https://doi.org/10.7840/kics.2022.47.9.1330

네트워크 최대 단대단 지연 시간의 최소화를 위한 강화학습 기반 스케줄러

권 주 혁•, 류 지 혜*, 정 진 우°

Network Scheduler Based on Reinforcement Learning for Minimizing the Maximum End-to-End Latency

Juhyeok Kwon•, Jihye Ryu*, Jinoo Joung°

요 약

본 논문에서는 네트워크 최대 단대단 지연 시간의 최소화를 위한 강화학습 기반 스케줄러를 single agent 환경과 multi agent 환경에서 구현하였다. 강화학습 모델은 double deep Q-network(DDQN)와 prioritized experience replay(PER)를 사용하였다. Multi agent 환경에서는 agent가 정확한 단대단 지연 시간을 알 수 없기에 추정 지연 시간을 사용해 상태(state)와 보상(reward)을 구했다. 강화학습 기반 스케줄러의 성능을 확인하기 위해 4가지의 네트워크 토폴러지들을 구현하여 강화학습 기반 스케줄러와 FIFO, round robin(RR), 간단한 heuristic algorithm(HA)들을 비교하였다. 시뮬레이션 결과 강화학습 기반 스케줄러는 고정적인 패킷 생성 시나리오의 토폴로지들에서 모두 최대 단대단 지연 시간의 최소화를 달성하였고 FIFO, RR은 모두 달성하지 못했으며 HA는 하나의 토폴로지에서 달성하지 못했다. 무작위로 플로우가 생성되는 시나리오에서 강화학습 기반 스케줄러는 FIFO, RR보다 좋은 성능을 보였으나 토폴로지에 따라 HA와 비교하여 같거나 떨어지는 성능을 보였다.

Key Words : min-max criterion, reinforcement learning, double deep Q-learning, prioritized experience replay, end-to-end latency, fairness

ABSTRACT

In this paper, a reinforcement learning(RL)-based scheduler to minimize the maximum network end-to-end latency is implemented in a single agent environment and a multi-agent environment. The RL model used the double deep Q-network (DDQN) with the prioritized experience replay (PER). Since the agents are unable to identify end-to-end latencies in the multi-agent environment, the state and reward were obtained using the estimated end-to-end latencies. Four network topologies were implemented and simulated to compare the reinforcement learning-based scheduler, FIFO, round robin (RR), and a simple heuristic algorithm (HA). As a result of simulation in fixed packet generation scenarios, the RL-based scheduler achieved the minimization of maximum end-to-end latency in all the topologies. The FIFO and RR schedulers could not minimize the maximum end-to-end latency in any of the topology, and the HA could not minimize the maximum end-to-end latency in a single topology. In scenarios with random flows generation, the RL-based scheduler performed better than the FIFO and RR, but performed the same as or worse than the HA, depending on the topology.

※ 본 연구는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2020R1F1A1058591).
◆ First Author : Sangmyung University, Department of AI & Informatics, 학생회원
° Corresponding Author : Sangmyung university, Department of Human-centered AI, jjoung@ smu.ac.kr, 정회원
* Sangmyung University, Department of AI & Informatics
논문번호 : 202205-107-B-RN, Received May 19, 2022; Revised June 28, 2022; Accepted July 2, 2022

Ⅰ. 서 론

Fairness는 패킷 스케줄링, congestion control, 대역폭 할당 등 네트워크의 많은 문제에서 중요한 고려 요소이다. 이러한 문제들에서 fairness를 높이기 위해 많은 연구가 수행되었다[1]. 최근 fairness를 높이기 위해 강화학습(reinforcement learning, RL)을 사용하는 연구가 제시되고 있다. Chen의 연구에서는 actor-critic 강화학습을 사용해 무선 환경에서의 스케줄링과 비디오의 quality of experience (QoE)를 최적화해 전통적인 알고리즘과 다른 러닝 알고리즘보다 fairness 측면에서 더 뛰어난 성능을 보였다[2]. López-Sánchez의 연구에서는 제안하는 deep RL 기반 스케줄링이 fairness와 평균 지연 시간 측면에서 다른 스케줄러보다 우수함을 보였다[3]. Jiang의 연구에서는 네트워크 환경은 아니지만 multi-agent 환경에서 계층적 강화학습(hierarchical RL)과 분산 강화학습(decentralized RL)을 통해 다양한 문제에서 우수한 fairness와 efficiency를 보였다[4]. 네트워크 분야에서 강화학습을 적용해 패킷 스케줄링을 수행하는 연구도 제안되었다. Kim의 연구에서는 Q-learning을 적용해 데이터의 특성 변화에 강인하면서 데이터의 지연 시간 요구사항을 만족하는 스케줄러를 제안하였다[5]. Guo의 연구에서는 패킷 스케줄링을 위한 자원 할당에 deep Q-network를 사용하여 round robin과 priority 기반 스케줄러보다 더 많은 패킷의 quality of service(QoS)를 만족하였다[6].

본 논문에서는 다른 경로의 패킷들이 겪는 지연 시간 측면에서의 fairness를 위해, 이들의 최대 단대단 지연 시간의 최소화를 달성하기 위한 강화학습 기반 스케줄러를 single agent 환경, multi agent 환경에 각각 구현해보았다. 또한, 시뮬레이션을 통해 다양한 토폴로지에서 강화학습 기반 스케줄러가 최대 단대단 지연 시간의 최소화를 달성하였음을 보였고 무작위 플로우가 생성되는 환경에서도 FIFO, round robin 스케줄러 보다 낮은 최대 단대단 지연 시간을 갖음을 보였다. 논문의 구성은 다음과 같다. 2장에서는 강화학습에 대해 알아본다. 3장에서는 간단한 네트워크 토폴로지로 시뮬레이션에서 네트워크 토폴로지와 강화학습 모델이 어떻게 구현되는지 알아본다. 4장에서는 구현한 네트워크 토폴로지에서 강화학습 기반 스케줄러의 결과를 살펴본다. 5장에서는 결과를 통해 얻은 결론을 서술한다.

Ⅱ. 관련 연구

어떠한 환경(environment)에 있는 에이전트(agent)가 현재 상태(state)에서 행동(action)을 수행하면 상태가 변하고 보상(reward)을 받는다. 에이전트가 이 과정을 반복하여 누적 보상 값이 최대가 되는 정책(policy)을 찾는 학습 방법을 강화학습이라고 한다. 정책을 찾기 위해서는 다양한 상태 행동 쌍을 경험해야만 한다. 정책을 따르지 않고 무작위로 행동을 정하는 것을 exploration, 정책을 따라 행동을 정하는 것을 exploitation이라고 한다. Exploration과 exploitation은 trade-off 관계로 충분한 exploration이 선행되어야 exploitation에서 높은 누적 보상 값을 얻을 수 있다. 강화학습의 대표적인 방법의 하나인 Q-learning은 행동에 대한 가치인 Q-value를 도입하여 Q-value가 높은 행동을 선택하는 방법이다[7]. Q-value는 수식 (1)을 통해 구할 수 있다.

$$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r+\gamma \max_a Q(s_{t+1},a)) \quad (1)$$

Q-learning은 유한 Markov decision process에서 최적의 정책을 학습할 수 있다는 사실이 증명되었다. 하지만 모든 상태와 행동 쌍에 대해 Q-value를 구해야 하기에 입력의 크기와 출력의 크기가 큰 환경에서는 사용하기 어렵다. 이러한 한계를 극복하기 위해 Q-learning과 딥러닝(deep-learning)을 합친 deep Q-networks(DQN)가 제안되었다[8]. DQN은 딥러닝

표 1. 수식에 사용한 기호 설명
Table 1. Symbols used in formulas

Symbol	Quantity
Q	상태와 행동을 입력으로 Q-value를 반환하는 value function, Q 테이블, 딥러닝 네트워크
s, S	상태
a, A	행동
r, R	보상
t	시간
α	학습률, 0-1사이의 값
γ	미래 Q-value에 대한 discount factor, 0-1사이의 값
θ	딥러닝 네트워크의 가중치
Y	목적 Q-value
ϵ	priority가 0이 되지 않도록 넣는 아주 작은 상수

네트워크에 상태를 입력하면 행동 개수만큼 Q-value를 출력하기에 모든 상태와 행동 쌍마다 Q-value를 구할 필요가 없다. 하지만 DQN에서는 특정한 행동의 Q-value가 계속 증가하는 overestimate 문제가 생길 수 있다. 이를 막기 위해 double Q-learning과 DQN을 합친 double deep Q-network(DDQN)가 제안되었다[9]. Double Q-learning이란 행동의 결정과 평가에 동일한 파라미터를 사용하지 않고 다른 파라미터를 사용해 결정과 평가를 분리하는 방법이다[10]. DDQN은 수식 (2)로 표현할 수 있다.

$$Y_t = R_t + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta), \theta^-) \quad (2)$$

DDQN은 overestimate 문제를 효과적으로 줄여 DQN보다 안정적이고 overestimate로 인한 성능 하락이 없기에 DQN보다 더 뛰어난 성능을 보임이 증명되었다[9].

딥러닝을 사용하는 강화학습 방법은 입력으로 사용할 샘플들을 저장할 버퍼가 필요하다. 기존 학습 방식에서는 버퍼에서 무작위로 샘플을 골라서 학습하였다. Prioritized experience replay (PER)는 버퍼에서 샘플을 무작위로 뽑지 않고 중요한 샘플을 뽑아 학습하도록 구상된 버퍼이다[11]. PER은 샘플에 대해 TD-error를 구해 샘플마다 priority를 정한다. 수식 (3)과 수식 (4)는 각각 DDQN 모델에서 TD-error와 priority를 구하는 수식이다.

$$TD\,error = \left| Y_t - Q(S_t, A_t, \theta) \right| \quad (3)$$

$$priority = (TD\,error + \epsilon)^{\alpha} \quad (4)$$

수식 (4)에서 α는 학습률이 아니고 TD-error에 의해 priority가 정해지는 수준을 정하는 0-1 사이의 변수이다. PER을 사용하는 강화학습 모델은 priority가 높은 샘플을 선택해 학습을 진행하여 PER을 사용하지 않는 경우보다 더 좋은 성능을 보인다[11].

자율 주행, 로봇, 게임 등 현실의 환경에서 접할 수 있는 분야들은 여러 에이전트가 서로에게 영향을 주는 환경이다. 이러한 환경에서 문제를 해결하기 위해 복수의 에이전트가 경쟁 또는 협력하며 학습하는 강화학습을 multi agent reinforcement learning(MARL)이라고 한다. 반대로 에이전트가 한 개인 강화학습을 single agent reinforcement learning(SARL)이라고 한다. 가장 간단한 MARL은 independent Q-learning(IQL)으로 에이전트가 서로 독립적으로 각자가 관찰한 상태만을 사용해 행동을 수행하는 분산 강화학습이다[12]. IQL은 에이전트 수가 많거나 에이전트의 종류가 다양하면 성능이 좋지 않지만 간단한 환경에서는 잘 동작한다.

Ⅲ. 시뮬레이션 구현 예시

3.1 네트워크 토폴로지 예시

구현하는 시뮬레이션의 네트워크 토폴로지는 그림 1과 같은 구조의 노드로 구성된다. 출력 버퍼 스위치를 기반으로 하는 노드로 각각의 출력 포트는 버퍼를 포함하는 출력 포트 모듈을 갖는다. 출력 포트 모듈의 버퍼는 입력 포트 개수만큼의 가상의 큐로 나누어진다. 각 큐는 입력 포트와 쌍을 이루어 해당하는 입력 포트에서 들어온 패킷을 저장한다. 이를 위해 패킷은 입력 포트 번호 정보를 담고 있어야 한다. 전체 큐의 개수는 입력 포트 수 × 출력 포트 수이다. 출력 포트 모듈의 스케줄러는 강화학습 모델이 정한 큐에서 패킷을 전송한다. 강화학습 모델을 사용하지 않는 노드의 스케줄러는 FIFO로 작동한다.

그림 2의 예시 토폴로지를 통해 시뮬레이션을 소개

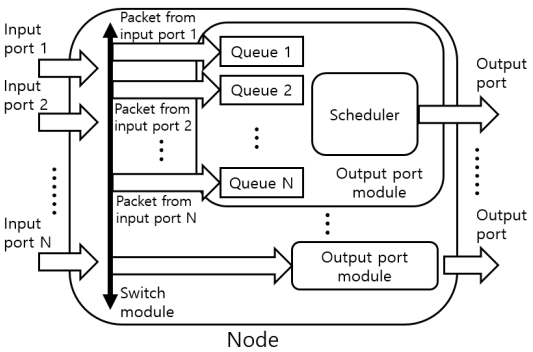

그림 1. 시뮬레이션에서 사용하는 노드의 구조
Fig. 1. Node architecture used in simulation

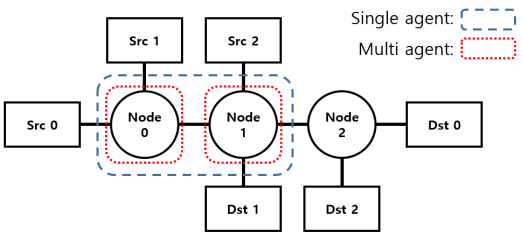

그림 2. 예시 토폴로지
Fig. 2. Example topology

표 2. 예시 토폴로지의 패킷 생성 시나리오
Table 2. Packet generation scenario in example topology

Time	Source 0	Source 1	Source 2
0	1	1	
T			1

하고자 한다. 예시 토폴로지는 표 2의 패킷 생성 시나리오를 따라 패킷을 생성한다. T는 임의로 정한 단위 시간이다. 시뮬레이션의 링크에서 발생하는 전송 지연 시간은 모든 패킷에 대해 항상 T만큼 걸린다고 가정한다. 이는 시뮬레이션에서 발생하는 모든 송수신을 T마다 수행되게 하여 결과 관찰을 쉽게 하기 위함이다. 또한, 소스와 노드를 연결하는 링크에서 발생하는 전송 지연 시간은 스케줄러를 통과하기 이전에 발생하는 지연 시간이므로 최대 단대단 지연 시간의 최소화 문제에 영향을 끼치지 않기에 고려하지 않는다. 이러한 환경에서 예시 토폴로지에서 발생할 수 있는 최대 단대단 지연 시간의 최솟값은 3T이다. 시간 0 시점에서, 노드 0에서 2개의 패킷이 동일한 링크로 나가고자 할 때, 소스 0에서 생성된 패킷이 더 많은 홉 수를 이동해야 하므로 소스 0에서 생성된 패킷을 먼저 내보내야 한다. 시간 T 시점에서는, 노드 1에서 2개의 패킷이 동일한 링크로 나가고자 할 때, 두 패킷의 남은 홉 수는 동일하나 소스 0에서 생성된 패킷이 이미 T만큼의 지연 시간을 겪었으므로 소스 0에서 생성된 패킷을 먼저 내보내야 한다. 이렇게 스케줄러가 작동하면 소스 0, 1, 2에서 생성된 패킷들의 단대단 지연 시간은 각각 3T, 2T, 2T로 최대 단대단 지연 시간이 최소화된다. 소스 0에서 생성된 패킷이 우선 전송되지 않는 경우 발생하는 최대 단대단 지연 시간은 4T이다. 이때의 평균 지연 시간은 최대 단대단 지연 시간이 3T인 경우의 평균 지연 시간보다 작아질 수 있다. 하지만 시뮬레이션으로 해결하고자 하는 문제는 오직 최대 단대단 지연 시간의 최소화이기 때문에 평균 지연 시간은 성능 평가 지표로 사용하지 않는다.

3.2 강화학습 적용

예시 토폴로지에서 SARL을 사용하는 경우 에이전트가 관찰할 수 있는 상태는 노드 0과 노드 1의 상태이다. SARL에서는 이 노드들과 연결된 컨트롤러가 있어 노드들에서 상태 정보를 받고 이를 바탕으로 컨트롤러는 노드의 스케줄러의 동작을 정한다. 컨트롤러는 이 메시지를 각 노드에 보내고 노드는 해당 메시지를 따라 스케줄러를 동작시킨다. 시뮬레이션에서 노드와 컨트롤러의 메시지 교환으로 인한 지연 시간은 고려하지 않는다. 예시 토폴로지에서 MARL을 사용하는 경우 에이전트는 노드 0과 노드 1에 2개의 에이전트를 구현한다. MARL으로는 IQL을 사용한다. IQL은 일반적으로 성능이 좋지 않지만 상정하는 네트워크 토폴로지에서는 에이전트가 모두 동일하고 순차적으로 동작하기에 서로에게 주는 영향이 적어 IQL이 충분한 성능을 보일 수 있다. 각 에이전트는 자신의 노드의 상태만을 관찰할 수 있다. MARL은 노드와 에이전트가 동일하므로 에이전트와 노드 간의 메시지 교환 없이 강화학습 모델을 통해 스케줄러를 동작시킬 수 있다. MARL에서 에이전트는 단대단 지연 시간을 직접 관찰할 수 없다. 이에 에이전트는 최대 단대단 지연 시간을 최소화하지 않고 노드에서 계산할 수 있는 추정 지연 시간의 최댓값을 최소화한다. 추정 지연 시간은 패킷의 예상되는 최소 단대단 지연 시간으로 수식 (5)로 구할 수 있다.

$$\begin{aligned}&\textit{The packet's estimated delay}\\&= \textit{the packet's remaining hop count}\\&+ \textit{the packet's experienced delay to the present}\\&+ \textit{order in the queue of the packet}\end{aligned} \tag{5}$$

노드에서 수식 (5)를 계산하기 위해 패킷은 생성된 시간 정보와 남아있는 홉 수 정보를 담고 있어야 한다. 추정 지연 시간은 결과적으로 단대단 지연 시간과 동일하기에 추정 지연 시간의 최댓값을 최소화하면 최대 단대단 지연 시간이 최소화된다. SARL은 에이전트가 목적지 노드의 상태 정보를 얻을 수 있도록 컨트롤러와 목적지 노드 간 메시지 교환을 추가하면 단대단 지연 시간을 얻을 수 있다. 하지만 컨트롤러에 부담이 늘어나고 단대단 지연 시간은 패킷이 목적지에 도달해야지만 얻을 수 있다는 단점이 있기에 SARL도 추정 지연 시간을 사용한다.

강화학습 모델은 에이전트가 관찰을 통해 얻은 상태를 입력으로 사용해 출력으로 행동을 결정한다. 에이전트는 행동을 통해 보상을 얻고 상태를 변화시킨다. 이를 반복해 누적 보상 값이 최대가 되도록 강화학습 모델을 학습시킨다. 강화학습을 사용해 네트워크의 최대 단대단 지연 시간의 최소화 문제를 해결하기 위해서 다음과 같이 상태, 행동, 보상을 정의하였다.

3.2.1 상태

상태는 출력 포트별 큐에서 얻을 수 있는 정보를 사용한다. 큐마다 첫 번째 패킷의 남은 홉 수, 첫 번째 패킷의 현재까지의 지연 시간, 큐 길이, 큐에 있는 패

킷들의 최대 추정 지연 시간 정보를 얻는다. 상태는 최대 단대단 지연 시간을 유추할 수 있는 정보이다. 첫 번째 패킷의 남은 홉 수와 현재까지의 지연 시간은 첫 번째 패킷의 추정 지연 시간을 계산할 수 있는 정보이다. 첫 번째 패킷은 추정 지연 시간과 실제 단대단 지연 시간이 가장 유사하기에 중요하다. 큐 길이는 다른 큐에 영향을 주는 정도를 유추할 수 있다. 큐에 있는 패킷들의 최대 추정 지연 시간은 목표인 최대 단대단 지연 시간과 가장 유사한 값이다. MARL에서 상태의 크기는 4 × 입력 포트 수 × 출력 포트 수이고 SARL에서 상태의 크기는 4 × 입력 포트 수 × 출력 포트 수 ×노드 수이다. 예시 토폴로지에서 노드 1의 경우 입력 포트 수가 2개, 출력 포트 수가 2개이므로 MARL에서 상태의 크기는 16이다. 하지만 출력 포트 중 1개만 패킷이 동시에 나가고자 하는 경우가 있고 다른 하나는 패킷이 1개만 지나기에 FIFO를 사용해도 된다. 따라서 1개의 출력 포트의 큐의 정보만을 사용해 상태의 크기를 8로 줄일 수 있다. SARL도 마찬가지로 상태의 크기를 32에서 16으로 줄일 수 있다. 고정된 토폴로지와 시나리오에서는 이렇게 상태의 크기를 줄여 강화학습 모델의 계산 비용을 줄일 수 있지만 고정된 환경이 아니라면 모든 정보를 사용해야 하므로 상태의 크기를 줄일 수 없다.

3.2.2 행동

행동은 출력 포트마다 전송할 큐를 선택하는 것이다. 행동은 그 시점에서 송수신이 모두 완료되었을 때 수행한다. 출력 포트당 큐의 개수는 입력 포트 수와 동일하므로 MARL에서 행동의 크기는 입력 포트 수 ^ 출력 포트 수이다. SARL에서 행동의 크기는 (입력 포트 수 ^ 출력 포트 수) ^ 노드 수이다. 행동의 크기도 상태의 크기처럼 줄일 수 있다. 예시 토폴로지에서 MARL으로 노드 1의 행동의 크기는 4지만 유효한 출력 포트는 1개이므로 행동의 크기를 2로 줄일 수 있다. 마찬가지로 SARL에서 행동의 크기는 8에서 4로 줄일 수 있다.

3.2.3 보상

보상은 알고리즘 1을 따라 정해진다. 보상은 출력 포트별로 행동이 수행된 이후 행동에 의한 송수신이 모두 완료되었을 때 얻는다. 알고리즘 1에 c는 상수로 출력 포트 모듈의 스케줄러가 모두 FIFO일 때 발생할 수 있는 최대 단대단 지연 시간보다 큰 수이다. c는 스케줄러가 작업 보존형(work-conserving) 스케줄러이기 위해 패킷을 전송하지 않는 행동을 수행했을 때 주는 감점이다. 패킷을 전송했으면 전송한 패킷의 추정 지연 시간과 모든 송수신이 끝난 후에 큐에 있는 패킷의 최대 추정 지연 시간을 이용해 보상 준다. 알고리즘 1의 핵심은 전송한 패킷의 추정 지연 시간이 클수록, 큐의 남아있는 패킷의 최대 추정 지연 시간은 작을수록 더 큰 보상을 주는 것이다. MARL에서 출력 포트별로 얻은 보상을 합하여 실제 보상으로 사용한다. SARL에서는 노드별로 출력 포트마다 얻은 보상을 모두 합하여 실제 보상으로 사용한다.

알고리즘 1 MARL 보상 알고리즘

```
N = number of output port
M = number of input port
OutputPortModule[1..N]
OutputPortModule.Queue[1..M]
reward = 0

for i in 1..N:
  if all OutputPortModule[i].Queue is not empty:
    if packet transmission is complete:
      if all queues are empty:
        reward += estimated delay of the
                  transmitted packet
      else:
        reward += estimated delay of the
                  transmitted packet - max(
                  estimated delay of packets in
                  all queues)
    else:
      reward += -c
return reward
```

Ⅳ. 시뮬레이션 결과

4.1 강화학습 모델

강화학습 모델은 TensorFlow를 사용해 구현하였다. TensorFlow는 머신러닝을 위한 엔드 투 엔드 오픈소스 플랫폼으로 머신러닝 모델을 개발하고 학습시키는 데 도움을 준다[13]. 시뮬레이션에서 사용하는 딥

표 3. 딥러닝 네트워크의 구조
Table 3. Deep learning network architecture

Layer	Size
Input	State size
Dense	64
ReLu	64
Dense	64
ReLu	64
Dense (Output)	Action size

러닝 네트워크의 구조는 표 3과 같다. Dense 레이어는 입력과 출력을 모두 연결해주는 레이어다. Rectified Linear Unit(ReLu)은 활성화 함수의 일종으로 수식 (6)과 같은 비선형 함수이다.

$$f(x) = \max(0, x) \tag{6}$$

강화학습의 버퍼는 PER을 사용하며 크기는 50000이다. 강화학습 모델의 하이퍼파라미터는 표 4와 같다. 강화학습 모델의 학습률은 수식 (7)을 따라 에피소드가 지날 때마다 감소한다.

$$Learning\ rate = Initial\ learning\ rate \times 0.5 \times (1 + \cos(\pi \times \frac{\min(number\ of\ episode, decay\ episode)}{decay\ episode})) \tag{7}$$

학습률을 감소시키는 이유는 학습 초기에는 높은 학습률로 loss를 빠르게 줄여 전역 최적해를 빠르게 찾고 전역 최적해를 찾은 이후에는 높은 학습률로 인해 전역 최적해를 벗어나지 않게 하기 위함이다.

에피소드는 시간 0부터 시작하여 모든 패킷이 목적지에 도달하거나 100T 시간이 되면 종료된다. 시뮬레이션 한번당 3000 에피소드 반복한다. 시뮬레이션의 처음 500 에피소드 동안은 강화학습 모델은 학습하지 않고 버퍼에 샘플을 저장한다. 이후 남은 2500 에피소드 동안 강화학습 모델이 학습된다. 시뮬레이션은 토폴로지마다 SARL, MARL로 구분하여 15번씩 반복해 결과를 얻었다.

표 4. 하이퍼파라미터
Table 4. Hyperparameter

Hyperparameter	Value
Number of episode	3000
Initial learning rate	0.002
Discount factor	0.99
Initial epsilon	1
Epsilon decay	0.997
Minimum epsilon	0.001
Target model update cycle	500 episode
Batch size	64
Decay episode	2500

4.2 토폴로지 1

네트워크 토폴로지는 SimPy를 사용해 구현하고 시뮬레이션하였다. SimPy는 파이썬 기반의 process based discrete-event 시뮬레이션 프레임워크로 멀티 에이전트 시스템을 구현할 수 있고 다양한 종류의 공유 자원을 제공한다[14]. 첫 번째 시뮬레이션은 그림 3과 같은 토폴로지 1에서 진행하였다. 토폴로지 1은 표 5를 따라 패킷을 생성한다. 토폴로지 1은 예시 토폴로지의 확장으로 소스 0에서 생성된 패킷이 우선 전송되어야 최대 단대단 지연 시간을 최소화할 수 있다. 예시 토폴로지와 같은 방법으로 구한 토폴로지 1의 최대 단대단 지연 시간의 최솟값은 5T이다. 강화학습 기반 스케줄러와 성능을 비교하기 위해 FIFO, round robin(RR), 제안하는 간단한 heuristic algorithm(HA)을 사용해 스케줄링해본다. HA는 송수신마다 모든 큐에 들어있는 패킷들의 추정 지연 시간을 구해 가장 큰 추정 지연 시간을 갖는 패킷이 들어있는 큐를 전송하는 스케줄러이다. 토폴로지 1뿐만 아니라 후술할 토폴로지 2, 토폴로지 3, 토폴로지 4도 이 3가지 스케줄러와 성능을 비교한다.

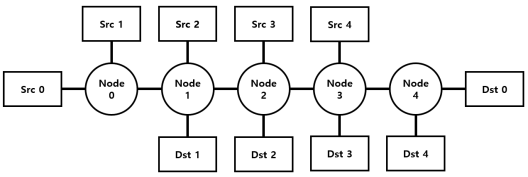

그림 3. 토폴로지 1
Fig. 3. Topology 1

표 5. 토폴로지 1의 패킷 생성 시나리오
Table 5. Packet generation scenario in topology 1

Time	Source 0	Source 1	Source 2	Source 3	Source 4
0	1	1			
T			1		
2T				1	
3T					1

그림 4는 토폴로지 1에서 강화학습 기반 스케줄러의 학습 결과 그래프이다. SARL과 MARL 모두 최대 단대단 지연 시간의 최소화를 달성했음을 확인할 수 있다. 처음 500 에피소드 동안에는 강화학습 모델이 학습하지 않고 버퍼에 샘플을 저장하는 기간이기에 보상의 합이 증가하거나 최대 단대단 지연 시간이 작아지지 않는다. 500 에피소드 이후 강화학습 모델이 학습을 시작하면서 보상의 합이 증가해 최댓값에 도달하였고 최대 단대단 지연 시간이 최소화되었다. MARL의 그래프를 보면 보상의 합의 이동 평균 그래

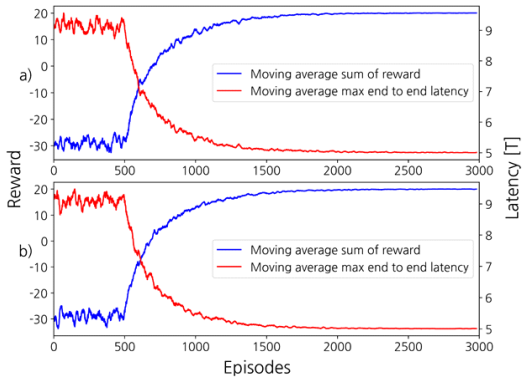

그림 4. 토폴로지 1에서 강화학습 모델의 학습 결과. a: SARL, b: MARL
Fig. 4. Training result of RL model in topology 1. a: SARL, b: MARL

프가 변해도 최대 단대단 지연 시간의 이동 평균 그래프는 변하지 않음을 볼 수 있다. 이는 강화학습 모델이 전역 최적해에 도달하지 못했어도 최대 단대단 지연 시간은 최소화될 수 있음을 의미한다. 하지만 강화학습 모델이 전역 최적해에 도달하면 최대 단대단 지연 시간은 항상 최소화되었다.

4.3 토폴로지 2

두 번째 시뮬레이션은 그림 5와 같은 토폴로지 2에서 진행하였다. 토폴로지 2는 표 6을 따라 패킷을 생성한다. 토폴로지 2는 트리 형태의 토폴로지로 단일 링크를 지나는 패킷의 수를 늘려 강화학습 모델이 더 다양한 샘플을 학습하는 환경이다.

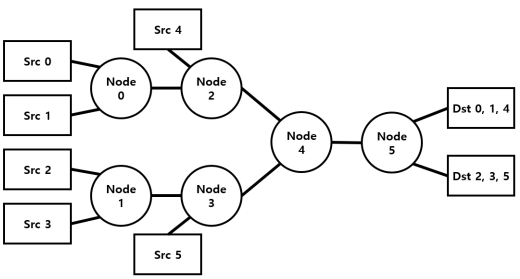

그림 5. 토폴로지 2
Fig. 5. Topology 2

표 6. 토폴로지 2의 패킷 생성 시나리오
Table 6. Packet generation scenario in topology 2

Time	Source 0	Source 1	Source 2	Source 3	Source 4	Source 5
0	1	1	1	1		
T					1	1

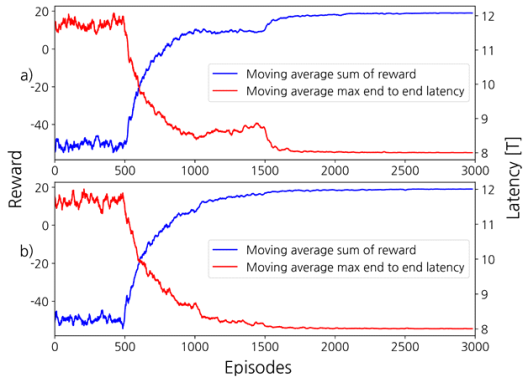

그림 6. 토폴로지 2에서 강화학습 모델의 학습 결과. a: SARL, b: MARL
Fig. 6. Training result of RL model in topology 2. a: SARL, b: MARL

그림 6은 토폴로지 2에서 강화학습 기반 스케줄러의 학습 결과 그래프이다. SARL과 MARL 모두 최대 단대단 지연 시간의 최소화를 달성했다. SARL의 경우 학습 초기 1000 에피소드 동안에 지역 최적해에 도달하고 이를 유지했음을 확인할 수 있다. 이러한 현상은 강화학습 알고리즘과 딥러닝 네트워크의 안정성에 의해 나타난다. 이는 토폴로지 2의 SARL에서 상태의 크기와 행동의 크기가 딥러닝 네트워크 구조에 비해 크기 때문이다.

이를 잘 볼 수 있도록 토폴로지 2에서 더 많은 패킷을 전송하는 시뮬레이션을 진행하였다. 패킷 생성 시나리오는 10개의 패킷을 무작위로 생성하되 시나리오의 최대 길이는 20T가 되도록 하였다. 그림 7은 이러한 환경에서 DQN 알고리즘과 DDQN 알고리즘간의 성능 비교 및 딥러닝 네트워크 구조에 따른 성능 차이를 보여준다. 기존 시뮬레이션과 달리 더 많은 패킷이 무작위로 생성되어 더 다양한 상태가 발생한다.

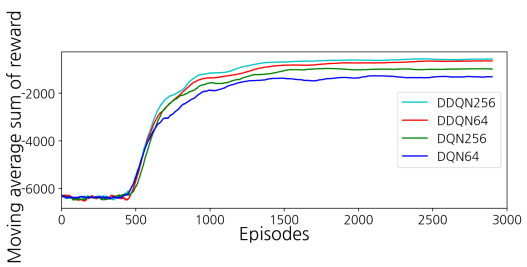

그림 7. 토폴로지 2에서 무작위 플로우가 생성될 때 SARL의 강화학습 알고리즘과 딥러닝 네트워크 구조에 따른 학습 결과
Fig. 7. Training results according to SARL's RL algorithm and deep learning network structure when random flows are generated in topology 2

이러한 환경에서 DQN 알고리즘은 딥러닝 네트워크 구조와 상관없이 DDQN보다 낮은 최적해에 수렴한다. 이는 DQN의 한계인 overestimated 문제로 DDQN에 비해 안정성이 낮기 때문이다. 또한 각 알고리즘의 성능은 딥러닝 네트워크 구조에 따라 달라지는데 토폴로지 2의 SARL에서 상태의 크기와 행동의 크기가 딥러닝 네트워크 구조에 비해 크고 무작위 플로우로 인해 발생하는 상태가 다양하기 때문이다. SARL에서 상태의 크기와 행동의 크기는 입력 포트 수, 출력 포트 수, 노드 수에 따라 결정된다. 특히 행동의 크기는 출력 포트 수와 노드 수의 제곱에 비례하여 증가하기에 노드 수가 많은 토폴로지 일수록 딥러닝 네트워크 구조의 크기를 늘려야 더 안정적으로 학습한다. 상태가 다양하지 않다면 그림 6처럼 지역 최적해에 잠깐 빠지는 정도이지만 상태가 다양해지면 딥러닝 네트워크 구조로 인한 안정성은 성능과 직결된다. 그림 7을 보면 딥러닝 네트워크 구조의 크기가 64x64에서 256x256으로 증가하자 DQN, DDQN 알고리즘 모두 더 좋은 최적해를 찾는 모습을 볼 수 있다.

4.4 토폴로지 3

세 번째 시뮬레이션은 그림 8과 같은 토폴로지 3에서 진행하였다. 토폴로지 3은 표 7을 따라 패킷을 생성한다. 토폴로지 3은 앞서 살펴본 토폴로지들과 달리 패킷을 생성할 때 이미 지연 시간을 겪었다고 가정한다. 소스 0에서 생성하는 패킷은 T, 소스 1에서 생성하는 패킷은 7T, 소스 2에서 생성하는 패킷은 3T만큼 지연 시간을 겪었다고 가정한다. 토폴로지 3은 나중에

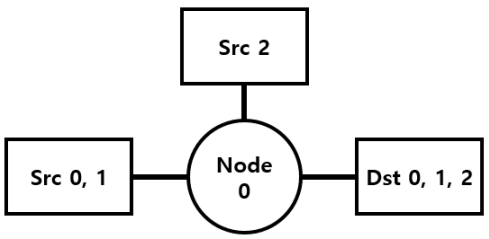

그림 8. 토폴로지 3
Fig. 8. Topology 3

표 7. 토폴로지 3의 패킷 생성 시나리오
Table 7. Packet generation scenario in topology 3

Time	Source 0	Source 1	Source 2
0	1		1
T		1	

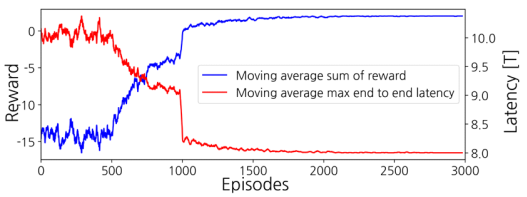

그림 9. 토폴로지 3에서 강화학습 모델의 학습 결과
Fig. 9. Training result of RL model in topology 3

큐에 들어오는 패킷이 최대 단대단 지연 시간을 결정할 때 이 패킷을 즉시 내보내기 위해 큐를 비워 놓아야 최대 단대단 지연 시간의 최소화를 달성할 수 있다. 토폴로지 3은 노드 하나로 이루어진 토폴로지이며 SARL과 MARL의 구현이 다르지 않기에 SARL만 시뮬레이션했다.

그림 9는 토폴로지 3에서 강화학습 기반 스케줄러의 학습 결과 그래프이다. SARL으로 최대 단대단 지연 시간의 최소화를 달성하였다. 그림 9에서 에피소드 1000 부근에서 보상의 합과 최대 단대단 지연 시간이 급격히 변하는 것을 볼 수 있다. 이는 토폴로지 3이 노드 1개로 이루어진 작은 토폴로지이고 시뮬레이션에 걸리는 타임 슬롯 또한 적어 상태가 다양하지 않기 때문이다. 이러한 토폴로지는 그림처럼 행동 하나로 누적 보상과 최대 단대단 지연 시간이 크게 변할 수 있다.

4.5 무작위 플로우

앞선 시뮬레이션에 따라 제안하는 강화학습 방법이 정해진 시나리오에서 잘 동작함을 확인하였다. 이번

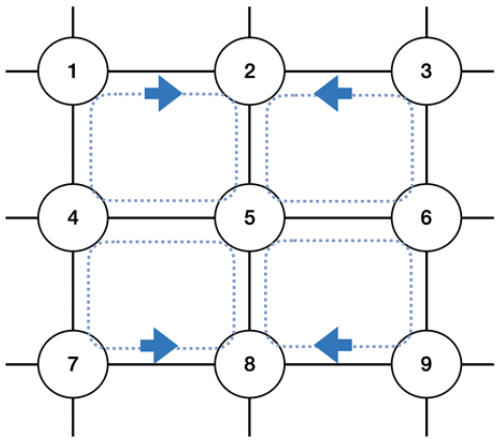

그림 10. 토폴로지 4
Fig. 10. Topology 4

표 8. 토폴로지 4에서 플로우의 경로
Table 8. The route of the flow in topology 4

Flow number	Route
0	1-2-5-6-9
1	4-1-2
2	4-7-8
3	7-8-5-6-3
4	3-2-5-4-7
5	6-3-2
6	6-9-8
7	9-8-5-4-1

시뮬레이션은 앞선 시뮬레이션과 달리 무작위 플로우를 생성하는 환경에서 강화학습 스케줄러의 성능을 알아본다. 무작위 플로우는 앞서 토폴로지 2에서 안정성에 따른 결과 변화를 알아보기 위한 시뮬레이션에서처럼 최대 20T 길이의 패킷 생성 시나리오에서 10개 패킷을 생성한다. 토폴로지는 기존 토폴로지들과 달리 symmetric한 사이클(cycle)이 존재하는 토폴로지에서 강화학습 스케줄러의 성능을 알아본다. 시뮬레이션에 사용한 토폴로지는 그림 10과 같다. 토폴로지 4는 특정한 플로우가 다른 모든 플로우와 겹치도록 시나리오를 구성할 수 없기에 무작위 플로우를 생성한다. 이전 시뮬레이션과 달리 강화학습 기반 스케줄러중 MARL만 구현했는데 토폴로지 4는 노드 수가 9개, 입력 포트 수와 출력 포트 수가 각각 2개로 SARL에서의 행동의 크기가 (2^2)^18로 너무 크기에 구현이 불가능하다. 이에 반해 MARL은 행동의 크기는 4이고, 상태의 크기는 16이기에 구현 가능하다.

그림 11은 토폴로지 4에서 무작위 플로우가 생성되는 네트워크 환경에서의 각 스케줄러의 최대 단대단 지연 시간의 이동 평균이다. HA가 가장 성능이 좋고 MARL, RR, FIFO 순서로 성능이 좋다. 토폴로지 4와 같이 사이클이 존재하는 환경에서는 한 노드가 나머지

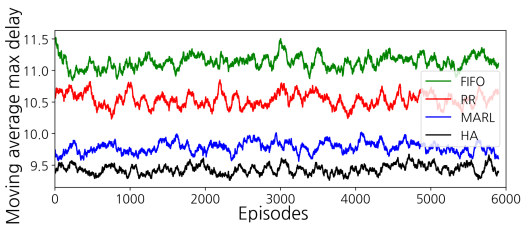

그림 11. 토폴로지 4에서 무작위 플로우가 생성될 때 스케줄러별 최대 지연 시간의 이동 평균
Fig. 11. Moving average of maximum end to end latency per scheduler when random flows are generated in topology 4

모든 노드에게 영향을 받을 수 있기에 IQL방식의 MARL로는 HA에 비해 성능이 안좋은 것으로 추측한다.

무작위 플로우가 생성되는 환경에서 SARL의 성능을 알아보기 위해 토폴로지 2에서 추가 시뮬레이션을 진행하였다. 무작위 플로우로 인해 상태가 다양해졌기에 강화학습 기반 스케줄러가 잘 학습할 수 있도록 딥러닝 네트워크 구조와 파라미터 일부를 변경하였다. SARL은 딥러닝 네트워크 구조를 (state size x 256 x 256 x 256 x action size)로 변경하였다. 또한 충분한 학습을 위해 SARL은 50000 에피소드동안 학습하였고 에피소드 수에 맞게 하이퍼파라미터를 변경하였다.

그림 12는 토폴로지 2에서 무작위 플로우가 생성되는 네트워크 환경에서의 각 스케줄러의 최대 단대단 지연 시간의 이동 평균이다. 제안하는 강화학습 기반 스케줄러는 FIFO, RR보다 뛰어난 성능을 보이고 HA와 유사한 성능을 보인다. 토폴로지 2는 고정된 시나리오에서도 강화학습 기반 스케줄러가 HA와 동일한

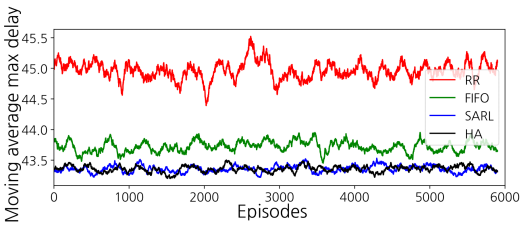

그림 12. 토폴로지 2에서 무작위 플로우가 생성될 때 스케줄러별 최대 지연 시간의 이동 평균
Fig. 12. Moving average of maximum end to end latency per scheduler when random flows are generated in topology 2

표 9. 스케줄러별 최대 단대단 지연 시간 평균 비교
Table 9. Comparison of average maximum end-to-end latency by scheduler

	Average maximum end-to-end latency				
	FIFO	RR	HA	Single agent	Multi agent
Topology 1	5.937 5T	5.937 5T	5T	5T	5T
Topology 2	9T	9T	8T	8T	8T
Topology 3	9T	9T	9T	8T	8T
Topology 2 with random flow	43.72 2T	44.96 3T	43.35 5	43.35 6T	-
Toplogy 4 with random flow	11.13 4T	10.53 7T	9.433 T	-	9.779 T

성능을 보였기에 SARL은 전역 최적해에 도달했다고 볼 수 있다. 토폴로지 4에서는 강화학습 기반 스케줄러가 HA보다 안 좋은 성능을 보였는데 이는 토폴로지의 사이클과 IQL방식의 한계로 보인다. MARL 강화학습 기반 스케줄러는 큐 길이를 노드마다 더 중요하게 여기거나 덜 중요하게 여기는 모습이 관찰되었다. 이로 인해 최대 추정 지연 시간의 차이가 크지 않다면 큐 길이에 따라 서비스 순서를 정할 수 있는데 이로 인해 HA보다 안 좋은 성능을 보이는 것으로 추측한다. SARL의 경우 학습은 오래 걸릴지라도 모든 노드의 정보를 알기에 큐 길이 정보에 좌우되지 않고 HA와 유사하게 동작한다.

V. 결 론

본 논문에서는 최대 단대단 지연 시간을 최소화하는 강화학습 기반 스케줄러를 구현하고 시뮬레이션을 통해 다른 스케줄러와 성능을 비교하였다. 강화학습 모델은 TensorFlow를 사용해 DDQN 구조로 구현하였다. 네트워크 토폴로지와 시뮬레이션은 SimPy를 사용해 구현하였다. 네트워크 토폴로지에서 노드는 출력 포트 모듈이 입력 포트 수만큼 논리적 큐를 갖고, 링크는 모든 패킷에 대해 전송 지연이 항상 T만큼 걸린다고 가정하였다. 강화학습 에이전트는 SARL과 MARL 두 가지로 구현하였다. SARL은 컨트롤러가 에이전트로 노드와 메시지 교환을 통해 정보를 받고 명령을 내린다. 시뮬레이션에서는 이러한 메시지 교환으로 인한 지연 시간은 고려하지 않았다. MARL은 노드별 에이전트가 노드 내의 정보를 바탕으로 독립적인 결정을 내리므로, 메시지 교환으로 인한 지연 시간이 발생하지 않는다. 강화학습 환경에서 상태는 큐들의 정보로 MARL은 단일 노드의 모든 출력 포트의 큐 정보를, SARL은 여러 노드의 모든 출력 포트의 큐 정보를 상태로 사용하였다. 강화학습 환경에서 에이전트는 행동으로 출력 포트마다 전송할 큐를 선택하였다. SARL은 에이전트인 컨트롤러가 관찰하는 노드마다 출력 포트별로 전송할 큐를 선택하였다. 강화학습 환경에서 보상은 전송한 패킷과 전송하지 않은 패킷의 추정 지연 시간을 통해 얻었다. 추정 지연 시간은 패킷이 현재 달성할 수 있는 최소 단대단 지연 시간을 의미한다. 시뮬레이션은 토폴로지 4개에서 진행하였고 토폴로지 2와 토폴로지 4에서는 무작위 플로우를 생성해 시뮬레이션을 진행하였다. 강화학습 기반 스케줄러와 성능 비교를 위해 FIFO, RR, HA의 시뮬레이션을 추가로 진행하였다. 시뮬레이션 결과 FIFO, RR은 모든 토폴로지에서 최대 단대단 지연 시간 최소화를 달성하지 못했다. HA는 토폴로지 1과 2에서는 최대 단대단 지연 시간 최소화를 달성했으나 토폴로지 3에서는 달성하지 못했다. 강화학습 기반 스케줄러는 모든 토폴로지에서 SARL, MARL 모두 최대 단대단 지연 시간의 최소화를 달성하였다. 무작위 플로우를 생성하는 시나리오에서 강화학습 기반 스케줄러는 기존 FIFO, RR보다 좋은 성능을 보였고 토폴로지에 따라 HA와 유사하거나 안좋은 성능을 보였다.

본 논문에서는 가치 기반(value-based) 강화학습 알고리즘을 사용한 강화학습 기반 스케줄러를 제안하였다. 하지만 가치 기반 강화학습 알고리즘은 행동의 크기가 무한한 환경에서는 적용하지 못하기에 토폴로지가 커질수록 제안하는 방법을 적용하기 어렵다. 이에 Proximal policy optimization 알고리즘과 같은 최신의 정책 기반(policy-based) 강화학습을 사용한 강화학습 기반 스케줄러를 제안하고 대규모 네트워크에서의 강화학습 스케줄러의 성능을 알아보고자 한다. 또한 단순 알고리즘 적용에서 나아가 네트워크 환경을 고려하여 강화학습 알고리즘의 개선을 시도해보고자 한다.

References

[1] R. Denda, A. Banchs, and W. Effelsbert, “The fairness challenge in computer networks,” *Int. Wkshp. Quality of Future Internet Serv.*, Springer, Berlin, hedelberg, 2000. (https://doi.org/10.1007/3-540-39939-9_17)

[2] J. Chen, Y. Wang, and T. Lan, “Bringing fairness to actor-critic reinforcement learning for network utility optimization,” *IEEE INFOCOM 2021-IEEE Conf. Computer Commun.*, 2021, pp. 1-10. (https://doi.org/10.1109/INFOCOM42981.2021.9488823)

[3] M. López-Sánchez, et al., “Latency fairness optimization on wireless networks through deep reinforcement learning,” *arXiv preprint arXiv:2201.10281*, 2022.

[4] J. Jiang and Z. Lu, “Learning fairness in multi-agent systems,” *Advances in NIPS*, vol. 32, 2019.

[5] D. Kim, et al., “Adaptive packet scheduling in IoT environment based on Q-learning,”

Procedia Comput. Sci., vol. 141, pp. 247-254, 2018. (https://doi.org/10.1016/j.procs.2018.10.178)

[6] B. Guo, et al., "Deep-Q-network-based multimedia multi-service QoS optimization for mobile edge computing systems," *IEEE Access,* vol. 7, pp. 160961-160972, 2019. (https://doi.org/10.1109/access.2019.2951219)

[7] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279-292, 1992. (https://doi.org/10.1007/bf00992698)

[8] V. Mnih, et al., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAi Conf. Artificial Intell.*, vol. 30, no. 1, 2016.

[10] H. Hasselt "Double Q-learning," *Advances in NIPS*, vol. 23, 2010.

[11] T. Schaul, et al., "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[12] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. tenth Int. Conf. Mach. Learn.*, pp. 330-337, 1993. (https://doi.org/10.1016/b978-1-55860-307-3.50049-6)

[13] TensorFlow, Retrieved May 13, 2022, from https://www.tensorflow.org/?hl=ko

[14] SimPy, Retrieved May 13, 2022, from https://simpy.readthedocs.io/en/latest/

권 주 혁 (Juhyeok Kwon)

2020년 8월 : 상명대학교 휴먼지능정보공학과 학사

2020년 9월~현재 : 상명대학교 지능정보공학과 석사

<관심분야> 유무선통신, 네트워크, 임베디드 시스템

류 지 혜 (Jihye Ryu)

2020년 8월 : 상명대학교 휴먼지능정보공학과 학사

2020년 9월~현재 : 상명대학교 지능정보공학과 석사

<관심분야> 네트워크, 강화학습, 딥러닝

정 진 우 (Jinoo Joung)

1992년 2월 : KAIST 전자공학과 학사

1994 8월 : NYU 전기전자공학과 Master

1997년 8월 : NYU 전기전자공학과 Ph.D.

1997년 10월~2005년 2월 : 삼성전자 종합기술원

2005년 3월~현재 : 상명대학교 휴먼지능정보공학과 교수

<관심분야> 유무선통신, 네트워크, 임베디드시스템

[ORCID:0000-0003-3053-9691]