https://doi.org/10.7840/kics.2024.49.4.567

DQN 기반 TCP 혼잡제어 알고리즘의 성능평가

서 상 진◆, 조 유 제°

Performance Evaluation of DQN-Based Congestion Control Algorithm for TCP

Sang-Jin Seo◆, You-Ze Cho°

요 약

기존의 TCP 혼잡제어는 느린 혼잡 윈도우 (cwnd) 증가율로 인해 전송 대역폭이 매우 크거나, 채널의 특성이 빈번하게 변화하는 환경에서 사용 가능한 대역폭을 충분히 활용하지 못하는 문제점이 있다. 이러한 문제를 개선하기 위해 기계학습을 적용한 적응형 TCP 혼잡제어에 관한 연구가 꾸준히 진행되고 있다. 본 논문에서는 강화학습의 일종인 Deep Q-Network (DQN)을 TCP 혼잡제어 알고리즘에 적용하여 성능을 개선한 DQN-based NewReno와 DQN-based CUBIC을 소개하였다. 구현한 알고리즘들은 NS-3 시뮬레이터를 통해 성능 평가를 수행하였으며, 실험 결과를 통해 DQN-based CUBIC이 특히 기존 혼잡제어에 비해 높은 처리량을 보이고, 서로 다른 혼잡제어 프로토콜 간의 공평성과 round trip time (RTT) 공평성 모두 향상됨을 확인하였다.

키워드 : TCP 혼잡제어, TCP CUBIC, Deep Q-Network, 강화학습
Key Words : TCP Congestion Control, TCP CUBIC, Deep Q-Network, Reinforcement Learning

ABSTRACT

The existing TCP congestion control suffers from the problem of slow congestion window (cwnd) increase, leading to underutilization of available bandwidth in environments where there is either a very large link bandwidth or frequent changes in channel characteristics. To address these issues, research on adaptive TCP congestion control using machine learning has been consistently progressing. In this paper, we propose DQN-based NewReno and DQN-based CUBIC, which enhance performance by applying a type of reinforcement learning, Deep-Q Network (DQN) to TCP congestion control algorithms. The implemented algorithms underwent performance evaluation using the Network Simulator 3 (NS3). Experimental results reveal that DQN-based CUBIC, in particular, demonstrates higher throughput compared to traditional congestion control. Additionally, fairness between different congestion control and round-trip time (RTT) fairness is also improved.

※ 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 the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2023R1A2C1003928).
◆ First Author : kyungpook national university, tjtkdwls1111@gmail.com, 정회원
° Corresponding Author : kyungpook national university, yzcho@ee.knu.ac.kr, 종신회원
논문번호 : 202308-039-B-RE, Received August 8, 2023; Revised November 2, 2023; Accepted December 9, 2023

Ⅰ. 서 론

TCP 혼잡제어는 호스트에서 네트워크 혼잡 상황을 감지하고, 이에 대응하여 트래픽 양을 조절함으로써 데이터를 안정적으로 전송하기 위한 기술이다. 네트워크에서 혼잡 상황은 호스트에서 링크 대역폭을 초과하는 과도한 트래픽의 전송을 시도할 때 주로 발생한다[1]. 네트워크에서 혼잡이 발생하면 패킷 손실로 인해 네트워크 품질이 저하되며, 이러한 상황이 지속되면 타임아웃으로 인한 호스트와의 연결 끊김과 같은 추가적인 문제가 발생할 수 있다.

1986년 10월, 네트워크 혼잡 상황을 방지하기 위해 TCP 혼잡제어가 제안되었다[2]. TCP 혼잡제어는 호스트가 한 번에 네트워크에 보낼 수 있는 패킷의 최대 개수를 나타내는 변수인 혼잡 윈도우 (cwnd)를 통해 송신하는 패킷의 양을 조절한다. 패킷이 성공적으로 전송되었음을 나타내는 응답인 ACK를 받으면 cwnd를 증가시켜 전송할 패킷의 양을 늘리고, 혼잡이 발생하면 감소시켜 전송할 패킷의 양을 줄인다.

상기한 동작 원리를 바탕으로 TCP Tahoe[2], TCP Reno[3], TCP NewReno (이하 NewReno)[4] 등이 개발되었다. 기본적인 TCP 혼잡제어 알고리즘인 NewReno는 cwnd 조절을 위해 additive increase multiplicative decrease (AIMD) 알고리즘을 사용하는데, AIMD는 혼잡이 발생하지 않고 정상적으로 통신이 이뤄질 때는 cwnd를 세그먼트 크기만큼 계속 증가시키고, 혼잡이 감지되면 cwnd를 절반으로 줄인다. 하지만, 이러한 방식은 대역폭-지연곱 (BDP)이 크면 클수록 한번 손실이 발생하였을 때 cwnd를 복구하기 위한 시간이 오래 걸리며, 링크가 추가적인 대역폭을 사용할 수 있을 경우 새로운 용량을 할당하는 데에도 매우 긴 시간을 필요로 한다[5]. 또한, 서로 다른 round trip time (RTT)을 가진 플로우끼리 경쟁하는 환경에서의 공평성 문제가 존재한다.

통신 기술의 발전으로 인해 크게 증가한 전송 대역폭에 맞춰 처리량이나 링크 이용률과 같은 성능을 최적으로 보장하기 위해 TCP CUBIC (이하 CUBIC)과 같은 고속 네트워크 환경을 상정한 혼잡제어 알고리즘들이 제안되었다[6]. CUBIC은 NewReno에 비해 cwnd의 크기가 더 빠르게 증가하여 대역폭을 보다 효율적으로 활용하는 TCP 혼잡제어 알고리즘이며, RTT 공평성 문제 또한 개선하였다. 그러나, CUBIC 또한 고정된 알고리즘을 사용하여 동작하기 때문에, 더욱 큰 BDP를 가진 네트워크[7]나 안정적이지 않고 다이내믹하게 변화하는 네트워크 환경[8]에서는 최적의 성능을 제공하는데 어려움을 가진다.

이러한 문제를 개선하기 위해 기계 학습을 혼잡제어 알고리즘에 적용하여 어떠한 링크 환경에서도 링크 대역폭을 최대한 사용하도록 하기위한 혼잡제어 알고리즘이나[9,10] 실시간 트래픽 환경에서 지연 시간을 최소화하는 혼잡제어 알고리즘[11] 등이 제안되고 있다. 현재 공개된 기계 학습 기반의 TCP 혼잡제어 알고리즘들은 AIMD 알고리즘을 기반으로 cwnd를 조절하는 경우가 많으며, 학습 방식과 목표로 하는 개선 방향이 서로 다른 특징을 가지고 있다.

본 논문 저자들 또한 TCP 혼잡제어의 처리량을 향상시키기 위해, 이전 연구에서 유선 네트워크를 가정한 환경일때 보다 높은 처리량을 제공하는 Deep Q-Network (DQN)-based NewReno를 제안했다[9]. DQN은 실시간 학습 및 학습 데이터의 재사용을 통한 효율성에 강점이 있는 학습 기법으로, 실시간으로 변화하는 데이터를 사용해야 하는 TCP 혼잡제어에 적합한 학습 방식이다. 그러나, DQN-based NewReno는 기존의 NewReno를 기반으로 설계되었기 때문에 RTT 공평성에 관한 단점을 그대로 가지고 있을 것으로 예측되었다. DQN-based NewReno의 단점을 개선하기 위해 우리는 DQN을 CUBIC에 적용한 DQN-based CUBIC또한 이전 연구에서 제안하였다[12,13]. DQN-based CUBIC은 높은 처리량을 제공하며, 서로 다른 플로우 간의 공평성을 유지할 수 있는 알고리즘이다. 하지만, 기존 논문에서의 DQN-based CUBIC은 실험 결과가 이상적이지 않거나[12], 서로 다른 혼잡제어를 사용하는 플로우들 간의 공평성에만 초점을 둬 명확한 성능 평가가 이뤄졌다고 보긴 어렵다[13]. 따라서 본 논문 저자들은 제안한 두 알고리즘들을 Network Simulator-3 (NS-3)에서 구현하고, 다양한 네트워크 시나리오에서 성능을 비교 및 평가하였다.

Ⅱ. TCP 혼잡제어와 강화 학습

2.1 TCP 혼잡제어

2.1.1 TCP혼잡제어 알고리즘과 연구동향

기본적인 TCP 혼잡제어인 NewReno는 cwnd 조절을 위해 AIMD 알고리즘[4]을 사용한다. Cwnd의 증가 속도가 느리기 때문에, NewReno는 넓은 대역폭을 가진 네트워크 환경에서는 높은 처리량을 보장하기 힘들다.

이러한 문제를 극복하기 위한 CUBIC은 고속 통신 환경에 초점을 두고 개발되었다. CUBIC은 삼차함수

를 사용하여 cwnd를 빠르게 증가시켜 높은 처리량을 제공한다. 또한, 패킷 손실로부터 경과된 시간을 cwnd 조정식에 사용함으로써 RTT 공평성에 대한 장점을 가지고 있다[6].

기존의 TCP 혼잡제어는 고정된 알고리즘으로 인해 정해진 대로만 동작하고, 이러한 정적인 알고리즘은 네트워크의 상태가 다양한 실제 환경에서 항상 최적의 성능을 제공하지는 못한다. 예를 들어, 가용 링크 대역폭이 크거나, 특성이 자주 변화하는 다이내믹한 환경에서 기존의 TCP 혼잡제어는 cwnd의 증가 속도가 느리기 때문에 대역폭을 충분히 활용할 수 없다는 문제가 제기되었다[14]. 언급한 문제 외에도 여러 문제점들이 존재하기에, 혼잡제어의 성능을 개선하기 위한 연구가 지속되고 있다.

2016년 Google은 bottleneck bandwidth and round-trip propagation time (BBR) 혼잡제어 알고리즘을 제안하였다[15]. Google 서버를 사용하는 네트워크 환경에서 실제로 적용되어 사용중인 BBR은 탁월한 성능을 보장하지만, 기존의 혼잡제어와 마찬가지로 모든 네트워크 상황에서 최적의 성능을 제공하지는 못한다. 그렇기 때문에, 어떠한 네트워크 환경에서도 적응적으로 동작하는 이상적인 혼잡제어를 개발하기 위해 기계 학습을 혼잡제어에 적용하는 연구가 꾸준하게 진행되고 있다[10].

2.1.2 온/오프라인 학습 기반 혼잡제어 알고리즘

온/오프라인 학습 기반 혼잡제어는 대표적으로 호스트의 행동과, 그로인한 결과의 연결을 지속적으로 관찰하여 학습하는 PCC[16], 미리 지정된 성능적 목표를 기반으로 패킷 손실과 대기열 지연을 감소시키기 위한 Remy[17], Phantheon으로 알려진 네트워크 실험 도구에서 수집된 데이터를 사용하여 학습하며 전송 패킷의 양을 조절하는 Indigo[18]등이 있다.

2.1.3 강화 학습 기반 혼잡제어 알고리즘

온/오프라인 학습 기반 혼잡제어는 변화하는 네트워크 환경에 따라 실시간으로 학습하기에 어려움이 있다. 그렇기 때문에 여러 환경에 적합한 모델을 학습하는 데 상당한 시간이 소요된다. 이러한 단점을 보완하고 통신 도중 실시간 학습을 통하여 혼잡제어의 성능을 향상시키기 위한 여러 강화 학습 기반의 혼잡제어 알고리즘이 제안되었다.

대표적으로 패킷 지연 시간의 최소화에 초점을 맞춘 속도 기반 혼잡제어 알고리즘인 Aurora[19], 온/오프라인 학습과 실시간 학습의 하이브리드인 Orca[20], 높은 처리량을 위한 DQN과 NewReno 기반의 TCP-DQN[21], 종단 간 기능 및 네트워크 신호 기반의 Owl[22]와 같은 혼잡제어 알고리즘들을 예로 들 수 있다.

2.2 강화 학습과 마르코프 결정 과정

강화학습의 구성은 크게 에이전트, 행동, 상태, 환경, 보상으로 나눌 수 있다[23]. 에이전트는 행동을 행하는 주체로써, 현재 환경의 상태에 따라 미리 정의된 행동을 실행하면 이에 따른 보상 값을 얻는다. 또한, 이전 행동에 대한 피드백을 받고 보상을 최대화하도록 학습하게 되는데, 이때 동작을 선택하는 의사 결정 전략을 정책이라고 한다.

환경은 에이전트가 속한 곳으로 미로, 체스판, 네트워크 통신 환경 등을 예시로 들 수 있으며 상태는 에이전트가 속한 환경의 현재 상태를 나타내는 집합이다. 상태는 학습 시 입력 값으로 사용되며, 에이전트가 동작을 수행하면 변화한다.

보상 값은 상태가 변화하기 전후의 값을 통해 계산되는데, 이는 특정한 상태에서 행동을 수행할 때 에이전트가 얻을 수 있는 이득으로 누적 보상 값이 가장 큰 동작을 선택함으로써 피드백을 받는다.

강화 학습은 마르코프 결정 과정 (Markov decision process, MDP)라는 확률적 모델[24]로 표현할 수 있으며, MDP의 목적은 무작위한 보상 값에 대한 누적 함수를 최대화하는 정책 (π)을 선택하는 것이다. 학습을 통해 목적에 부합하는 정책을 얻은 에이전트는 상태에 따라 적절한 행동을 선택하여 높은 보상을 얻도록 동작한다.

2.3 Deep Q-Network

2013년, Google DeepMind는 강화학습을 기반으로 기존의 Q-러닝을 개선한 DQN[23]을 제안하였다. Q-러닝은 환경 변수가 복잡할수록 더 큰 메모리 용량과 긴 Q-table의 탐색 시간을 필요로 한다는 단점을 가지고 있다. DQN은 이러한 문제를 극복함과 Convolutional Neural Network (CNN)을 적용하여 데이터 효율성을 높이고, 데이터와 상태 간의 상관 관계를 줄이며, Local minima 현상으로 인한 학습 방해를 방지하도록 개선되었다.

DQN은 평균 제곱 오차(Mean Squared Error, MSE)와 같은 gradient descent를 사용하여 손실 함수를 계산하고, 최적의 정책을 업데이트 하도록 동작하며, 매 에피소드마다 재생 버퍼에 [상태(s), 보상 값(r), 행동 (a), 다음 상태 (s^')]로 구성된 튜플 θ를 저

장한다. 재생 버퍼에서 여러 무작위 매개변수를 샘플링하여 학습에 사용하기 때문에 연속된 샘플 간의 상관관계를 줄일 수 있고, 그렇기 때문에 통신이 발생하는 네트워크 환경에서 실시간으로 학습하기에 적합하다.

따라서 본 논문 저자들은 <표 1>과 같이 DQN 변수를 통신 환경에 매핑하였으며, 학습에 사용된 상태 값은 혼잡제어에서 사용되는, 측정 가능한 RTT 및 cwnd와 같은 변수를 사용한다. 행동은 각 혼잡제어의 cwnd 조정 알고리즘을 기반으로 수정하였으며, 보상 값은 높은 처리량과 낮은 패킷 손실과 같은 최상의 성능을 목표로 학습하도록 수 있도록 설정하였다.

표 1. DQN 변수와 TCP 혼잡제어에서 사용되는 매개변수의 매핑
Table 1. Mapping DQN's variables to parameters used in TCP congestion control.

DQN 변수	TCP 혼잡제어의 매개변수
에이전트	혼잡제어 알고리즘
환경	혼잡제어를 사용하여 통신을 수행하는 네트워크 환경
a	cwnd 조절
s	통신 중 에이전트에서 확인할 수 있는 Cwnd, RTT와 같은 상태 변수들로, 학습 시 사용되는 값
r	에이전트가 특정 *a*를 취했을 때 얻을 수 있는 값, 에이전트는 *r*을 최대화하도록 Cwnd를 조절

Ⅲ. TCP 혼잡제어와 강화 학습

3.1 DQN-based NewReno

NewReno는 기본적인 혼잡제어 알고리즘으로, AIMD 알고리즘을 사용하여 cwnd를 조절한다. 전송한 패킷과 동일한 수의 ACK를 수신하면 cwnd를 1 세그먼트 크기만큼 선형적으로 증가시키기 때문에 증가속도가 더디다. 또한, NewReno는 손실 기반 알고리즘이기 때문에, 혼잡이 발생할 때까지 cwnd를 무조건 증가시킨다. 혼잡이 발생할 경우 cwnd를 절반으로 감소시키고 패킷을 재전송 하기 때문에 처리량이 낮아지게 되는 문제점을 가지고 있다.

3.1.1 DQN-based NewReno의 흐름도 및 상태 공간

문제점의 해결 및 성능 개선을 위하여 본 논문 저자들은 기존의 NewReno에 DQN을 적용한 알고리즘을 구현하였으며, DQN 모델 학습을 위해 사용한 3개의 파라미터는 다음과 같다.

1) Threshold: 혼잡제어의 느린 시작과 혼잡 회피 동작을 판단하기 위한 값으로, 마지막 혼잡발생 시점의 cwnd에 0.7의 감소지수를 곱하여 계산된다.
2) cwnd: TCP가 ACK 패킷을 수신하기 전까지 네트워크로 전송할 수 있는 최대 데이터양을 나타내는 상태 변수로써, 패킷 단위로 표시된다.
3) RTT: 패킷이 목적지에 도착한 후 응답이 발신자에게 돌아올 때까지 걸리는 시간으로, 네트워크 속도와 안전성을 진단할 때 사용할 수 있다. 단위는 ms를 사용하였다[13].

Q-value를 계산하기 위해 3개의 hidden layer와 30개의 노드를 가진 DQN 모델을 사용하였으며, 활성화 함수로 ReLU와 Dropout을 사용하였다.

DQN-based NewReno는 <그림 1>과 같이 동작한다. 통신이 시작되면 slow start는 기존의 NewReno와 동일한 알고리즘대로 동작하며, congestion avoidance 동작에 진입하면 학습을 위한 튜플 θ가 2000개 쌓일 때까지 학습이 진행되지 않고 랜덤한 행동을 선택하여 동작한다. 이후 실시간으로 학습이 진행되며, 전송한 패킷 수와 동일한 ACK가 도달하여 cwnd를 상승

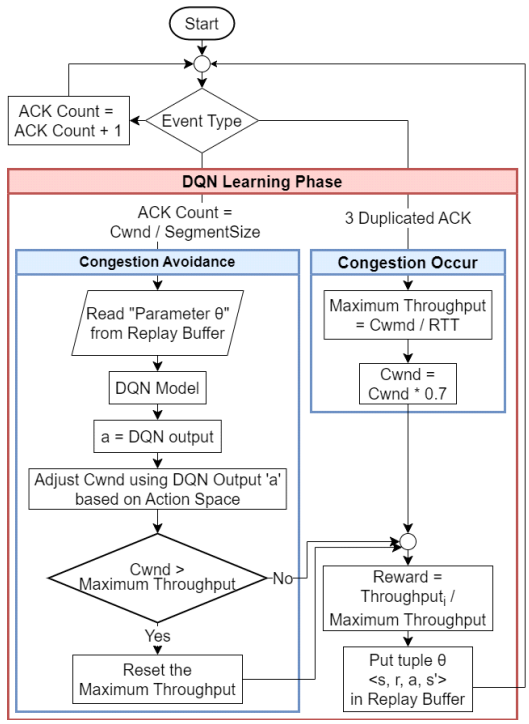

그림 1. DQN-based NewReno의 흐름도
Fig. 1. Flowchart of DQN-based NewReno.

시켜야 할 때[4] 학습에 따른 행동을 선택하여 cwnd를 유동적으로 조절하며 동작한다.

3.1.2 DQN-based NewReno의 행동공간[13]

$$a = \begin{cases} 0: Cwnd = Cwnd - (10 \times segmentSize) \\ 1: Cwnd = Cwnd - (3 \times segmentSize) \\ 2: Cwnd = Cwnd - segmentSize \\ 3: Cwnd = Cwnd \\ 4: Cwnd = Cwnd + segmentSize \\ 5: Cwnd = Cwnd + (3 \times segmentSize) \\ 6: Cwnd = Cwnd + (10 \times segmentSize) \end{cases} \quad (1)$$

$$Action\ Select\ Probability = \begin{cases} \varepsilon: Select\ Random\ Action \\ 1-\varepsilon: Select\ Best\ Reward\ Action \end{cases}, \ \varepsilon = 0.015 \quad (2)$$

우선, DQN 학습 이후 출력 값은 <식 1>과 같이 0~6의 값을 가진다. 이를 통해 에이전트는 출력 값 a가 3보다 작으면 혼잡 발생 위험이 크다고 판단하여 cwnd를 감소시키고, 3보다 크면 더 많은 대역폭을 사용할 수 있다고 판단하고 cwnd를 증가시킨다. 이 cwnd 조절식을 사용한 이유는 owl 혼잡제어 알고리즘에 의해 제안된, 이상적인 값이기 때문이다[22].

행동의 선택은 epsilon-greedy 알고리즘을 통해 이루어지며, <식 2>와 같이 동작한다. DQN 에이전트는 랜덤 행동을 통해 네트워크 통신 환경이 변화하여도 일정한 확률로 무작위 동작을 선택함으로써 변화한 네트워크 환경에 맞는 새로운 최적 정책을 탐색할 수 있게 된다.

3.2 DQN-based CUBIC

NewReno기반의 DQN-based NewReno는 AIMD 알고리즘을 사용하기 때문에 링크 대역폭의 크기가 증가할수록 최대 처리량에 도달할 때까지 소요되는 시간이 증가하게 된다. 또한, 여러 플로우가 동일한 병목 링크를 공유하는 네트워크 환경에서 RTT가 다른 플로우간의 공평성과 관련된 문제가 여전히 남아 있다.

이러한 문제는 ACK를 통해서만 혼잡을 판단하기 때문으로, 이를 개선하기 위해 본 논문 저자들은 리눅스 커널에서 표준으로 사용되는 혼잡제어 알고리즘인 CUBIC에 주목하였다.

CUBIC은 고속 네트워크 환경을 위해 제안된 혼잡제어 알고리즘으로, 삼차 함수를 사용하여 비선형 cwnd 증가식을 사용하는 것이 가장 큰 특징이다. 또한, CUBIC은 cwnd 조절식에 마지막 혼잡 이벤트부터 측정된 시간을 매개변수로 사용하여 NewReno에 비해 양호한 RTT 공평성을 가진다. 하지만, CUBIC 또한 고정된 알고리즘 대로만 동작하기 때문에 단일 플로우가 링크를 사용하는 환경에서도 혼잡이 발생할 수 있다. 또한, 통신 기술의 발전으로 인해 동적 환경에서 사용 가능한 링크 대역폭이 크게 증가하였기 때문에 최대 cwnd를 탐색하는데 더 많은 시간이 필요하게 되었다. 해당 알고리즘은 기존에 제안했던 DQN-based CUBIC[12]과 전체적인 틀은 유사하지만, 세부적인 코드 최적화를 통하여 보다 실제 환경과 가깝게 학습 및 매개변수 측정이 가능하도록 수정한 버전이다.

$$K = \sqrt[3]{\frac{w_{max}(1-\beta)}{C}} \quad \text{or} \quad 0 \quad (3)$$

CUBIC은 혼잡이 발생하면, w_max를 계산하고, cwnd를 줄인다. 이후, T가 <식 3>을 통해 계산된 K보다 작을 경우, cwnd를 빠르게 증가시킨다.

w_max는 <표 2>에 따라 혼잡이 발생한 시점의 cwnd에 0.85를 곱한 값으로, 해당 값 주위에서는 cwnd의 변화가 크지 않다. 일정한 시간이 지난 이후에도 혼잡이 발생하지 않으면 CUBIC은 혼잡이 발생할 때까지 cwnd를 매우 빠르게 증가시킨다. 하지만, 사용 가능한 링크 대역폭이 클 때 cwnd 변화가 적은 구간이 매우 길어져 링크 대역폭을 제대로 사용하지 못하는 문제가 있다. 그렇기에, DQN 학습을 통한 C의 변수화로 cwnd 증가 및 업데이트 속도를 빠르게 하여 CUBIC의 성능 향상을 시도하였다.

표 2. CUBIC에서 사용하는 변수
Table 2. Variables of CUBIC.

변수명	변수 설명
β	cwnd 감소 지수 (기본 0.7)
w_{max}	$w_{max} = cwnd \times (1+\beta) / 2$
T	마지막 cwnd 감소 이후 경과 시간
C	스케일링 상수 (기본 0.4)

3.2.1 DQN-based CUBIC의 흐름도 및 상태 공간

DQN-based CUBIC은 <그림 2>의 흐름도를 따라 동작한다. 학습을 통해 Cwnd_cnt를 즉시 줄이거나, cwnd 스케일링 상수를 조절함으로써 유동적으로 네트워크 환경에 따라 동작을 변화하도록 학습한다. 학

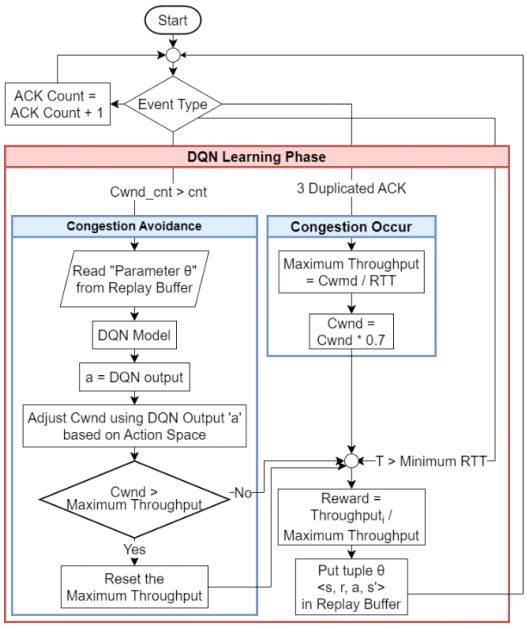

그림 2. DQN-based CUBIC의 흐름도
Fig. 2. Flowchart of the DQN-based CUBIC.

습을 위해 T가 측정된 최소 RTT보다 크거나 cwnd가 변화하는 경우, 에이전트는 재생 버퍼에 매개변수를 삽입한다.

학습에 사용되는 매개변수는 5개로, DQN-based NewReno에서 사용한 threshold, cwnd, RTT에 w_max와 T를 추가였고, 이유는 다음과 같다.

1) w_max: CUBIC 혼잡제어 알고리즘에서 오목 함수와 볼록 함수를 구분하기 위한 변수로, <표 2>에서 표기한 식을 통해 계산된다. Cwnd의 증가 속도를 조절하기 위한 변수로 적합하며, 패킷 단위를 사용하였다.
2) T: 마지막 혼잡 이벤트 발생 이후 경과된 시간을 측정한 값으로, CUBIC은 네트워크 활용률, flow 간의 공평성 및 안정성을 위하여 시간을 측정한 뒤 cwnd 조절 함수에 적용한다. CUBIC의 장점을 유지하기 위해 학습에 사용하였으며, 단위는 ms이다[13].

DQN-based CUBIC에 사용된 모델은 5개의 입력 값, 각 50개의 노드를 가진 5개의 hidden layer로 구성하였으며, 활성화 함수로는 ReLU와 Dropout을 사용하였다. 에이전트는 현재 상태를 학습한 뒤 다음과 같이 4가지 범주로 분류하고, 그에 따라 행동을 선택한다. DQN-based CUBIC 또한 slow start 부분은 기존 CUBIC과 동일하게 동작하며, 학습을 위한 튜플 θ가 replay buffer에 2000개 쌓일 때까지 학습이 진행되지 않는다. CUBIC의 동작에 따라 cwnd_cnt값이 cnt값보다 커지면[6], 학습 결과에 따른 행동을 선택하여 유동적으로 cwnd를 조절한다.

3.2.2 DQN-based CUBIC의 행동 공간[13]

$$Target\ Cwnd = \begin{cases} (C \times a)(T-K)^3 + w_{max}, K = \sqrt[3]{\frac{w_{max}(1-\beta)}{C \times a}}, & a = 3 \\ (T-K)^3 + w_{max}, \quad K = \sqrt[3]{\frac{w_{max}(1-\beta)}{C}}, & a = 2 \end{cases} \tag{4}$$

$$Avoid\ Congestion = \begin{cases} Cwnd = Cwnd, & a = 1 \\ Cwnd = Cwnd - SegmentSize, & a = 0 \end{cases} \tag{5}$$

에이전트는 <식 4, 식 5>와 같이 DQN 출력 값에 따라 cwnd 조정식을 선택하는데, a가 2 혹은 3일 때 <식 4>를 사용하여 cwnd를 빠르게 증가시키거나, 일반적인 CUBIC과 동일한 속도로 증가시킨다. 하지만, <식 5>와 같이 출력 값이 0일 경우 혼잡 위험이 높다고 판단하고 cwnd를 세그먼트 크기만큼 감소시키고, a가 1이면 현재 cwnd를 유지한다. DQN-based CUBIC의 행동 공간은 0~3의 값을 가지며, 이는 cwnd 조절이 과도하게 공격적이지 않도록 방지하기 위함이다. DQN-based CUBIC 또한 <식 2>의 epsilon-greedy 알고리즘을 통해 행동을 선택한다.

3.3 보상 함수

$$\begin{cases} Rewards = \frac{Throughput_i}{Maximum\ Throughput} \\ Rewards = -1,\ when\ packet\ loss\ occurs \end{cases} \tag{6}$$

$$Throughput_i = \frac{Cwnd_i}{RTT_i} \tag{7}$$

<식 6>은 링크 이용률을 나타내는 보상 함수이다. 링크 이용률은 플로우에서 현재 사용중인 처리량을 링크에서 사용 가능한 최대 처리량으로 나눈 값을 나타내며, 단위는 bps (bit per second)를 사용한다. $Cwnd_i$ 및 RTT_i는 DQN 에이전트가 혼잡 발생 없이 전송이 완료된, i^{th} 세그먼트에 대한 ACK를 수신할 때 측정된 매개변수로, 이를 사용하여 $throughput_i$ 값을 계산한다. Maximum Throughput은 DQN 학습 및 네

트워크에서 통신 중 혼잡이 발생하는 순간 측정된 최대 값을 나타낸다.

링크 사용률이 1에 가까울수록 에이전트가 전송하는 데이터와 측정된 링크의 최대 전송 용량이 비슷함을 나타내며, 특히 단일 플로우가 링크를 사용할 때 최대의 처리량을 보장할 수 있도록 모델의 학습 방향을 유도한다. 패킷 손실과 같은 혼잡으로 인한 cwnd의 감소가 발생하면 해당 상태에서의 보상 값을 -1로 설정하여 혼잡 위험이 있음을 학습할 수 있도록 하였다. 본 논문에서는 처리량을 최대화하는 것을 목표로 학습을 시도하였기에 제안한 두 알고리즘 모두 동일한 보상 함수를 사용하였다.

Ⅳ. 성능 평가

4.1 성능 평가를 위한 실험 환경

실제 환경과 유사하게 네트워크 통신 실험을 구성하기 위해 오픈 소스 이산 네트워크 시뮬레이터인 Network Simulator-3 (NS3)를 사용하였다. NS-3는 패킷 데이터 네트워크를 위한 모델과 C++를 이용한 시뮬레이션 실험용 엔진을 제공한다. 또한, OpenAI Gym toolkit의 일종인 NS3-gym 프레임워크를 이용한 기계 학습의 적용이 가능하여 실험에 적합하다.

<그림 3>은 NS3-gym의 구조를 나타낸 것으로, NS-3와 OpenAI Gym 간의 정보 교환을 위해 C++과 Python으로 작성한, 두개의 모듈로 구성된 toolkit이다. 통신은 프로토콜 버퍼를 사용하여 ZMQ 소켓을 통해 연결된 뒤 이루어진다.

OpenAI Gym은 NS-3로부터 네트워크 상태에 관한 정보를 수신하고, DQN 학습을 진행하며, cwnd를 조절하는 행동을 취하는 에이전트로 구성되었다. 에이전트에 의해 조절된 cwnd가 다시 NS-3로 업데이트되면, 이에 따라 패킷 전송이 진행되며 해당 작업을 반복하며 DQN 학습이 진행된다. 에이전트가 혼잡제어 역할을 하기에, NS-3는 OpenAI Gym은 통신 시작 후 동일한 내부 타이머를 통해 시점을 동기화한다.

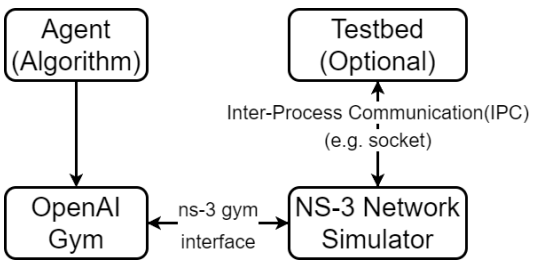

그림 3. OpenAI Gym의 구조
Fig. 3. Architecture for OpenAI Gym.

4.2 실험 구성

실제 통신이 이루어지는 네트워크 환경은 미래 상태를 예측하기 어렵고, 다양한 상태가 존재하기 때문에 여러 환경에서 비교 실험이 필요하다. 기존의 DQN-based CUBIC을 제안한 논문에서는 서로 다른 혼잡제어를 사용하는 플로우 간의 공평성에만 초점을 두고 실험을 진행하였기에[13], 명확한 성능 비교 및 분석이 추가로 필요하다. 따라서 본 논문에서는 다음과 같은 토폴로지를 사용해 실험을 진행하였다.

<그림 4>는 단일 플로우 토폴로지 실험 환경을 나타낸다. 네트워크 통신 중 병목 링크의 대역폭이 사용자와 라우터를 연결하는 액세스 링크의 대역폭보다 작으면 하나의 플로우가 네트워크를 사용하는 경우에도 혼잡이 발생할 수 있다. 이는 기존 혼잡제어 알고리즘의 고질적인 문제로, 해당 문제를 개선하여 항상 사용가능한 최대 처리량을 유지할 수 있도록 구현한 뒤 성능 평가 실험을 진행하였다.

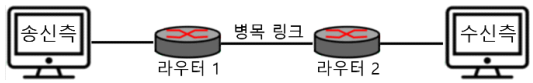

그림 4. 단일 플로우 토폴로지
Fig. 4. Single flow topology.

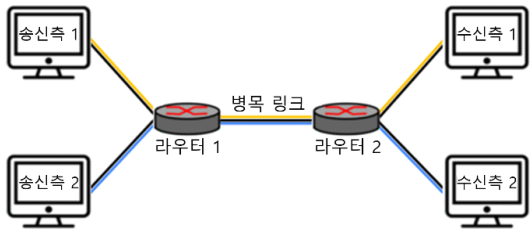

그림 5. 다중 플로우 토폴로지
Fig. 5. Multi flow topology.

표 3. 기본 실험 설정
Table 3. Default experiment settings.

구분	실험 설정
사용- 혼잡제어 알고리즘	NewReno, CUBIC DQN-based NewReno, DQN-based CUBIC
병목 링크 지연	30 ms
액세스 링크 지연	0.01 ms
병목 링크 대역폭	100 Mbps
액세스 링크 대역폭	150 Mbps
실험 시간	50 Seconds

<그림 5>는 여러 플로우들이 동일한 병목 링크에서 경쟁하는 환경을 구성한 토폴로지로, 실제 네트워크 환경에서는 단일 플로우가 홀로 링크를 독점하는 경우가 거의 없기 때문에, 더욱 중요한 실험이다.

서로 다른 RTT를 가진 플로우 또는 서로 다른 혼잡제어 알고리즘을 사용하는 플로우끼리 경쟁하는 환경 등의 여러 실험이 수행되었으며, 실험의 기본 설정은 <표 3>과 같다.

4.3 실험 및 성능 평가

4.3.1 단일 flow 환경에서 성능 비교

첫 번째 실험으로, 각각의 TCP 혼잡제어 알고리즘을 사용하는 하나의 플로우에 대한 처리량을 비교하였다. 실험에 사용된 매개 변수는 <표 3>과 대부분 동일하지만, 병목 상황을 구성하기 위해 병목 링크 대역폭을 50 Mbps, 액세스 링크 대역폭을 100 Mbps로 설정하였다.

뒤이어 두 번째는 실험은 설정을 <표 3>과 동일하게 하여 첫번째 실험에 비해 넓은 대역폭을 가진 네트워크 환경에서 각 혼잡제어 알고리즘의 성능 비교 실험을 진행하였다. 두 실험에서는 <그림 4>의 토폴로지로 구성하였다.

NewReno와 CUBIC은 병목 링크와 액세스 링크의 대역폭 차이로 인해 단일 플로우에서도 혼잡이 계속 발생하지만, DQN 기반 혼잡제어는 링크에서 사용가능한 최대 처리량을 학습하고, cwnd를 유지하도록 동작하기 때문에 <그림 6, 그림 7>과 같이 학습이 완료된 시점부터 처리량을 일정하게 유지한다.

<그림 8>는 실험으로 측정된 각 혼잡제어 알고리즘의 평균 처리량 결과를 나타내며, NewReno는 모든 실험에서 가장 낮은 평균 처리량을 보였으며, CUBIC의 평균 처리량은 DQN 학습 기반 혼잡제어보다 약간 낮았다.

DQN-based NewReno는 빠른 cwnd 증가로 인해 평균 처리량이 기존의 혼잡제어 알고리즘보다 높고 DQN-based CUBIC과 비슷함을 확인할 수 있다. 하지만 학습이 완료되기 전까지 혼잡이 자주 발생해 불필요한 패킷 손실이 발생하고, 학습이 완료되어 cwnd가 유지되기까지 소요되는 시간이 약 25초 정도 걸리는 등의 불안정함을 보였다.

DQN-based CUBIC은 가장 높은 평균 처리량을 보이고, 패킷 손실 또한 가장 적었다. 최대 처리량을 학습한 뒤 최대 cwnd로 수렴하는 데 걸리는 시간 또한 가장 짧지만, 네트워크 상태의 변화에 더 민감하기 때문에 cwnd가 지속적으로 미세하게 변화한다.

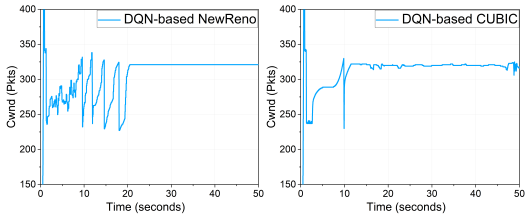

그림 6. DQN 기반 혼잡제어의 cwnd (병목 링크 대역폭: 50 Mbps)
Fig. 6. Cwnd of DQN-based congestion controls (bottleneck link bandwidth: 50 Mbps).

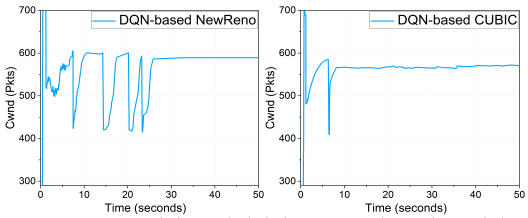

그림 7. DQN 기반 혼잡제어의 cwnd (병목 링크 대역폭: 100 Mbps)
Fig. 7. Cwnd of DQN-based congestion controls (bottleneck link bandwidth: 100 Mbps).

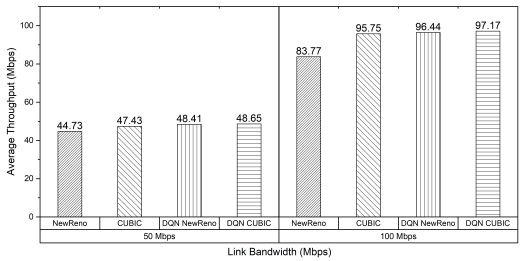

그림 8. 각 혼잡제어 알고리즘의 평균 처리량
Fig. 8. Average throughput of each congestion controls

이러한 실험을 통해 단일 플로우가 병목 링크를 통과할 때 대역폭이 증가할수록 DQN 학습 기반 혼잡제어 알고리즘들이 기존 혼잡제어보다 높은 처리량과 링크 이용률을 가짐을 확인할 수 있었다. 이후, 우리는 보다 정확한 성능 평가를 위해 다중 플로우 토폴로지를 사용하여 추가 실험을 진행하였다.

4.3.2 다른 혼잡제어를 사용하는 플로우 사이의 공평성 비교 실험

서로 다른 혼잡제어 알고리즘을 사용하는 플로우 사이의 공평성 비교 실험은 서로 다른 혼잡제어 알고리즘을 가진 플로우가 동일한 병목 링크를 사용하며 경쟁할 때 공평성 및 공격성을 평가하기 위한 실험이다. 다양한 혼잡제어를 사용하는 실제 네트워크 환경

의 특성상 cwnd를 공격적으로 빠르게 증가시키는 플로우가 링크를 선점하여, 다른 혼잡제어

알고리즘을 사용하는 플로우가 대역폭을 제대로 활용할 수 없게 만드는 공평성 문제가 발생할 수 있다. 제안한 DQN 기반 혼잡제어 알고리즘은 항상 최대 처리량을 유지하도록 학습되기 때문에 다른 혼잡제어 알고리즘과의 공평성을 확인할 필요성이 있다. 해당 실험 설정은 <표 4>와 같이 구성하였고, 네트워크 토폴로지는 <그림 5>와 같이 구성하였다.

실험 4.3.2의 첫번째는 DQN-based NewReno가 NewReno 혹은 CUBIC을 사용하는 플로우와 동일한 병목 링크에서 경쟁할 때의 성능 비교 실험으로, 결과는 <그림 9>, <표 5>와 같다.

제안한 DQN-based NewReno는 다른 플로우의 대역폭 사용에 느리게 반응하여 혼잡이 발생하며, cwnd 증감의 속도가 매우 빠르기 때문에 처리량의 변화폭이 크다. 그렇기 때문에 DQN-based NewReno가 일반 NewReno와 경쟁할 때 링크 대역폭을 선점하여 보다 높은 처리량을 보였으며, 두 플로우 간의 공평성을 비교하는 지표인 Jain's fairness index[25]는 0.9886이다.

표 4. 서로 다른 혼잡제어를 사용하는 플로우 간의 공평성 비교 실험 설정
Table 4. Setting of fairness comparison experiment between flows using different congestion controls.

구분	실험 설정
플로우 1의 혼잡제어 알고리즘	DQN-based NewReno or DQN-based CUBIC
플로우 2의 혼잡제어 알고리즘	NewReno or CUBIC
액세스 링크 지연	0.01 ms
병목 링크 지연	30 ms
액세스 링크 대역폭	100 Mbps
병목 링크 대역폭	100 Mbps
실험 시간	50 Seconds

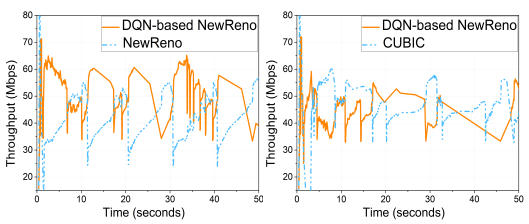

그림 9. DQN-based NewReno와 기존 혼잡제어 알고리즘 간의 처리량 비교[13]
Fig. 9. Comparison of throughput between DQN-based NewReno and existing congestion control algorithm.

표 5. DQN-based NewReno와 기존 혼잡제어 알고리즘이 같은 병목링크에서 경쟁할 때의 평균 처리량
Table 5. Average throughput when DQN-based NewReno and existing congestion control algorithm compete on the same bottleneck link.

실험 구분	플로우 구분	평균 처리량
DQN-based NewReno vs NewReno	플로우1: DQN-based NewReno	53.60 Mbps
	플로우2: NewReno	43.19 Mbps
DQN-based NewReno vs CUBIC	플로우1: DQN-based NewReno	48.39 Mbps
	플로우2: CUBIC	47.74 Mbps

반면 CUBIC은 NewReno에 비해 빠른 cwnd 상승 속도를 가지고 있기 때문에 DQN-based NewReno와 비슷하게 링크 대역폭을 사용하였다. Jain's fairness index 또한 높은 0.9999의 수치를 보였고, 링크 이용률은 두 실험 모두에서 96%로 측정되었다.

다음으로 DQN-based CUBIC이 NewReno 혹은 CUBIC과 같은 병목 링크를 사용하며 경쟁하는 네트워크 환경에서 실험을 진행하였다. 실험을 통해 측정된 결과는 <그림 10>, <표 6>과 같다.

DQN-based CUBIC은 링크를 공격적으로 점유하지 않고, NewReno의 cwnd 증가에 따라 혼잡이 발생하지 않도록 조절하였다. 해당 실험에서 링크 이용률은 95%, Jain's fairness index는 0.9937로 DQN-based NewReno보다 공평성은 양호하였으나,

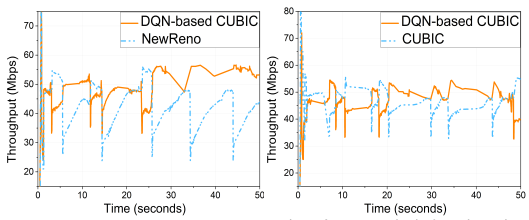

그림 10. DQN-based CUBIC과 기존 혼잡제어 알고리즘 간의 처리량 비교[13]
Fig. 10. Comparison of throughput between DQN-based CUBIC and existing congestion control algorithm.

표 6. DQN-based CUBIC과 기존 혼잡제어 알고리즘이 같은 병목링크에서 경쟁할 때의 평균 처리량
Table 6. Average throughput when DQN-based CUBIC and existing congestion control algorithm compete on the same bottleneck link.

실험 구분	플로우 구분	평균 처리량
DQN-based CUBIC vs NewReno	플로우1: DQN-based CUBIC	51.14 Mbps
	플로우2: NewReno	43.58 Mbps
DQN-based CUBIC vs CUBIC	플로우1: DQN-based CUBIC	49.07 Mbps
	플로우2: CUBIC	48.40 Mbps

cwnd를 조절하는 과정 때문에 링크 이용률은 낮게 측정되었다.

DQN-based CUBIC이 일반 CUBIC과 경쟁하는 네트워크 환경에서 두 혼잡제어 알고리즘의 처리량 차이는 1 Mbps가 넘지 않을 정도로 공평성이 우수했으며, 링크 이용률 또한 97%로 높게 측정되었다.

본 실험을 통해 DQN-based NewReno가 기존의 NewReno와 경쟁할 때는 비교적 공격적인 cwnd 상승으로 인해 비교적 나쁜 공평성을 보였으나, CUBIC과 경쟁하는 환경에서는 양호한 공평성을 보였다. 하지만, DQN-based CUBIC은 너무 공격적인 cwnd 증가를 보이지 않고 공평성을 유지하도록 동작하기 때문에, 기존의 NewReno와 CUBIC 두 혼잡제어 알고리즘과 사용해도 공평성에 문제가 없음을 확인할 수 있었다.

4.3.3 RTT 공평성 비교 실험

세번째로는 동일한 혼잡제어 알고리즘을 사용하는 두 플로우가 하나의 병목 링크를 공유하며 경쟁하는 네트워크 환경에서의 RTT 공평성을 비교하기 위한 실험을 진행하였다.

하나의 병목 링크를 공유하는 두 플로우가 같은 혼잡제어 알고리즘을 사용하는 경우, 두 플로우는 서로 경쟁하며 링크 대역폭을 사용한다. 그러나, 두 플로우의 RTT 값이 다를 경우 NewReno와 같은 혼잡제어는 짧은 RTT를 가진 플로우가 링크를 선점하게 되고, 다른 플로우는 링크 대역폭을 제대로 사용하지 못하는 문제가 존재한다. CUBIC은 cwnd 조절을 RTT에 의존하지 않기 때문에 해당 문제가 해결되었으나, DQN-based NewReno의 경우 기존 NewReno를 기반으로 한 알고리즘이기 때문에 해당 문제가 여전히 존재할 수 있다. 처리량이 향상되었을지라도, RTT 공평성 문제가 해결되지 않았으면 현실에 적용하기 어렵기 때문에 RTT 공평성 측정 실험의 필요성을 느끼고 비교 및 평가 실험을 진행하였다. 실험 설정은 <표 7>과 같고, 토폴로지는 <그림 5>와 같이 구성하였다.

세 번째 실험에 대한 결과는 <그림 11>의 처리량 그래프와 <표 8>의 평균 처리량으로 정리된다.

두 플로우가 혼잡제어 알고리즘으로 일반 NewReno를 사용한 실험에서는 RTT가 짧은 플로우가 링크 대역폭을 더 공격적으로 점유하여 4 Mbps 높은 평균 처리량을 보였고, Jain's fairness index는 0.9979로 측정되었다. 반면 CUBIC은 RTT 공평성을 개선한 혼잡제어 알고리즘이기 때문에 두 플로우 간의 평균 처리량 차가 NewReno보다 작았고, 공평성 지수 또한 0.9996으로 비교적 양호했다.

제안한 DQN-based NewReno는 RTT 공평성이 매우 떨어짐을 확인할 수 있다. 짧은 RTT로 인해 먼저 링크 대역폭을 점유한 플로우가 최대 cwnd를 학습하고 유지하기 때문으로, RTT가 긴 플로우는 RTT가 짧은 플로우에 비해 약 70 Mbps 낮은 평균 처리량을 가

표 7. RTT 공평성 비교 실험 설정
Table 7. Setting of RTT fairness comparison experiments.

구분	실험 설정
혼잡제어 알고리즘	NewReno, CUBIC, DQN-based NewReno, DQN-based CUBIC
플로우1의 RTT	40 ms
플로우2의 RTT	60 ms
액세스 링크 대역폭	100 Mbps
병목 링크 대역폭	100 Mbps
실험 시간	50 Seconds

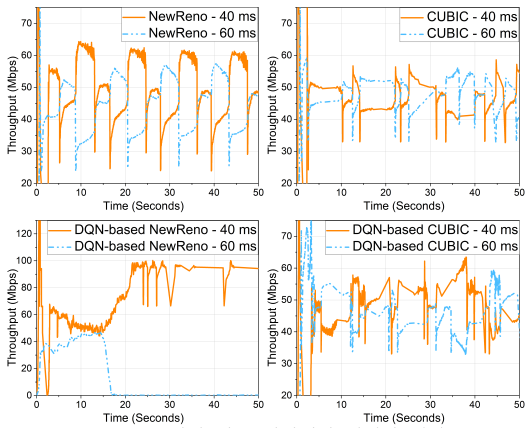

그림 11. RTT 공평성 비교 실험에서 처리량 변화
Fig. 11. Throughput changes in RTT fairness comparison experiments.

표 8. RTT 공평성 비교 실험의 평균 처리량
Table 8. Average throughput of RTT fairness comparison experiments.

혼잡제어 알고리즘	RTT	평균 처리량
NewReno	40 ms	49.94 Mbps
	60 ms	45.58 Mbps
CUBIC	40 ms	47.38 Mbps
	60 ms	49.22 Mbps
DQN-based NewReno	40 ms	80.64 Mbps
	60 ms	11.91 Mbps
DQN-based CUBIC	40 ms	48.53 Mbps
	60 ms	48.02 Mbps

졌다. Jain's fairness index는 0.6445이며, 링크 이용률 또한 92.5%로 낮았다.

그러나 DQN-based CUBIC은 cwnd 증가가 RTT에 의존하지 않기 때문에 서로 다른 RTT를 가진 두 플로우가 비슷한 평균 처리량을 보이며 링크 대역폭을 이용하였다. Jain's fairness index는 0.9999이고, 링크 이용률 또한 96.5%로 높았다.

4.3.4 사용가능한 링크 대역폭이 변화하는 네트워크 환경에서의 성능 확인 실험

정적인 네트워크 환경에서는 학습이 간단히 이루어지고, 학습에 따른 성능 향상이 명확하게 확인되지만, 학습된 환경이 변화할 때의 성능을 확인할 수 없다.

따라서, 마지막으로 동적인 환경에서 DQN 학습 기반 혼잡제어 알고리즘들의 성능을 확인하기 위해, 사용가능한 링크 대역폭이 계속 변화하는 네트워크 환경에서 성능비교 실험을 진행하였다. 실험을 위해 <그림 5>의 토폴로지를 사용하였고, 실험 설정은 <표 9>와 같이 구성하였다.

우선, DQN-based NewReno는 앞선 실험들의 결과에서 볼 수 있듯, <그림 12>와 같이 공격적인 cwnd 증가로 인해 매우 잦은 혼잡이 발생하여 불안정한 모습을 보였다.

DQN-based NewReno를 사용한 마지막 실험의 평균 처리량은 <표 10>과 같다. 0~49초의 구간에서 경쟁하는 다른 플로우로 인해 잦은 혼잡으로 인한 패킷 손실이 발생하여 92.8 %의 링크 이용률을 가졌다. 50 ~ 99초의 실험 구간에서는 50 Mbps의 대역폭을 가진 병목 링크에서 두 플로우가 경쟁하였고, 링크 이용률은 92.4 %이다. 마지막 100 ~ 150초의 실험 구간에서는 CUBIC의 단점 중 하나인, cwnd가 정체되었고, DQN-based NewReno는 링크를 선점하여 빠르게 cwnd를 복구하였으나, 지속적으로 혼잡이 발생하였다.

DQN-based NewReno는 빠른 cwnd 복구와 높은 처리량을 보였지만, 잦은 혼잡이 발생하였고 150초의 실험 시간동안 플로우1에서 손실된 패킷의 수는 88개로 측정되었다.

반면 DQN-based CUBIC은 사용 가능한 링크 대역폭이 감소하거나, 경쟁하는 플로우가 많은 대역폭을 사용해도 <그림 13>과 같이 혼잡이 비교적 적게 발생하였다. <표 11>에 따르면, 0~49초의 구간에서 학습 완료 후 처리량을 유지함에 따라 94.8 %의 링크 이용률을 보였다. 50 ~ 99초의 실험 구간에서는 50 Mbps의 병목 링크 대역폭을 두 플로우가 경쟁하였고, DQN-based NewReno보다 높은 95 %의 링크 이용률이 측정되었다. 마지막 100 ~ 150초의 실험 구간에서 DQN-based CUBIC은 빠른 cwnd 증가로 인해 기존 CUBIC보다 높은 처리량을 보임과 동시에 측정된 손

표 9. 가용 링크 대역폭이 변화하는 네트워크 환경에서 성능 확인 실험
Table 9. Setting of performance comparison experiments in environments where the available link bandwidth is changing.

구분	실험 설정
플로우1의 혼잡제어 알고리즘	DQN-based NewReno or DQN-based CUBIC
플로우2의 혼잡제어 알고리즘	CUBIC
플로우1의 액세스 링크 대역폭	100 Mbps
플로우2의 액세스 링크 대역폭	10 → 50 → 100 Mbps
병목 링크 대역폭	100 → 50 → 100 Mbps
대역폭 변경 주기	50 s
실험 시간	150 Seconds

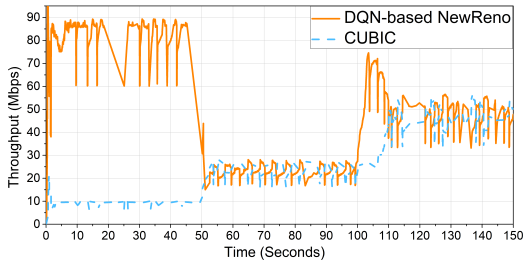

그림 12. 가용 링크 대역폭이 변화하는 실험 환경에서 DQN-based NewReno의 처리량의 변화
Fig. 12. Changes in throughput of DQN-based NewReno in an experiment environment where available link bandwidth is changing.

표 10. 가용 링크 대역폭이 변화하는 실험 환경에서 DQN-based NewReno의 평균 처리량
Table 10. Average throughput of DQN-based NewReno in an experiment environment where available link bandwidth is changing.

혼잡제어 알고리즘	실험 시간	평균 처리량
DQN-based NewReno	0~49 s	83.36 Mbps
	50~99 s	23.51 Mbps
	100~150 s	50.79 Mbps
CUBIC	0~49 s	9.42 Mbps
	50~99 s	22.69 Mbps
	100~150 s	44.43 Mbps

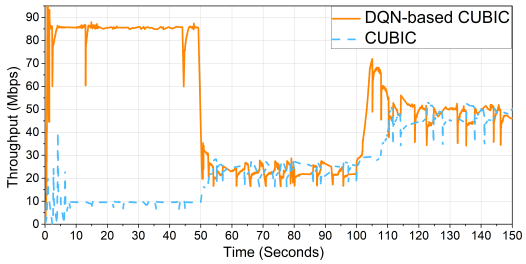

그림 13. 가용 링크 대역폭이 변화하는 실험 환경에서 DQN-based CUBIC의 처리량의 변화
Fig. 13. Changes in throughput of DQN-based CUBIC in an experiment environment where available link bandwidth is changing.

표 11. 가용 링크 대역폭이 변화하는 실험 환경에서 DQN-based CUBIC의 평균 처리량
Table 11. Average throughput of DQN-based CUBIC in an experiment environment where available link bandwidth is changing.

혼잡제어 알고리즘	실험 시간	평균 처리량
DQN-based CUBIC	0~49 s	85.25 Mbps
	50~99 s	24.1 Mbps
	100~150 s	52.18 Mbps
CUBIC	0~49 s	9.53 Mbps
	50~99 s	23.39 Mbps
	100~150 s	44.70 Mbps

실 패킷은 44개로, DQN-based NewReno보다 안정적이었다.

두 알고리즘 모두 네트워크 환경이 변화할 때 완벽하게 재학습을 완료하지는 못하였으나, 각 DQN 기반 혼잡제어 알고리즘들이 이전 실험에서 보여준 특징을 유지하는 것을 확인할 수 있었다.

V. 결 론

본 논문에서는 기존 혼잡제어 알고리즘들이 가진 문제를 해결하고, 성능을 개선하기 위해 DQN-based NewReno와 DQN-based CUBIC을 제안하였고, 성능 평가를 위해 다양한 환경에서 성능 비교 실험을 진행하였다.

DQN-based NewReno는 단일 플로우 실험에서 기존의 NewReno 보다 약 13 Mbps, CUBIC 보다 약 1 Mbps 높은 평균 처리량을 보였다. 하지만, 서로 다른 혼잡제어 알고리즘을 사용하는 두 개의 플로우 간의 공평성 비교 실험에서는 기존의 NewReno와의 공평성이 DQN-based CUBIC에 비해 좋지 않았고, RTT 공평성 실험에서는 Jain's fairness index가 0.6445로, 기존의 혼잡제어 알고리즘보다도 공평성이 낮았다. 또한, 공격적인 cwnd 증가로 인해 혼잡이 비교적 자주 발생한다는 단점을 보였다.

반면, DQN-based CUBIC은 단일 플로우 실험에서 기존의 NewReno보다 최대 14 Mbps, CUBIC보다 1.4 Mbps 높은 처리량을 보였고, 서로 다른 알고리즘을 사용하는 플로우 간의 공평성 실험과 RTT 공평성 실험에서 가장 높은 링크 이용률 및 양호한 공평성을 보였다. DQN-based CUBIC은 비교적 안정적으로 동작하며, 처리량과 공평성 모두 향상됨을 확인하였다.

모든 실험들의 결과를 통해 DQN-based NewReno는 기존 혼잡제어보다 높은 처리량을 보였으나, 공평성은 오히려 악화되어 많은 개선이 필요함을 확인할 수 있었다. 하지만, DQN-based CUBIC은 모든 실험에서 기존의 혼잡제어에 비해 높은 처리량을 보였고, 공평성또한 양호함을 확인할 수 있었다. 그렇기 때문에, DQN-based CUBIC을 차세대 혼잡제어 알고리즘의 발전 방향으로 볼 수 있지만, 평균 처리량을 더욱 높이거나 사용 가능한 링크 대역폭이 변화하는 네트워크 환경에서도 학습한 cwnd를 꾸준히 유지하도록 알고리즘을 수정하는 등 추가적인 성능 개선이 필요함을 확인하였다.

References

[1] J. Nagle, "*Congestion Control in IP/TCP Internetworks*," RFC896, Jan. 1984. (https://doi.org/10.17487/RFC0896)

[2] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314-329, Nov. 1988. (https://doi.org/10.1145/52324.52356)

[3] K. Fall, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, pp. 5-21, Jul. 1996. (https://doi.org/10.1145/235160.235162)

[4] S. Floyd, et al., "*The newreno modification to TCP's fast recovery algorithm*," RFC 2582, Apr. 1999. (https://doi.org/10.17487/RFC2582)

[5] M. Kühlewind, "*TCP SIAD: Congestion control supporting high speed and low latency*," ETH TIK Technical Report 367,

Dec. 2016.
(https://doi.org/10.48550/arXiv.1612.07947)

[6] S. Ha, et al., "CUBIC: A new TCP-Friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, pp. 64-74, Jul. 2008.
(https://doi.org/10.1145/1400097.1400105)

[7] M. A. Alrshah, et al., "Comparative study of high-speed linux TCP variants over high-BDP networks," *J. Netw. and Comput. Appl.*, vol. 43, pp. 66-75, 2014.
(https://doi.org/10.1016/j.jnca.2014.04.007)

[8] G. H. Kim, et al., "Performance evaluations of TCP in 5G mmWave cellular network," *J. KICS*, vol. 46, no. 12, pp. 2237-2250, 2021.
(https://doi.org/10.7840/kics.2021.46.12.2237)

[9] S. J. Seo and Y. Z. Cho, "Fairness enhancement of TCP congestion control using reinforcement learning," *ICAIIC IEEE*, pp. 288-291, 2022.
(https://doi.org/10.1109/ICAIIC54071.2022.9722626)

[10] W. Wei, et al., "Congestion control: A renaissance with machine learning," *IEEE Netw.*, vol. 35, pp. 262-269, Jul./Aug. 2021.
(https://doi.org/10.1109/MNET.011.2000603)

[11] J. Fang, et al., "Reinforcement learning for bandwidth estimation and congestion control in real-time communications," in *Proc. NeurIPS Wkshp. Mach. Learn. Syst.*, 2019.
(https://doi.org/10.48550/arXiv.1912.02222)

[12] S. J. Seo, et al., "A DQN-based CUBIC for TCP congestion control," *27th APCC*, pp. 419-420, 2022.
(https://doi.org/10.1109/APCC55198.2022.9943650)

[13] S. J. Seo and Y. Z. Cho, "Inter-protocol fairness evaluation of DQN-based congestion control algorithms," *ICAIIC*, pp. 693-696, 2023.
(https://doi.org/10.1109/ICAIIC57133.2023.10067107)

[14] M. Zhang, et al., "Transport layer performance in 5G mmWave cellular," *2016 IEEE Conf. Comput. Commun. Wkshp.*, pp. 730-735, 2016.
(https://doi.org/10.1109/INFCOMW.2016.7562173)

[15] N. Cardwell, et al., "BBR: Congestion-based congestion control," *Commun. ACM*, vol. 14, pp. 20-53, Oct. 2016.
(http://dx.doi.org/10.1145/3009824)

[16] M. Dong, et al., "PCC: Re-architecting congestion control for consistent high performance," *USENIX Symp. NSDI*, pp. 395-408, 2015.
https://doi.org/10.48550/arXiv.1409.7092)

[17] K. Winstein, et al., "TCP ex machina: Computer-generated congestion control," *ACM SIGCOMM*, pp. 123-34, 2013.
(https://doi.org/10.1145/2534169.2486020)

[18] F. Y. Yan, et al., "Pantheon: The training ground for internet congestion-control research," *Annu. Tech. Conf.*, pp. 731-43, 2018.
https://dl.acm.org/doi/abs/10.5555/3277355.3277426)

[19] N. Jaya, et al., "A deep reinforcement learning perspective on internet congestion control," *ICML*, 2019.
(https://proceedings.mlr.press/v97/jay19a.html)

[20] S. Abbasloo, et al., "Classic meets modern: A pragmatic learning-based congestion control for the internet," *SIGCOMM*, pp. 632-647, 2020.
(https://doi.org/10.1145/3387514.3405892)

[21] Y. Wang, et al., "An intelligent TCP congestion control method based on deep Q network," *Future Internet, MDPI*, vol. 13, no. 10, pp. 1-14, Oct. 2021.
(https://doi.org/10.3390/fi13100261)

[22] A. Sacco, et al., "Owl: Congestion control with partially invisible networks via reinforcement learning," *IEEE INFOCOM 2021*, pp. 1-10, 2021.
(https://doi.org/10.1109/INFOCOM42981.2021.9488851)

[23] V. Mnih, et al., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
(https://doi.org/10.48550/arXiv.1312.5602)

[24] A. Wrobel, "On Markovian decision models with a finite skeleton," *Math. Methods of Oper. Res.*, vol. 28, pp. 17-27, 1984. (https://doi.org/10.1007/BF01919083)

[25] R. Jain, et al., "*A quantitative measure of fairness and discrimination for resource allocation in shared computer systems*," Research Report TR-301, Dec. 1984. (https://doi.org/10.48550/arXiv.cs/9809099)

서 상 진 (Sang-jin Seo)

2021년 : 경북대학교 전자공학과 졸업

2023년 : 경북대학교 전자전기공학부 석사

2023년~현재 : ㈜한울소재과학 연구원

<관심분야> 차세대 전송 계층 프로토콜, TCP 혼잡제어, 강화학습 적용 혼잡제어 알고리즘

[ORCID:0000-0002-5464-8131]

조 유 제 (You-Ze Cho)

1982년 : 서울대학교 전자공학과 졸업

1983년 : 한국과학기술원 전기전자공학 석사

1988년 : 한국과학기술원 전기전자공학 박사

1989년~현재 : 경북대학교 전자공학부 교수

1992년~1994년 : Univ. of Toronto in Canada, 객원교수

2002년~2003년 : NIST(미국국립표준연구소) 객원연구원

<관심분야> 전차세대 이동 네트워크, 무선 애드혹 네트워크, 이동성 관리 기술, 차세대 전송 계층 프로토콜

[ORCID:0000-0001-9427-4229]