-space, so in this research, we have used the *Deep Q-Network* algorithm to find *Q* values. *Deep Q-Network* is one of the popular reinforcement algorithms for estimating *Q* values in a system modeled with MDP. By choosing an action, and sending traffic and delivery, QoS parameters such as throughput and delay are measured.

In this algorithm, the agent is not directly guided to what to do. It tries to recognize the value of each permitted action in each environment's state by trial and error. It may also use exploration and exploitation. The algorithm finds a model for appropriate behavior and maximizes rewards. Reinforcement learning behavior is based on the reward hypothesis [22] in which all goals are defined by maximizing the mathematical expectation of cumulative reward.

RL is a step-by-step decision-making process that seeks to maximize reward in each step but may choose actions in the next step that do not necessarily have immediate and large rewards, but the goal is to maximize cumulative rewards.

After implementing the reinforcement learning algorithm, it trains with repeated episodes. To implement this algorithm, we have used a deep neural network with a dense layer with a linear activity function. To solve each RL problem, we must specify the values for the states, reward, and set of actions. To run the algorithm in the training phase, we consider several episodes. In the algorithm *States*: Takes observations or state of the environment, *Reward*: Earns a numerical reward by following an action, *Episod_done*: Indicates the successful end of an episode (*Episod_done* value determines the completion of each episode). In the following, the details of the algorithm are described:

<u>*States:*</u> We consider the experience at any point in time as a state:

1. *current_bw:* The amount of bandwidth available for each server.
2. *min_bw:* Minimum bandwidth for each server (minimum SLA bandwidth allowed).
3. *server_id*: The ID of each server that responded to the request.
4. *current_delay:* The amount of current latency for each server.
5. *max_delay:* Maximum delay (maximum SLA delay allowed).

The above parameters are measured at each step when the agent performs an action.

Actions: Here, the action $a \in A = \{0, 1, 2\}$ as one of three server *s1*, *s2*,*s3* could be choosing between the *CDN* servers.

Reward: The most essential part of designing an RL algorithm is the design of the reward function. The design method of the reward function strongly affects the performance of the RL algorithm. The goal here is to maximize *the bandwidth* usage or minimize *the delay*, assuming that each server's CAPEX differs.

To define a reward function for the problem, the following points should be considered:

1- The agent should avoid choosing a server with a higher cost, so choosing a server with a lower cost, a positive reward, and selecting a server with a higher cost, a negative reward are calculated.

2- The service level agreement or SLA must be maintained at all costs. In the other words, the bandwidth obtained must be higher than the service level specified in the agreement. In the design of rewards, when the bandwidth falls below the values specified in the service level agreement, a severe penalty should be considered. Also, in the design of rewards, severe penalties should be considered when increasing the delay above the SLA values.

3- Regardless of any choice made, the goal is to deliver the data in any way, so a positive reward is considered in each successful delivery.

To obtain the reward, QoS evaluating functions and other factors that affect the reward are calculated as follows:

$$R_d^t = \begin{cases} \frac{max\_delay - current\_delay}{max\_delay} \\ , current\_delay < max\_delay \\ delay\_sla\_penalty \\ current\_delay > max\_delay \end{cases}, \quad (1)$$

$$R_{bw}^t = \begin{cases} 1 - \left(\frac{current\_bw - min\_bw}{min\_bw}\right) \\ , current\_bw > min\_bw \\ bw\_sla\_penalty \\ , current\_bw < min\_bw \end{cases} \quad (2)$$

$$Reward = \alpha_1.(R_d^t) + \alpha_2.(R_{bw}^t) + \alpha_3.(R_{episode}^t) + \alpha_4.(R_{costi}^t) \quad (3)$$

Where, SLA constrains violation (2, 4) punished and by α_1 to α_4 coefficients, the impacts of reward factors for *the delay*, *bw*, *successful episodes*, and *cost* respectively could be tuned. We set $\alpha_1 = \alpha_2 = 1/2$ and $\alpha_3 = \alpha_4 = 1$ and $R_{costi}^t = i$, $i = 1,2,3$.

Episodes: In an RL algorithm, we can consider a final execution limit, so this experience ends at some point in the execution. If we do not consider this final limit, we can continue to run the algorithm indefinitely. Of course, an episode may be stopped due to an error. For example, when the SLA is not satisfied several times, it decides to end the episode, in which case the agent restarts.

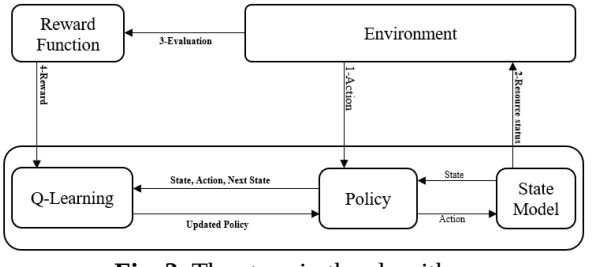

Fig. 2. The steps in the algorithm.

As indicated in Fig. 2, the proposed algorithm has five main steps. The agent, by taking each action could cause to increase or decrease the amounts of QoS parameters. By monitoring and evaluating QoS parameters, agent gets informed by a reward and based on, the agent adjusts its policy (*Q-values*).

5. EXPERIMENTAL RESULTS

In this section, we are going to show the result of the evaluation of the proposed architecture. We use Mininet [23] to implement SDN based environment.

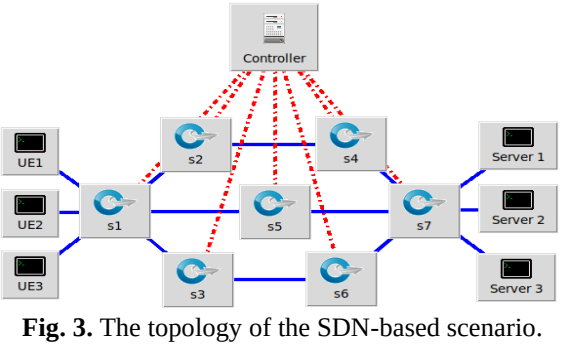

Fig. 3. The topology of the SDN-based scenario.

The configuration setting is shown in TAable 1. To evaluate the performance of the proposed algorithm, we compare simulation results with the random method, Round-robin (RR) algorithm [24] in which resource allocates in turn to the UEs and each UE cannot be served in two successive periods and the work in [5].

Table 1. Configuration setting of simulated network.

OS	Ubuntu 18.04
Environment	Mininet 2.3.0d6
SDN-controller	OpenFlow 1.3
TCP window size	85.3 Kbytes (standard)
UDP buffer size	208 Kbytes
ICMP (ping) packet size	1500 bytes (1472+20+8, data, IP header, ICMP header)
Number of Controllers	1
Number of OpenFlow switches	7
Number of hosts	6
Time (seconds) to listen for new traffic connections	1
Interpacket gap (IPG)	280 us
Link Bandwidth	10 Mbps
SDN Controller Framework	Ryu
Language	Python 3.6
Reward decay (γ)	0.95
Learning Rate(α)	0.001
Size of the Memory buffer	1000000
Size of mini-batch	20
The maximum value of ε	0.9
Number of episodes	500
Number of Timestep Per episode	100
Number of hidden layers	2
Number of nodes per layer	24

First, we create a topology based on the proposed architecture, as shown in Fig. 2, and the designed algorithm implemented on the controller.

Using modules that can be used in hosts, parameters such as *transmission throughput*, *jitter*, *latency*, and *packet loss* can also be achieved. To access this information, a Linux operating system API called *Netlink* [25], [26] can be used to update the status matrix. Also, *sFlow* [27], [28] from the infrastructure of SDN that consists of network devices such as switches and router could collect network status information and send this information to management application.

By using *OpenAIGym* [29], we can create a playground for the agent to try and learn the optimal behavior in the environment. *OpenAIGym* acts as an interface between the DRL agent and the smart network environment.

To train the network, we use a dataset called MQTTset [30], as it is captured from the MQTT protocol from IoT sensors. It contains 11,915,716 network packets. For the training phase of our algorithm, we consider 500 episodes. As shown in Fig. 4, the average cumulated reward obtained from the experiment increased from an episode near 200. Some fluctuations are seen in reward in Fig. 4. These fluctuations are because of the maximum value of ε, which is used greedily for action selection during the training phase. We use 0.9, which means that the agent with a probability of 0.9 chooses the best action, and with a probability of 0.1 may choose a bad action at each iteration that causes a punishment or reduced reward.

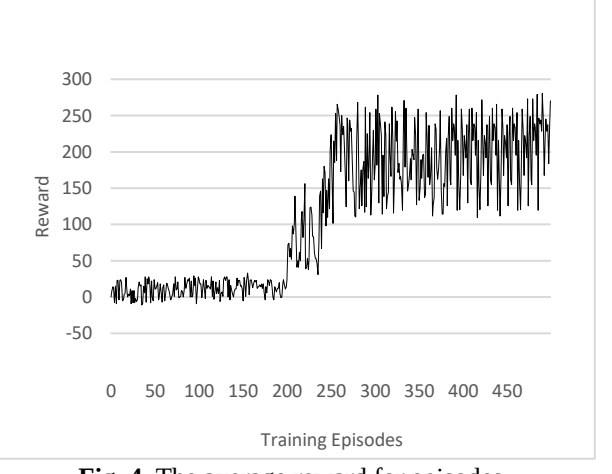

Fig. 4. The average reward for episodes.

To evaluate the proposed algorithm we generate traffic with the *iPerf* [31] with a constant rate of 100, 300, 500, 700, and 900 packets per second. When a client request is received, the agent should decide which server is best in terms of QoS parameters. In other words, the clients send requests and the servers are selected by the agent. To make the simulation closer to real scenarios, we generate background traffic between other hosts at a constant rate.

The performance of the proposed algorithm is compared with Round Robin and Random methods and the [5] method, in terms of throughput and delay. Fig. 5 indicates that the proposed method outperforms two other technics and in all incoming traffic, from 100 to 900 packets per second is higher than SLA (60 % total bandwidth). The throughput increases while increasing traffic to 700 packets per second but throughput in 900 packets per second in random and round-robin methods decreased except in the proposed and the [5] method. The slight difference between the proposed method and the [5] is due to fluctuations of reward.

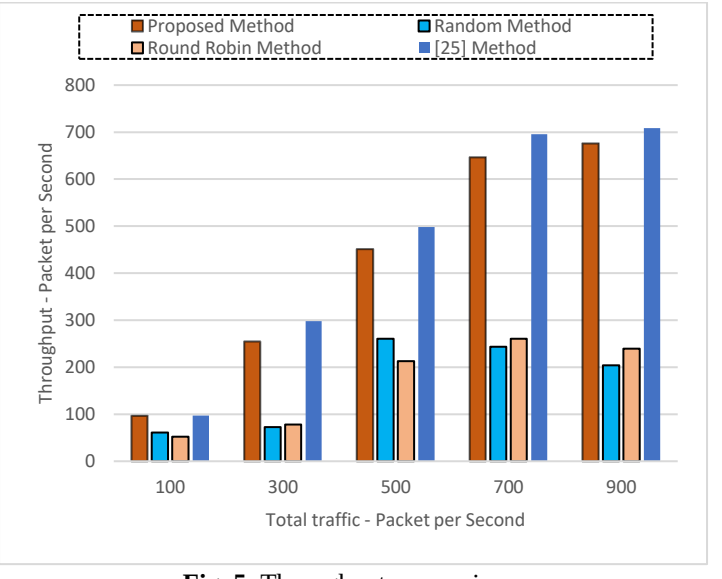

Fig. 5. Throughput comparison.

Fig. 6 shows delay comparisons between methods. It can be seen that the Random and Round Robin methods have a high delay and the proposed method and [5] have an approximately identical delay. As observed from 700 packets per second, Random and Round Robin methods suffered a high delay.

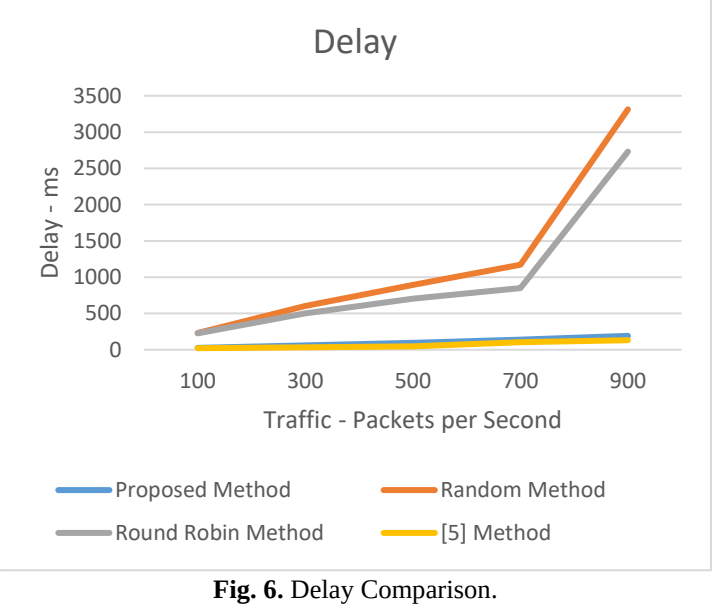

Fig. 6. Delay Comparison.

6. CONCLUSION AND FUTURE WORK

IoT is an emerging technology that consists of millions of heterogeneous devices and creates smart environments such as smart cities, etc. One of the most important aspects of these environments is the traffic and resource management framework to meet the QoS requirements of the user end device requests. However, matching the requests to the resources while satisfying QoS for IoT traffic is a challenging task considering the large volume of traffics from a massive number of devices and resource constraints.

To deal with the challenge, we designed an SDN-based solution to maximize QoS inspired by the success of Deep Reinforcement Learning to handle complex problems. The proposed framework was implemented in Mininet as an SDN testbed. Furthermore, the performance of the proposed framework is validated in comparison to the Random and Round Robin methods and [5] method. Simulation results show that the proposed algorithm achieves better performance

compared to similar methods. A number of future work can be considered, for examples, as our proposed framework considers single SDN-controller, it would be interesting to study IoT QoS improvement in the multi controller SDN framework. In addition, the use of other AI and machine learning methods to predict the workload of controllers can be considered in future work.

REFERENCES

[1] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco." [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed: 23-Feb-2020].

[2] **"Software-Defined Networking (SDN) Definition - Open Networking Foundation."** [Online]. Available: https://opennetworking.org/sdn-definition/. [Accessed: 07-Dec-2020].

[3] **"Software-Defined Networking (SDN) Definition - Open Networking Foundation."** [Online]. Available: https://www.opennetworking.org/sdn-definition/?nab=1&utm_referrer=https%3A%2F%2Fsdn.itrc.ac.ir%2F%3Fq%3Dfa%2Fcontent%2Fsdn. [Accessed: 08-Jan-2020].

[4] **"Examples of Network Programmability and SDN > Introduction to Controller-Based Networking | Cisco Press."** [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=2995354&seqNum=3. [Accessed: 07-Dec-2020].

[5] A. Montazerolghaem and M. H. Yaghmaee, **"Load-Balanced and QoS-Aware Software-Defined Internet of Things,"** *IEEE Internet Things J.*, Vol. 7, No. 4, pp. 3323–3337, Apr. 2020.

[6] S. Din, M. M. Rathore, A. Ahmad, A. Paul, and M. Khan, **"SDIoT: Software Defined Internet of Thing to Analyze Big Data in Smart Cities,"** in *Proceedings - 2017 IEEE 42nd Conference on Local Computer Networks Workshops, LCN Workshops 2017*, 2017, pp. 175–182.

[7] H. E.-K. S. G.-B. Mohammadreza Moslehi, **"Improving QoS using Software-Defined Networking for Smart City (SDSC),"** *Int. J. Futur. Gener. Commun. Netw.*, Vol. 13, No. 3, pp. 2757–2767–2757–2767, Aug. 2020.

[8] S. K. Tayyaba, M. A. Shah, O. A. Khan, and A. W. Ahmed, **"Software Defined Network (SDN) Based Internet of Things (IoT),"** in *Proceedings of the International Conference on Future Networks and Distributed Systems - ICFNDS '17*, 2017, pp. 1–8.

[9] D. Sinh, L. V. Le, B. S. P. Lin, and L. P. Tung, **"SDN/NFV - A new approach of deploying network infrastructure for IoT,"** in *2018 27th Wireless and Optical Communication Conference, WOCC 2018*, 2018, pp. 1–5.

[10] Y. Njah, C. Pham, and M. Cheriet, **"Service and Resource Aware Flow Management Scheme for an SDN-Based Smart Digital Campus Environment,"** *IEEE Access*, Vol. 8, pp. 119635–119653, 2020.

[11] X. Guo, H. Lin, Z. Li, and M. Peng, **"Deep-Reinforcement-Learning-Based QoS-Aware Secure Routing for SDN-IoT,"** *IEEE Internet Things J.*, Vol. 7, No. 7, pp. 6242–6251, Dec. 2019.

[12] K. S. Bhandari, I. H. Ra, and G. Cho, **"Multi-Topology Based QoS-Differentiation in RPL for Internet of Things Applications,"** *IEEE Access*, vol. 8, pp. 96686–96705, 2020.

[13] G. C. Deng and K. Wang, "An Application-aware QoS Routing Algorithm for SDN-based IoT Networking," in *Proceedings - IEEE Symposium on Computers and Communications*, pp. 186–191, 2018.

[14] K. Streit, C. Schmitt, and C. Giannelli, **"SDN-Based Regulated Flow Routing in MANETs,"** 2020, pp. 73–80.

[15] N. Hu, F. Luan, X. Tian, and C. Wu, "**A Novel SDN-Based Application-Awareness Mechanism by Using Deep Learning,"** *IEEE Access*, Vol. 8, pp. 160921–160930, 2020.

[16] M. Naeem, S. T. H. Rizvi, and A. Coronato, **"A Gentle Introduction to Reinforcement Learning and its Application in Different Fields,"** *IEEE Access*, Vol. 8, pp. 209320–209344, Nov. 2020.

[17] Yichen Qian, Jun Wu, Rui Wang, Fusheng Zhu, and Wei Zhang, **"Survey on Reinforcement Learning Applications in Communication Networks,"** *J. Commun. Inf. Networks*, Vol. 4, No. 2, pp. 30–39, 2019.

[18] A. Alharin, T.-N. Doan, and M. Sartipi, "Reinforcement Learning Interpretation Methods: A Survey," *IEEE Access*, Vol. 8, pp. 171058–171077, Sep. 2020.

[19] V. Mnih *et al.*, **"Playing Atari with Deep Reinforcement Learning,"** Dec. 2013.

[20] V. Mnih *et al.*, **"Human-level control through deep reinforcement learning.,"** *Nature*, Vol. 518, No. 7540, pp. 529–33, Feb. 2015.

[21] Y. He, N. Zhao, and H. Yin, **"Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach,"** *IEEE Trans. Veh. Technol.*, Vol. 67, No. 1, pp. 44–55, Jan. 2018.

[22] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction Second edition, in progress."

[23] "Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet." [Online]. Available: http://mininet.org/. [Accessed: 22-Mar-2020].

[24] M. Runsungnoen and T. Anusas-amornkul, "**Round Robin Scheduling Based on Remaining Time and Median (RR_RT&M) for Cloud Computing,**" in *Smart Innovation, Systems and Technologies*, Vol. 165, pp. 21–29, 2020.

[25] "RFC 3549 - Linux Netlink as an IP Services Protocol." [Online]. Available: https://datatracker.ietf.org/doc/rfc3549/. [Accessed: 11-Dec-2020].

[26] A. Kuznetsov, J. Salim, A. Kleen, and H. Khosravi, **"Linux Netlink as an IP Services Protocol.**"

[27] "sFlow.org - Making the Network Visible." [Online]. Available: https://sflow.org/index.php. [Accessed: 26-Nov-2020].

[28] "sFlow-RT." [Online]. Available: https://sflow-rt.com/. [Accessed: 17-Dec-2020].
[29] "Gym." [Online]. Available: https://gym.openai.com/. [Accessed: 11-Nov-2020].
[30] "MQTTset | Kaggle." [Online]. Available: https://www.kaggle.com/cnrieiit/mqttset. [Accessed: 26-Dec-2020].
[31] "esnet/iperf: iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool." [Online]. Available: https://github.com/esnet/iperf. [Accessed: 05-Dec-2020].