applied sciences

MDPI

Article

HTTP Adaptive Streaming Framework with Online Reinforcement Learning

Jeongho Kang and Kwangsue Chung *

Department of Electronics and Communications Engineering, Kwangwoon University, Seoul 01897, Korea; jhkang@cclab.kw.ac.kr
* Correspondence: kchung@kw.ac.kr

Abstract: Dynamic adaptive streaming over HTTP (DASH) is an effective method for improving video streaming's quality of experience (QoE). However, the majority of existing schemes rely on heuristic algorithms, and the learning-based schemes that have recently emerged also have a problem in that their performance deteriorates in a specific environment. In this study, we propose an adaptive streaming scheme that applies online reinforcement learning. When QoE degradation is confirmed, the proposed scheme adapts to changes in the client's environment by upgrading the ABR model while performing video streaming. In order to adapt the adaptive bitrate (ABR) model to a changing network environment while performing video streaming, the neural network model is trained with a state-of-the-art reinforcement learning algorithm. The proposed scheme's performance was evaluated using simulation-based experiments under various network conditions. The experimental results confirmed that the proposed scheme performed better than the existing schemes.

Keywords: dynamic adaptive streaming over HTTP (DASH); quality of experience (QoE); reinforcement learning; online learning

Citation: Kang, J.; Chung, K. HTTP Adaptive Streaming Framework with Online Reinforcement Learning. *Appl. Sci.* **2022**, *12*, 7423. https://doi.org/10.3390/app12157423

Academic Editor: Cheonshik Kim

Received: 23 June 2022
Accepted: 22 July 2022
Published: 24 July 2022

1. Introduction

With the proliferation of various smart devices and network development, the number of users accessing video streaming services via the Internet has recently increased. According to the Cisco Annual Internet Report, the total number of Internet users worldwide will increase from 51% of the population in 2018 to 66% of the population by 2023 [1]. Video streaming services such as YouTube and Netflix account for the majority of internet traffic. With the growing importance of video streaming services, HTTP adaptive streaming is gaining traction as a technology to provide users with a high quality of experience (QoE) [2]. Dynamic adaptive streaming over HTTP (DASH) was established as a standard for HTTP adaptive streaming technology in 2011 as a solution to provide efficient and smooth video streaming [3]. DASH has high reliability and is not restricted by firewalls and network address translations (NATs) because it uses the existing TCP-based HTTP protocol. In addition, it has high scalability because it improves QoE by adjusting the quality of video segments delivered through the network on the client side. Commercialized services include Microsoft's Smooth Streaming, Apple's HTTP Live Streaming, and Adobe's HTTP Dynamic Streaming [4–6].

To guarantee user QoE according to the time-varying network conditions between the client and the server, it is necessary to design an adaptive bitrate (ABR) algorithm for the DASH system. To prevent QoE degradation, the DASH client requests the next video segment with the appropriate quality based on the network's bandwidth and playback status (buffer length, playback interruption, and requested video quality) using the ABR algorithm [7]. Therefore, designing an efficient ABR algorithm is a challenge for the DASH system. In general, ABR algorithms can be classified into two types: model-based algorithms and learning-based algorithms. A model-based ABR algorithm is a heuristic-based algorithm that controls ABR decisions based on a model in which the segment request

method according to the network bandwidth and playback state is defined in advance. However, it is difficult to achieve optimal QoE with such a fixed-control rule because video playback environments and network conditions are different for each user. To overcome the limitations of model-based ABR algorithms, a learning-based ABR algorithm employs a learning method such as reinforcement learning (RL) to empirically learn the optimal ABR determination method in advance using network state data. However, learning-based ABR algorithms have a limitation in that if the user's environment changes, the training must be repeated because performance is degraded if not performed in the targeted network environment.

In this study, we propose an adaptive streaming scheme based on online reinforcement learning to solve the problems of the existing learning-based ABR algorithms. The main contributions of the proposed scheme are as follows:

- We propose an HTTP adaptive streaming framework with online reinforcement learning. When a decrease in QoE is confirmed, the client updates the ABR model at the same time as performing video streaming to adapt to the change in the client's network status.
- Using a network classification scheme, the network traces used as the dataset and the current user's network environment are classified according to characteristics. The proposed scheme can adapt the ABR model to the time-varying network conditions using classified network data.
- If a learning-based ABR algorithm is used, it can be extended using the proposed scheme regardless of the learning method.
- The neural network model is trained with a state-of-the-art RL algorithm to adapt the ABR model to the network environment that changes simultaneously with video streaming. Training is performed at high speed because it is asynchronous due to creating agents in parallel.
- The proposed scheme was compared with the existing schemes through simulation-based experiments. The experimental results show that the proposed scheme outperforms the existing schemes in terms of overall QoE for clients.

The rest of this study is organized as follows. In Section 2, we present the related work on the operation process of the DASH system and the ABR algorithm. In Section 3, we describe the design of the framework of the proposed scheme. In Section 4, we show the simulation-based experiments performed to evaluate the proposed scheme, and we conclude the study in Section 5.

2. Related Work

In this section, we first describe the DASH-based adaptive streaming method. We then classify and present the representative model-based ABR algorithms of DASH. Finally, we illustrate RL-based adaptive streaming and introduce the existing schemes utilizing it.

2.1. DASH-Based Bitrate Adaptation

HTTP adaptive streaming, standardized as DASH, is the main scheme for video delivery over the Internet today. HTTP-based video delivery allows content providers to utilize existing content delivery networks at a low cost without additional modifications. In DASH, video is pre-stored on a server as multiple segments, each segment containing a portion of the video. A video segment is encoded to have various qualities for the purpose of performing quality adaptation according to the time-varying network conditions. Figure 1 shows the structure of DASH-based HTTP adaptive streaming. The client requests and downloads the media presentation description (MPD) file for the video to be transmitted to the server when video streaming begins. The MPD file describes information, such as the bitrate, size, playback length, and request address of an encoded segment in eXtensible Markup Language (XML) format. After that, the client requests the video segment through the ABR algorithm. This algorithm uses the information described in the MPD file and contextual information such as measured available bandwidth and current buffer level to

determine the quality of the next segment. Because DASH-based HTTP adaptive streaming selects the quality adaptively to suit the network situation, it provides a seamless streaming service to users and high QoE [8]. To provide a seamless streaming and high QoE to users, an ABR algorithm should be designed that takes into account the various contextual factors that affect the user's QoE. ABR algorithms can be broadly classified into model-based algorithms and learning-based algorithms. These can be further classified according to their contexts of use.

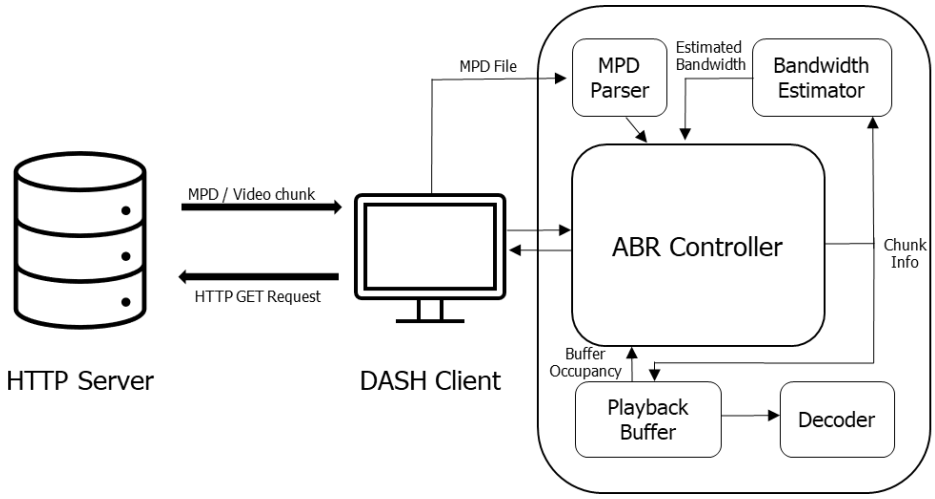

Figure 1. Structure of DASH-based HTTP Adaptive Streaming.

2.2. Model-Based ABR Algorithms

A model-based ABR algorithm focuses on building a segment request method in advance based on network bandwidth and playback conditions and makes ABR decisions based on that algorithm. This type of algorithm is also called a heuristic-based algorithm because it defines and uses a policy for optimizing the quality selection of the segment to be requested in the form of a fixed algorithm. Model-based ABR algorithms can be subdivided into three types: throughput-based models, buffer-based models, and hybrid models.

A throughput-based model estimates the available bandwidth using the time taken to download the segment, and the DASH client requests the segment with the maximum quality lower than the expected available bandwidth. Representative throughput-based models include conventional and adaptive streaming of audiovisual content (ASAC) [9,10]. The conventional method measures and flattens the next bandwidth using the exponentially weighted moving average (EWMA). The EWMA can minimize the error due to noise because it flattens the sudden change in bandwidth by simultaneously reflecting the previous bandwidth and the currently measured bandwidth. When measuring the current bandwidth, the size of the downloaded segment and the time taken to download it are used. The ASAC technique dynamically determines the flattening coefficient to adaptively determine the quality according to the bandwidth change. The flattening coefficient is increased when the difference between the measured and flattened bandwidth is large, and it is decreased when the difference is small. The throughput-based model, on the other hand, has the disadvantage of causing unnecessary quality changes and average quality degradation due to incorrect bandwidth measurement in an environment where network bandwidth changes rapidly.

The buffer-based model uses buffer-related data such as the current remaining buffer amount or predicted future buffer amount to determine the quality of the segment to be requested. Representative buffer-based models are buffer-based approach to rate adaptation (BBA) and buffer-based adaptation for adaptive HTTP streaming (BAHS) [11,12]. The BBA method divides the area of the buffer occupancy and maps the video quality to the area to determine the quality of the segment to be requested according to the current buffer occupancy. The BAHS technique adjusts the number of selectable qualities according to the

buffer occupancy. When the buffer occupancy is high, the number of selectable qualities is increased, and when the buffer occupancy is low, the number of selectable qualities is decreased. The quality to be requested is determined using the buffer occupancy, the number of selectable qualities, and a fixed buffer threshold. Buffer-occupancy-based quality adaptation techniques have the advantage of minimizing playback interruption even when bandwidth is rapidly reduced. However, there is a disadvantage in that QoE deteriorates due to frequent quality changes when the buffer occupancy is changed in the fixed buffer critical region.

To compensate for the shortcomings of the throughput-based model and the buffer-based model, a hybrid model approach that further improves the ABR algorithm was proposed. Representative hybrid models include adaptive rate-based intelligent HTTP streaming+ (ARBITER+) and model predictive control (MPC) [13,14]. ARBITER+ guarantees QoE by integrating the available bandwidth estimation method using the harmonic average, the hybrid method bandwidth sampling, the proportional integral controller, and the short-term actual bitrate meter to determine the quality of the next segment. An MPC uses an algorithm that combines throughput and buffer information for optimal quality selection. MPC uses the throughput predictor to select the optimal quality for the next five segments. In an environment with large network fluctuations, the throughput predictor does not work properly, and if the number of encoded quality segments is large, the computational complexity becomes very large when determining the segment to be requested.

A model-based ABR algorithm determines the quality of the segment to be requested through a fixed-control rule based on the observed information. In other words, in the short term, only the optimal quality at the current time is selected. However, since the ABR method of DASH is a kind of dynamic knapsack problem in which the maximum overall long-term reward must be obtained through a series of selections in a dynamic high-dimensional state space, it is difficult to achieve optimal QoE with a fixed-control rule.

2.3. Learning-Based ABR Algorithms

To overcome the shortcomings of existing model-based ABR algorithms, several learning-based ABR algorithms such as RL-based approaches have been proposed in recent years. RL is an appropriate learning method for achieving optimal QoE because it aims to maximize the expected cumulative reward for a given environment [15].

An RL-based ABR algorithm should be designed in consideration of the following issues: First, the network state fluctuates over time and has a large deviation depending on the client's environment. A high bitrate must be maintained even when the amount of remaining buffer is reduced to achieve optimal QoE in a link with a high and stable average bandwidth. In the opposite case, the client should consider the remaining buffer amount as a priority when determining the quality of a video segment to minimize playback interruption. Second, adaptive streaming must balance multiple goals, such as maximizing average quality, minimizing playback interruption, and minimizing quality fluctuations. Streaming by requesting only high-quality segments increases the average quality but increases the risk of playback interruption. Also, changing the quality of a segment whenever the bandwidth of the network fluctuates leads to frequent quality fluctuations, which is a cause of QoE degradation. Finally, the factors that affect QoE can vary greatly from user to user [16,17]. High QoE cannot be achieved when users who value maximum quality as the most important element of video streaming use an algorithm that aims to minimize playback interruption. Therefore, it is necessary to optimize QoE for various network conditions.

An RL-based ABR algorithm helps the agent experience various network conditions through learning. The goal of RL is to learn a policy suitable for the agent through trial and error, using the feedback from the actions as experience. After finding the optimal policy, the agent can dynamically adapt to various environmental conditions to maximize the predicted cumulative reward. In order to apply RL to the DASH system, ABR algorithms

applying the deep-Q learning (DQN) method and the actor–critic algorithm have been proposed [18,19]. However, the existing RL-based ABR algorithms have a disadvantage in that their performance is degraded when they are not used in the targeted network environment [20]. To address this shortcoming, we propose an online learning-based HTTP adaptive streaming technique that can update the learned neural network based on the network environment of a specific user.

3. Framework Design

This section describes the basic assumptions and framework of the proposed scheme. Because video content consists of scenes with exponentially distributed duration, the ABR algorithm of DASH can be modeled as a Markov decision process (MDP) problem, which is mainly used as a basic optimization method. Figure 2 shows how RL is applied to adaptive streaming. After receiving the state, the ABR agent decides the quality of the next requested segment through the neural network. The result after playing the requested segment (quality of the requested segment, playback interruption time, etc.) is reflected in the reward through the defined QoE model. The information related to the network and playback state after video playback is used as the state in making the next ABR decision. To design the framework of the proposed scheme, the state, action, reward, and neural network models should be specified, and the operation process of the framework should be configured in an algorithmic form.

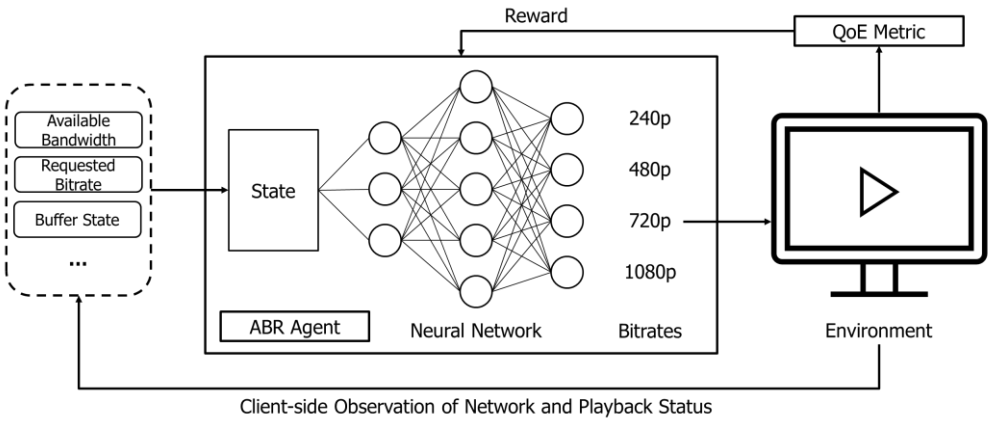

Figure 2. Adaptive streaming with reinforcement learning.

3.1. Basic Assumptions

The basic assumptions of the proposed scheme are as follows. First, the client has a criteria neural network model for performing early-stage adaptive streaming. The framework of the proposed scheme is intended to improve the performance degradation that occurs when an already trained neural network model operates in a non-targeted network environment through online learning. In this study, Pensieve is used as a criteria neural network model in the initial stage [21]. The model used does not necessarily have to be Pensieve, and content providers can choose arbitrarily according to the environment they want to apply. Second, during video streaming, the client's network status is collected and stored in real time. This is to improve the neural network model by identifying the network state when performance degradation occurs and performing additional online learning in a similar network environment. Finally, the client calculates the actual QoE according to the QoE model defined during video streaming and, at the same time, calculates the maximum QoE obtained in the current network environment. The two calculated values are used to fairly compare the QoE values obtained in different network environments.

3.2. MDP Problem Formulation

The MDP problem for modeling DASH's ABR algorithm consists of five elements as shown in Equation (1).

$$\mathcal{M} = < \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma > \quad (1)$$

$\mathcal{S}$ is the set of states as the state space, $\mathcal{A}$ is the set of actions as the action space, $\mathcal{P}$ is the transition probability function, $\mathcal{R}$ is the reward function, and γ is the discount factor for the future reward.

3.2.1. State Space

The agent creates and improves the policy π for selecting an action in the action space considering the input state. Therefore, it is important to decide which observation to use as the state to be used as an input. Because the input required for policy learning is insufficient when the state space is small, the optimal policy cannot be learned. When the size of the state space is large, the deviation greatly increases in the policy-learning process, which can lead to a local optimum, and the load according to the amount of computation increases. So as to quickly reach the global optimum and reduce the amount of computation, the trade-off according to the state space should be considered, and a state suitable for the optimization goal should be selected. The proposed scheme's optimization goal is to create a policy to improve the QoE value, and to do so, the state must use the observations related to the adaptive streaming QoE metric. We configured the state for time step t as shown in Equation (2).

$$\mathcal{S} = \left\{ \vec{x}_t, \vec{d}_t \vec{n}_t, b_t, f_t, s_t \right\} \quad (2)$$

$\vec{x}_t$ is a vector of the throughput of previously requested segments, $\vec{d}_t$ is a vector of download times for previously requested segments, $\vec{n}_t$ is a vector of the size of the next available video segment, b_t is the client's current remaining buffer, f_t is the predicted buffer occupancy, and s_t is the size of the last requested segment.

Unlike Pensieve, we use the predicted future buffer state as the state rather than the number of remaining segments. Since general RL-based adaptive streaming aims to maximize the expected cumulative reward, QoE can be improved when the remaining number of segments is used as the state. In particular, Pensieve tends to request more high-quality segments at the end of video streaming. This means that there is a situation in which a higher quality segment can be requested, but a lower quality segment is requested to maximize the expected cumulative reward. Therefore, the predicted buffer occupancy is used as the state to request the highest quality segment that does not cause playback interruption within the available bandwidth. The predicted buffer occupancy is calculated using Equation (3), where τ means the playback length of the segment.

$$f_t = b_t - d_t + \tau \quad (3)$$

3.2.2. Action Space

When the requested segment is completely downloaded, the client determines the size of the next requested segment in the action space according to the learned policy, considering the obtained state. We constructed the action space as in Equation (4).

$$\mathcal{A} = \{q_1, q_2, \ \ldots, \ q_L\} \quad (4)$$

L is the number of encoded segment qualities, and q_k means video segments of quality level k. We simplify the action space by assuming that the number of encoded segment qualities is fixed. There are several neurons in the output layer of the neural network model, and each neuron represents a selection probability of q_k.

3.2.3. Reward Space

Setting the reward in reinforcement learning is a very important task for learning the optimal policy. As mentioned in Section 3.2.1, the optimization goal of the proposed scheme is to create a policy to improve the QoE value. Therefore, in order to define the reward function, QoE should be modeled and reflected. Factors such as video quality, quality variations, and playback interruption affect a client's QoE. The existing method for modeling the client's QoE function linearly combines factors that generally affect the QoE value, as shown in Equation (5)

$$QoE = \sum_{k=1}^{K} r_k + \lambda \sum_{k=1}^{K-1} |r_{k+1} - r_k| + \rho T_n \tag{5}$$

r_k denotes a bitrate of the selected quality, T_n denotes a reproduction interruption time, and λ and ρ denote weights used to control the effects of quality change and playback interruption, respectively.

In the existing RL-based adaptive streaming techniques, a situation of requesting a low-quality segment occurs in a situation where a higher-quality segment can be requested to maximize the expected cumulative reward. Therefore, if these techniques don't operate in the targeted network environment, performance degradation occurs. To solve this problem, we extend the QoE function by considering the bandwidth usage rate when designing the reward function. The reward function is designed using Equation (6).

$$r = \mu_q q(R_i) + \mu_S T_i + \mu_{QS} |q(R_i) - q(R_{i-1})| + \mu_G G \tag{6}$$

$q(R_i)$ means the quality of the i-th video segment with bitrate R_i, G is the bandwidth usage rate, and μ_q, μ_S, μ_{QS}, and μ_G represent the normalization weights for reward used to scale between each QoE metric. In adaptive streaming, if a client requests a higher quality than the measured bandwidth, the probability of playback interruption increases due to the decreased buffer occupancy. If the difference between the bitrate of the requested quality and the measured bandwidth is large, the QoE is degraded because the bandwidth is not fully utilized. Therefore, in order to check whether high quality can be requested without any playback interruption, G is defined as shown in Equation (7).

$$G = \left| \frac{BW - R_{q,i}}{R_{q+1,i} - R_{q,i}} \right| \tag{7}$$

$R_{q,i}$ means the size of the i-th segment corresponding to the quality level q, and BW means the measured bandwidth. That is, G is a value obtained by normalizing the measured bandwidth based on the segment size of the requested quality and the segment size of a higher quality than the requested quality. When the G approaches 1, it means that the bandwidth utilization rate is small, and a penalty is given accordingly.

3.3. Neural Network Model

To learn the optimal policy, the neural network model must be properly configured. Neural network models should be large enough to generate optimal policies and low in complexity to minimize the load from training. We added a dropout layer and an early stopping technique to a simple structure using only a convolution layer and a fully-connected layer to increase the convergence speed of the model and reduce the variance of training [22,23]. The proposed scheme uses a learning algorithm based on the actor–critic algorithm. This method trains the actor network and the critic network simultaneously. The actor network plays a role in determining the action given a state, and the critic network evaluates the current value of the state and passes this information to the actor network to help update the weights. The existing RL algorithm takes a large amount of time to learn and has a problem in that the training result does not converge. This is due to the fact that when training episodes are long or the optimization target is complex, training

becomes more distributed. On the other hand, the actor–critic algorithm has the advantage of reducing the variance of training through a complementary training process.

Whenever a video segment is downloaded, the agent passes the state to the neural network model. Figure 3 shows how the state is passed to the neural network model, as well as the structure and output of the actor–critic network. A non-vector state is transmitted to a normal neuron, and a vector-type state is transmitted to a 1D-convolutional neural network (1D-CNN) neuron. The 1D-CNN layer is effective in extracting features for multiple inputs. The extracted features are transferred to the fully connected layer and used in the hidden layer. According to the parameters calculated in the hidden layer, the output layer of the actor network calculates the selection probability for video quality. As the calculated probability value selects a higher quality, the reward value increases, and a policy is created and trained based on it. The output layer of the critic network computes the current value of the state. This value is expressed as an integer and is used to update parameters in the actor network.

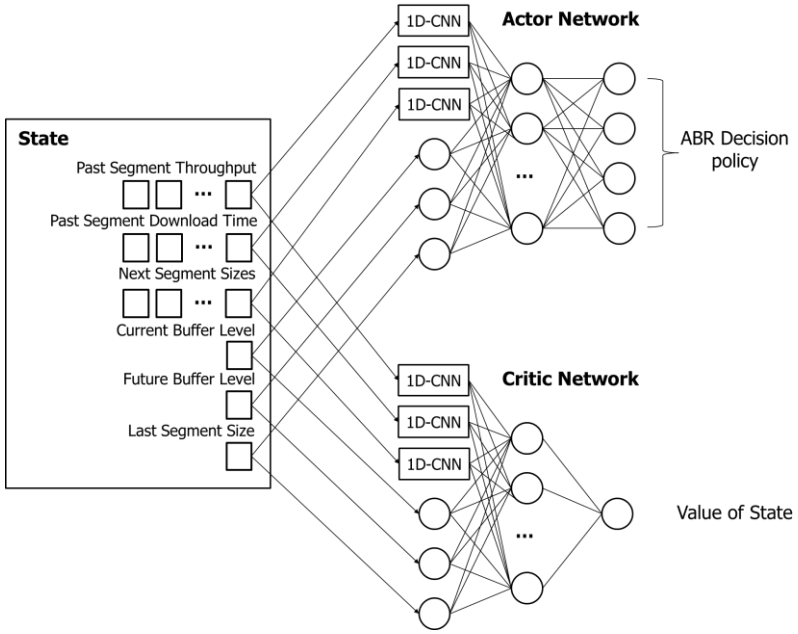

Figure 3. Architecture of actor network and critic network.

The proposed scheme trains the policy using the policy gradient method. The policy gradient method estimates the gradient of the expected total reward by observing the actions taken in accordance with the policy [24]. By updating the parameters of the neural network model, the agent adjusts the neural network model to select high-reward tasks more frequently. The slope of the expected cumulative reward is calculated as in Equation (8) by considering the policy parameters.

$$\nabla_\theta \mathrm{E}_{\pi\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] = \mathrm{E}_{\pi\theta}[\nabla_\theta \log\ \pi_\theta(s,a)A^{\pi_\theta}(s,a)] \tag{8}$$

γ is a discount factor to control the effect of future reward, and r_t is the reward value at time step t. θ means policy parameters, $\pi_\theta(s,a)$ is a policy that performs action a according to state s, and $A^{\pi_\theta}(s,a)$ is an advantage function, meaning the difference between the predicted reward when selecting action a in state s and the predicted reward from the policy π_θ. That is, the advantage function compares how much better a specific action is compared to the average action taken according to the policy.

For the purpose of maximizing the agent's predicted cumulative reward, it is important to update the parameters for the actor network and the critic network. The expression for updating the parameters in the actor network is shown in Equation (9).

$$\theta \leftarrow \theta + \alpha \sum_{t} \nabla_{\theta} \log \ \pi_{\theta}(s_t, a_t) A(s_t, a_t) \tag{9}$$

α denotes the learning rate, and s_t and a_t denote the state and action, respectively, at time step t. $\nabla_{\theta} \log \ \pi_{\theta}(s_t, a_t)$ specifies how to change the policy parameters to increase the probability of $\pi_{\theta}(s_t, a_t)$.

To calculate the advantage function $A(s_t, a_t)$, we need to calculate the value function $v^{\pi_{\theta}}(s)$. $v^{\pi_{\theta}}(s)$ means the total reward expected according to the policy π_{θ} starting from the state s. The main role of the critic network is to learn how to predict the value function from the calculated reward values. In this study, unlike Pensieve, which uses the Monte Carlo method to quickly update the parameters of the critic network, the temporal difference (TD) method is used as shown in Equation (10) [25,26].

$$\theta_v \leftarrow \theta_v + \alpha' \sum_{t} \nabla_{\theta_v} \left(r_t + \gamma^{V^{\pi_{\theta}}}(s_{t+1}; \theta_v) - V^{\pi_{\theta}}(s_t; \theta_v) \right)^2 \tag{10}$$

$V^{\pi_{\theta}}(s_t; \theta_v)$ is the output of the critic network, meaning the estimated value of the value function, and α' means the learning rate of the critic network. If the TD method is used, the difference between the calculated value functions can be used as an advantage function. The critic network only participates in parameter updates of the actor network. After training on the actor network and the critic network is finished, the agent performs adaptive streaming using only the actor network.

Finally, in order for the agent to learn the optimal policy, it needs to properly navigate the action space during training. To this end, we added an entropy regularization term to Equation (9), which is an updated method of the actor network, as shown in Equation (11).

$$\theta \leftarrow \theta + \alpha \sum_{t} \nabla_{\theta} \log \ \pi_{\theta}(s_t, a_t) A(s_t, a_t) + \beta \nabla_{\theta} H(\pi_{\theta}(\cdot|s_t)) \tag{11}$$

$H(\pi_{\theta}(\cdot|s_t))$ means the probability distribution for the action of the policy at each time step t. By adding this entropy regularization term, we update θ in the direction of increasing entropy so that the search for the action space occurs more actively. β is the entropy weight, which is set to a large value at the beginning of learning and gradually decreases with time.

3.4. Operation of Framework

3.4.1. Framework Workflow

In this study, we propose an HTTP adaptive streaming framework with online reinforcement learning to address the disadvantage that existing RL-based ABR algorithms degrade in performance when not employed in the target network environment. If the QoE of the client decreases, the proposed scheme adapts the ABR algorithm to changes in the network environment by updating the ABR neural network model while video streaming is performed. Here, the network environment change does not mean network fluctuations in the DASH environment but a case in which the ABR neural network model operates in an environment different from the environment in which the learning is performed. Because the proposed scheme is a framework for improving the previously trained model, it is assumed that the client has a pre-trained model when video streaming starts. We use Pensieve as a pre-trained reference model. The operational structure of the proposed scheme is shown in Figure 4, and the pseudocode to describe the complete algorithm in detail is presented in Algorithm 1. The client performs adaptive video streaming using the criteria model. At the same time, the state of the network is stored, and the network is classified using the average and deviation of the bandwidth. At this time, the QoE value calculated during video streaming and the maximum QoE value obtained from the current

network state are calculated to normalize the QoE value. The network state and normalized QoE values are stored in a first-in-first-out (FIFO) manner, with a video of length of 2 min. When the normalized QoE value is lower than the threshold set by the algorithm, the client creates an agent and updates the model through reinforcement learning. The network trace data has been classified in the same way that the current network is classified. The client uses a network trace similar to the current network state for training. How to calculate the normalized QoE and how to classify networks will be described in detail later. If a training agent is created based only on the normalized QoE value, the risk of overfitting increases because an agent that conducts training using the same network trace data is generated. Therefore, we make the created agents learn by using different network trace data. Each agent uses an early stopping technique to terminate training early when the increase in validation accuracy becomes small. The neural network model is updated whenever each created agent's training ends. The proposed scheme uses an asynchronous advantage actor–critic (A3C) network because there is no central network, and the model needs to be updated asynchronously every time an agent completes training [27].

Algorithm 1 HTTP Adaptive Streaming with Online Reinforcement Learning

$\langle \mathcal{N} \rangle$: classified network trace data
QoE_{nom} : normalized QoE
th : threshold for normalized QoE
Bw : network bandwidth
$\langle \mathcal{S} \rangle$: streaming trace data
$\langle \mathcal{D} \rangle$: duplicate trace check vector
$\langle \mathcal{A} \rangle$: waiting agent
$\langle \mathcal{T} \rangle$: each agent's trace data
Initialize the $k = 0$, $i = 1$
While Streaming session
  Compute Network bandwidth Bw_i
  Compute Normalized QoE QoE_{nomi}
  If $\langle \mathcal{S} \rangle$'s length $<$ 2 − minutes
    Enqueue $\langle Bw_i\ ,\ QoE_{nomi} \rangle$ to $\langle \mathcal{S} \rangle$
  Else
    Dequeue $\mathcal{S}_1$
    Enqueue $\langle Bw_i\ ,\ QoE_{nomi} \rangle$ to $\langle \mathcal{S} \rangle$
    Compute Average and variance of $\mathcal{S}(Bw_i)$ in the form of Bw_{avg_i} and Bw_{var_i}
    Leveling the Bw_{avg_i} and Bw_{var_i} to $\langle \hat{B}w_{avg_i}, \hat{B}w_{var_i} \rangle$
    If Average of $\mathcal{S}(QoE_{nom}) < th$ & $\langle \hat{B}w_{avg_i}, \hat{B}w_{var_i} \rangle$ is not in $\mathcal{D}$
      Enqueue $\langle \hat{B}w_{avg_i}, \hat{B}w_{var_i} \rangle$ to $\mathcal{D}$
      If $\mathcal{A}'s$ length is less than # of thread that can be used
        Training $Agent$ using $\mathcal{N}$that matches $\langle \hat{B}w_{avg_i}, \hat{B}w_{var_i} \rangle$
      Else
        Enqueue $\langle \hat{B}w_{avg_i}, \hat{B}w_{var_i} \rangle$ to $\mathcal{T}$
        Enqueue $Agent$ to $\mathcal{A}$
        Dequeue $\mathcal{A}(1)$ and $\mathcal{T}(1)$ & training $\mathcal{A}(1)$ using $\mathcal{T}(1)$

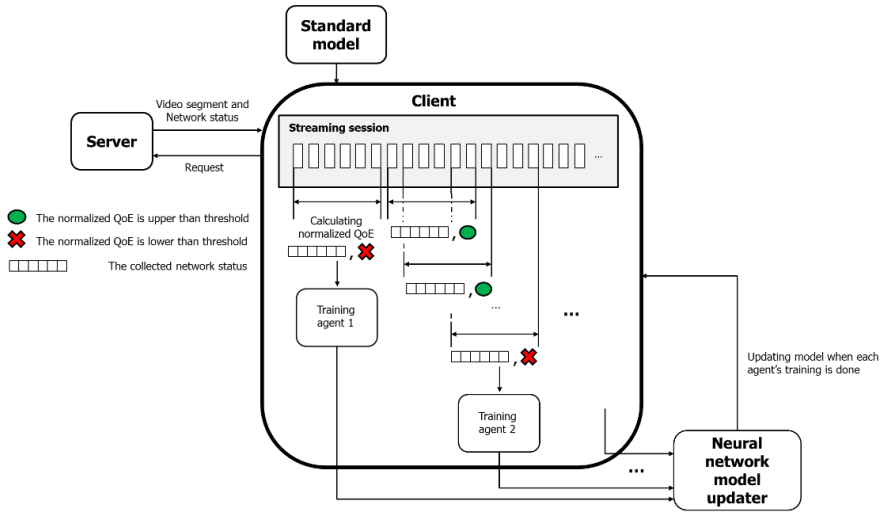

Figure 4. Operation of the proposed scheme.

3.4.2. QoE Normalization

When comparing algorithm performance in various network environments, there is a limit to using the actual QoE value. When comparing the performance of algorithms operating in a rich network environment and in a poor network environment, the actual QoE is lower in a poor network environment. Therefore, to use the QoE value as a threshold in a changing network environment, it is necessary to normalize the QoE value. QoE normalization is performed using the actual QoE value calculated using the QoE function of Equation (5) and the maximum QoE value obtained from the current network state. The formula for QoE normalization is expressed as Equations (12) and (13).

$$Q_{max} = \sum_{k=1}^{K} \hat{r}_k - \lambda \sum_{k=1}^{K-1} |\hat{r}_{k+1} - \hat{r}_k| \tag{12}$$

$$Q_{nom} = \frac{Q}{Q_{max}} 100 \tag{13}$$

Q_{max} means the maximum QoE value in the current network state, and Q_{nom} means the normalized QoE value. Since Q_{max} means the maximum QoE value only at the current time in the short term, it is calculated using the maximum quality $\hat{r}_k$ of the segment k that can be requested in the available bandwidth without considering the metric related to playback interruption.

3.4.3. Network Classification

If the client's performance is degraded in a specific network environment, the performance should be improved by intensive learning in the network environment. Therefore, the proposed scheme should classify networks by characteristics. The network characteristics include the capacity of the network and the degree of fluctuation. Network classification is performed using the average ($\frac{\sum_{i=1}^{n} b_{\Delta_i}}{n}$) and variance ($\frac{\sum_{i=1}^{n} (b_{\Delta_i} - BW_e)^2}{n}$) of the bandwidth. The average and variance of the bandwidth are calculated by quantizing the network traces collected over a 2 min length video into time steps ($\Delta_1 \ldots \Delta_n$). We used a total of eight levels, from 0 Mbps to 7 Mbps, based on the rounded-down value of the average bandwidth. This classification level can be set variably according to the environment in which the learning is performed. We further classify the networks in more detail based on their fluctuation levels. As a result, we classified the networks into a total of 80 classes (an average of 8 levels and a variance of 10 levels).

4. Performance Evaluation

In this section, we first describe the implementation details of our neural network model and then show the network datasets used for training and testing the neural network model, the existing schemes for performance comparison, and the evaluation metrics. The evaluation result compares the normalized QoE values of the proposed scheme with those of the existing schemes.

4.1. Implementation

In this study, TensorFlow is used to implement the neural network model of the proposed scheme, and the TFLearn deep learning library is used for training and testing [28,29]. There are 128 filters in the 1D-CNN layer of each neural network model. The size of each filter is 4, and the size of the stride is 1. The features of the state extracted through the 1D-CNN layer are transferred to the hidden layer using 128 neurons. In the actor network, the number of output layers equals the number of available video qualities. Table 1 describes the parameters used in the proposed scheme.

Table 1. Parameters and their values used in the proposed scheme.

Parameters	Description	Value
γ	Discount factor	0.99
λ, ρ	Weight parameters used in the QoE function	1.0, 4.3
α, α'	Learning rate for the actor-critic network	10^{-4}, 10^{-3}
β	Entropy weight	1 to 0.1 (over 10^6 iterations)

The ideal way to train a neural network model is to learn by using real video data and network environments. To perform training in a real environment, the agent must update the parameters of the neural network model after video streaming is fully completed. However, since reinforcement learning requires many iterations, there is a limit to learning in a real environment. Therefore, a simulator based on the DASH system is implemented and used to train the reinforcement-learning-based ABR algorithm. The neural network model of the proposed scheme can perform adaptive streaming for hundreds of hours of video in a short time using this simulator.

4.2. Experimental Settings

To evaluate the performance of the proposed scheme and the existing schemes, a real network dataset and a dataset generated by adjusting the network capacity and seed value with Network Simulator 3 (NS3) were used [30]. For the real network dataset, a broadband dataset provided by the FCC, a 3G/HSDPA mobile dataset from Norway, and a 4G/LTE mobile dataset from Belgium were used [31–33]. The real network dataset was used as the agent's training data, and the dataset created using NS3 was used to operate the algorithm of the proposed scheme. Table 2 shows the characteristics of the network dataset used for the test, and Table 3 shows the characteristics of the existing schemes to be compared with the proposed scheme.

Before presenting the experimental results of the proposed scheme and the existing schemes, the video used, the evaluation scenario, and other factors are described. The video is constructed based on the real video of the DASH-246 JavaScript reference player [34]. The video is encoded with {240p, 360p, 480p, 720p, 1080p} resolution and the H.264/MPEG-4 codec. The segment playback length is set to 2 s. Clients can experience different network environments by randomly choosing a bandwidth from the test set. The maximum buffer level of the client is set to 60 s, and online learning is performed using a 24 h dataset. The experimental results are derived based on the model after the online learning is completed.

Table 2. Throughput information of network datasets.

	Trace Dataset						
Characteristics	#1	#2	#3	#4	#5	#6	#7
Mean Throughput (Mbps)	0.98	1.57	2.62	3.23	4.75	5.79	6.52
Max Throughput (Mbps)	3.45	3.38	6.54	8.97	8.23	9.65	11.53
Min Throughput (Mbps)	0.09	0.23	0.21	0.38	0.26	0.36	0.39
Coefficient of Variation	0.76	0.44	0.56	0.72	0.35	0.39	0.37

Table 3. Analysis of existing video adaptation schemes for performance evaluation.

Scheme	Characteristics
robustMPC	1. Model predictive control-based adaptation 2. Traditional video adaptation scheme
Pensieve	1. Reinforcement learning (Server-side) 2. QoE model with perceptual quality 3. QoE degradation for specific network

4.3. Results

First, according to the evaluation scenario described in Section 4.2, normalized QoE values and target reward values for the proposed scheme and the existing schemes were measured. Figure 5 shows the normalized QoE values and target reward values measured for each test dataset. The proposed scheme has the highest normalized QoE and reward values compared to the existing schemes. This shows the effectiveness of the method of updating the neural network model according to the network environment of the proposed scheme. In particular, when the proposed scheme and Pensieve are compared using Dataset #1 and Dataset #2 with low average bandwidth values, a more pronounced performance improvement is shown. Through this, we show that the proposed scheme effectively improves QoE in an environment where the performance of the existing schemes has deteriorated.

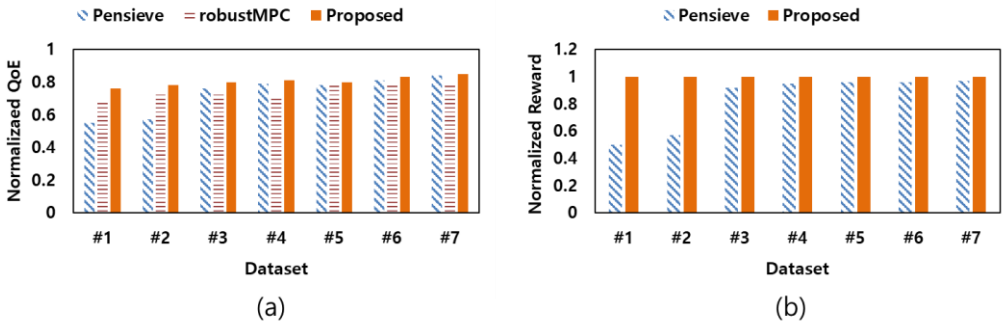

Figure 5. Normalized QoE and reward according to the dataset, (**a**) normalized QoE, (**b**) normalized reward.

Figure 6 shows how much performance improvement occurs and when it converges with time from the time online learning is started. This experiment was conducted based on Dataset #2, in which the performance improved most clearly by about 21%. The proposed scheme uses the A3C network to update the model. As a result, it can be confirmed that the performance increases linearly according to the time step despite the asynchronous model updating of each agent. Also, through the experimental results, it can be confirmed that the performance improvement converges after about 18 h. This is due to the time it takes for each agent's online learning to complete and the model to update, implying that additional techniques are required to improve the learning rate.

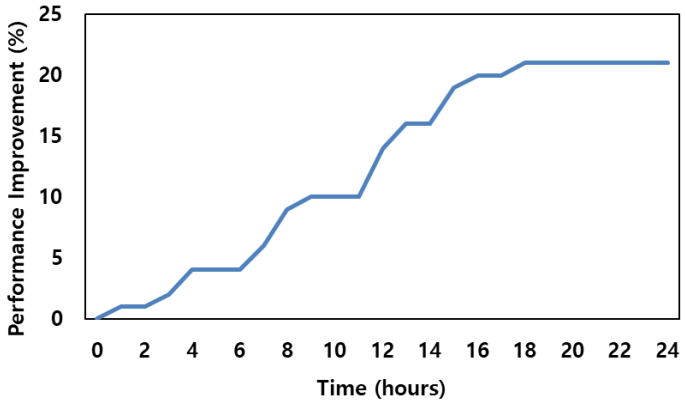

Figure 6. The degree of performance improvement over time in online learning.

Figure 7 shows the results of experiments in the same environment after increasing the number of threads used by each agent to improve the training speed of each agent. As the number of threads increases, the degree of performance improvement increases with time. In particular, when four threads are used, the degree of performance improvement converges after about 12 h has elapsed. If more threads are used, the performance improvement is predicted to occur faster, but when several agents are training at the same time, the risk of an agent that cannot use multi-threading increases depending on the computing power. In addition, for the model to be updated, the normalized QoE value must fall below the threshold, and the state of the network must also change. As a result, even if more threads are used, performance cannot be improved beyond a certain point unless the network has very high bandwidth fluctuations.

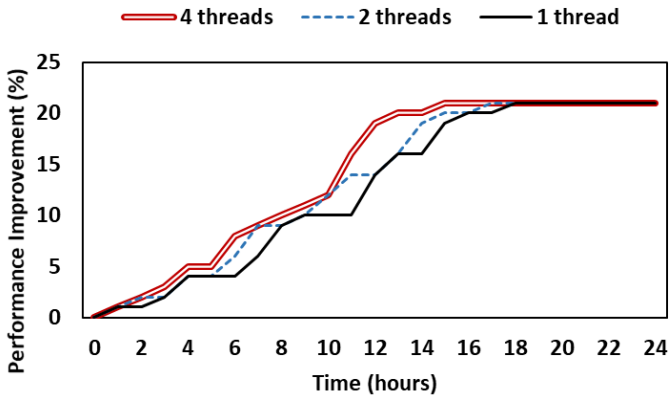

Figure 7. The degree of performance improvement according to time and number of threads in online learning.

Figure 8 shows the results of an experiment using a dataset with a very wide bandwidth fluctuation to check the degree of performance improvement according to an increase in the number of threads in a rapidly changing network environment. The performance improvement is more pronounced when two threads are used than when one thread is used. This is due to the rate at which the model is updated becoming faster as a large number of agents are created in accordance with occurring QoE degradation and network fluctuations rapidly in the early stage of online learning (0 to 10 h) and the number of threads increases.

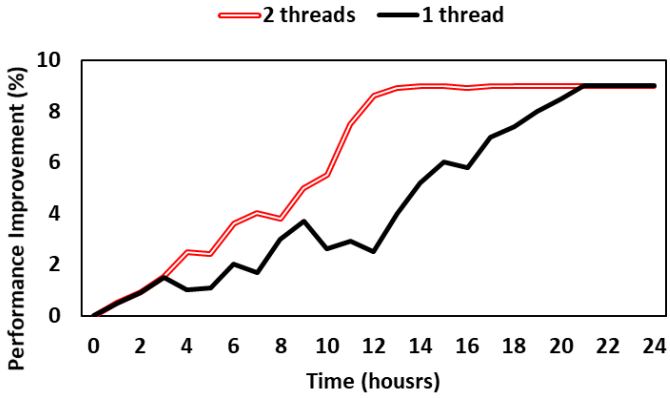

Figure 8. The degree of performance improvement according to time and number of threads of online learning in the dynamic network environment.

Figure 9 shows the results of the experiment by changing the threshold value, which is the criterion in generating the agent, from 0.3 to 0.9 by comparing the normalized QoE. Experimental results show that the normalized QoE value increases as the threshold increases. However, if the threshold value increases, the performance evaluation criteria for the neural network model decrease, and the number of agents generated accordingly also increases. Since the increase in the number of agents leads to serious computational load, it is necessary to solve the trade-off between QoE improvement and load. To this end, we set the threshold at 0.7.

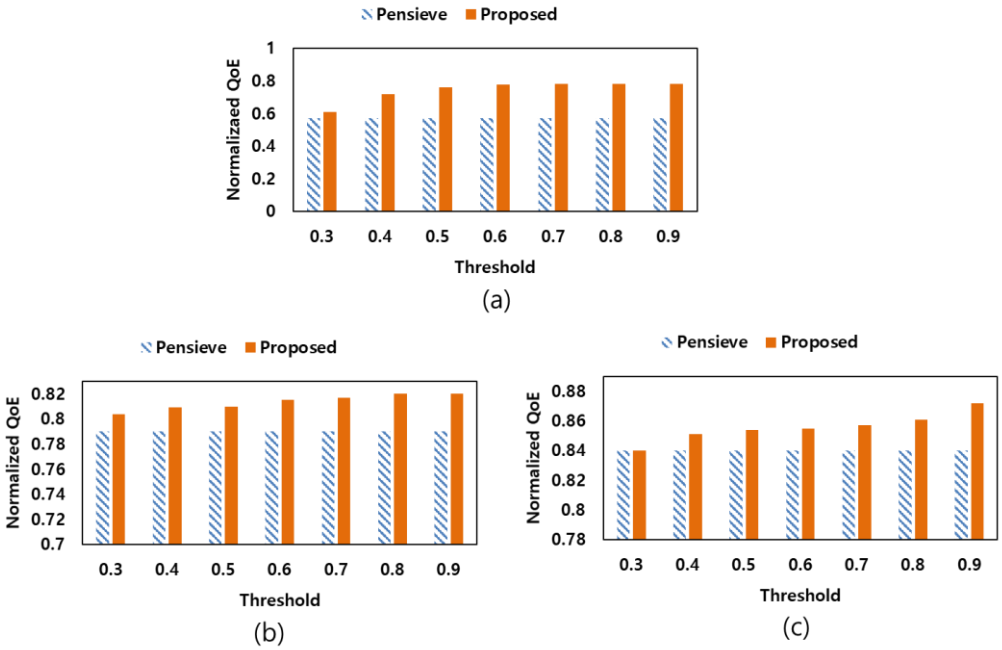

Figure 9. Change of the normalized QoE according to change of threshold value, (**a**) Dataset #2, (**b**) Dataset #4, (**c**) Dataset #7.

4.4. Discussions

From the experimental results, it can be confirmed that the proposed scheme has better performance than the existing schemes used as the reference model. In particular, we see a noticeable performance improvement of about 21% in Dataset #1 and Dataset #2 in low-bandwidth environments of the network. This is due to the mechanism of adapting neural network models to network environments by detecting QoE degradation in the proposed scheme.

However, considerable time is required for performance improvement to adapt the neural network model to the network environment. Therefore, to increase the efficiency of the proposed scheme, it is necessary to shorten the learning time. Experimental results confirm that the proposed scheme reduced the performance improvement rate of neural network models by regulating the number of threads used for training. Users can expect better performance by appropriately adjusting the number of threads according to the computing power and the characteristics of the network environment in which streaming is performed.

5. Conclusions

In this study, we present an adaptive streaming scheme based on online reinforcement learning. The proposed scheme aims to provide a network-aware adaptive streaming scheme by improving the performance of the existing learning-based adaptive streaming scheme from being degraded according to the network environment. The proposed scheme uses a state-of-the-art RL algorithm, trains the neural network model by improving the state and reward, and adapts the neural network model to the changing network environment by introducing network classification and QoE regularization. Because it is a framework that improves its performance based on the existing learning-based adaptive streaming technique, the proposed scheme can be used to extend learning-based ABR algorithms regardless of the learning method. Through the experimental results, we confirmed that the proposed scheme successfully improves QoE according to the changing network environment. In future research, we plan to study how to further increase the convergence speed of the model and how to allow the model to have generality by sharing parameters between updated models in different network environments.

Author Contributions: Conceptualization, J.K. and K.C.; methodology, J.K.; simulation, J.K.; validation, J.K. and K.C.; formal analysis, J.K.; investigation, J.K.; resources, J.K. and K.C.; data curation, J.K.; writing—original draft preparation, J.K.; writing—review and editing, J.K. and K.C.; visualization, J.K. and K.C.; supervision, K.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (No. 2020R1F1A1048627, Collaborative Media Streaming Model Based on Mobile Edge Computing). It was also supported by the Kwangwoon University Excellent Researcher Support Program in 2022.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cisco Public. Cisco Annual Internet Report (2018–2023) White Paper. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed on 15 March 2022).
2. Kua, J.; Armitage, G.; Branch, P. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP. In *IEEE Communication Surveys & Tutorials*; IEEE: Piscataway, NJ, USA, 2017; pp. 1842–1866.
3. Timmerer, C.; Sodogar, I. *Ad Hoc on HTTP Streaming of MPEG Media*; ISO/IEC JTC1/SC29/WG11/M176 57; ISO: Geneva, Switzerland, 2010.
4. Zambelli, A. *IIS Smooth Streaming Technical Overview*; Microsoft Corporation: Redmond, WA, USA, 2009.
5. Pantos, R.; May, W.; HTTP Live Streaming. IETF Draft. August 2017. Available online: https://datatracker.ietf.org/doc/html/rfc8-216 (accessed on 20 April 2022).
6. Adobe HTTP Dynamic Streaming. Available online: http://www.adobe.com/products/httpdynamicstreaming/ (accessed on 20 April 2022).
7. Petrangeli, S.; Hooft, J.V.D.; Wauters, T.; Turck, F.D. Quality of Experience-Centric Management of Adaptive Video Streaming Services: Status and Challenges. *ACM Trans. Multimed. Comput. Commun. Appl.* **2018**, *14*, 1–29. [CrossRef]

8. Bae, S.; Jang, D.; Park, K. Why is HTTP Adaptive Streaming So Hard? In Proceedings of the Asia-Pacific Workshop on Systems, Tokyo, Japan, 27–28 July 2015; pp. 1–8.
9. Li, Z.; Zhu, X.; Gahm, J.; Pan, R.; Hu, H.; Began, A.C.; Oran, D. Probe and Adapt: Rate Adaptation for HTTP Video Streaming at Scale. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 719–733.
10. Thang, T.; Ho, Q.; Kang, J.; Pham, A. Adaptive Streaming of Audiovisual Content Using MPEG DASH. *IEEE Trans. Consum. Electron.* **2012**, *58*, 78–85.
11. Huang, T.; Johari, R.; McKeown, N.; Trunnell, M.; Waston, M. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In Proceedings of the ACM Conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 187–198.
12. Le, H.; Nguyen, D.; Ngoc, N.; Pham, A.; Thang, T.C. Buffer-based Bitrate Adaptation for Adaptive HTTP Streaming. In Proceedings of the IEEE International Conference on Advanced Technologies for Communications, Ho Chi Minh, Vietnam, 16–18 October 2013; pp. 33–38.
13. Zahran, A.H.; Raca, D.; Sreenan, C.J. Arbiter+: Adaptive Rate-based Intelligent HTTP Streaming Algorithm for Mobile Networks. *IEEE Trans. Mob. Comput.* **2018**, *17*, 2716–2728. [CrossRef]
14. Yin, X.; Jindal, A.; Sekar, V.; Sinopoli, B. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, New York, NY, USA, 22–26 August 2015; pp. 325–338.
15. Huang, T.; Zhang, R.X.; Zhou, C.; Sun, L. Video Quality Aware Rate Control for Real-time Video Streaming based on Deep Reinforcement Learning. In Proceedings of the 26th ACM International Conference on Multimedia, New York, NY, USA, 22–26 October 2018; pp. 1208–1216.
16. Mok, R.K.P.; Chan, E.W.W.; Luo, X.; Chang, R.K.C. Inferring the QoE of HTTP Video Streaming from User-Viewing Activities. In Proceedings of the First ACM SIGCOMM Workshop on Measurements up the Stack, Toronto, ON, Canada, 19 August 2011; pp. 31–36.
17. Mok, R.K.P.; Chen, E.W.W.; Chang, R.K.C. Measuring the Quality of Experience of HTTP Video Streaming. In Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops, Dublin, Ireland, 23–27 May 2011; pp. 485–492.
18. Gadaleta, M.; Chiariotti, F.; Rossi, M.; Zanella, A. D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Trans. Cogn. Commun. Netw.* **2017**, *20*, 703–718. [CrossRef]
19. Konda, V.R.; Tsitsiklis, J.N. Actor-Critic Algorithms. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2000; pp. 1008–1014. Available online: https://proceedings.neurips.cc/paper/1999/file/6449f44a102fde8-48669bdd9eb6b76fa-Paper.pdf (accessed on 9 April 2022).
20. Zhang, G.; Lee, J.Y. Ensemble Adaptive Streaming–A New Paradigm to Generate Streaming Algorithms via Specializations. *IEEE Trans. Mob. Comput.* **2020**, *19*, 1346–1358. [CrossRef]
21. Mao, H.; Netravali, R.; Alizadeh, M. Neural Adaptive Video Streaming with Pensieve. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017; pp. 197–210.
22. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
23. Duvenaud, D.; Maclaurin, D.; Adams, R. Early Stopping as Nonparametric Variational Inference. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, Cadiz, Spain, 9–11 May 2016; pp. 1070–1077.
24. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Proceedings of the 12th International Conference on Neural Information Processing Systems, Denver, CO, USA, 29 November–4 December 1999; pp. 1057–1063.
25. Lazaric, A.; Restelli, M.; Bonarini, A. Reinforcement Learning in Continuous Action Spaces Through Sequential Monte Carlo Methods. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 3–8 December 2007.
26. Kurth-Nelson, Z.; Redish, A.D. Temporal-Difference Reinforcement Learning with Distributed Representations. *PLoS ONE* **2009**, *4*, e7362. [CrossRef]
27. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
28. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. TensorFlow: A System for Large-Scale Machine Learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
29. TFLearn. TFLearn: Deep Learning Library Featuring a Higher-Level API for TensorFlow. Available online: http://tflearn.org/ (accessed on 23 February 2022).
30. The Network Simulator NS-3. Available online: http://www.nsnam.org (accessed on 29 December 2021).
31. Federal Communications Commission (FCC). Raw Data-Measuring Broadband America Mobile Data. Available online: https://www.fcc.gov/reports-research/reports (accessed on 17 March 2022).
32. Riiser, H.; Vigmostad, P.; Griwodz, C.; Halvorsen, P. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In Proceedings of the 4th ACM Multimedia Systems Conference, Oslo, Norway, 28 February 2013; pp. 114–118.

33. Hooft, J.V.D.; Petrangeli, S.; Wauters, T.; Huysegems, R.; Alface, P.R.; Bostoen, T.; Turck, F.D. HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks. *IEEE Commun. Lett.* **2016**, *20*, 2177–2180. [CrossRef]
34. DASH Industry Forum. Reference Client 2.4.0. Available online: https://reference.dashif.org/dash.js/v2.4.0/samples/dash-ifreference-player/index.html (accessed on 15 May 2021).