Study of a Smarter AQM Algorithm to Reduce Network Delay

Aya Abdullah Ismael[1], Didem Kıvanç Türeli [2]
[1]*Deparement of Advanced Electronic and Communication Engineering Technology, Istanbul, Turkey, ayaabdullah600@gmail.com, ORCID: 0000-0001-7048-6550*
[2]*Deparement of Advanced Electronic and Communication Engineering Technology, Istanbul, Turkey, aismael3@stu.okan.edu.tr, ORCID: 0000-0001-9981-4829*

The focus of our study was to study the behavior of the smart RED algorithm with parameter adaptation based on a neural network structure. Previously Kim et. al. [2] and Basheer et. al. [1], introduced the use of deep reinforcement learning for active queue management (AQM). This work studies the performance deep reinforcement learning, using a simple topology. Transmitters and receivers communicate and all information is routed over a single bottleneck link. This work studies the effect of changing the bottleneck link bandwidth and bottleneck latency on this algorithm. It is observed that increasing training data size will increase the performance of the algorithm. This paper shows a detailed flowchart of the training process and the specific hardware configuration and also demonstrates results.

Keywords: Neural Network, Testing, Training, DQN.

© 2022 Published by *AIntelia*

1. Introduction

The goal of queue management is to selectively drop packets from the queue of packets to be transmitted. Packets are dropped to reduce latency for all packets in the network, to reduce the probability of time-out for even more packets and to ensure fairness so that a single user does not send so many packets so as to block all other users at intermediate nodes in the network (lock-out phenomenon). There have been many proposed Active Queue Management (AQM) algorithms such as random early detection (RED), Proportional Integral Controller Enhanced (PIE), etc.
The RED algorithm was chosen to teach a neural network model to take action. The model updates the REQ parameters according to the environment by measuring the reward after each action performed on the algorithm.

2. Background and Prior Work

This paper studies a link which bridges multiple communicating nodes on two networks which is overloaded with traffic. The goal is to study the reinforced deep learning based Deep Q-Network (DQN) algorithm and to compare it to the conventional random early detection (RED) management algorithm.

A. Active Queue Management Algorithms

Active queue management (AQM) is used in routers and switches which receive a lot of packets from the upstream network in order to minimize delays and prevent the system from locking up. Packets are selectively dropped to keep traffic moving into the network. The goal is to reduce local congestion in the network and to improve end-to-end latency. Latency is particularly important for real-time algorithms such as voice and video conferencing, and video streaming which are increasing as a proportion of internet traffic. Latency is also particularly important for real time systems which use data to make decisions in new internet of things applications [1].

To regulate and prevent network congestion, it is critical to use resilient active queue management (AQM) algorithms. Minsu Kim *et. al.* [1] have proposed a novel self-learning network management algorithm using deep reinforcement learning for use on fog/edge nodes in an Internet of Things (IoT) network. In their work they compare a Deep Reinforcement Learning (DRL) based active queue management algorithm called the Deep Q-Network (DQN) algorithm [1]. A similar algorithm with some differences in the formulation of the reward function has been implemented by Ma *et. al.* [2]. Szyguła *et. al.* [3] use a Convolutional Neural Network (CNN) to train the parameters of a Proportional Integral (PI$^{\alpha}$) AQM algorithm. Abbas *et. al.* [4] summarize recent work in AQM. AQM algorithms recommended by the IETF for use on the internet include random early detection (RED), Proportional Integral Controller Enhanced (PIE), express congestion notification (ECN), and controlled delay (CoDel) [5].

To study the DQN algorithm we compare it to an existing algorithm, the Random Early Detection (RED) algorithm which is widely used today.

B. Deep Reinforcement Learning

This paper studies a new approach to queue management using reinforcement learning. To design a reinforcement learning algorithm the system must be described in terms of its state, the actions that can be taken, and rewards. In each state, the set of actions an agent can take are ranked according to the rewards that they can return. The system updates the expected reward of its actions as it operates. The actions taken are chosen so as to maximize the reward of the system but occasionally the agent takes suboptimal action to explore the possibility of getting more reward.

In deep reinforcement learning the neural network is trained using the reward functions to rank possible actions and suggest the best course of action in any state. Deep reinforcement learning has been applied to many problems in communications and networking [8,9].

The reward in a queueing system will be the throughput (percentage of offered load which is successfully transmitted) and the latency of received packets. In general, it is true that as the length of a queue grows, the latency incurred by all packets will increase. Dropping packets will decrease the length of the queue but will impact throughput since not all packets will be successfully forwarded to the network. These two goals are balanced using a tunable parameter in the DQN algorithm.

3. System Model

In the network topology under consideration there are multiple nodes on two sides of a bottleneck link. Data is being transmitted from nodes P_1, P_2, … P_K to nodes Q_1, Q_2, … Q_K. There is a low bandwidth bottleneck link between the two networks. In simulations as detailed in Table 4 the number of transmitter/receiver pairs is varied from K = 2 to 8.

The bottleneck link is the one that connects Node N_1 to Node N_2. The bandwidth and the link delay on the bottleneck link is controlled in simulations. N_1 is the router where the AQM algorithm is implemented.

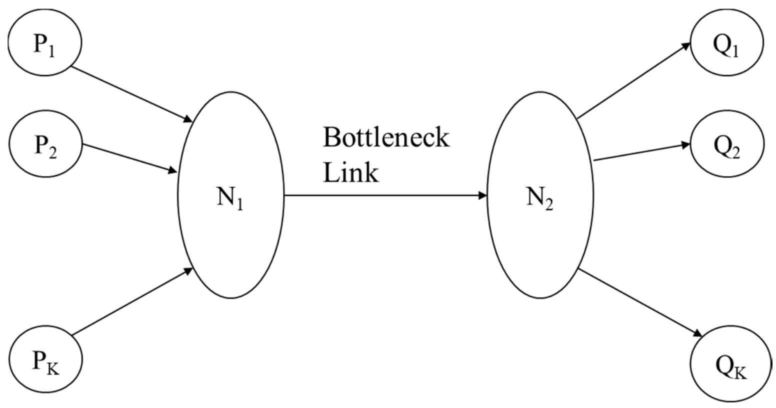

***Figure* 1.** Network structure.

The algorithm works by balancing two types of utility functions, referred to as rewards:

- The delay reward increases with decreased delay time.
- The enqueue reward increases as the rate of packet drops decreases.

The two rewards must be balanced as when most packets are dropped the delay reward will increase (but many packets will be lost) whereas if most packets are kept the enqueuer reward will increase but packets may experience very long delays. The balance between the two is controlled using a scale factor parameter δ.

$\boldsymbol{R = clip\Big((\delta \times R_{\mathrm{D}}) + \big((1-\delta) \times R_{\mathrm{E}}\big), -1, 1\Big)}$	(1)
$\boldsymbol{R_{\mathrm{D}} = d_{\mathrm{QD}} - d_{\mathrm{QC}}}$	(2)
$\boldsymbol{R_{\mathrm{E}} = (d_{\min} - d_{\mathrm{QD}}) \times \left(\frac{N_{en}}{N_{en}+N_{drop}}\right)}$	(3)
$\boldsymbol{d_{\mathrm{QC}} = \frac{L_{\mathrm{Q}}}{N_{\mathrm{drop}}} \times (1\ second)}$	(4)
$\boldsymbol{d_{\min} = \frac{L_{\mathrm{Q}}}{B}}$	(5)

where Equation (4) is as used in the Proportional Integral Controller Enhanced (PIE) AQM protocol. The clipping function is a nonlinear function which limits the reward to be between -1 and 1 to enhance the stability of the algorithm.

Table 1. *Equation variables.*

R	reward	Reward parameter used to tune the queue size
δ	Scale Factor	Scales the balance between enqueue reward and delay reward
R_{D}	Delay Reward	increases with decreased delay time of packets
R_{E}	Enqueue Reward	increases as the rate of packet drops is reduced.
d_{QD}	Desired Queue Delay	Delay that can be tolerated by user, default value is 0. May be used to define different classes of service.
d_{QC}	Current Queue Delay	Delay currently experienced by a user. Estimated using the formula used by the Proportional Integral Controller Enhanced (PIE) AQM protocol.
$d_{\min}$	Minimum Expected Delay	Delay experienced by the packet at the end of the queue if transmit channel is continuously in use.
L_{Q}	Average Queue Length	Average queue length for the user (measured, in units of bytes).
B	Bandwidth in Bytes	Total network link bandwidth, available for use.
N_{en}	Enqueue Counter	Counts number of packets in the queue which are not dropped.
N_{drop}	Dequeue Counter	Counts number of packets in the queue which are dequeued/dropped.

4. Simulation Parameters and Results

Table 2 summarizes the hardware and software used in testing. Table 3 shows the NS3 Training Simulation Parameters.

The DQN and RED algorithms were implemented on the simulation platform ns3 [11][12]. The deep learning was realized using Tensorflow under Python. Table 4 shows the values of bandwidth and latency that were used for the bottleneck link, the number of transmitter and receiver nodes as well as the flow rate for each node. In every case the bottleneck link was highly loaded with offered traffic. Through simulations the RED and DQN algorithms' queue length, throughput, delay, and packet loss rate data are collected. Simulation time is set to sixty seconds.

Table 2. *Hardware and Software used in testing*

OS Version	Ubuntu 20.04.3 64 bit
CPU	Intel Core i7 4500U
RAM	8GB DDR3
Storage	SSD 128GB
Python Version	Python 3.8.13
Tensorflow	Tensorflow Version 2.4.1
Python IDE	VS Code 1.67.1
NS3 Version	3.35

Table 3. *System Parameters*

Parameter	**Value**
Bottleneck Bandwidth	2Mbps to 8Mbps
Bottleneck Delay	5ms
Access Bandwidth	1000Mbps
Access Delay	0.1ms
Number of sender-sink agents	2 to 8
Flow Data Rates	1Mbps to 3Mbps
Simulation Duration	60s

Table 4. *Test Parameters*

Test No.	Simulation Duration in seconds	Bottleneck Link		Sender/Sink Apps	
		Bandwidth in Mbps	Delay in MS	Number	Flow Rate in Mbps
1	60	2	5	2	1
2	60	2	5	3	1
3	60	2	5	4	1
4	60	4	5	3	2
5	60	4	5	4	2
6	60	4	5	5	2
7	60	8	5	6	3
8	60	8	5	7	3
9	60	8	5	8	3

Figure 2 shows the average round trip time (RTT) for the DQN based AQM algorithm compared to the RED algorithm for the scenarios listed in Table 4. It is observed that the RTT is almost the same for the two algorithms, with the DQN based algorithm having slightly lower RTT in every scenario. The improvement in RTT for each scenario is shown in Figure 3.

Figure 4 shows the values of RTT over the 60 second simulation duration for Scenario 1 in Table 4. It can be seen that DQN RTT is lower for specific scenarios. Unfortunately, as with RED the DQN algorithm also seems to lead to oscillations in round trip time and this is a scenario for further work.

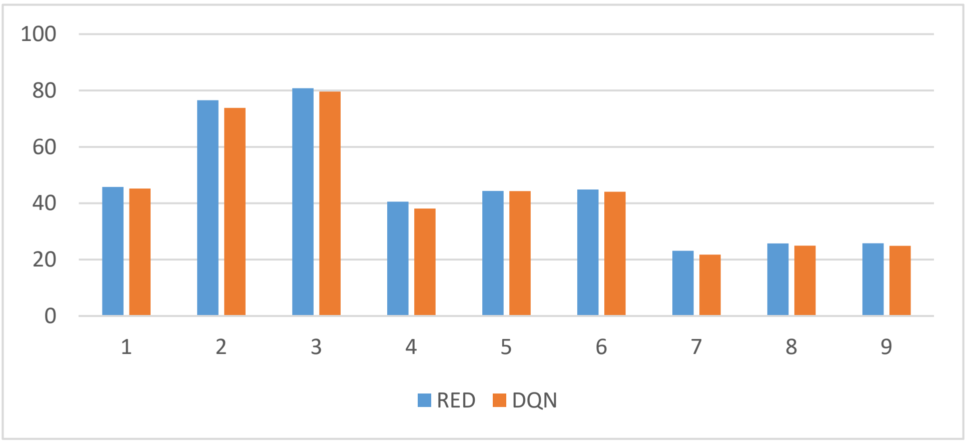

***Figure** 2.* Average RTT values.

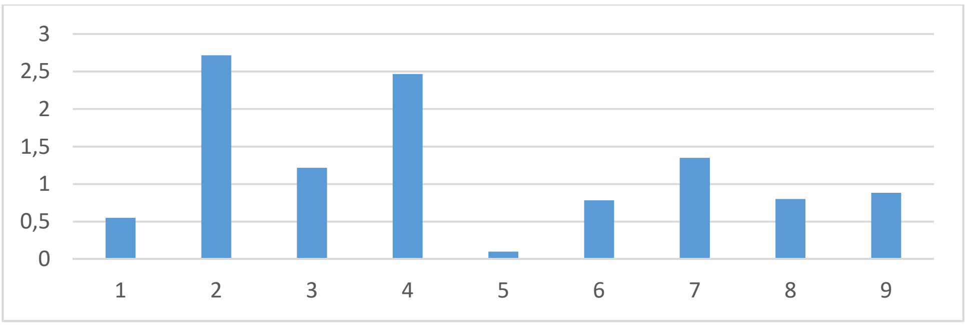

***Figure* 3.** Improvement in RTT for DQN over RED.

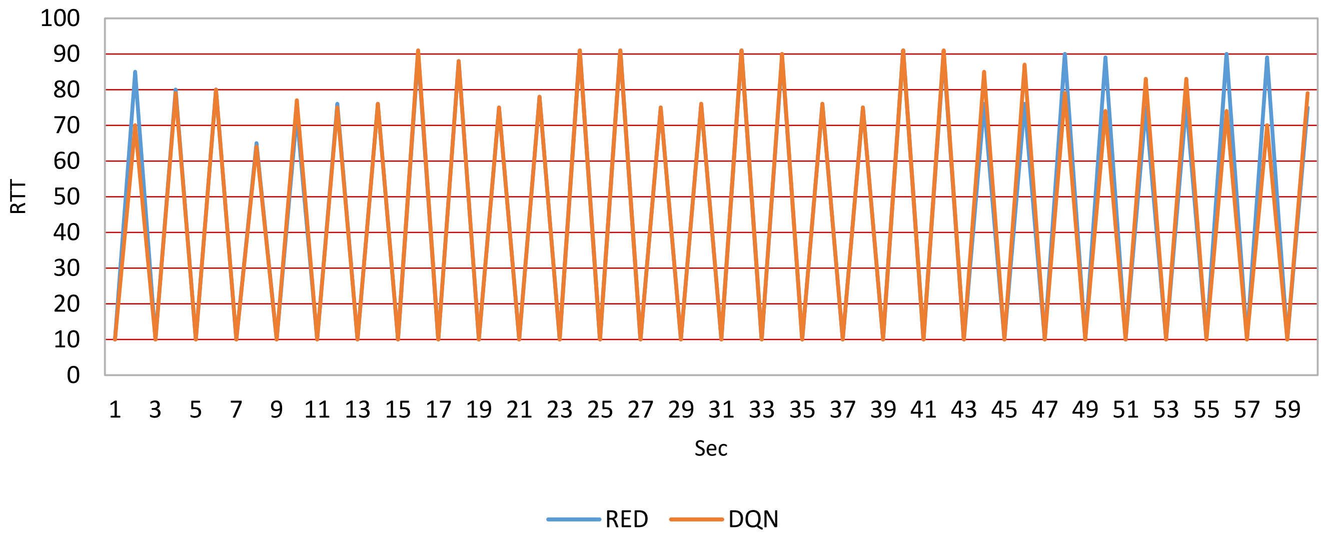

***Figure* 4.** RTT over time for DQN and RED in Scenario 1.

5. Conclusion

This paper performed simulations of two active queue management algorithms: RED and DQN-QM. Simulations were performed using ns3 and tensorflow. The deep reinforcement learning based AQM was found to outperform RED in terms of latency and throughput under many scenarios.

Acknowledgment

My most humble and sincere thanks to:
Thank Allah for making us capable and allowing us to complete this work.
My deepest thanks go to my supervisor, Dr. Didem KIVAN TÜREL, for her helpful advice and patience, as well as to my family, especially my brother Mohammed.

REFERENCES

[1] M. Kim, M. Jaseemuddin, and A. Anpalagan, “Deep Reinforcement Learning Based Active Queue Management for IoT Networks,” *J. Netw. Syst. Manag.*, vol. 29, no. 3, 2021, doi: 10.1007/s10922-021-09603-x.

[2] H. Ma, D. Xu, Y. Dai, and Q. Dong, “An intelligent scheme for congestion control: When active queue management meets deep reinforcement learning,” *Comput. Networks*, vol. 200, no. October, p. 108515, Dec. 2021, doi: 10.1016/j.comnet.2021.108515.
[3] J. Szyguła, A. Domański, J. Domańska, D. Marek, K. Filus, and S. Mendla, “Supervised Learning of Neural Networks for Active Queue Management in the Internet,” *Sensors*, vol. 21, no. 15, p. 4979, Jul. 2021, doi: 10.3390/s21154979.
[4] G. Abbas, Z. Halim, and Z. H. Abbas, “Fairness-Driven Queue Management: A Survey and Taxonomy,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 324–367, 2016, doi: 10.1109/COMST.2015.2463121.
[5] Baker, F., Fairhurst, G.: IETF Recommendations Regarding Active Queue Management. In: *Internet Engineering Task Force (IETF)*, RFC 7567, 2015.
[6] M. M. Hamdi, S. A. Rashid, M. Ismail, M. A. Altahrawi, M. F. Mansor, and M. K. Abufoul, “Performance Evaluation of Active Queue Management Algorithms in Large Network,” *ISTT 2018 - 2018 IEEE 4th Int. Symp. Telecommun. Technol.*, pp. 3–8, 2018, doi: 10.1109/ISTT.2018.8701716.
[7] R. Adams, “Active Queue Management: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 15, no. 3, pp. 1425–1476, 2013, doi: 10.1109/SURV.2012.082212.00018.
[8] C. Li, “Deep Reinforcement Learning,” in *Reinforcement Learning for Cyber-Physical Systems*, November, Boca Raton, Florida : CRC Press, [2019]: Chapman and Hall/CRC, 2019, pp. 125–154.
[9] N. C. Luong *et al.*, “Applications of Deep Reinforcement Learning in Communications and Networking: A Survey,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019, doi: 10.1109/COMST.2019.2916583.
[10] M. Fajri and K. Ramli, “Design of network traffic congestion controller with PI AQM based on ITAE index,” Int. J. Electron. Telecommun., vol. 66, no. 4, pp. 715–721, 2020, doi: 10.24425-ijet.2020.134032/754.
[11] D. A. Alwahab and S. Laki, “A simulation-based survey of active queue management algorithms,” *ACM Int. Conf. Proceeding Ser.*, no. February, pp. 71–77, 2018, doi: 10.1145/3193092.3193106.
[12] L. Campanile, M. Gribaudo, M. Iacono, F. Marulli, and M. Mastroianni, “Computer network simulation with ns-3: A systematic literature review,” *Electron.*, vol. 9, no. 2, pp. 1–25, 2020, doi: 10.3390/electronics9020272.