EasyChair Preprint

№ 2149

Deeplive: QoE Optimization for Live Video Streaming through Deep Reinforcement Learning

Zhao Tian, Laiping Zhao, Lihai Nie, Peiqi Chen and Shuyu Chen

December 12, 2019

Deeplive: QoE Optimization for Live Video Streaming through Deep Reinforcement Learning

Zhao Tian, Laiping Zhao*, Lihai Nie, Peiqi Chen, Shuyu Chen
Tianjin Key Laboratory of Advanced Networking, College of Intelligence and Computing, Tianjin University, China
{tianzhao,laiping,nlh3392,chenpeiqi,3017218119}@tju.edu.cn

***Abstract*—A new broad of video services that support live streaming has become tremendously popular in recent years. Compared with traditional video-on-demand (VOD) services, live video streaming has much higher requirements on Quality-of-Experience (QoE), including low rebuffering, high definition, low latency and low bitrate oscillations. While previous adaptive bitrate algorithms (ABR) solely optimize bitrate for ensuring QoE of VOD, live video streaming has a larger decision space, making the optimization problem more difficult to solve. We propose *Deeplive*, which maximizes QoE through deep reinforcement learning (DRL), so it does not rely on fixed rules. To accelerate the training process of *Deeplive*, we further propose optimization including *window completion with historical data* and *quick-start with rate-based algorithm*. We compare *Deeplive* with other advanced ABR algorithms in a frame-level dynamic adaptive video streaming simulator using different network traces, QoE definitions, and video categories. In all experiments, we find that *Deeplive* not only has significant improvement in training time, but also shows an average of 15-55% improvement on QoE than the state-of-the-art ABR algorithms.**

***Index Terms*—live video streaming, QoE, deep reinforcement learning.**

I. Introduction

Internet traffic statistics show that video traffic demand has reached 76% in 2016, and is expected to increase to 82% by 2021 [1]. Streaming media transmission has become the top priority in today's network transmission. In recent years, with the rise of live streaming services, the number of people watching live video streaming has increased dramatically. It is particularly important to ensure the Quality-of-Experience (QoE) of live video streaming transmission. Compared with video-on-demand (VOD), live streaming QoE has much higher requirements on high bitrate and less rebuffering, where bitrate refers to the definition of the video and the higher bitrate means that video can be played in higher definition; rebuffering implies the time video stops for buffering and less rebuffering means users can watch the video smoothly. Moreover, minimizing the latency between when a video frame is generated and when it is received by the end user is also extremely important for ensuring the real-time interaction.

They mainly rely on the prediction on throughput (e.g., *FESTIVE* [2] and *Rate-based algorithm* [3]) or buffer size (e.g., *Buffer-based algorithm* [4] and *BOLA* [5]), or a combination of them (e.g., *DYNAMIC* [6] and *MPC* [7]). However, these algorithms cannot be applied into a complex network with oscillations, and cannot meet the various QoE demand for live video streaming users. The state-of-the-art *Pensieve* [8] which uses deep reinforcement learning (DRL) has been proven to have an average QoE improvement of 12-25% over other traditional algorithms and can adapt to different network conditions. However, these algorithms do not take latency into consideration and have a poor performance in live video streaming scenarios.

In live video streaming, we can only get a few seconds of video ahead at every moment. It means that there is much less information that can be utilized to make optimal live video streaming decisions. In addition, due to the limitation of network conditions, choosing a high bitrate means that more transmission time is needed, which inevitably brings latency or even rebuffering. It is also particularly important to ensure the stability of transmission during the live video streaming.

In this paper, we present *Deeplive*, a QoE optimization algorithm for live video streaming using DRL, which can deal well with complex scenarios with perplexing network conditions and various video content, and can choose appropriate actions from the decision space for live video streaming according to the state. Unlike previous optimization goals, we improved video quality from multiple metrics rather than just bitrate. We establish a DRL model to select bitrate,target buffer and latency limit for future video chunks based on states observed in the live video streaming. By using the past status observed from the client and instead of the raw video pictures, the state space of the *Deeplive* can be reduced efficiently. We also implement optimizations based on *Deeplive*, making it able to converge in a short time in the training phase. In general, *Deeplive* can optimize the bitrate, rebuffering and latency in the live video streaming an average 15-55% improvement over the performance of other algorithms.

II. System Model

In this section, we describe the model of the Dynamic Adaptive Streaming over HTTP (DASH) system, including the overview of live video streaming process and the corresponding QoE definition.

A. Overview

As shown in the Fig. 1, DASH system adopts a chunked transfer encoding mechanism for splitting video data into multiple chunks. Video chunks are encoded into multiple video

*Corresponding author : laiping@tju.edu.cn

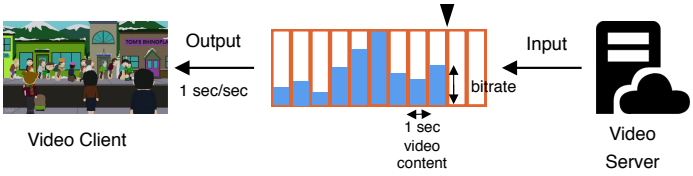

Fig. 1. Overall processing of DASH system.

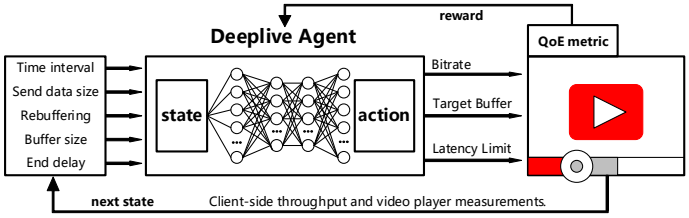

Fig. 2. The overview of *Deeplive*.

TABLE I
PARAMETERS FOR DEFINING QOE.

Params	Notations	Params description
FrameTimeLen	λ_f	Length of each frame
Bitrate	λ_b	Bitrate of video
Rebuffering	λ_r	Freezing time
EndDelay	λ_{ed}	End-to-end delay
SkipFrameTimeLen	λ_{sf}	Length of skip frame
Smooth	λ_s	Frequency of bitrate switching

frames. Our algorithm can work at the frame level DASH system. Rectangular vertical bars with equal spacing represent video over a while. The height of the video bar represents the bitrate of this chunk. Higher bar implies higher definition of video and more data transmission. The total length of all buffered chunks is the size of the buffer, which freezes when the buffer is empty.

A pull request towards the next chunk is issued at the end of last chunk. Adaptive bitrate algorithms (ABR) needs to derive bitrate, target buffer and latency limit for maximizing user QoE. In particular, bitrate decides the video definition, target buffer means that if video freezes at this moment, it needs to buffer the content of target buffer capacity before it can be played. And latency limit is the threshold for the frame skipping, i.e., when the latency reaches this threshold, in order to reduce the latency, the frames are skipped and the newer ones are downloaded. Then, ABR algorithm is to find the best decision so that live video playback can get a maximum QoE.

B. QoE Definition

We focus on the five aspects that affect the QoE in live video streaming (parameters are listed in the Tab. I):

- Frame video quality: QoE of the k^{th} frame is denoted by $Q(k) = \lambda_f \cdot \lambda_b$. The higher the bitrate, the larger the corresponding value.
- Rebuffering: The first the k^{th} frame video rebuffering time of QoE notes for $R(k) = \lambda_r$. The longer the rebuffering time, the greater the penalty.
- Latency: The delay time QoE of video in the k^{th} frame is marked as $L(k) = \lambda_{ed}$. The higher the delay, the greater the penalty.
- Frame skipping: Frame skipping QoE of the k^{th} frame video is recorded as $F(k) = \lambda_{sf}$. If frame skipping occurs, there is a penalty for skipping frames.
- Bitrate switch: The QoE of the i^{th} chunk switching frequency is denoted as $S(i) = \lambda_s$. If frequent switching occurs, a penalty will result. Switching across the bitrate (from super definition to low definition) can increase penalties.

The choice of bitrate is related to $Q(k)$, $R(k)$ and $S(k)$. Target buffer and latency limit are used to control the $L(k)$. There is a close relationship between latency limit and time $F(k)$.

We define the QoE of a chunk of live video streaming as following.

$$QoE_{chunk}(i) = \mu_q \sum_{k=1}^{K} Q(k) + \mu_r \sum_{k=1}^{K} R(k) + \mu_l \sum_{k=1}^{K} L(k) + \mu_f \sum_{k=1}^{K} F(k) + \mu_s S(i) \tag{1}$$

Then, for the whole video, the calculation formula of QoE is as follows:

$$QoE = \sum_{i=1}^{I} QoE_{chunk}(i) \tag{2}$$

In addition, μ_q is reward coefficient for QoE while μ_r, μ_l, μ_f and μ_s are penalty coefficient. By changing the values of the parameters (μ_q, μ_r, μ_l, μ_f, μ_s), we were able to implement different QoE metrics.

III. DEEPLIVE DESIGN

In this section, we describe the design of *Deeplive* and present optimizations on accelerating the training process. *Deeplive*'s training algorithm uses Double-DQN [9]. The detailed structure of *Deeplive* is explained below.

A. Design of our algorithm

As shown in Fig.2, *Deeplive* deploys an agent at each user client. The agent utilizes a neural network to determine which action should be selected for a larger user QoE.

1) Agent: The agent consists of two neural networks with the same structure. Target network predicts the target value(Q_{target}) and evaluation network predicts the realistic value(Q_{eval}). The difference between them updates the parameter of the networks. In particular, target network is slow to update, while the other always updates its parameters during the training process. The neural network has a 1-dimentional convolution layer and multiple full-connection layers. Each node in the final output layer represents one action. The node value represents the reward of the action. So the agent chooses the action with the highest score.

Params	Notations	Params description
time_interval(s)	$\vec{t}$	Duration in this cycle
send_data_size(bit)	$\vec{s}$	The data size downloaded in this cycle
rebuffering(s)	$\vec{r}$	The rebuffring time of this cycle
buffer_size(s)	$\vec{b}$	The current buffer size
end_delay(s)	$\vec{e}$	The current end-to-end delay

TABLE II
PARAMETERS OF STATE. *Deeplive* USES THESE VARIABLES PROVIDED BY THE SIMULATOR TO MAKE DECISIONS.

2) State: As shown in Tab. II, during the live video, the algorithm obtains the information in the list from the client. It just reaches the chunk boundary at time t, and the algorithm starts to make decisions. *Deeplive* takes s_t = ($\vec{t}$, $\vec{s}$, $\vec{r}$, $\vec{b}$, $\vec{e}$) as the input of neural network, where $\vec{t}$, $\vec{s}$, $\vec{r}$, $\vec{b}$ and $\vec{e}$ stand for the vector consisting of the duration, the size of data downloaded, rebuffering time, buffer size and end-to-end delay for the past k video frames respectively. In the real training process, we find that when we normalize the values of these variables to the interval of [-1,1], the neural network tends to converge faster.

3) Action: Agent makes decisions on bitrate, target buffer and latency limit. The value range of bitrate is determined by the video trace provided, while the value range of target buffer and latency limit can be set by users. To maintain the consistency of the experiment, we used the set of values provided above throughout the experiment. The nodes of the neural network are the combination of the above three vectors. We use π for decision-making strategies. The decision made at time t can be expressed as:

$a_t = \pi\ (s_t) \rightarrow [bitrate,\ target\text{-}buffer,\ latency\text{-}limit]$

B. Policy

The pseudocode of *Deeplive* is described in Algorithm 1. First, we initialize the model, including the setting of some hyperparameters and the initialization of neural network weights. At t moment, the agent gets state s_t from the environment. At this point, the agent has a probability of ϵ to randomly select an action a_t, otherwise the agent will choose the a_t with the maximum reward according to the neural network $a_t = \underset{a}{argmax}\, Q_{eval}(s_t)$. After the agent selects a_t, video continues to play, and the agent will get the next state s_{t+1} and the real reward r_t, and the agent will save the quadruple $[s_t, a_t, r_t, s_{t+1}]$ in replay experience pool REP. In the process of training, agents will randomly select some tuples from REP for playback of training at intervals. Loss function is Huber Loss, denoted by $L(y_t, y_p)$.

$$L(y_t, y_p) = \begin{cases} 0.5|y_t - y_p|^2, & |y_t - y_p| < \delta \\ 0.5\delta^2 + \delta \cdot (|y_t - y_p| - \delta), & otherwise \end{cases} \quad (3)$$

In the formula above, y_p is the real reward of playing on video,

$$y_t = r_t + \gamma \cdot \max_a Q_{target}(s_{t+1}, \underset{a}{argmax}\, Q_{eval}(s_{t+1})) \quad (4)$$

Algorithm 1: The pseudocode of *Deeplive*

Input : Multiple video samples, hyperparameters;
Output: Neural network parameter;

[Parameters]:
video m; {*c*hoose a video file from input.}
chunk t; {*d*ownload t^{th} chunk from the video.}
frame k; {*d*ownload k^{th} frame from the chunk.}

Initialize replay experience pool REP;
Initialize Q_{eval} Network and Q_{target} Network with random weights;
for *video m = 1, 2, ..., M* **do**
  Observe initial state s_t;
  for *chunk t = 1, 2, ..., T* **do**
    With probability ε randomly select action a_t;
    Otherwise action $a_t = \underset{a}{argmax}\ Q_{eval}(s_t)$;
    for *frame* $k = 1, 2, ..., K$ **do**
      Download k^{th} video frame;
      Observe reward r_k;
      if *it's time to train* **then**
        Randomly sample tuples from REP;
        if *the video ends* **then**
          $y_t = r_t$;
        else
          Compute for y_t using Formula (4);
          Train the network using Huber Loss as loss function;
      if *it's time to update target network* **then**
        Set the weights of Q_{target} to be the same as that of Q_{eval};
    Set reward $r_t = \sum r_k$ and next state s_{t+1};
    Store tuple (s_t, a_t, r_t, s_{t+1}) into REP.

[9] and δ = 1.0 by default. Every once in a while, eval-network assigns its weight parameters to target-network.

C. Optimizations

1) Window completion with historical data: In the actual situation, the frame rates of the two bitrates are inconsistent, which creates different chunk video with a different number of frames after switching the bitrate. However, the input to our neural network is fixed. For the input which has frames less than inout size, our solution is window completion with historical data(WCHD) which fills in the frame information before this chunk.

2) Quick start with Rate-based algorithm: The startup phase refers to the beginning of video or after switching video. At this time, the DRL algorithm has no frame history information to refer to. *Pensieve* uses a default action for this situation, which is obviously not rigorous enough. *Deeplive* starts the Rate-based algorithm, which only depends on the historical network condition and has a better performance

TABLE III
THE QOE METRICS WE USE IN OUR EVALUATION.

Name	μ_q	μ_r	μ_l	μ_f	μ_s
QoE_{al}	0.0001	-1.5	-0.01	-1.0	-0.01
QoE_{ar}	0.001	-3.0	0	0	0
QoE_{hd}	0.002	-0.5	0	0	0
QoE_b	0.001	-1.5	-0.005	-0.5	-0.02

than taking the default action. This provides a way for ABR algorithms of reinforcement learning to perform better in the startup phase.

IV. EVALUATION

A. Experimental setup

We compare *Deeplive* with four other state-of-the-art algorithms: *BBA* [4], *RBA* [3], *DYNAMIC* [6] and *Pensieve* [8]. And we use a DASH video streaming simulator [10] which is a frame-level simulator with a pluggable client module for bitrate control and latency control. The datasets we use include network trace and video trace [11]. The network trace contains the network throughput over time, which is used to simulate the dynamic change of network conditions. We divide the network trace into four different types: high speed network, medium speed network, low speed network and fluctuation network (i.e., mixed of the previous three network conditions). Video trace provides the time and size of each frame arriving at CDN collected on the transcode server and CDN. The video trace consists of three datasets: AsianCup_China_Uzbekistan, Fengtimo_2018_11_3 and YYF_2018_08_12 [11]. For evaluating QoE, we consider four different scenarios, and the weights are listed in Tab. III. In particular, QoE_{al} prefers low latency of the video playing; QoE_{ar} prefers to avoid video rebuffering; QoE_{hd} places emphasis on high bitrate; and QoE_b favors a balance among the low latency, less rebuffering and high bitrate.

B. QoE Evaluation

Tab. IV shows the performance of *RBA*, *BBA*, *DYNAMIC*, *Pensieve* and *Deeplive* under four different network conditions: high, medium, low and mixed, respectively. The values are normalized with regard to the maximum QoE achieved by all algorithms. These network conditions are unaware by these algorithms. In the case of low network speed, performance improvement space is limited. Thus, the improvement by *Deeplive* is not significant compared with other algorithms. When the network is in good condition, we find that *Deeplive* performs better than other algorithms in different network conditions. It improves QoE by an average of 40% than *RBA*, 53% than *BBA*, 34% than *DYNAMIC*, and 18% than *Pensieve*.

Next, we evaluate the performance of the algorithms using three different video datasets under all QoE settings. Tab. V shows that *Deeplive* increase QoE by 32% than *RBA*, 39% than *BBA*, 30% than *DYNAMIC*, and 16% than *Pensieve*.

Finally, we compare the performance of the algorithm under different QoE and network settings. As shown in Fig.3, we recorded the average QoE per chunk during video playback on QoE_b. *Deeplive* has more video chunks with higher QoE. As a result, *Deeplive* can be applied to different network condition with better performance.

TABLE IV
COMPARING *Deeplive* WITH EXISTING ABR ALGORITHM BY ANALYZING THEIR PERFORMANCE ON THE QOE METRICS LISTED IN TAB. III. RESULTS ARE EVALUATED FOR THE $AsianCup\_China\_Uzbekistan$ VIDEO AND FOUR THROUGHPUT DATASETS($high, medium, low, mixed$).(RB:RBA, BB:BBA, DY:DYNAMIC, PE:PENSIEVE, DL:DEEPLIVE.)

Throughput		high	medium	low	mixed		high	medium	low	mixed
QoE_{al}	RB	0.67	0.60	0.62	0.64	QoE_{ar}	0.82	0.77	0.79	0.80
	BB	0.50	0.61	0.97	0.64		0.58	0.68	0.92	0.70
	DY	0.60	0.65	0.88	0.67		0.73	0.79	0.99	0.81
	PE	0.77	0.80	0.93	0.82		0.87	0.88	0.96	0.90
	DL	**1.00**	**1.00**	**1.00**	**1.00**		**1.00**	**1.00**	**1.00**	**1.00**
QoE_{hd}	RB	0.75	0.72	0.71	0.73	QoE_b	0.76	0.70	0.67	0.72
	BB	0.53	0.61	0.80	0.63		0.55	0.64	0.85	0.66
	DY	0.67	0.72	0.88	0.74		0.68	0.72	0.87	0.74
	PE	0.85	0.89	0.96	0.89		0.78	0.77	0.72	0.77
	DL	**1.00**	**1.00**	**1.00**	**1.00**		**1.00**	**1.00**	**1.00**	**1.00**

TABLE V
COMPARING *Deeplive* WITH EXISTING ABR ALGORITHM BY ANALYZING THEIR PERFORMANCE ON THE QOE METRICS LISTED IN TAB. III. RESULTS ARE EVALUATED FOR THE MIXED THROUGHPUT AND THREE VIDEO DATASETS.(AS:$AsianCup\_China\_Uzbekistan$, FE:$Fengtimo\_2018\_11\_3$, YY:$YYF\_2018\_08\_12$, RB:RBA, BB:BBA, DY:DYNAMIC, PE:PENSIEVE, DL:DEEPLIVE.).

Metric	*QoEal*			*QoEar*			*QoEhd*			*QoEb*		
Video	AS	FE	YY	AS	FE	YY	AS	FE	YY	AS	FE	YY
RB	0.64	0.67	0.63	0.80	0.92	0.88	0.73	0.81	0.77	0.72	0.82	0.79
BB	0.64	0.69	0.67	0.70	0.87	0.84	0.63	0.73	0.70	0.66	0.79	0.76
DY	0.67	0.67	0.63	0.81	0.92	0.88	0.74	0.80	0.75	0.74	0.82	0.78
PE	0.81	0.90	0.86	0.90	0.94	0.94	0.90	0.91	0.89	0.77	0.76	0.77
DL	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

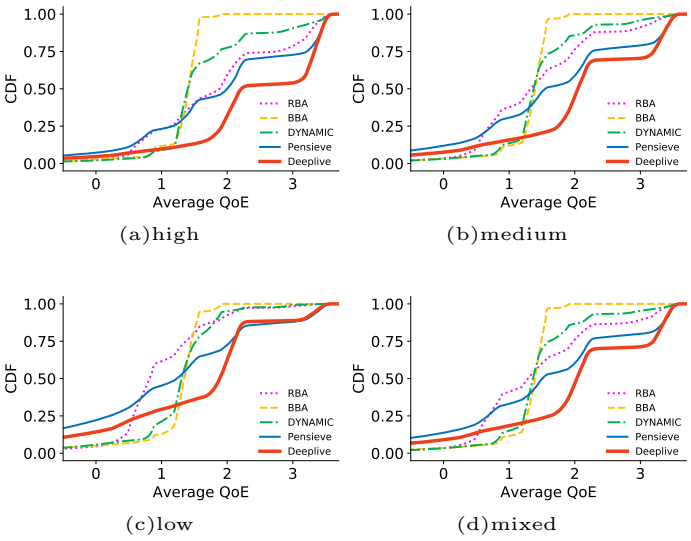

Fig. 3. Comparing *Deeplive* with existing ABR algorithm by analyzing their performance on the QoE_b definition. Results are evaluated for the $AsianCup\_China\_Uzbekistan$ video and four throughput datasets($high, medium, low, mixed$).

V. Related Work

We commit to the research of ABR algorithm. In recent years, traditional ABR algorithms are mainly divided into three categories: rate-based, buffer-based and hybrid. The traditional ABR algorithm is usually based on the fixed rules to dynamically adjust the bitrate. Among them, *FESTIVE* [2], *RBA* [3], *BBA* [4], *BOLA* [5] and *DYNAMIC* [6] all have limitations. They make decisions based solely on buffer or throughput prediction, and do not fully utilize the available information. The application of hybrid model based Model Predictive Control(*MPC*) [7] makes use of both buffer and throughput. But when the bandwidth changes dramatically, it's just hard to predict throughput. In addition, when the length of the optimization interval is large, *MPC* will spend a lot of time in solving the optimal solution. *MPC* needs to optimize the future video interval, so it needs to know the video information stored in CDN for a period of time in the future. In the case of low delay, the CDN basic cumulative frame is basically empty. They perform poorly in live scenarios with low latency.

In recent years reinforcement learning has been applied to ABR algorithms, the most famous of which is the *Pensieve* [8]. Compared with the traditional approach, the *Pensieve* achieves better results with an improvement of about 12-25%. But *Pensieve* performs poorly in a low-latency scenario. Considering the live video streaming, adding auxiliary delay control decision variables to the decision will increase the decision space, and the convergence speed of *Pensieve* will drop sharply. And *Pensieve* takes a default set of decisions in the startup phase of video which is a poor method.

Recently, an algorithm called *QARC* [12] has been developed to control the bitrate in real-time video streaming via DRL. *QARC* can be used to optimize video quality in live broadcast. But it relies on predictions of future video quality that is unconvincing for different user real experience of video quality [13]. In the process of video optimization, it may choose video with lower bitrate in order to save bandwidth, and its QoE optimization target does not consider the rebuffering situation.

To sum up, the state-of-the-art current ABR algorithms have a series of problems in live video streaming, the startup phase, training time, resource occupation and so on. We have carefully studied these existing problems and put forward corresponding solutions. After verification our algorithm has been tested to perform better than these ABR algorithms.

VI. Conclusion

We propose *Deeplive*, a DRL method that is used for optimizing QoE in live video streaming. We experimentally evaluate the performance of *Deeplive*, and find that *Deeplive* increase the QoE by an average of 15-55% than the state-of-the-art *RBA*, *BBA*, *DYNAMIC* and *Pensieve*, under different network conditions, video traces and QoE constraints. *Deeplive* also performs faster training speed and less overhead than the same DRL-based *Pensieve*. In the future, we would like to further optimize the QoE in live video streaming, and evaluate its performance in real systems.

VII. Acknowledgment

This work is supported by the National Natural Science Foundation of China under grant 61872265, U1836214; the new Generation of Artificial Intelligence Science and Technology Major Project of Tianjin under grant 18ZXZNGX00190.

References

[1] "Cisco visual networking index: Forecast and methodology 2016-2021.(2017) http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual- networking-index-vni/complete-white-paper-c11-481360.html. high efficiency video coding (hevc) algorithms and architectures https://jvet.hhi.fraunhofer.," 2017.

[2] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, (New York, NY, USA), pp. 97–108, ACM, 2012.

[3] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, (New York, NY, USA), pp. 272–285, ACM, 2016.

[4] 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*, SIGCOMM '14, (New York, NY, USA), pp. 187–198, ACM, 2014.

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "Bola: Near-optimal bitrate adaptation for online videos," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, April 2016.

[6] K. Spiteri, R. Sitaraman, and D. Sparacio, "From theory to practice: Improving bitrate adaptation in the dash reference player," in *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, (New York, NY, USA), pp. 123–137, ACM, 2018.

[7] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "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*, SIGCOMM '15, (New York, NY, USA), pp. 325–338, ACM, 2015.

[8] 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*, SIGCOMM '17, (New York, NY, USA), pp. 197–210, ACM, 2017.

[9] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100, AAAI Press, 2016.

[10] NGnetLab, "https://github.com/ngnetlab/live-video-streaming-challenge," 2019.

[11] NGnetLab, "https://github.com/ngnetlab/live-video-streaming-challenge/tree/master/dataset," 2019.

[12] T. Huang, R.-X. Zhang, C. Zhou, and L. Sun, "Qarc: 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*, MM '18, (New York, NY, USA), pp. 1208–1216, ACM, 2018.

[13] R. Rassool, "Vmaf reproducibility: Validating a perceptual practical video quality metric," in *2017 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–2, June 2017.