DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

Intelligent Parameter Tuning using Deep Q-network in Adaptive Queue Management Systems

Ayman Basheer Yousif[1], Hassan Jaleel Hassan[2], Ghaida Muttasher[3]
[1]*Department of Computer Technology Engineering, Al-Esraa University College, Baghdad, Iraq*
[2,3]*Computer Engineering Department, University of Technology, Baghdad, Iraq*
[1]*ayman.bashir@esraa.edu.iq*, [2]*60012@uotechnology.edu.iq*, [3]*drghaida7@gmail.com*

Abstract— *Network traffic has risen in recent years to the point that it is obviously and astonishingly in 2020, with the increase predicted to double in the following days. Up to 23 Teraa bit every month is an incredible amount. The Active Queue Management (AQM) algorithm is one of the most significant study areas in network congestion control; nevertheless, new self-learning network management algorithms are needed on nodes to cope with the huge quantity of traffic and minimize queuing latency, used reinforcement learning for automatic adaptive parameter with the AQM algorithm for effective network management, and present a novel AQM algorithm that focuses on deep reinforcement learning to deal with latency and the trade-off between queuing delay and throughput; choose Deep Q-Network (DQN) as the foundation for our scheme and equate it with Random Early Detection (RED) Results based on Network simulation (NS3) simulation suggest that the DQN algorithm has good and better results were obtained from RED, where the difference reached a drop rate of 2%, and this percentage is considered good, in addition to the percentage of throughput and the packet transfer rate of 3% is better in the proposed algorithm.*

Index Terms— *Adaptive Queue Management, Network congestion, Network Traffic Management, Deep Q-Network, Reinforcement Learning.*

I. INTRODUCTION

It is vital to avoid excessive packet failure rates over the Internet. If a package is lost before it reaches its intended destination, it loses all of the energy it used in transit. In severe circumstances, this condition will cause congestion to collapse [1]. Active Queue Management (AQM) has arisen as the sophisticated network control tool for selectively sending and receiving packets for effective management when it comes to queuing networks [2]. Unlike passive queue such as First-In-First-Out (FIFO), AQM implements the smart drop of network packets to minimize network congestion by adjusting the parameters of AQM, such as the possibility of packet-drop adaptation to the environment. Online Innovation Task Force (IETF) common usage of and approved AQM schemes [3]. Machine Learning (ML) has developed into an essential technology in the industries and our quality of life. In fact, Deep Learning (DL) appears to outperform numerous ML methods in diverse fields such as effective data coding and modeling artifacts (unsupervised learning), as well as usual classification and prediction employment (supervised learning)[4]. DL has also been extended to Reinforcement Learning (RL), which is an ML type that looks at how the program agent decides to take appropriate action on such states to get the highest total reward [5]. In this paper suggest an AQM (RED)

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 *ISSN (Online) 2617-3352*

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

implemented deep reinforcement learning framework for effective network control and research the trade-off between queuing latency and throughput. Our program is built using Deep Q-Network (DQN). Like key Q-network and goal network equipped with experience replay [6]. Based on the current state consisting of dequeue_rate, enqueue_rate, drop rate and avg_queue_lengh it chooses a packet drop or non-drop operation at the packet departure point. Once an event is chosen, a compensation is determined on the basis of the many factors to be explained in the following [7].

II. RELATED WORK

Bouacida et al. [14] introduced Learn Queue AQM algorithm focused on wireless networking reinforcement learning. Through dynamically modifying a buffer size utilizing Q-Learning in a specified period, they change the Q-table and refine the Q-function strategy, however check their method for just two and three scenarios deployed.

Bisoy et al. in [15] proposed an AQM scheme focused on a shallow neural network with one secret layer consisting of three neurons to resolve the non-linearity of the networking framework and the queuing latency, but their research did not deal with the trade-off between throughput and delay performance .

Reinforcement Learning-Queuing Delay Limitation (RL-QDL) AQM algorithm suggested in by Vucevic et al. [8]. RL agents provide topology details from the bandwidth broker that handles resource management and QoS provisioning based on what QoS requirements are met in egress routers (ERs). This supports Class-Based Queuing (CBQ) by endorsing three separate classes: Expedited Forwarding (EF), guaranteed forwarding (AF), and Best Effort (BE) trac to provide end-to - end QoS to customers with specific service types. With respect to network scheduling algorithms, Chen et al.[9] suggested automated computation offloading strategy focused on Deep Reinforcemnt Laerning (DRL) by implementing a double DQN on the edge node. Comparing with standard algorithms, their solution implied the optimum tradeoff between task latency and drop.

Xu et al. [10] applied DRL to network trace engineering in by implementing actor-critical approach with a replay of prioritized experiences. Authors contrasted their algorithm with the commonly used baseline solutions, such as Shortest Path (SP), Load Balance (LB), and Network Utility Maximization (NUM), and checked that their model performs better than specified baseline solutions.

Minus Kim B.Eng. [6] proposed the design of Deep Reinforcement Learning based Active Queue Management. As a baseline model of the design, selected Deep Q-Network since the state transition in networking is discrete and it is able to be expressed as a finite Markov Decision Process which is the fundamental principle of reinforcement learning. Applying deep learning framework deployed the AQM scheme at the interface of the fog/edge device connected to the cloud gateway, and our proposed scheme achieved substantial performances such as low queuing delay and jitter on the stochastic IoT environment simultaneously maintaining good throughput comparing with widely used AQM schemes.

III. NETWORK SIMULATION-3 SIMULATION

Ns-3 simulator is a discrete-event network simulator targeted primarily for research and educational use. The ns-3 project, started in 2006, is an open-source project developing *ns-3*.[6]

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

Ns-3 has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, ns-3 provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use ns-3 include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work. Users will note that the available model set in ns-3 focuses on modeling how Internet protocols and networks work, but ns-3 is not limited to Internet systems; several users are using ns-3 to model non-Internet-based systems. [6]

IV. PROPOSED SYSTEM

First give an explanation of Deep Q-Network (DQN) which is the baseline of our system. Then describe the design of our system in terms of the state, action, and reward, and the algorithm is followed by focusing on how to give a reward to the agent in detail.

A. Deep Q-Network

The reinforced learning is achieved through the interaction of the agent with the environment in sequential time steps (t = 1, 2, 3, …..) randomly. At each time step, the agent tests an action out of set of actions At ∈ A (s) that come from the state St ∈ S. After the A(t) action is tested, the agent receives a reward, and a new state is assigned S(t+1). Through repeating this method (operation), each notion in the path will be suitable to express MDP (Markov Decision Process) Process shown in *Fig 1*, as following:

(s1, a1, r1), (s2, a2, r2)…(sn, an, rn) which s is state, a is action, r is reward.

Instead of transforming MDP , the Markov characteristic, which has no random memory. In the random operation, the future stats depend on the current state not on the complete path of the actions [5].

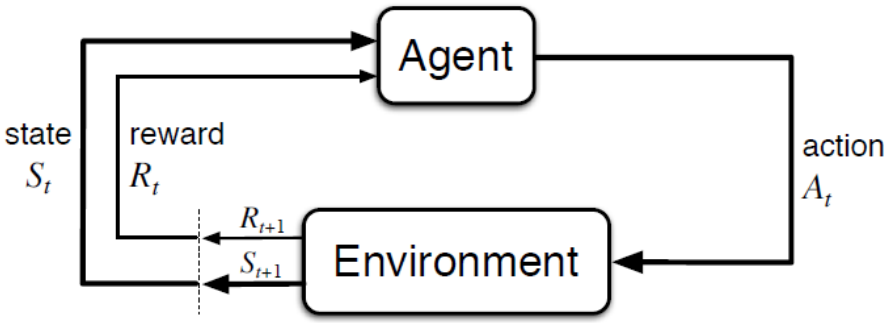

FIG. 1. ILLUSTRATES THE MDP OPERATION.

In the network waiting queue, packets will enter the waiting queue, and the agent will be able to monitor the current state of that queue to decide whether to drop the packet or not. It also monitors the next stack and can obtain Reward. RL can be added as a representation of the sequence in each step (t). There are two important features of RL, they are searching for the experiment and error and searching for the Reward [5].

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

B. DQN based AQM System Design

In this section introduce the state, action and reward function design for our AQM algorithm based on the DQN.

Step1: Process of Selecting Action.

With respect to the state of RL, consider four elements: dequeue rate, enqueue rate, drop rate and Avrg_Queue_Len. At each time step t, state st is defined as st = {dequeue rate, enqueue rate, drop rate and Avrg_Queue_Len} which is an input of multilayer perceptron (MLP) consisting of three hidden layers of 16-32-16 neurons for each layer. For selecting an action, main Q-network is used and it returns two probabilities as an output (drop/non-drop probability). utilize the explore/exploit approach to discover a better action on a given state. This implies that the agent either performs an action based on its own selection (exploit) or chooses a random action uniformly based on a specific probability (explore). For the explore/exploit strategy starting from a highly random probability of action for the explore/exploit strategy. The exploring probability is set at 90 percent based on the round of the episode at the first episode of the network simulation, and it diminishes to 0 percent through the episode. Section 3.2 explains the selection process for an action.

Step2: Reward Engineering

During the interval Tint, the RL agent waits for the next state st+1 after performing an action. A reward function evaluates the chosen action. The most essential aspect of constructing the reward function is to optimize the trade-off between queuing time and drop-rate, as well as to prevent endless or non-drop packet states. refer learn Queue's reward function as a baseline

$$Reward = clip\left((\square * delay\_rerword) + ((1 - \square) * enqueue\_reward), -1, 1\right) \quad (1)$$

There are two main components for a reward: delay_rerword and enqueue_reward for queuing delay and packet drop-rate respectively, and the ϒ is scaling factor used to balance between delay_rerword in Eq. 2. and enqueue_reward

delay_rerword is

$$delay\_rerword = desiredQueueDelay - current\_delay \quad (2)$$

when desiredQueueDelay is Expected delay and the default is 0, and the current_delay is in Eq. 3.

$$current\_delay = Avrg\_Queue\_Len \,/\, dequeue\_Rate \quad (3)$$

where = Avrg_Queue_Len is current queue length in bytes, and dequeue_Rate in Eq. 4. is the average dequeue rate per sec

$$dequeue\_Rate = dequeueCounter \,/\, timeDiffrence \quad (4)$$

where dequeueCounter is the Packet numbers will not enter the queue, where timeDiffrence is set by user. enqueue_reward in Eq. 5. is defined as:

$$enqueue\_reward = (min\_delay - \_desiredQueueDelay) * enqueue\_rate \quad (5)$$

where min_delay in Eq. 6. is define as:

$$min\_delay = avrgQueueLenInBytes \,/\, band\_in\_bytes \quad (6)$$

where avrgQueueLenInBytes is the current queue length of the device in bytes, band is the physical bandwidth (data rate) of Peer-to-Peer (P2P) link connected to the device, and enqueue_rate in Eq. 7. is defined as:

$$enqueue\_rate = enqueueCounter/(\_enqueueCounter + dropCounter) \quad (7)$$

where enqueueCounter is the number of enqueued packets and dropCounter is the number of dropped packets

Step3: Training Process.

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

Since using DQN as our model, at each time stage t the agent stores an experience et = (st, at, rt, st+1) in tuple format to the replay memory. When the number of replay memory experiences approaches the mini-batch size, the agent randomly selects samples of the memory experiences in a consistent manner, In the first episode, initialize the weights of the initializer MLPs, which assigns the weights of the layers from a Gaussian distribution [12], and minimize the loss using the optimizer Adam [13] to train the model.

Fig. 2 shows the flowchart of DQN based AQM training process. In the flowchart, tcurr is current time step t, and C is target update step to update the target network periodically.

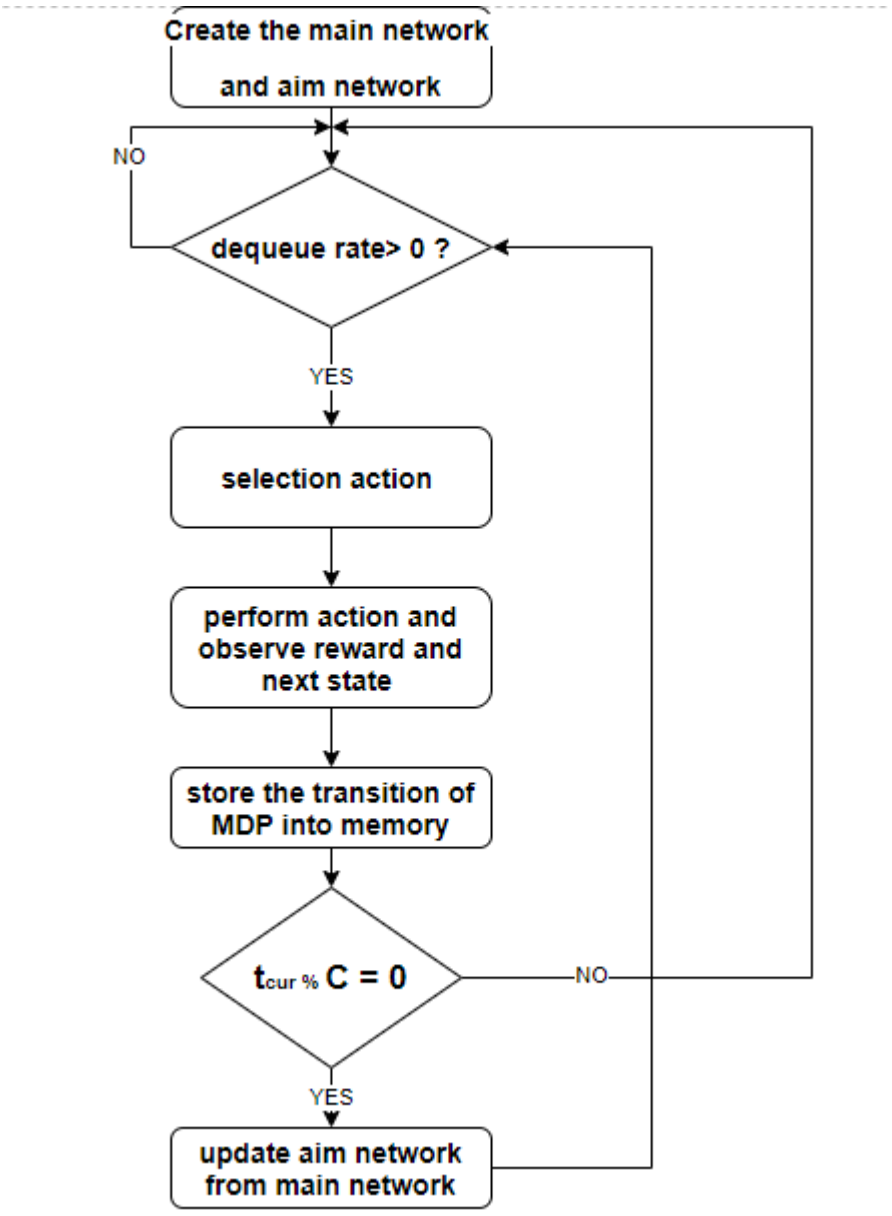

FIG. 2. FLOWCHART OF DQN BASED AQM TRAINING.

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

V. SIMULATION ANALYSIS

This section validates the validity and performance of the NS3 simulation experiment of the designed DQN algorithm, the simulation uses the typical single-bottleneck network topology as shown in *Fig. 3* The network has n senders (S_1~S_n), receivers (r_1 ~ r_n), and 1 router (n_2). The bandwidth and delay between each sender (n_1) and (n2) is 100Mbps and 0.1ms, and the bandwidth and delay between each receiver and (n_2) are 100Mbps and 5ms too. To compare, and analyze the RED algorithm and DQN algorithm's queue length, throughput, delay, and packet loss rate under changing load, respectively. The performance of the algorithm, the simulation time is 100 seconds. Table I shows the queue length and standard deviation of the RED algorithm and the DQN algorithm. As can be seen from Table I the average queue length of the RED algorithm is larger than that of the DQN algorithm. So, the DQN algorithm reduces the drop probability, and reduces the delay.

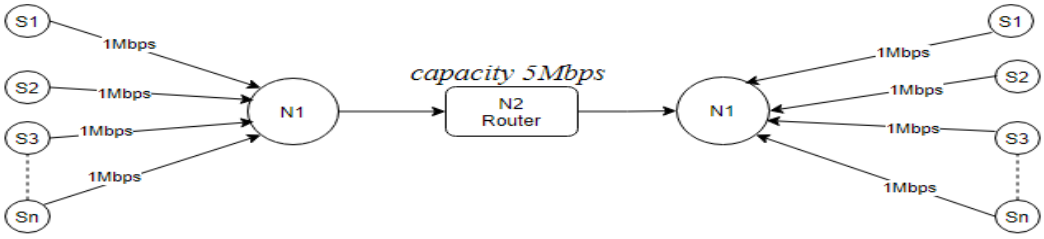

FIG. 3. SIMULATED NETWORK STRUCTURE.

VI. RESULT AND DISCUSSION

1- For Low load (number of sessions (n) = 60)

The ability of two algorithms (DQN, RED) to keep queue length near to desired value is shown in *Fig. 4* a,b. Can be observed in Table I, with the DQN method having a little advantage. the number N factor is 60 indicates that by increasing the number of training classes, higher outcomes may be reached.

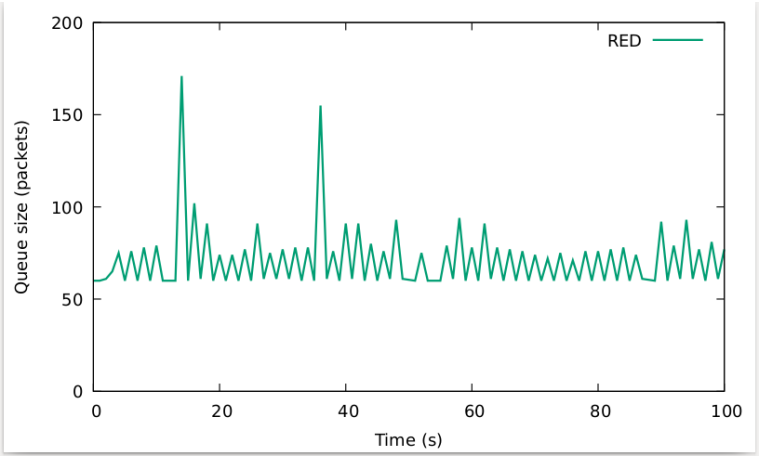

FIG. 4. A. RED ALGORITHM UNDER LOW LOAD.

Received 15/October/2021; Accepted 13/December/2021

ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

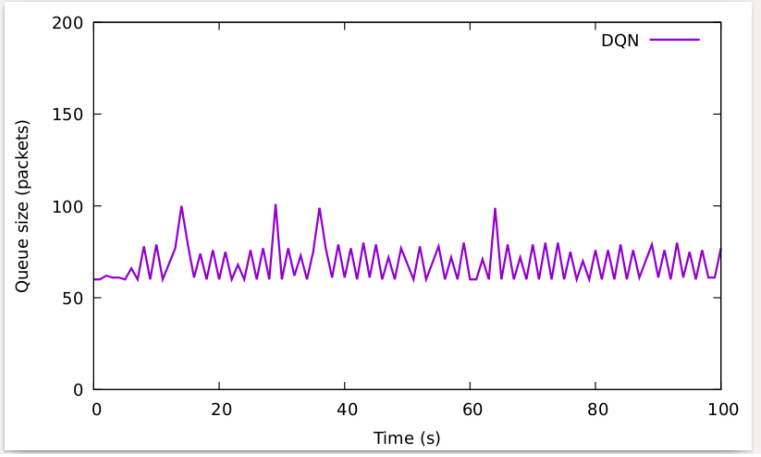

FIG. 4. B. DQN ALGORITHM UNDER LOW LOAD.

TABLE I. MEAN AND STANDARD DEVIATION OF THE QUEUE LENGTH UNDER LOW LOAD

Scenario	Mean (packet)	Standard Deviation (packet)
RED	68.958	10.33
DQN	71.319	17.37

2- For Mid load (number of session (n) = 80)

Table II shows the ability of two algorithms (DQN and RED) to keep queue lengths close to desired values shown in *Fig 5* a,b , with the DQN approach having a little edge.

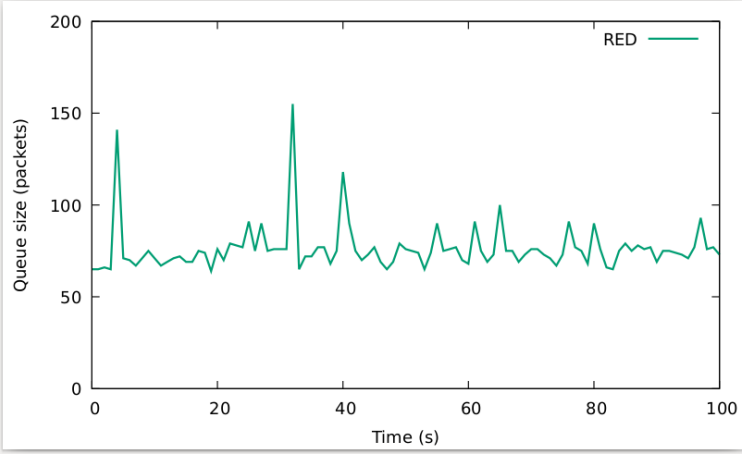

FIG. 5. A. RED ALGORITHM UNDER MID LOAD.

Received 15/October/2021; Accepted 13/December/2021

ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

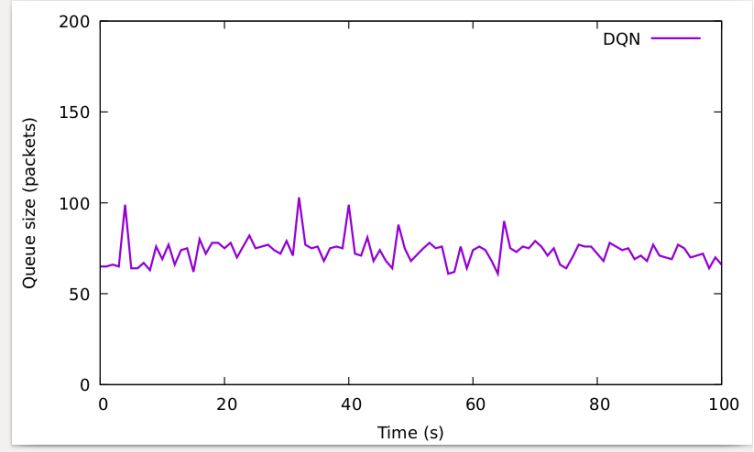

FIG. 5. B. DQN ALGORITHM UNDER MID LOAD.

TABLE II. MEAN AND STANDARD DEVIATION OF THE QUEUE LENGTH UNDER MID LOAD

Scenario	Mean (packet)	Standard Deviation (packet)
RED	74.23	13.187
DQN	76.222	7.322

3- For High load (number of session (n) = 100

When changing the number of sessions to 100, it can be seen the ability of two algorithms (DQN, RED) to keep queue length near to desired value, shown *in Fig. 6* a,b can be observed in Table III, with DQN method having a little advantage.

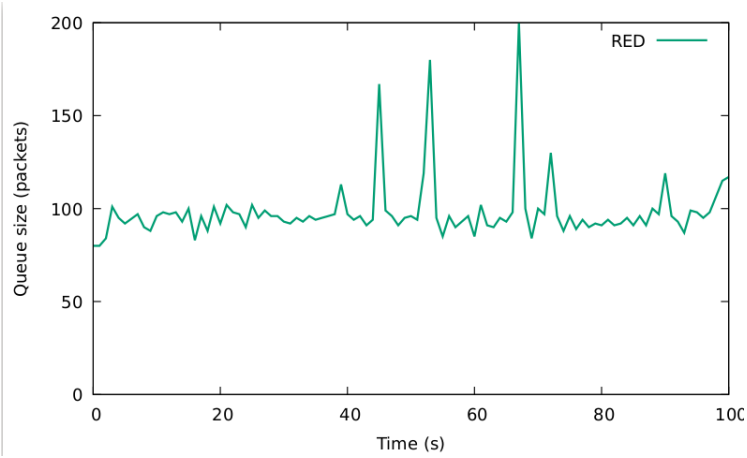

FIG. 6. A. RED ALGORITHM UNDER HIGH LOAD.

Received 15/October/2021; Accepted 13/December/2021

ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

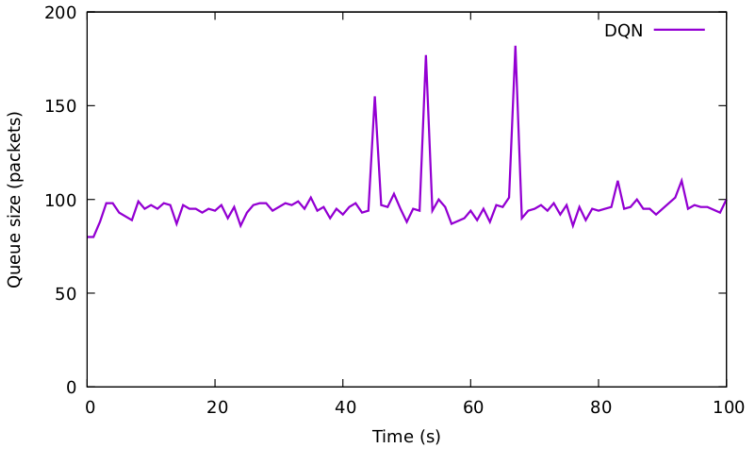

FIG. 6. B. DQN ALGORITHM UNDER HIGH LOAD.

TABLE III. MEAN AND STANDARD DEVIATION OF THE QUEUE LENGTH UNDER HIGH LOAD

Scenario	Mean (packet)	Standard Deviation (packet)
RED	98.204	14.532
DQN	97.214	17.01

TABLE V. COMPARE MEAN PACKET, THROUGHPUT AND DROP RATE BETWEEN RED ALGORITHM AND DQN ALGORITHM UNDER CHANGE LOAD

N	Mean (ckets RED)	Mean packets DQN	Throughput RED	Throughput DQN	Drop RED	Drop DQN
60	68.958	71.319	4.7446	4.7462	11.00%	10.00%
80	74.23	76.222	4.5263	4.5654	12.70%	12.10%
100	98.204	97.214	4.5334	4.5551	16.90%	15.80%

VII. DISCUSSION

The statistics of the throughput, delay, and drop rate of the RED algorithm and DQN algorithm are shown in Table V. It can be found that in the dynamic network environment, the performance of the DQN algorithm is better than that of the RED algorithm.

Through the aforementioned simulations and analysis, the performance of DQN algorithm including throughput, delay, and the drop rate is consistent with the expected results. Although unilateral performance improvement is not too much, the overall performance in different network scenarios is better than the RED algorithm. Therefore, DQN algorithm can improve the sensitive parameters of RED algorithm to a certain extent, so that the DQN algorithm achieves better network performance and can select appropriate parameters deceptively according to different network scenarios (changing the load factor network (N) 60-100).

VIII. CONCLUSIONS

To boost the parameter settings of the RED algorithm, and allow the algorithm to achieve improved network efficiency, the RED algorithm selects the correct parameters according to the learning process. This paper discusses an improved RED algorithm,

Received 15/October/2021; Accepted 13/December/2021
 ISSN (Print) 1811-9212 ISSN (Online) 2617-3352

DOI: https://doi.org/10.33103/uot.ijccce.22.3.6

known as DQN. The Q-learning algorithm is used in this method to pick the highest likelihood parameter for the decrease of packets. The parameters of the RED algorithm are immune to faults and can anticipate complex shifts in network networks. The learning structure gets the optimum Network Quality Management System. This will tailor the highest likelihood of packet drop for the algorithm, ensuring better efficiency of the network while preventing congestion. The simulation tests validate the DQN algorithm's benefits, which can be applied in the network to preserve reliability, minimize latency, Boost throughput, etc. The overall network output of the DQN algorithm is better than that created by the RED algorithm. Throughout our future research, shall find enhancing the quality of the improvement of the act.

REFERENCES

[1] S. Athuraliya, S. Low, V. Li, and Q. Yin. REM Active Queue Management. IEEE Network Magazine, 15(3), May 2001.

[2] 2. N. Kuhn, P. Natarajan, N. Khademi, and D. Ros, "Characterization Guidelines for Active Queue Management (AQM)," in Internet Engineering Task Force (IETF), RFC 7928, July 2016. [Online]. Available: https://tools.ietf.org/html/rfc7928I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[3] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," in Internet Engineering Task Force (IETF), RFC 7567, July 2015. [Online]. Available: https://tools.ietf.org/html/rfc7567.

[4] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep Learning for IoT Big Data and Streaming Analytics: A Survey," IEEE Communications Surveys Tutorials, June 2018. [Online early access]. Available: https://doi.org/10.1109/COMST.2018.2844341.

[5] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. MIT Press, second ed., 2018.

[6] [14] M. Kim, "Deep Reinforcement Learning based Active Queue Management for IoT Networks," no. April, 2019, doi: 10.13140/RG.2.2.24361.65126.

[7] [15] S. Whiteson and others, Adaptive representations for reinforcement learning, vol. 291, Springer, 2010. pp. 7

[8] N. Vucevic, J. Perez-Romero, O. Sallent, and R. Agusti, "Reinforcement Learning for Active Queue Management in Mobile All-IP Networks," in 2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, September 2007. [Online]. Available: https://doi.org/10.1109/PIMRC.2007.4394713.

[9] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized Computation Offloading Performance in Virtual Edge Computing Systems via Deep Reinforcement Learning," IEEE Internet of Things Journal, October 2018. [Online early access]. Available: https://doi.org/10.1109/JIOT.2018.2876279.

[10] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experiencedriven Networking: A Deep Reinforcement Learning based Approach," in IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, April 2018. [Online]. Available: https://doi.org/10.1109/INFOCOM.2018.8485853

[11] N. Bouacida and B. Shihada, "Practical and Dynamic Buffer Sizing using LearnQueue," IEEE Transactions on Mobile Computing, September 2018. [Online early access]. Available: https://doi.org/10.1109/TMC.2018.2868670.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 249–256, March 2010. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in arXiv preprint arXiv:1412.6980v9, January 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[14] N. Bouacida and B. Shihada, "Practical and Dynamic Buffer Sizing using LearnQueue," IEEE Transactions on Mobile Computing, September 2018. [Online early access]. Available: https://doi.org/10.1109/TMC.2018.2868670.

[15] S. K. Bisoy, P. K. Pandey, and B. Pati, "Design of an active queue management technique based on neural networks for congestion control," in 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), December 2017. [Online]. Available: https://doi.org/10.1109/ANTS.2017.8384104.

[16] D. A. Duwaer, "On Deep Reinforcement Learning for Data-Driven Traffic Control," 2016.

Received 15/October/2021; Accepted 13/December/2021

 ISSN (Print) 1811-9212 *ISSN (Online) 2617-3352*