Fairness Enhancement of TCP Congestion Control Using Reinforcement Learning

Sang-Jin Seo*
School of Electronic and Electrical Engineering
Kyungpook National University
Daegu, Korea
chil258@knu.ac.kr

You-Ze Cho
School of Electronic and Electrical Engineering
Kyungpook National University
Daegu, Korea
yzcho@ee.knu.ac.kr

***Abstract*—In TCP congestion control research, the use of machine learning to solve the issue of unused link bandwidth and to improve performance, such as maximizing link utilization or minimizing latency, is steadily increasing. Among such approaches, the Deep Q Network (DQN)-based TCP congestion control algorithm improves the link utilization but suffers from performance degradation when a specific link bandwidth is exceeded. In addition, inter-protocol fairness with other TCP congestion control algorithms has not been verified. In this paper, on a NS3 simulator, we conducted the experiments to enhance the improvement of the DQN-based TCP congestion control algorithm v2 in single flow and an inter-protocol fairness when several flows share the same bottleneck link. Our results confirmed that the average throughput was improved, and our approach is fairer than existing congestion control algorithms.**

Keywords—Deep Q Network, TCP congestion control

I. INTRODUCTION

TCP congestion control is a network congestion avoidance algorithm that involves slow start and adjustment of the congestion window (Cwnd), i.e., the maximum number of unacknowledged packets that can be transmitted. As an internet host function implemented in the operating system's protocol stack, TCP congestion control is used to reduce packet loss and avoid congestion collapse by limiting the Cwnd. Since TCP congestion control was first proposed in 1988 [1], various versions of the algorithm have been studied and proposed, e.g., NewReno [2], CUBIC [3], and BBR [4].

Research to improve the performance of TCP congestion control is still ongoing. In particular, since 2017, interest in congestion control research by applying machine learning has increased [5]. Machine learning is an approach that automatically improves performance through experience and learning. Machine learning has been applied to many congestion control algorithms such as an algorithm that has been improved to use the available link bandwidth as much as possible in various link environments [6], an algorithm to minimize latency in real-time traffic environments [7], and so on.

Among machine learning-based TCP congestion control algorithms, the Deep Q Network (DQN)-based TCP congestion control algorithm v1 we proposed earlier [6] has a higher average throughput than CUBIC or NewReno when the link bandwidth is less than 50 Mbps. However, when the link bandwidth exceeds 50 Mbps, the average throughput is relatively low. Furthermore, inter-protocol fairness with existing congestion control was not verified. In this paper, on a NS3 simulator, we conducted an experiment to validate the improvement of the DQN-based TCP congestion control algorithm in single flow and an experiment to check the fairness when existing or DQN-based TCP congestion control algorithms share the same bottleneck link.

II. RELATED WORK AND MOTIVATION

Existing congestion control algorithm such as NewReno and CUBIC use hand-tuned heuristics, so they only operate with a fixed Cwnd adjustment algorithm in any network environment. As Cwnd increases according to a fixed algorithm, the existing congestion control unconditionally causes congestion during the Cwnd increase.

Nowadays, due to the development of communication technology the internet speed increases, and the link bandwidth become larger. As the link bandwidth becomes larger such as 5G or over 100 Mbps of network environment, the existing congestion control algorithm has some issues that takes a longer time to increase Cwnd to use all link bandwidth, and the unused link bandwidth also increases due to unconditionally occurring congestion.

To overcome these issues by making it more adaptable to network environment, several algorithms, such as Orca [8], a hybrid congestion control protocol that depends on TCP fine-grained control action with DDPG, Aurora [9], a rate-based congestion control algorithm with PPO, and DQN-based TCP congestion control algorithm [6][10], that apply machine learning from various perspectives, have been proposed.

Among the machine learning methods, DQN is learned in real time by selecting an action that obtains the best reward from the state, it is suitable for adapting to the real-time changing network environment that has many state parameters, so there have been many attempts to apply it to the congestion control.

The DQN-based TCP congestion control algorithm v1 we proposed [6], has higher average throughput when the link bandwidth is under 50 Mbps but has lower average throughput than CUBIC when the link bandwidth is over 50 Mbps, and the inter-protocol fairness or round-trip time (RTT) has not yet been validated. So, to improve the issues of DQN-based TCP congestion control algorithm v1 and verify the inter-protocol fairness, we propose the DQN-based TCP congestion control algorithm v2.

III. DQN-Based Congestion Control Algorithm

A. DQN Model

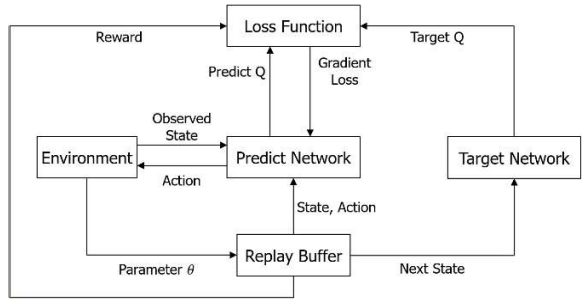

Figure 1. DQN algorithm.

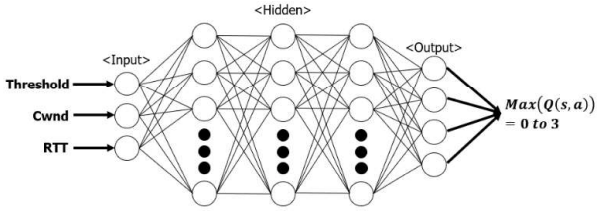

Figure 2. Deep Neural Network model for action.

Figure 1 is a diagram of DQN algorithm. The environment refers to the network environment in which communication is performing using congestion control, and the DQN agent is congestion control in environment. The DQN attempts to find a policy that maximizes the $Q(s,a)$, which is the value obtained when the agent takes an action 'a' in a certain state 's'. The Q value can be obtained as an output value of the deep neural network using the state as an input, and the one with the largest Q value is selected as the action like Figure 2. The parameter θ is a tuple that [state (s), reward (r), action (a), next state (s')].

At the target network, the agent calculates the target Q using Bellman equation, as (1). The $R(s,a)$ is the final reward value calculated using the equation described in section C, Reward function. The γ is a discount factor for weight, the s' is next state after action, and a' is the best action that the agent chose. So, $\max_{a'} Q(s',a')$ means the maximum possible Q value at the next state, estimated by target network.

When the parameter is θ_i, the loss for gradient descent is calculated with the loss function (2) using Mean Squared Error (MSE), and $Q_{predict}$ is Q value of the current state after the DQN, from predict network.

$$Q_{target}(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a') \quad (1)$$

$$L_i(\theta_i) = \mathrm{E}_{s,a,r,s'}\left[\left(Q_{target}(s,a;\theta_i) - Q_{predict}(s,a;\theta_i)\right)^2\right] \quad (2)$$

The DQN-based TCP congestion control algorithm v1 [6], used [Threshold, Cwnd, RTT, Throughput] for state space. However, because the throughput overlaps the value calculated by Cwnd and RTT, we excluded the throughput from state space of the DQN-based TCP congestion control algorithm v2 to improve the learning performance. In this model, Cwnd is a byte unit.

In the hidden layer, the previous model used only ReLU as an activation function, so the model training was not good in some cases. We improved the performance by adding Dropout so that the model can be trained well in various network environments by reducing overfitting and using all nodes.

B. DQN output & Action Space [6]

$$a = \begin{cases} 0: Very\ high\ possibility\ of\ congestion \\ 1: High\ possibility\ of\ congestion \\ 2: Low\ possibility\ of\ congestion \\ 3: Very\ low\ possibility\ of\ congestion \end{cases} \quad (3)$$

$$Action - Cwnd\ Adjustment = \begin{cases} Cwnd = Cwnd + SegmentSize \times (a-1), & a > 0 \\ Cwnd = Cwnd - SegmentSize, & a = 0 \end{cases} \quad (4)$$

$$Epsilon - Greedy\ Probability = \begin{cases} \varepsilon, & Random\ Action \\ 1-\epsilon, & Best\ Rewards\ Action' \end{cases} \quad \varepsilon = 0.015 \quad (5)$$

At first, after DQN learning, the output is the same value as (3), and the model learns by recognizing the output as network state. Action is selected using (4) according to the DQN output. At this time, the output value 'a' is used as a variable in the Cwnd adjustment equation. By increasing/decreasing Cwnd according to the network state, the algorithm adjusts to maintain the link utilization rate learned through learning as much as possible.

Action selection probabilistically chooses between the best performance action and the random action using the Epsilon-greedy policy of (5), with the epsilon value as 0.015. Therefore, even if the network environment changes, the DQN model can explore to learn the new best performance action.

C. Reward function [6]

$$Rewards = \frac{Throughput_i}{Throughput_{DQN}} \quad (6)$$

$$Throughput_i = \frac{Cwnd_i}{RTT_i} \quad (7)$$

Equation (6) is a reward function indicating the link utilization rate, which is the improvement direction that this algorithm focuses on. $Throughput_i$ is the throughput calculated by (7), and uses measured $Cwnd_i$ and RTT_i, parameters when the DQN agent receives ACK_i for the i^{th} segment and confirms that the transmission completed without congestion. $Throughput_{DQN}$ is the maximum throughput that will not cause congestion, obtained through DQN learning. By adjusting Cwnd using a Rewards function, it can minimize the congestion occurring in a micro-changing network environment.

D. Overall algorithm

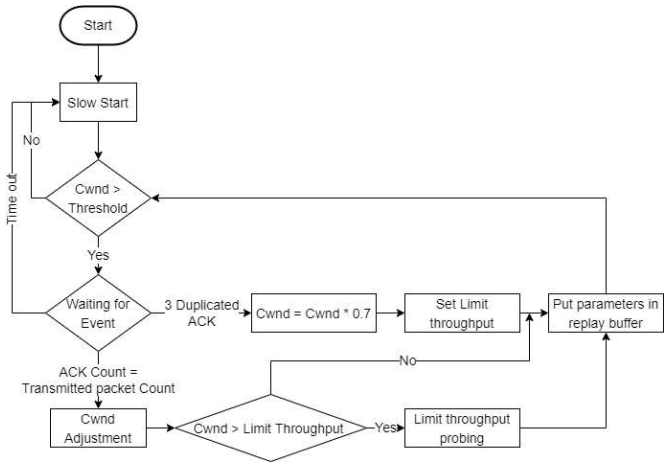

Figure 3. Flowchart of DQN-based TCP congestion control algorithm.

Figure 3 is a flowchart of the DQN-based TCP congestion control algorithm v2. When communication starts, it enters and operates in the standard TCP slow start until the Cwnd is bigger than threshold. After the slow start, the algorithm waits for the event. First, when three duplicated ACKs occur, it is judged as congestion, and Cwnd is reduced by using the standard CUBIC Cwnd reduction index of 0.7 which aims for fast Cwnd increase and scalability. Afterwards, set the limit throughput for $Throughput_{DQN}$. Second, when a timeout occurs, it re-enters a slow start according to the standard TCP operation. Finally, when ACKs equal to the number of all transmitted packets are received, it operates in an algorithm that takes an action based on (4). If Cwnd is bigger than the limit throughput without 3 duplicated ACKs, the agent enters limit throughput probing, otherwise it input parameters into the replay buffer.

IV. Experiment Environment

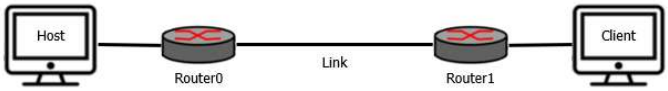

Figure 4. NS3 simulator experiment A setup.

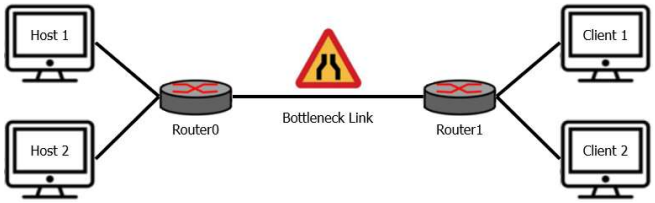

Figure 5. NS3 simulator experiment B setup.

Figure 4 illustrates the setup of experiment A on NS3. The experiment A involves checking the performance improvement of the DQN-based TCP congestion control algorithm v2 compared to existing congestion control algorithms. The performance is compared by transmitting NewReno, which uses the same AIMD algorithm, CUBIC, which is widely used as a standard, the DQN-based TCP congestion control algorithm v1, and the v2 with only one host and client in a single flow. The RTT of the experiment environment is set to 100ms, and the experiment time is 300 seconds. Link bandwidths of 5, 25, 50, 75, and 100 Mbps are used.

Figure 5 illustrates the setup of experiment B on NS3. The experiment B compares fairness when two flows using different congestion control algorithms share a bottleneck link. The experiment is conducted with a bottleneck link bandwidth of 50 Mbps, an RTT of 100 ms, and an experiment time of 300 seconds with two hosts and client. The competition uses the DQN-based TCP congestion control algorithm v2 vs CUBIC or NewReno.

V. Evaluation

A. Congestion control comparison when single flow

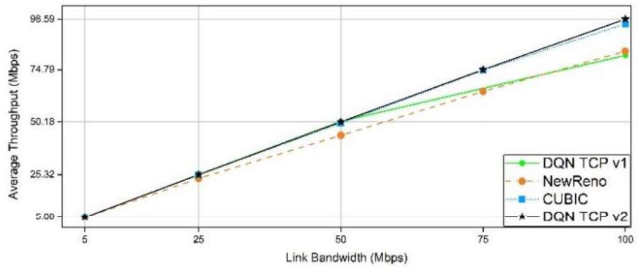

Figure 6. Average throughput comparison of each congestion control.

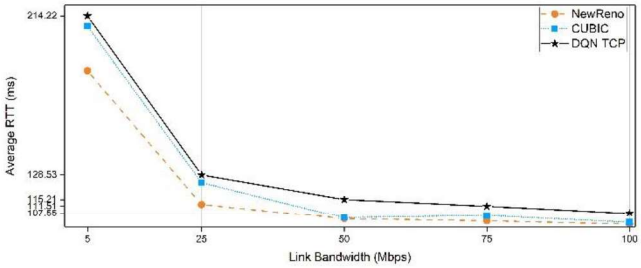

Figure 7. Average RTT comparison of each congestion control.

Figure 6 is a graph comparing the average throughput of each congestion control during a single flow. The DQN-based TCP congestion control algorithm v1 suffers from lower average throughput than CUBIC and NewReno when the link bandwidth exceeds 50 Mbps. However, we confirmed that the proposed algorithm has a higher average throughput than the existing congestion control because there is almost no Cwnd reduce operation due to congestion.

Figure 7 is a graph comparing the average RTT of each congestion control. Although the average RTT of the proposed algorithm was higher than existing congestion control in all link bandwidths, the difference was insignificant, i.e., approximately 5 ms.

B. Comparison of fairness when sharing bottleneck link

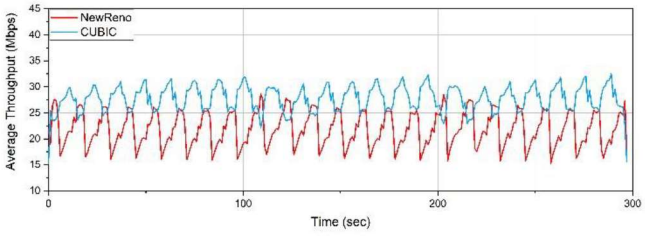

Figure 8. NewReno vs CUBIC average throughput.

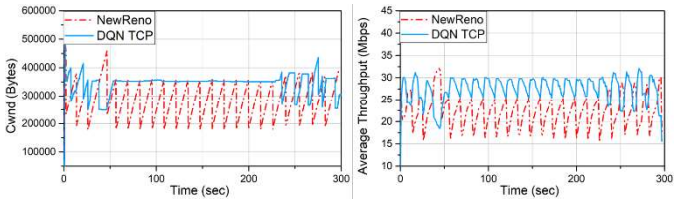

Figure 9. NewReno vs the proposed algorithm average throughput & Cwnd.

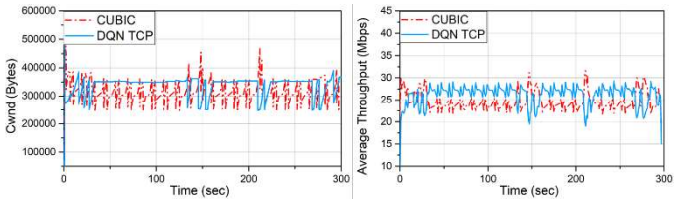

Figure 10. CUBIC vs the proposed algorithm average throughput & Cwnd.

Figure 8 is a graph of average throughput when NewReno and CUBIC compete by sharing the same bottleneck link. Ideally, each should have 25 Mbps of average throughput, but CUBIC has 27.7 Mbps and NewReno has 21.85 Mbps for average throughput because CUBIC has a faster Cwnd increase speed and small reduction index than NewReno.

Figure 9 is a graph of Cwnd and average throughput when NewReno and the proposed algorithm compete by sharing the bottleneck link. NewReno reduces Cwnd using 0.5 of reduction index when congestion occurs, and the Cwnd increase speed is slower than proposed algorithm or CUBIC. The proposed algorithm maintains the learned Cwnd, and NewReno repeats the increase and decrease in Cwnd due to congestion. Even if congestion occurs because of exploration of DQN or Cwnd increase of NewReno, the proposed algorithm increases the Cwnd back to the previous Cwnd again after decreasing it, and maintains. As a result, the average throughput of NewReno is 22.28 Mbps and that of the proposed algorithm is 27.39 Mbps, which is not significantly improved compared to CUBIC vs NewReno.

Figure 10 is a graph of Cwnd and average throughput when CUBIC and the proposed algorithm compete by sharing the same bottleneck link. CUBIC selects a concave/convex function as a Cwnd increment algorithm according to w_{max}. When the proposed algorithm occupies the bottleneck link by learning the result, CUBIC adjusts Cwnd using the remaining link bandwidth. The proposed algorithm maintains Cwnd as much as possible and even if congestion occurs, and the two algorithms have a faster Cwnd increase rate and a smaller reduced index than NewReno. For these reasons, the average throughput of CUBIC is 24.62 Mbps, and that of the proposed algorithm is 26.08 Mbps, confirming that is fairer than when NewReno competing with CUBIC.

VI. Conclusion

In this paper, we propose an improved DQN-based TCP congestion control algorithm v2 and simulate the performance difference of each algorithm in a single flow as well as the fairness comparison when sharing a bottleneck link on an NS3 simulator. As a result of improving the DQN model, the average throughput improved to higher than existing TCP congestion control algorithms for all bandwidths. Through a fairness experiment when sharing a bottleneck link, the proposed algorithm and CUBIC confirmed that fairness was good by using the link bandwidth at an approximately 49:51 ratio. However, the proposed algorithm and NewReno do not significantly improve compared to CUBIC vs NewReno by using the link bandwidth at about 55:45. In future research, we will attempt to adjust Cwnd adaptively and make it more TCP-friendly in a more dynamic network environment by applying a learning method such as DDPG with continuous output rather than DQN with discrete output.

Acknowledgment

This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by Ministry of Education (No. NRF-2018R1A6A1A03025109) and by National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1006249).

References

[1] V. Jacobson, "Congestion avoidance and control,", ACM SIGCOMM Computer Communication Review, vol. 25, pp. 157-187, January 1995.

[2] S. Floyd, A. Gurtov, and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 2582, April 1999.

[3] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," ACM SIGOPS Operating Systems Review, vol. 42, pp. 64-74, July 2008.

[4] N. Cardwee et al., "BBR: Congestion-Based Congestion Control," Commun. ACM, vol. 60, pp. 58-66, February 2017.

[5] W. Wei, H. Gu, and B. Li, "Congestion control: A Renaissance with Machine Learning," IEEE Network, vol. 35, pp. 262-269, July/August 2021.

[6] S. J. Seo and Y. Z. Cho, "DQN-Based TCP Congestion Control Algorithm to Improve Link Utilization," 2nd Korea Artificial Intelligence Conference, September 2021.

[7] J. Fang et al., "Reinforcement Learning for Bandwidth Estimation and Congestion Control in Real-Time Communications," Proc. NeurIPS Workshop on Machine Learning for Systems, 2019.

[8] S. Abbasloo et al., "Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet," Proc. ACM SIGCOMM, pp. 632-647, 2020.

[9] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," 36th International Conference on Machine Learning, 2019.

[10] Y. Wang, L. Wang, and X. Dong, "An Intelligent TCP Congestion Control Method Based on Deep Q Network," Future Internet, vol. 13, no. 10, pp. 261, October 2021.