Computer Science and Information Systems 20(1):157–173 ttps://doi.org/10.2298/CSIS220927055N

Reinforcement Learning - based Adaptation and Scheduling Methods for Multi-source DASH ⋆

Nghia T. Nguyen[1,2], Long Luu[1,2], Phuong L. Vo[1,2], Thi Thanh Sang Nguyen[1,2],
Cuong T. Do[3], and Ngoc Thanh Nguyen[4]

[1] School of Computer and Engineering, International University, Ho Chi Minh City, Vietnam
[2] Vietnam National University, Ho Chi Minh City, Vietnam
{ntnghia, vtlphuong, nttsang}@hcmiu.edu.vn
ITITIU18079@student.hcmiu.edu.vn
[3] Department of Computer Engineering, Kyung Hee University, 446-701, Korea
dtcuong@khu.ac.kr, dothecuong@gmail.com
[4] Faculty of Information and Communication Technology,
Wroclaw University of Science and Technology, Poland
ngoc-thanh.nguyen@pwr.edu.pl

Abstract. Dynamic adaptive streaming over HTTP (DASH) has been widely used in video streaming recently. In DASH, the client downloads video chunks in order from a server. The rate adaptation function at the video client enhances the user's quality-of-experience (QoE) by choosing a suitable quality level for each video chunk to download based on the network condition.

Today networks such as content delivery networks, edge caching networks, content-centric networks, *etc.* usually replicate video contents on multiple cache nodes. We study video streaming from multiple sources in this work. In multi-source streaming, video chunks may arrive out of order due to different conditions of the network paths. Hence, to guarantee a high QoE, the video client needs not only rate adaptation, but also chunk scheduling.

Reinforcement learning (RL) has emerged as the state-of-the-art control method in various fields in recent years. This paper proposes two algorithms for streaming from multiple sources: *RL-based adaptation with greedy scheduling* (RLAGS) and *RL-based adaptation and scheduling* (RLAS). We also build a simulation environment for training and evaluation. The efficiency of the proposed algorithms is proved via extensive simulations with real-trace data.

Keywords: multi-source streaming, reinforcement learning, proximal policy optimization, dynamic adaptation streaming over HTTP.

1. Introduction

A significant part of Internet traffic today is video streaming [1]. Dynamic adaptive streaming over HTTP (DASH) is the primary technique to stream a video from a server to a video player. In DASH, videos are encoded in multiple quality levels. Furthermore, videos are

⋆ This is an extended version of the conference paper [20].
This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number DS2020-28-01. Dr. Phuong L. Vo is the corresponding author.

partitioned into video chunks. Each chunk contains media data in a short interval of playback time. Video players request the chunks with suitable quality levels based on the current network condition [2–5]. The downloaded chunks are buffered in the client's memory before being played. Buffer size is the total playing time of the wait-to-be-played chunks. When a new video chunk is successfully downloaded, the buffer size increases by a chunk length. When a chunk is played, the buffer size is decreased by the chunk length. The buffer size has an upper threshold level. When the buffer exceeds the threshold, the client will pause downloading a new chunk, wait for the buffer to decrease below the threshold, and then resume downloading. The client *rebuffers* when the chunk will be played is not in the buffer. Rebuffering causes video freezes.

The rate adaptation function in video clients is essential in providing a high quality-of-experience (QoE) for the user. Various adaptation methods are proposed for DASH. Throughput-based adaptation method chooses the quality level for the next chunk such that it does not exceed the estimated throughput [6,7]. The throughput is usually estimated by the mean or harmonic mean of several last requested chunks. The buffer-based methods observe the buffer level to decide the encoding quality level [8, 9]. Both the throughput-based method and BOLA, a buffer-based method [8], are employed in Dash.js reference client [6]. Some methods combine both these two approaches [10].

On the other hand, several networks today such as content delivery networks, edge caching networks, content-centric networks, *etc.* replicate popular videos at the routers to reduce network congestion and delay. Utilizing multiple sources to stream a video to a user is studied in this paper. When streaming from multiple sources, quality control is much more complicated than streaming from a single source. In multi-source streaming, the quality control includes not only *rate adaptation*, *i.e.*, choosing the quality levels for the chunks, but also *chunk scheduling*, *i.e.*, which chunk indices are requested on each path (see Fig. 1). Due to the difference in the network conditions of the connections, the chunks may arrive at the video client out of order. For example, assume that the maximum buffer size of the client is 3 chunks and there are two paths. With bad scheduling, path 2 is downloading chunk 2 while path 1, with very high throughput, already downloaded chunks 1, 3, 4. Therefore, the buffer is full, however, the video playing is frozen since the client waits for chunk 2.

Some previous works have studied multi-source streaming [11–13]. In [11], MSPlayer can download video content from multiple servers. The authors in [11] consider the chunks with only one quality level, however, the chunk size varies. They focus on the chunk scheduling problem. The client estimates the path quality to request chunk indices and chunk sizes for the paths. In work [12], MP-H2 protocol is designed on top of HTTP/2. MP-H2 splits the video into many chunks, and the client requests chunks over multiple network connections such as wi-fi and cellular. Chunk sizes are calculated based on bandwidth and round-trip-time of the connections. A chunk scheduling algorithm is then used to download the chunks over multiple paths. No adaptation method is proposed in [12]. The work [13] has proposed a bitrate adaptation algorithm for DASH, called DQ-DASH, that allows downloading multiple video chunks from various servers in parallel to enhance QoE. Distributed queueing theory is applied to address the situation when multiple clients send requests to many servers simultaneously. Fair QoE across clients is considered in the model. Different from [11–13], our proposed framework jointly considered rate adaptation and chunk scheduling.

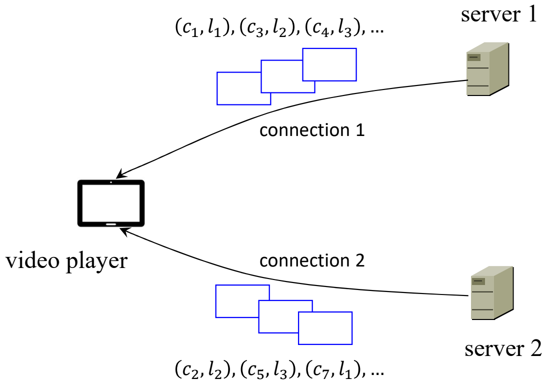
$(c_1, l_1), (c_3, l_2), (c_4, l_3), \ldots$
server 1
connection 1
video player
connection 2
server 2
$(c_2, l_2), (c_5, l_3), (c_7, l_1), \ldots$

Fig. 1. Multi-source video streaming.

Reinforcement learning (RL) has been widely used in many fields recently. The works [14–16] have applied RL algorithms for single-source adaptive streaming. The actions of RL agents in these works are the quality levels of video chunks. The work [14] applied a Q-learning method for DASH. Buffer and network bandwidth are discretized for the discrete state space. The study [15] applied Asynchronous Advantage Actor-Critic (A3C) algorithm for rate adaptation. The work [16] has proposed D-DASH that applied a Deep Q-learning to choose the quality level for the chunks.

In this work, we also use RL algorithm for rate adaptation and chunk scheduling in streaming from multiple sources. However, there are several challenges we cannot simply extend the RL framework for single-source streaming in [15,16] to multi-source streaming straightforwardly:

- The action space of multi-source streaming must be redesigned to integrate scheduling. An action must include a chunk index and a quality level.
- The RL algorithms need a simulation environment to train the model. In the environment for single-source streaming, when the agent takes action, *i.e.*, downloads a new chunk, the environment immediately returns a reward value associated with that chunk, which is calculated from the utility of the chunk's quality, the quality switch penalty between two consecutive chunks and the rebuffering penalty. The rebuffering penalty is calculated when action is taken. However, in multi-source streaming, where the chunks may arrive to the client out of order, the simulation environment cannot estimate the rebuffering time right when an action is taken.
- The simulation environment for the single-source streaming is open [15]. However, the simulation environment for the multi-source streaming is not available in the literature, as far as we know.

Multipath transmission control protocol (MPTCP) [17–19] also utilizes multiple paths to transmit data from a source to a destination. It is shown that MPTCP achieves high throughput, provides a smooth hand-off, and improves the high availability of TCP connection. However, the MPTCP adoption has been prolonged because of the middlebox problem. Moreover, it requires modifying the kernels of both client and server. Our proposed framework can be applied to stream a video from a single source to a video player over multiple paths as MPTCP does. However, the MPTCP is a source-control protocol at the transport layer, whereas our proposed multi-source streaming is a client-based control protocol at the application layer. Therefore, the proposed multi-source streaming does not need to modify the kernel as well as overcomes the middlebox problem.

The contributions of our work include:

1. We propose two RL-based frameworks for rate adaptation and chunk scheduling in multi-source streaming called RL-based adaptation with greedy scheduling (RLAGS) and RL-based adaptation and scheduling (RLAS).
2. We build an environment, which is an event-driven simulation that simulates a client downloading chunks from multiple sources and playing the chunks for training and testing.
3. We conduct extensive simulations with real-trace bandwidth to evaluate the performance of the proposed methods. Both RLAGS and RLAS outperform the other baseline methods used with greedy scheduling for multi-source streaming, *i.e.*, throughput-based and BOLA. The source code is available at `https://github.com/ntnghia1908/Master_Thesis`.

This is the extended work of our conference paper [20]. In this paper, RLAS improves the proposed algorithm in [20] by using invalid action masking to avoid duplicate downloads. In addition, we propose RLAGS algorithm with greedy scheduling. We also add more evaluations in various network scenarios. The outline of the paper is as follows. Section I has presented the motivation and related works. Section II describes the RL model applied in rate adaptation and chunk scheduling for video streaming from multiple sources. The simulation environment and results are presented in Section III, and Section IV concludes the work.

2. Reinforcement learning frameworks for DASH

This section describes the RL framework, including reward function, action space, and state space. Two chunk scheduling policies are considered, *i.e.*, greedy and RL-based scheduling, which leads to two proposed algorithms, RLAGS and RLAS, respectively.

2.1. Reward

We apply a similar reward function used in [15,16], which captures utility, switch penalty, and rebuffering penalty. Assume that a time step begins when the client requests a video chunk. The episode ends when the client finishes playing the video.

Reward for single-source streaming: Assume that step t starts when the client requests for chunk t, the reward at step t in single-source adaptive streaming is given by [15, 16] (see Table 1 for the notation descriptions):

$$r_t = q_t - \beta \mid q_t - q_{t-1} \mid -\gamma\phi_t - \delta[\max(0, B^{min} - B_t)]^2, \quad t = 2, \ldots, N, \qquad (1)$$

where

Table 1. Main notations

Notations	**Descriptions**
$B^{\max}$	maximum buffer size (in seconds)
N	number of video chunks
L	number of quality levels in action space
W	number of chunks in action space
$\mathcal{A}$	action space
r_t	reward estimated at step t
s_t	environment state at step t
q_i	utility of quality level i
β	quality-switch coefficient
γ	rebuffering coefficient

- q_t is the utility corresponding to the quality level of chunk t;
- $\mid q_t - q_{t-1} \mid$ penalties the difference in quality levels between two consecutive chunks;
- ϕ_t is rebuffering time is seconds;
- $[\max(0, B^{min} - B_t)]^2$ is an optional penalty that is applied whenever the buffer level is below a threshold B^{min}. This term helps to reduce the risk of rebuffering.

If d_t, *i.e.*, the download time of chunk t, is greater than remaining time in buffer, which is B_t, then rebuffering time ϕ_t is $d_t - B_t$, otherwise, there is no rebuffering. Hence, the rebuffering time associated with chunk t in single-source streaming is given by the following formula

$$\phi_t = \max(0, d_t - B_t). \tag{2}$$

Reward for multi-source streaming: Formula (2) is no longer correct in the multi-source streaming environment since the buffer at the client may not store consecutive chunks. For example, the buffer may have chunks 3, 5, 6, and 7, while chunk 4 has not fully received on the low-throughput path. Therefore, in the multi-source environment, the reward is estimated when playing chunks in a step. Let a step start when the client requests a chunk and end when the client requests a new chunk or reaches the end of the episode. The reward returned at step t in the multi-source streaming environment is given by

$$r_t = \sum_i q_i - \beta \sum_i \mid q_i - q_{i-1} \mid - \gamma \phi_t, \tag{3}$$

where i is any chunk index played, and ϕ_t is the cumulative rebuffering time in step t. The terms $\sum_i q_i$, $\beta \sum_i \mid q_i - q_{i-1} \mid$, and $\gamma \phi_t$ are called *utility*, *switching penalty*, and *rebuffering penalty*, respectively.

2.2. State space

The state s of the proposed reinforcement learning frameworks includes the following components

- vector of network throughput measurements of last 06 video chunks on each path;
- vector of chunk sizes of L quality levels of next W chunks count from playing chunk (length $W \times L$);
- the vector of quality levels of next W chunks counted from the playing chunk, if the chunks have not yet downloaded, their quality level is set to 0;
- current buffer size in seconds;
- number of remaining chunks that have not yet played;
- quality level of the playing chunk; and
- download times of last 06 video chunks on each path.

2.3. Scheduling policies and action spaces

We assume that the request for a new chunk is sent on a path right after the downloading chunk on that path is fully received if the buffer size is under $B^{\max}$. Otherwise, the client will pause sending a new request. Let's consider two scheduling policies, *i.e.*, greedy scheduling and RL-based scheduling, corresponding to two proposed methods, RLAGS and RLAS, respectively.

Greedy scheduling In greedy scheduling, the chunk is requested in order. When the client downloads a new chunk from a source, it requests the chunk index, the smallest index that has not been or is being downloaded. Therefore, RLAGS agent only decides the quality level of the chunk to request. The action space of RLAGS includes the quality levels of video chunks:

$$\mathcal{A}^{\mathrm{RLAGS}} = \{l_i | i = 1, \ldots, L\}, \tag{4}$$

where L is the number of quality levels of video.

For example, in Fig. 2 (upper figure), chunks 1-3 have been played, chunk 4 is playing, and chunk 5 has already been requested. The next request is for chunk 6, with the quality level decided by RLAGS.

RL-based scheduling RLAS method uses RL-based scheduling. When the client requests a new chunk, the RL agent decides both the index and quality level. Assuming that the maximum number of chunks that can be stored in the video buffer is W, the number of quality levels is L. Action space of RLAS method is defined as

$$\mathcal{A}^{\mathrm{RLAS}} = \{(c_i, l_j) | c_i \in [1, W], j \in [1, L]\}. \tag{5}$$

If the playing chunk is c_t and the RL agent takes action $a_t = (c_i, l_j)$ to download chunk on a path at time step t, the client will download chunk index $c_t + c_i$ at quality l_j on this path.

For example, in Fig. 2, if quality levels for each chunk are *low*, *medium*, and *high* ($L = 3$), $W = 4$. Assume that at current time t, the playing chunk is $c_t = 4$ and the RL agent takes action $a_t = (3, 2)$. It means that the agent will download chunk index $4 + 3 = 7$ with quality level 2, which is *medium* quality (see Fig. 2).

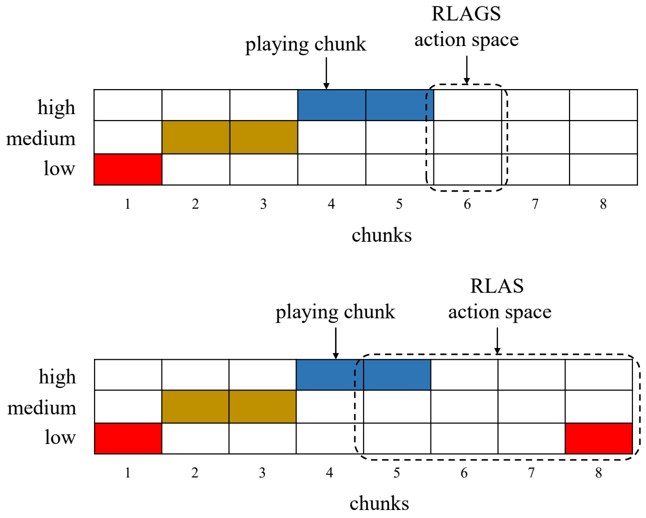

Fig. 2. The action spaces of RLAGS and RLAS. (The shade regions represent the chunks that have been requested.)

Invalid action masking With RLAS, there are invalid actions in some steps. Firstly, the two-dimension action space of RLAS allows the possibility of re-download the same chunk index again if that chunk has not been played. The chunk index already downloaded is considered an invalid action to avoid duplicate downloads. Secondly, since RLAS's action space is a sliding window that shifts forward by one chunk when a new chunk is played, some actions are *invalid* when the number of remaining chunks is less than the window side W. Hence, the valid actions of RLAS are given by

$$\{(c_i, l_j) \mid c_i \in [1, \min{[W, N - c_t]}, c_i \text{ has not been requested}, j \in [1, L]\}, \qquad (6)$$

where N is the number of chunks of the video, and chunk c_t is the chunk being played.

There are several approaches to dealing with invalid actions. Two common ones are *invalid action penalty* and *invalid action masking*. With the invalid action penalty approach, the rewards resulting from the invalid actions are set to negative values. With invalid action masking, the action is sampled among the valid actions in each step. These approaches are well investigated and implemented in the work [21]. With policy gradient algorithms, invalid action masking is shown theoretically and empirically that it outperforms the other approaches, particularly with the state-of-the-art proximal policy optimization (PPO) algorithm in the experiments [21]. Therefore we also apply PPO in our evaluations.

2.4. PPO for multisource DASH

PPO is a policy gradient algorithm that uses two networks, actor and critic, like A2C or A3C. The actor network estimates the policy directly from the state. A baseline is subtracted from the return to reduce the variance of a policy gradient algorithm. A common-used baseline is the value function, which is estimated by a critic network. To accelerate the training, PPO algorithm could also use multiple copied environments in parallel, similar to A2C and A3C.

However, PPO is an improvement from A2C/A3C. To prevent the catastrophic drop in the performance of the traditional actor-critic algorithms, PPO constraints the change in policy between two consecutive training steps by introducing a new clipped surrogate objective. PPO has shown a reliable performance and is used in many RL applications. Please see the detail of PPO algorithm in [25].

We utilize Stable Baseline3 [26] library to implement PPO in training and evaluation. Stable Baseline3 includes a set of reliable implementations of deep RL algorithms and is used in many applications. The invalid action masking function is also provided with PPO in Stable Baseline3.

3. Evaluations

3.1. Event-driven environment

We build an environment that simulates the streaming from two sources, emulating the practical scenarios, *e.g.*, a cell phone uses Wi-Fi and 4G to connect to video servers, or a laptop connects via Ethernet and Wi-Fi simultaneously. Scenarios with more than two sources can be easily extended by modifying *reset* function. The simulation environment follows Gym interface to be able to use Stable Baseline3 [26].[5]

The environment emulates a client downloading chunks on two paths parallelly and playing the received chunks. An array-type buffer, which stores downloaded chunk indices, is maintained during an episode. When the client fully receives a chunk, the chunk index is appended to the buffer, and the buffer size increases by a chunk length. The client plays the chunks stored in the buffer sequentially. If a chunk is played, that chunk index is removed from the buffer, and the buffer size increases by a chunk length.

There are four main events, *i.e.*, DOWN, PAUSE, PLAY, REBUFFER. Every event has a timestamp, and the program runs through the events iteratively in time order until the end of the episode. DOWN and PAUSE events are associated with a path index, whereas PLAY and REBUFFER events are not.

- A DOWN event simulates sending a request for a chunk, say c_t, on a path. When the program encounters a DOWN event at time t, at timestamp *t + downtime*, where downtime is the time from sending the request for c_t to fully receiving the chunk, a new DOWN event associated with a new chunk is generated if the buffer size is less than $B^{\max}$. The index and the quality level of the new chunk are decided by RLAGS or RLAS methods. Otherwise, a PAUSE event is generated if the buffer size exceeds $B^{\max}$.
- A PAUSE event simulates pausing the download on a path due to the buffer size exceeding $B^{\max}$. If a PAUSE event is encountered at time t, with timestamp $t + sample$, where sample is a short period (0.05 second in our program), a new PAUSE

[5] The source code of the environment is available at `https://github.com/ntnghia1908/Master_Thesis/blob/main/RLAS/menv_baseline.py`.

event is generated if the buffer size exceeds $B^{\max}$; otherwise, a new DOWN event associated with a new chunk is generated.

- A PLAY event occurs when the client starts playing a chunk, say chunk c_t. After a PLAY event, at time $t + chunk\_length$, a new PLAY event associated with chunk $c_t + 1$ is generated if this chunk is available in the buffer; otherwise, a REBUFFER event is generated.
- A REBUFFER event occurs when the chunk going to be played is not in the buffer. After a REBUFFER event, at time $t + sample$, a new PLAY event associated with chunk $c_t + 1$ is generated if this chunk is fully received; otherwise, a REBUFFER event is generated.

3.2. Simulation settings

We evaluate RLAGS and RLAS with Big Bug Bunny video [4]. There are seven quality levels $300, 700, 1200, 1500, 3000, 6000, 8000$ Kbps ($L = 7$). Assume that the maximum buffer size of the client is $B^{\max} = 30$ seconds and the video chunk length is 4 seconds. The number of chunks in the action space is $W = \lfloor 30/04 \rfloor = 7$, which means that if the agent is playing chunk i, the maximum chunk index stored in the buffer is $i + 7$. We train with the first 60 chunks, which results in 240 seconds per episode. Table 2 shows the parameters of the simulation environment.

Two real-trace datasets are used: a broadband dataset provided by US Federal Communications Commission (FCC) [22] and a 4G LTE Dataset collected from two major Irish mobile operators [24].

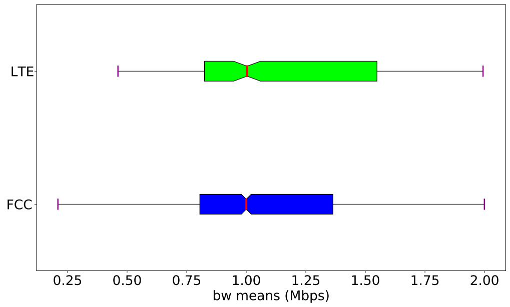

Fig. 3. The distributions of the means of all traces of the datasets.

The **FCC dataset** contains over one million throughput traces in the “download speed” category with a granularity of 10 seconds per sample [23]. (It is 5 seconds before 2016.) The **4G dataset** has 135 traces, with around 15 minutes per trace, at 1-second granularity. The traces are collected from Irish mobile operators with five mobility patterns: static, pedestrian, car, bus, and train [24].

Since the real bitrates of 8,000 Kbps quality level of the almost chunks are less than 4000 Kbps [4], we choose the traces with the average throughputs in $[0.1, 2.0]$ Mbps, in which 1800 traces in the FCC dataset and 400 traces in the LTE one. Fig. 3 shows the distributions of the average throughputs of the traces from two datasets. We randomly select 80% of traces in each dataset for training, the remaining ones are for testing.

Table 2. Simulation parameters

Environment parameter	**Notation**	**Value**
maximum buffer size	$B^{\max}$	30 seconds
number of video chunks	N	60 chunks
number of quality levels in action space	L	7
number of chunks in action space	W	7
quality levels	l_i	$[300, 700, 1200, 1500, 3000, 6000, 8000]$ Kbps
utility	q_i	$\ln(\frac{l_i}{l_1})$
quality-switch coefficient	β	1
rebuffering coefficient	γ	3.3

We compare RLAGS and RLAS methods with two well-known adaptation methods, *i.e.*, throughput-based [2] and BOLA [8] (a buffer-based) methods. These adaptations are originally designed for single-source video streaming. We apply greedy scheduling to extend them to multi-source streaming. In the throughput-based method, the quality of the next download chunk on one path is the highest quality level which is smaller than the harmonic mean of the last six chunks downloaded on that path.

Table 3 lists some tuned hyper-parameters for RLAGS and RLAS. The not-listed hyperparameters are used with the default values provided by Stable Baseline3. We use fully connected neural networks with 64 nodes for each hidden layer. We tuned the number of hidden layers for the algorithms. Round-trip-times of the network connections are uniformly random in $[50, 100]$ ms.

Each proposed algorithm is trained in five runs, $30,000$ episodes each run. Each episode chooses a random trace in the training set and starts at a random point. The throughput trace is circulated if the time from starting point of an episode to the end of the throughput trace is not enough for the time playing the episode. The results are the average values in five runs.

3.3. Results

Fig. 4 shows the convergence of both RLAGS and RLAS algorithms in training with turned parameters given in Table 3. We can see that RLAGS converges faster than RLAS since RLAGS has fewer actions than RLAS, which are only quality levels. However, RLAS yields a higher average reward than RLAGS.

We test the case when one path is a broadband connection, and the other path is an LTE connection. The reward, utility, switch penalty, and rebuffering penalty are given

Table 3. Tuned hyperparameters used in RLAGS and RLAS.

Hyperparameters	Descriptions	Tuning ranges	RLAGS	RLAS
learning rate	learning rate	uniform: [0.0001, 0.001]	0.000125	7.61e-05
batch size	minibatch size	uniform: [59, 590]	411	530
n epochs	number of epoch when optimizing the surrogate loss	values: [10, 20, 30]	10	10
gamma	reward discount factor	values: [0.99, 1.0]	0.99	1
gae lambda	factor for trade-off of bias vs. variance for generalized advantage estimator	values: [0.9, 0.95]	0.9	0.95
clip range	clipping parameter	values: [0.2, 0.3]	0.3	0.2
vf coef	value function coefficient for the loss calculation	uniform: [0.2, 0.5]	0.317708	0.286954
ent coef	entropy coefficient for the loss calculation	values: [0.0, 0.00001, 0.00000001]	0	0
act func	value function coefficient for the loss calculation	values: [128, 256, 512]	256	512
features dim	value function coefficient for the loss calculation	values: [tanh, relu]	relu	tanh
policy net arch layer	number of policy network layer	values: [1, 2, 3, 4]	1	3
policy net arch units	policy network unit.	values: [64, 128, 256, 512]	512	256
value net arch layers	number of value network layer	values: [1, 2, 3, 4]	3	4
value net arch units	value network unit.	values: [64, 128, 256, 512]	512	256

in Table 4. The rewards yielded by RLAGS and RLAS are higher than the reward by throughput-based and BOLA methods. RLAS achieves the highest reward, and RLAGS results in a smaller rebuffering penalty.

We consider the performance of multisource streaming in the case when the difference between two paths increases gradually. Particularly, the mean bandwidth of the first path is from 1.5 Mbps to 2 Mbps and the mean bandwidth of the second path decreases gradually, in $[1.5, 2.0]$ Mbps, in $[1.0, 1.5]$ Mbps, in $0.5, 1.0$ Mbps, and less than 0.5 Mbps.

We can see from Fig. 5 that the rewards of multisource streaming of all the methods decrease gradually. The rewards of RLAS are the highest in most of the cases, which shows the efficiency of the RL-based chunk scheduling. BOLA yields a higher reward

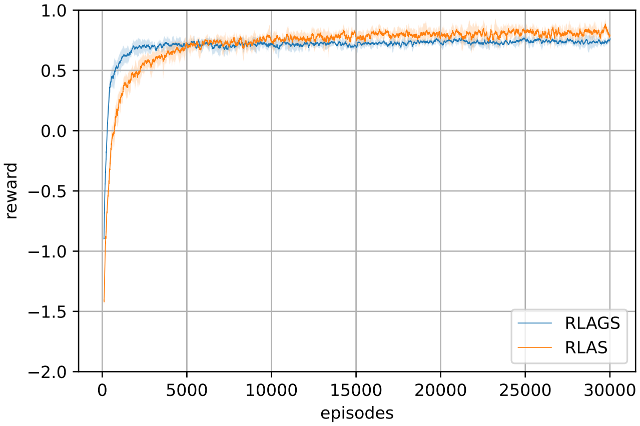

Fig. 4. Convergence of training phases of RLAGS and RLAS methods. The lines and shadows are the means and the standard deviations of the running average rewards of five runs, respectively.

Table 4. Rewards of ABR methods when one path is broadband and another path is LTE connection.

Methods	**reward**	**utility**	**switch penalty**	**rebuffering penalty**
THGHPUT	42.10	68.56	21.40	5.06
BOLA	77.80	129.75	27.26	24.70
RLAGS	88.35±2.04	97.86±2.29	8.53±1.01	0.98±0.52
RLAS	**107.75±1.91**	130.68±5.87	17.51±4.88	5.42±2.62

than RLAGS. However, in the extreme case when the mean bandwidth of two paths is very different, RLAGS and RLAS outperform the traditional methods.

Table 5. Rewards of ABR methods with the mean bandwidths of the first path is in $[1.5, 2.0]$ Mbps and of the second path is less than 0.5 Mbps.

Methods	reward	utility	switch penalty	rebuffering penalty
THGHPUT	17.34	68.20	32.03	18.83
BOLA	-35.78	120.11	19.10	136.78
RLAGS	**66.61±4.94**	92.68±2.66	9.74±1.92	16.33±7.69
RLAS	57.55±3.99	105.69±1.77	17.54±1.49	30.59±3.76

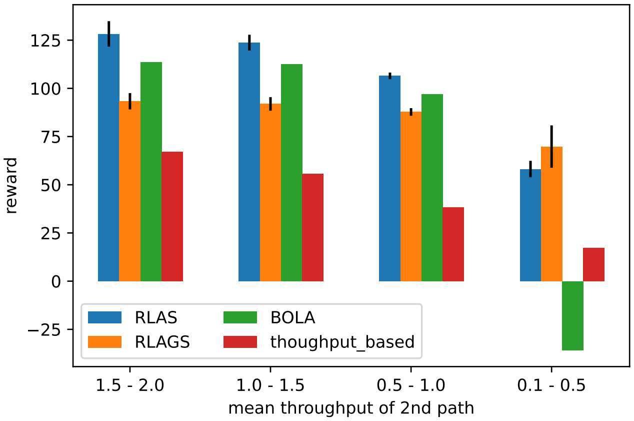

Fig. 5. The test rewards of different adaptation methods when the mean bandwidth of the first path is in $[1.5, 2]$ Mbps and the mean bandwidth of the second path decreases gradually.

Table. 5 shows the performance of the adaptation methods in the extreme case: the average throughput of the first path is in $[1.5, 2.0]$ Mbps, and of the second path is less than 0.5 Mbps. Overall, RLAGS and RLAS outperform BOLA method, and their rewards are much higher than those of the throughput-based method. The reward of RLAGS is a bit higher than RLAS because it has fewer actions in the action space. Hence, the agent may be easier to learn the optimum. The throughput-based method has the least rebuffering; however, it also has the lowest utility. BOLA has the highest utility but also the highest rebuffering penalty. RLAGS balances the objectives: high utility, low number of switches, and small rebuffering penalties.

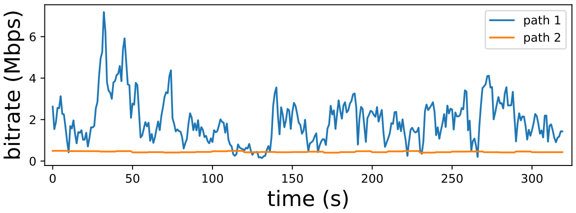

Fig. 6. A sample of throughput traces in the extreme case: the mean bandwidths of the first path is in $[1.5, 2.0]$ Mbps and of the second path is less than 0.5 Mbps.

Fig. 7 shows video quality levels selection and buffer occupancy when the client experiences a pair of throughput traces shown in Fig. 6 with different methods. The video

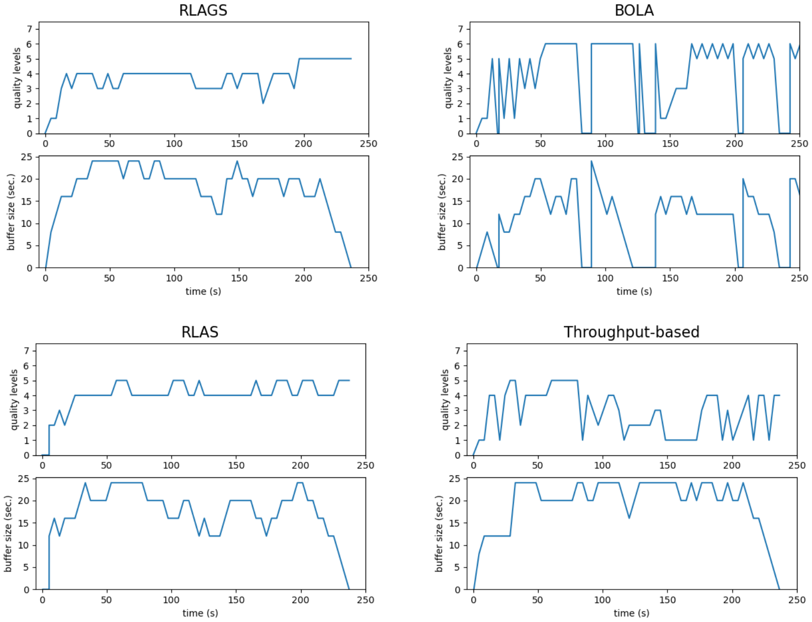

Fig. 7. The quality levels and the buffer occupancy with the sample throughput traces in Fig. 6.

played by the RL-based methods is more stable than by the other methods. We see that the proposed methods have a smarter buffer occupancy so that they can download higher quality levels with fewer switches than other methods.

4. Conclusions

We have proposed two novel adaptation and scheduling methods for video streaming from multiple sources, *i.e.*, RL-based adaptation and greedy scheduling (RLAGS) and RL-based adaptation and scheduling (RLAS). The state space, action space, and reward are defined for the methods. We have also built a GymAI-compatible environment for training and evaluation. Extensive simulations have shown that the proposed methods outperform the baseline methods in terms of the user's QoE. Model-free reinforcement learning algorithms could not work well in transfer learning [27]. If running the model in an untrained environment, the model could yield a low reward. In the future, we will apply model-based algorithms to bitrate adaptation.

References

1. Cisco: Cisco Visual Networking Index: Forecast and Methodology, 2016-2021.
2. T. Stockhammer: Dynamic adaptive streaming over HTTP: standards and design principles. In Proceedings of the second annual ACM conference on Multimedia systems, 133-144. (2011)

3. I. Sodagar: The MPEG-DASH Standard for Multimedia Streaming Over the Internet. IEEE MultiMedia, Vol. 18, Issue 4, 62-67. (2011)
4. S. Lederer, C. Müller and C. Timmerer: Dynamic Adaptive Streaming over HTTP Dataset. In Proceedings of the ACM Multimedia Systems Conference, 22-24. (2012) Online: `https://dash.itec.aau.at/dash-dataset/`.
5. ISO/IEC 23009-1:2014: Dynamic Adaptive Streaming over HTTP (DASH)– part 1: Media Description and Segments format.
6. DASH Reference Client. Accessed: Jun. 28, 2019. [Online]. Available: https://reference.dashif.org/dash.js/
7. J. Jiang, V. Sekar, and H. Zhang: Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In Proceedings of CoNEXT. (2012)
8. K. Spiteri, R. Urgaonkar, and R. K. Sitaraman: BOLA: Near-optimal bitrate adaptation for online videos. In Proceedings of 35th Annual IEEE International Conference on Computer Communications (INFOCOM). (2016)
9. T. Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson: A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In Proceedings of the 2014 ACM conference on SIGCOMM, 187-198. (2014)
10. Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran: Probe and adapt: Rate adaptation for HTTP video streaming at scale. IEEE Journal on Selected Areas in Communications, Vol. 32, No. 4, 719-733. (2014)
11. Y.C. Chen, D. Towsley, and R. Khalili: MSPlayer: Multisource and multi-path video streaming. IEEE Journal on Selected Areas in Communications, Vol.34, Issue 8, 2198-2206. (2016)
12. A. Nikravesh, Y. Guo, X. Zhu, F. Qian, and Z. M. Mao: MP-H2: a Client-only Multipath Solution for HTTP/2. In Proceedings of The 25th Annual International Conference on Mobile Computing and Networking, 1-16. (2019)
13. A. Bentaleb, P.K. Yadav, W.T. Ooi, and R. Zimmermann: DQ-DASH: A Queuing Theory Approach to Distributed Adaptive Video Streaming. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), Vol. 16, No. 1, 1-24. (2020)
14. M. Claeys, S. Latre, J. Famaey, and F. De Turck, "Design and evaluation of a self-learning HTTP adaptive video streaming client," *IEEE communications letters*, vol. 18, issue 4, pp. 716–719, 2014.
15. H. Mao, R. Netravali, and M. Alizadeh: Neural adaptive video streaming with pensieve. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 197-210. (2017)
16. M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella: D-DASH: A deep Q-learning framework for DASH video streaming. IEEE Transactions on Cognitive Communications and Networking, Vol. 3, Issue 4, 703-718. (2017)
17. D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley: Design, implementation and evaluation of congestion control for multipath TCP. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, Vol. 11, 8-8. (2011)
18. C. Raiciu, M. Handley, and D. Wischik: Coupled congestion control for multipath transport protocols, RFC6356. (2011)
19. Phuong Luu Vo, Tuan Anh Le, S. Lee, C. S. Hong, B. Kim, H. Song: mReno: a practical multipath congestion control for communication networks. Computing, Vol. 96, No. 3, 189-205. (2014)
20. Nghia T. Nguyen, Phuong L. Vo, Thi Thanh Sang Nguyen, Quan M. Le, Cuong T. Do, and Ngoc-Thanh Nguyen: A Reinforcement Learning Framework for Multi-source Adaptive Streaming. In Proceedings of International Conference on Computational Collective Intelligence, 416-426. (2021)

21. S. Huang and S. Ontanon: A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. In Proceedings of the Thirty-Fifth International Florida Artificial Intelligence Research Society Conference, (FLAIRS 2022), Florida, USA, May 15-18. (2022)
22. US Federal Communications Commission (FCC). [Online]. Available: https://data.fcc.gov/download/measuring-broadband-america/2019/data-raw-2019-sept.tar.gz
23. Tenth Measuring Broadband America Fixed Broadband Report [Online]. Available: Measuring Fixed Broadband - Tenth Report — Federal Communications Commission (fcc.gov)
24. D. Raca, J.J. Quinlan, A.H. Zahran, C.J. Sreenan: Beyond Throughput: a 4G LTE Dataset with Channel and Context Metrics. In Proceedings of ACM Multimedia Systems Conference (MMSys 2018), Amsterdam, The Netherlands. (2018)
25. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov: Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347. (2017)
26. A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus and N. Dormann: Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research, Vol. 22, No. 268, 1-8. (2021)
27. T. M. Moerland, J. Broekens, and C. M. Jonker: Model-based reinforcement learning: A survey. arXiv preprint arXiv:2006.16712. (2020)

Nghia Trung Nguyen received his bachelor's and master's degrees from Computer Science from International University - Vietnam National University Ho Chi Minh City in 2019 and 2022, respectively. His main research interest is to apply Reinforcement Learning to enhance the efficiency of various applications in computing.

Minh-Long Luu holds a bachelor's degree of Computer Science from International University - Vietnam National University Ho Chi Minh City. His main research areas are Reinforcement Learning and Computer Vision for bitrate adaptation, image classification, and out-of-distribution generalization.

Phuong L. Vo received her B.Eng and M.Eng degrees in electrical-electronics engineering from Ho Chi Minh City University of Technology, Vietnam in 1998, 2002, respectively, and Ph.D. degree at Kyung Hee University, Korea in 2014. Currently, she is an Associate Professor at School of Computer Science and Engineering at International University – VNUHCM. Her research interest is to apply machine learning, optimization, and game theory to contemporary networks.

Thi Thanh Sang Nguyen is a lecturer at the School of Computer Science and Engineering, International University, Vietnam National University, Hochiminh City, Vietnam. She has received her PhD degree in Software Engineering from the University of Technology, Sydney (UTS) in 2013. Her Ph.D. thesis is about Semantic-enhanced Web-page Recommender Systems. She was supervised by A.Prof. Haiyan Lu and Prof. Jie Lu. She received her master's degree in computer engineering from the University of Technology (VNU-HCMC) in 2006. She has more than 20 published research papers in the field of Web mining. Her research interests include Web mining, Semantic Web, knowledge discovery and business intelligence. Her profile on ResearchGate is https://www.researchgate.net/profile/Sang-Nguyen-7, and her publications are on https://dblp.org/pid/55/8981.html.

Cuong T. Do received his BS degree from Hanoi University of Science and Technology and Ph.D degree from Kyung Hee University, in electrical and computer engineering, in 2008 and 2014, respectively. His research interests include Queueing Theory, Game Theory, Machine Learning and their applications in Communication Networks.

Ngoc Thanh Nguyen (Senior Member, IEEE) is currently a Full Professor with the Wroclaw University of Science and Technology, and the Head of Information Systems Department, Faculty of Computer Science and Management. He is the author or coauthor of five monographs and more than 350 journal and conference papers. He has given 22 plenary and keynote speeches for international conferences, and more than 40 invited lectures in many countries. His research interests include collective intelligence, knowledge integration methods, inconsistent knowledge processing, and multi-agent systems.

Received: September 27, 2022; Accepted: November 20, 2022.