DRL360: 360-degree Video Streaming with Deep Reinforcement Learning

Yuanxing Zhang[†*], Pengyu Zhao[†*], Kaigui Bian[†‡], Yunxin Liu[§], Lingyang Song[†‡] and Xiaoming Li[†]
[†]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
[‡]National Engineering Laboratory for Big Data Analysis and Applications, Beijing, China
[§]Microsoft Research, Beijing, China
Email: [†]{longo, pengyuzhao, bkg, lingyang.song, lxm}@pku.edu.cn, [§]yunxin.liu@microsoft.com

***Abstract*—360-degree videos have gained more popularity in recent years, owing to the great advance of panoramic cameras and head-mounted devices. However, as 360-degree videos are usually in high resolution, transmitting the content requires extremely high bandwidth. To protect the Quality of Experience (QoE) of users, researchers have proposed tile-based 360-degree video streaming systems that allocate high/low bit rates to selected tiles of video frames for streaming over the limited bandwidth. It is challenging to determine which tiles should be allocated with a high/low rate, because (1) the video playbacks include too many features that dynamically change over time when making the rate allocation; (2) most of the state-of-the-art systems focus on a fixed set of heuristics to optimize a specific QoE objective, while users may have various QoE objectives that need to be optimized in different ways. This paper presents a Deep Reinforcement Learning (DRL) based framework for 360-degree video streaming, named *DRL360*. The DRL360 framework helps improve the system performance by jointly optimizing multiple QoE objectives across a broad set of dynamic features. The DRL-based model adaptively allocates rates for the tiles of the future video frames based on the observations collected by client video players. We compare the proposed DRL360 to the existing systems by trace-driven evaluations as well as conducting a real-world experiment over a wide variety of network conditions. Evaluation results reveal that DRL360 can adapt to all considered scenarios, and outperform the state-of-the-art approaches by 20%–30% on average given different QoE objectives.**

***Index Terms*—Virtual reality, deep reinforcement learning, video streaming, quality of experience**

I. Introduction

360-degree videos have received more and more attentions from the public in recent years, as they provide immersive scenes for users, increasing the Quality of Experience (QoE) of both video-on-demand and live-video streaming [1]. Many large-scale commercial video service providers such as YouTube, iQIYI and Hotstar have provided services on 360-degree video streaming. These emerging video services help attract new consumers, leading to a potential of great profits for the video streaming business.

360-degree videos are usually in high resolution, e.g., 4K, to ensure a good quality of content displayed to users. However, transmitting the whole 4K video content through the Internet requires extremely high bandwidth, while only the contents inside the viewports of the videos can be seen by the users [2]. Hence, researchers have provided viewport-based streaming strategies [3] such as tile-based 360-degree video streaming [4], and many works have been conducted to tackle this emerging streaming task [5]. Fig. 1 gives an illustration of a typical tile-based 360-degree streaming system, where tiles in the viewport are allocated with a high bit rate and other tiles are allocated with a low bit rate, to reduce the bandwidth consumption.

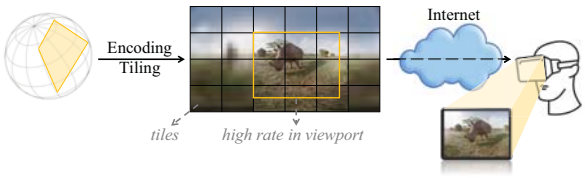


Fig. 1: Illustration of the tile-based 360-degree streaming.

There are two main challenges for streaming viewport-based 360-degree videos.

- **Vast number of potential time-variant features**: There are many time-variant factors, i.e., features, that may affect the decision making on rate allocation for tiles, for example, network conditions and locations of viewpoints. These features fluctuate over time and are difficult to predict, which complicate the rate allocation strategy. It is also hard to figure out which features are important to what extent for rate allocation for tiles.
- **QoE objectives vary with scenarios**: Streaming of 360-degree videos is an optimization problem with multiple QoE objectives, such as maximizing video quality in the viewport, minimizing rebuffering events, or maintaining the temporal and spatial video quality smoothness. Previous fixed heuristic methods fail to achieve the optimal performance across a broad set of dynamic features and multiple QoE objectives.

To address these challenges, it is desirable to design a streaming strategy to map the features to the QoE objectives, and then construct an optimization model for finding the best rate allocation scheme. Reinforcement Learning (RL) [6] is appropriate to adaptively optimize the long-term QoE reward. RL seeks to numerically conclude the states of a given system, and execute an action to optimize an accumulated reward. Deep learning further enables RL to scale to decision-making problems that were previously intractable, i.e., the problems with settings of high-dimensional state space and action space [7]. Many researchers have proposed RL-based models to learn the relationships between the features and the objectives, and achieved success in various application

* The first two authors contributed equally to this work.

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

domains.

In this paper, we present a general video-content independent framework of rate allocation for tiles in 360-degree video streaming, named *DRL360*. In the framework, we define the states and actions of video streaming, and propose a DRL-based model to adaptively learn the optimal rate allocation. The model leverages the high precision of the bandwidth prediction and the viewport prediction by Recurrent Neural Network (RNN), and then learns the long-term QoE reward by policy gradient. Evaluation shows that the proposed model can adapt to the network condition and the future changes in viewports, and outperform existing 360-degree streaming algorithms over the data with a number of QoE objectives. Our contributions are summarized as follows.

- We provide a DRL-based video-content independent framework of streaming 360-degree videos, where any possible feature can be included, given any QoE objective;
- We utilize the LSTM-based ACTOR-CRITIC (AC) model for optimizing various QoE objectives, which can adapt to the network condition and the future changes in viewports;
- We implement a prototype of the system. Evaluation on a number of 360-degree videos over various network conditions show that the proposed method outperforms state-of-the-art algorithms by 20%–30% on average, given different QoE objectives.

II. System Overview

A. 360-degree Video Streaming

A 360-degree video is composed of a series of images, called *frames*. The *frame per second* (FPS) indicates the frequency (frame rate) at which consecutive frames appear on a display. A *video chunk* is a short segment of the video, which gathers a number of consecutive frames. A 360-degree video streaming system usually compresses the video stream at the video chunk level. The video is divided into chunks with the same durations, denoted as T. The chunks are indexed from 1 to C, where C represents the number of chunks in the video.

Rate allocation for tiles: Each raw 360 video chunk is spatially segmented into small tiles based on their relative position, which are then compressed separately. In this paper, we segment the frames of the chunks to grids with $\mathcal{I}$ rows and $\mathcal{J}$ columns in a given shape. The split grids of a frame are the tiles of the frame, and grids at the same location compose the tiles of a chunk. Let (i, j) denote the location of the i-th row and the j-th column. The tiles are allocated and encoded by specific bit rates from a *candidate rate set*, i.e., the streaming system could determine the rate for each tile of a chunk based on the rate allocation strategy. Let $\mathcal{R}$ denote the candidate rate set, and let $r_{ij}(c) \in \mathcal{R}$ represent the rate for the tile at location (i, j) of the c-th chunk. After allocating rates to the tiles, the corresponding tiles are collected, which are then sent through the network. It is possible that a user may change his/her viewpoint during the playback of a 360-degree video. Due to the limited bandwidth, it is not wise to request streaming all tiles with a high rate for each chunk during the playback. Instead, an ideal system is to only allocate a high rate to tiles inside the viewport.

Two rules of rate selection: The proposed system DRL360 follows the DASH (Dynamic Adaptive Streaming over HTTP) structure [8], and adaptively optimizes the QoE objectives over the monitored playback statistics. The system follows two rules to allocate rates for the tiles of chunks.

- Rule 1: The tiles outside the viewport are allocated with a fixed non-zero rate lower than the tiles inside the viewport, to handle the case that the user may occasionally show an orientation against the prediction. Originally, these tiles are allocated with the lowest rate, which will be adjusted during the playback. Thus, the user will never view a "blank" screen during the playback.
- Rule 2: The tiles inside the viewport are allocated with the same selected rate, so that there will not be spatial variations inside the viewport. Hence, no distinct borders can be observed between the adjacent tiles.

Based on these two rules, the DRL360 system, which will be introduced in Sec. III, selects one rate for the tiles inside the viewport.

B. Feature Selection

We select the following features that are important for learning the optimal rate allocation scheme.

1) Viewpoint and viewport: During the playback of a 360-degree video, only a specific region of the video can be viewed by the user. The region is called *viewport*, where the center of the viewport is called *viewpoint*. The viewpoint for the c-th chunk can be seen as a pair of longitude and latitude. Based on the viewpoint and the field-of-view of the client, we could locate the coverage of the viewport. Let an indicator matrix $\mathbf{v}(c) \in \{0, 1\}^{\mathcal{I} \times \mathcal{J}}$ represent whether a tile is inside the viewport, where the element on the i-th row and the j-th column is denoted by $v_{ij}(c)$. If the tile at (i, j) is in the viewport, $v_{ij}(c) = 1$; otherwise, $v_{ij}(c) = 0$.

2) Video-content independent features: The attribute of a 360-degree video can affect the streaming. The size of the file and the encoding algorithm influence in which rate the tiles can be downloaded. The tiling strategy requires the client to find a balance: on one hand, changing the tiling scheme such as changing the shape of grids or segmenting the chunk to more tiles can help accurately allocate a high rate to the viewport and a low rate for the tiles outside the viewport [9]; on the other hand, splitting into more tiles or into special shapes would increase the size of the chunk and the overhead on the client [1]. The size for the tile at (i, j) of the c-th chunk under a coding algorithm with specific tiling scheme would be denote by $d_{c,ij}(r)$, where $r \in \mathcal{R}$. The size of the c-th chunk, denoted as $z(c)$, can then be computed as

$$z(c) = \sum_{i=1}^{\mathcal{I}} \sum_{i=1}^{\mathcal{J}} d_{c,ij}(r_{ij}(c)). \tag{1}$$

Meanwhile, the total rates inside the viewport, denoted as $f(c)$, could be thereby calculated by:

$$f(c) = \sum_{i=1}^{\mathcal{I}} \sum_{i=1}^{\mathcal{J}} d_{c,ij}(r_{ij}(c)) v_{ij}(c). \tag{2}$$

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

3) Bandwidth: The video streaming over the network is in a dynamic process, where the bandwidth at timestamp t is denoted as $N(t)$. Suppose the client begins to send the requests of the c-th chunk at time t_c, and the average download speed for this chunk is denoted by N_c. We assume that the requests to the tiles of the same chunk is executed continuously, while there may be a short delay Δt_c between the c-th and $(c+1)$-th chunks. Hence, the beginning of time of downloading two consecutive chunks can be calculated by:

$$t_{c+1} = t_c + \frac{z(c)}{N_c} + \Delta t_c, \tag{3}$$

where

$$N_c = \frac{1}{t_{c+1} - t_c - \Delta t_c} \int_{t_c}^{t_{c+1}-\Delta t_c} N(t)dt. \tag{4}$$

4) Buffer Occupancy: The chunks are downloaded into a *playback buffer*, which is actually a queue of the not-yet-viewed video content. The buffer is divided into slots. Each slot contains a single chunk. Define $B(t) \in [0, B_{\max}]$ as the remaining playback time of the video content in the buffer, a.k.a., *buffer occupancy*, at timestamp t, where $B_{\max}$ denotes the *buffer capacity*. As watching the 360-degree video involves a large amount of interaction, $B_{\max}$ is usually set to a few seconds of video content.

The buffer occupancy changes along with time during the playback. According to the setting, one chunk in the buffer will be consumed in every T seconds. Meanwhile, whenever the chunk has been downloaded completely, the buffer occupancy will increase by T seconds. Let $B_c = B(t_c)$ stand for the buffer occupancy when the request for the c-th chunk is sent. During the startup stage, to avoid an empty buffer, the client is expected to download several chunks with all tiles before the playback begins. Suppose that the clients should download $\mathcal{S}$ chunks in the startup stage, for $c \leq \mathcal{S}$, the buffer occupancy follows: $B_{c+1} = c \times T$. After the startup stage when $\mathcal{S} < c \leq C$, the buffer occupancy changes according to the downloading of the new chunk and the playing of the chunks in the buffer, i.e.:

$$B_{c+1} = \rho\Big(B_c - \frac{z(c)}{N_c}\Big) + T - \Delta t_c, \tag{5}$$

where $\rho(x) = \max\{x, 0\}$ denotes the rectified linear function.

Rebuffering and rate adaption: The playback and downloading strategy should adapt to the buffer occupancy. If the buffer occupancy reaches 0, i.e., there is no chunk in the buffer, the playback will suffer from *rebuffer*. The playback will not recover until there is at least one chunk in the buffer. In this condition, the rate for the tiles outside the viewport will be downgraded to a smaller one if it has not become the lowest rate. Besides, the downloading of two consecutive chunks should be continuous to minimize the unnecessary delay. However, in the case when there is no space for inserting a chunk into the buffer, the downloading has to be suspended for a short delay. Meanwhile, the rate for the tiles outside the viewport will be upgraded to a greater one if it has not become the highest rate. Formally, when $B_c + 1 > B_{\max}$, the delay can be computed by:

$$\Delta t_c = \rho\bigg(\rho\Big(B_c - \frac{z(c)}{N_c}\Big) + T - B_{\max}\bigg). \tag{6}$$

Otherwise when $B_c + 1 \leq B_{\max}$, $\Delta t_c = 0$. Note that the rate for the tiles outside the viewport should never be greater than the rate of tiles inside the viewport. Hence, the rate for the tiles outside the viewport should be immediately degraded to an appropriate level when the allocated rate for the tiles inside the viewport decreases.

C. Quality of Experience Metrics

The quality of experience perceived by users determines how long a user will engage in a video. Users may have various preferences on the QoE of the playback under different scenarios. The proposed deep reinforcement learning model is able to optimize any QoE objective. To be consistent with the common streaming systems for 360-degree videos, we focus on the following metrics that may contribute to the QoE between the $\underline{C}$-th chunk and the $\bar{C}$-th chunk:

- *Average Viewport Quality* [Mb]: As only the content inside the viewport can be viewed, the average quality of one tile in viewports over all chunks is important to the QoE. The reinforcement learning models can handle any types of quality representations. We take the linear representation of average quality as an example:

$$Q^{\bar{C}}_{1,\underline{C}} = \frac{1}{\bar{C} - \underline{C} + 1} \sum_{c=\underline{C}}^{\bar{C}} \frac{f(c)}{\sum_{i=1}^{\mathcal{I}} \sum_{i=1}^{\mathcal{J}} v_{ij}(c)}. \tag{7}$$

- *Rebuffer* [s]: When the downloading time of a chunk is greater than the buffer occupancy, the client would suffer from rebuffer until the new chunk is ready to play. The overall rebuffer time can be calculated by:

$$Q^{\bar{C}}_{2,\underline{C}} = \sum_{c=\underline{C}}^{\bar{C}} \rho\Big(\frac{z(c)}{N_c} - B_c\Big). \tag{8}$$

- *Average Viewport Temporal Variations* [Mb]: The rate changes between two viewports of consecutive chunks may impulse users with physiological symptoms such as dizziness and headache. Hence, the changes in the quality of viewports should not be drastic, which can be measured by:

$$Q^{\bar{C}}_{3,\underline{C}} = \frac{1}{\bar{C} - \underline{C} + 1} \sum_{c=\underline{C}}^{\bar{C}} \Big| \frac{f(c)}{\sum_{i=1}^{\mathcal{I}} \sum_{i=1}^{\mathcal{J}} v_{ij}(c)} - \frac{f(c-1)}{\sum_{i=1}^{\mathcal{I}} \sum_{i=1}^{\mathcal{J}} v_{ij}(c-1)} \Big|. \tag{9}$$

Then we define the QoE objective over the entire video by a weighted summation formulation:

$$\text{QoE}^{\bar{C}}_{\underline{C}} = \eta_1 Q^{\bar{C}}_{1,\underline{C}} - \eta_2 Q^{\bar{C}}_{2,\underline{C}} - \eta_3 Q^{\bar{C}}_{3,\underline{C}}, \tag{10}$$

where $\boldsymbol{\eta} = (\eta_1, \eta_2, \eta_3)$ are the non-negative weighting parameters corresponding to the elements.

Therefore, to provide a high QoE for users, we propose a rate allocation scheme to determine the rate to the tiles of all chunks and maximize the QoE objective:

Problem 1.

$$\arg \max_{\{r_{ij}\}_{\mathcal{I} \times \mathcal{J}(1 \ldots C)}} \text{QoE}^{C}_{1}$$

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

subject to

Constraints (3)–(6).

III. Deep Reinforcement Learning Framework for 360-degree Video Streaming

The video streaming system contains high-dimensional action space and state space, which is not appropriate for a legacy RL model. The Deep RL model is usually used for processing high dimensional data. Specifically, we design two Long Short-Term Memory (LSTM) [10] models to predict the bandwidth and the viewport in the future, and an LSTM based deep neural network to reveal the relationship between the features and the QoE objectives.

A. The Agent-Environment Interface

The legacy RL methods consider a general setting in which an *agent* interacts with an *environment*. The decision maker is called the agent. The thing it interacts with is called the environment. At each step, the agent makes an *action* based on its current *state*. The environment responds to the action and reveals new *observations* to the agent, leading the agent to update its state. The environment also gives a rise to a *reward*, i.e., a special numerical value. The agent seeks to maximize the rewards over steps through its choice of actions.

More specifically, in 360-degree video streaming, the client requesting the playback can be seen as the agent, and everything outside the agent during the playback can be treated as the environment. Initially, the agent maintains a state s_0. The agent and environment interact when each chunk is going to be requested. At each chunk c, the agent maintains a state s_c, describing the historical and current information of the streaming system. On that basis the agent selects an action, a_c, i.e., allocating rates for tiles in chunk $(c+1)$. One chunk later, in part as a consequence of its action, the agent receives a numerical QoE reward, τ_{c+1}, and obtains a new observation o_{c+1}, i.e., some statistics of the system, leading itself to a new state s_{c+1}.

Thereby, a sequence can be generated as:

$$s_0, a_0, \tau_1, o_1, s_1, a_1, \tau_2, o_2, s_2, a_2, \tau_3, o_3, \ldots, \tau_C, o_C, s_C.$$

B. Definitions

States: In practice, the client observes and records the temporal chunk ID c, viewport $\mathbf{v}(c)$, bandwidth $N(t_c)$, buffer occupancy B_c. Besides, the client has access to the size of the tiles of the next chunk, i.e., $d_{c,ij}(\cdot)$. However, due to dynamic viewport and dynamic bandwidth of 360-degree video streaming, the system is expected to estimate the viewport and the bandwidth in the future.

Let $N'(t)$ denote the estimated bandwidth at timestamp t, and let $\mathbf{v}'(c)$ correspond to the predicted viewport of the c-th chunk. Then, an observation can be described as a tuple:

$$o_c = <c+1, t_{c+1}, B_{c+1}, \mathbf{v}'(c+1), d_{c+1,ij}(\cdot), N'(t_{c+1}+1), \ldots, N'(t_{c+1}+\iota)>,$$

where ι stands for a prediction period of the future bandwidth.

We notice that the playback is a continuous process. Suppose there is a function $H(\cdot)$ which can conclude the historical observations, we define the state (except for the initial state s_0) as a representation of the historical observations, which reflects the valuable information of the previous playback that may contribute to the streaming:

$$s_c = H(o_1, o_2, \ldots, o_c). \tag{11}$$

Note that the observations are influenced by the selected rates, the allocated rates for tiles of the previous chunks are thereby embedded in the state.

Actions: The DRL-based model seeks to determine the rate for the tiles inside the viewport for the $(c+1)$-th chunk based on the current state s_c, representing the action $a_c \in \mathcal{R}$.

Rewards: The client records the average rate of the tiles in the viewports of each chunk along with the playback and monitors the rebuffer time. Therefore, the value of the QoE objectives can be obtained in real time, which is taken as the reward for the RL model. Specifically, the reward of the c-th chunk τ_c would be calculated by $\tau_c = QoE_c^c$.

Policy: The goal of the reinforcement learning is to maximize the expected cumulative discounted reward, i.e., $\mathbb{E}[\sum_{c'=c}^{C} \gamma^{c'-c}\tau_{c'}]$, where $\gamma \in (0,1]$ is a factor discounting future rewards. To reach the maximum of the rewards, the actions should be selected based on a stochastic policy, defined as a probability distribution over actions $\pi_{\boldsymbol{\theta}} : (s_c, a_c) \to [0,1]$, where $\boldsymbol{\theta}$ represents the parameters of the DRL-based model. $\pi_{\boldsymbol{\theta}}(s_c, a_c)$ is the probability that action a_c is taken given the current state s_c. The policy takes all the historical information of the playback, and on that basis generates the best strategy of allocating rates for tiles.

Then, Problem 1 can be transformed into:

$$\pi_{\boldsymbol{\theta}}^* = \arg\max_{\pi_{\boldsymbol{\theta}}} V^{\pi_{\boldsymbol{\theta}}}(s_0), \tag{12}$$

where

$$V^{\pi_{\boldsymbol{\theta}}}(s_c) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\sum_{c'=c+1}^{C} \gamma^{c'-c-1}\tau_{c'}]$$

is the *state value function* with polity $\pi_{\boldsymbol{\theta}}$, i.e., the expectation of the cumulative discounted QoE reward by $\pi_{\boldsymbol{\theta}}$ from state s_c to the end of the playback. Finally, according to the learned policy $\pi_{\boldsymbol{\theta}}^*$, DRL-based model could determine the action with the greatest value and select the corresponding rate for the tiles inside the viewport.

C. Models for Predicting Bandwidth and Viewport

The DRL360 system requires the estimated bandwidth and viewport to be included in the state. RNN models have been proved to be efficient in solving the time series prediction problem. The DRL360 system employs an anti-long-term-dependency version of RNN, i.e., LSTM network, for the prediction. The LSTM cell can encode the historical information in its hidden state and make accurate prediction.

The LSTM sets a *forget gate* and an *input gate* to determine which playback information from the inputs should be discarded, and how much information should be included in the state of the cell. Let θ_{LSTM} denote the parameters of the LSTM network. Given the inputs $x_0, \ldots, x_t$ indicating the statistics of the playback, the LSTM model can be abbreviated to

$$x'_{t+1} = \text{LSTM}(x_0, \ldots, x_t; \theta_{\text{LSTM}}), \tag{13}$$

where x'_{t+1} is the prediction from the model.

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

Therefore, we could use the LSTM model to predict the bandwidth in time level, following:

$$N'(t+1) = \text{LSTM}(N(0), \ldots, N(t); \theta_{\text{N,LSTM}}), \tag{14}$$

where $\theta_{\text{N,LSTM}}$ stands for the parameters for predicting bandwidth. The predicted bandwidth can be recurrently fed into the LSTM model, so that we can get ι bandwidth values for the observation. Similarly, we could predict the viewport in the future in chunk level by LSTM, obtained by:

$$\mathbf{v}'(c+1) = \text{LSTM}(\mathbf{v}(1), \ldots, \mathbf{v}(c); \theta_{\mathbf{v},\text{LSTM}}), \tag{15}$$

where $\theta_{\mathbf{v},\text{LSTM}}$ denotes the parameters for predicting viewport.

D. DRL-based Optimization for QoE Objectives

In policy-based reinforcement learning approaches, policy $\pi_{\boldsymbol{\theta}}(s_c, a_c)$ is parameterized by $\boldsymbol{\theta}$ and is iteratively improved by updating $\boldsymbol{\theta}$. We use the ACTOR-CRITIC (AC) [6] as the RL training algorithm, and optimize the QoE objectives. We define the utility function for choosing actions in specific states as:

$$J(\boldsymbol{\theta}) = \sum_{c'=1}^{C} \gamma^{c'-1} \tau_{c'}. \tag{16}$$

Policy gradient algorithms estimate the QoE gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ over the entire playback process following the policy $\pi_{\boldsymbol{\theta}}(s_c, a_c)$. Specifically, the QoE gradient could be unbiasedly computed by:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_{c=0}^{C-1} \left[\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s_c, a_c) \sum_{c'=c+1}^{C} \gamma^{c'-c-1} \tau_{c'} \right] \tag{17}$$

based on a given playback trajectory.

However, this method must wait until the simulation to the end of the playback to know the value of R_c and then the QoE gradient can be determined. In this case, we introduce the AC algorithm which needs to wait only until the next video chunk, so that the methods are fully online and incremental. The *AC error* is defined to evaluate the precision of the estimation of the state-value, calculated by

$$\delta_c = \tau_{c+1} + \gamma \hat{V}(s_{c+1}, \mathbf{w}) - \hat{V}(s_c, \mathbf{w}), \tag{18}$$

where $\hat{V}(s_c, \mathbf{w})$ is an estimation of the state value and $\mathbf{w}$ is the weight vector, named *state-value weight*. Obviously, DRL360 should minimize the AC error, so that it will make an unbiased estimation with the one-step QoE reward. Then the QoE gradient can be transferred to be based on the AC error, i.e.,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_{c=0}^{C-1} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s_c, a_c) \delta_c. \tag{19}$$

Therefore, the update rule could be derived by:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{20}$$

where α is the preset learning rate (could be adjusted dynamically).

The training methodology of DRL360 is concluded in Alg. 1. It first trains the parameters of the prediction models for bandwidth and viewport (line 2). Then, the model repeatedly simulates the viewing events and calculates the QoE objective of the viewing records (line 6–14). Based on the updating rules, the parameters are trained, and finally the trained models will be returned.

Algorithm 1 Training methodology of DRL360.

Require:
Factor discounting future reward, γ; Learning rate of policy parameterization, $\alpha^{\boldsymbol{\theta}}$; Learning rate of state-value parameterization, $\alpha^{\mathbf{w}}$; Training data, D.

Ensure:
Parameters of LSTM, policy and state-value estimation, $\theta_{\text{N,LSTM}}$, $\theta_{\mathbf{v},\text{LSTM}}$, $\boldsymbol{\theta}$ and $\mathbf{w}$.

1: Initialize parameters $\theta_{\text{N,LSTM}}$, $\theta_{\mathbf{v},\text{LSTM}}$, $\boldsymbol{\theta}$ and $\mathbf{w}$;
2: Train the LSTM models on D, and retrieve $\theta_{\text{N,LSTM}}$ and $\theta_{\mathbf{v},\text{LSTM}}$;
3: **repeat**
4: **for** each data trajectory in D **do**
5: Initialize s_0;
6: **for** $c \in [0, C-1]$ **do**
7: Sample $a_c \sim \pi_{\boldsymbol{\theta}}$;
8: Select rate by a_c, obtain τ_{c+1};
9: Predict $\mathbf{v}'(c+2)$ and $N'(t_{c+2}+1), \ldots, N'(t_{c+2}+\iota)$;
10: Retrieve o_{c+1} and s_{c+1};
11: $\delta \leftarrow \tau_{c+1} + \gamma \hat{V}(s_{c+1}, \mathbf{w}) - \hat{V}(s_c, \mathbf{w})$;
12: $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^c \delta \nabla_{\mathbf{w}} \hat{V}(s_c, \mathbf{w})$;
13: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^c \delta \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s_c, a_c)$;
14: **end for**
15: **end for**
16: **until** Reward (QoE) will not increase anymore.
17: **return** $\theta_{\text{N,LSTM}}$, $\theta_{\mathbf{v},\text{LSTM}}$, $\boldsymbol{\theta}$ and $\mathbf{w}$.

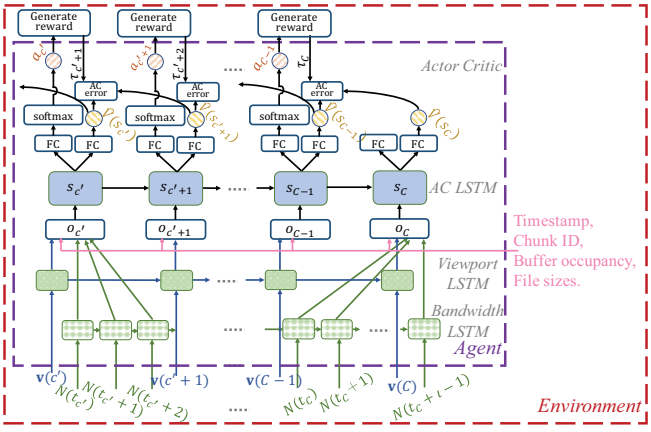


Fig. 2: The structure of the DRL-based model in the DRL360 streaming system.

In practice, we implement an LSTM based model for function $H(\cdot)$ in Eqn. (11), named *AC LSTM*. The tasks of learning policy and estimating state-value function share the parameters of the AC LSTM, denoted as θ_{R}. The AC LSTM cell takes the observation from the playback environment as the inputs, and receives the output from the previous cell to understand the historical playback conditions. The c-th AC

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

LSTM cell computes:

$$s_c = \text{LSTM}(o_1, o_2, \ldots, o_c; \theta_{\text{R}}). \tag{21}$$

We construct a fully-connected (FC) network g_v with parameter $\mathbf{w}'$ to approximate the accumulated reward, by:

$$\hat{V}(s_c; \mathbf{w}) = g_v(s_c; \mathbf{w}'), \tag{22}$$

where $\mathbf{w} = \{\theta_{\text{R}}, \mathbf{w}'\}$. Similarly, we have a softmax-activated [11] FC network g_π with parameter $\boldsymbol{\theta}'$ to learn the policy $\pi_{\boldsymbol{\theta}}$, deduced by:

$$\pi_{\boldsymbol{\theta}}(s_c, a_c) = g_\pi(s_c; \boldsymbol{\theta}'), \tag{23}$$

where $\boldsymbol{\theta} = \{\theta_{\text{R}}, \boldsymbol{\theta}'\}$. Based on the learned policy, the action with the greatest softmax value is selected, and the corresponding rate is allocated to the tiles inside the predicted viewport. Within the environment, the state is transferred, and the system can make the request for the next video chunk when the buffer has vacant space.

E. Procedure of DRL360 for 360-degree Video Streaming

The structure of the DRL-based model is depicted in Fig. 2. DRL360 system requires an interactive procedure with the environment for the agent. The environment keeps simulating the playback of the following chunks, and observes the features during the entire playback. The features are recurrently fed to the DRL-based model, so that the model can update the parameters during training and provide appropriate actions for optimizing QoE objectives. When requesting for a new chunk, the environment provides the previous bandwidth records and viewport observations to the prediction LSTM models respectively. The prediction LSTM models generate one viewport and bandwidth values in the future ι timestamps, which are then concatenated to the other information in observation. The observation goes through the AC LSTM cell, the output of which (i.e., the current state) is leveraged to estimate the state value and generate action by two distinct FC networks. The generated action affects the environment, and the environment brings back the reward and the new observation, which can be leveraged to update the parameters during training process.

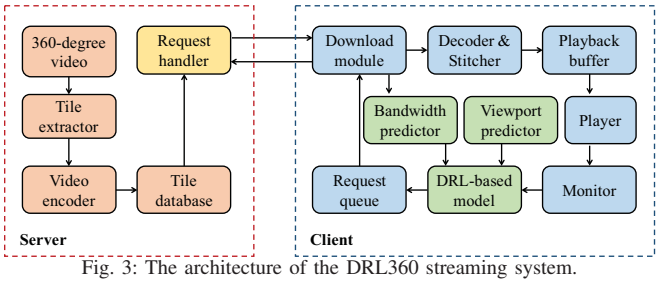


Fig. 3: The architecture of the DRL360 streaming system.

IV. Implementation

The architecture of our system consists of two major parts: server-side modules and client-side modules, as illustrated in Fig. 3.

The server is implemented on a cluster with Gigabit-NIC, which contains two categories of modules: The **offline modules** include the *tile extractor* and *video encoder*, which prepare the tiles of chunks with all candidate rates of all videos offline. The tiles are then stored in the *tile database*. Regarding **online modules**, the server sets an interface as the *request handler* to respond to the requests from the clients. According to the requests, the server retrieves the tiles from the tile database. Then, the server sends the tiles to the client through HTTP/2.0 connection.

A client application is deployed on the client, which displays the videos, and records the statistics of the playback. The *bandwidth predictor*, *viewport predictor* and DRL-based model are embedded in the client application, and the parameters are received from the server. At the beginning of the playback, the client would first request for a XML file, which records the information of tiles of the video, including the number of chunks, durations and the size of each tile. Then during playback, the application will find the state of the playback s_c, and execute an action a_c to select a rate for the tiles inside the viewport, based on the learned policy $\pi_{\boldsymbol{\theta}}(s_c, a_c)$ of the DRL360 system. The action is transformed to a request for chunk, and inserted to the *request queue*. The *download module* requests for chunks in turn. The downloaded chunks are decoded by the *decoder* and stitched by the *stitcher* to a complete chunk. The chunk can finally be displayed by the *player* with high resolution.

V. Evaluation

In this section, we evaluate the performance of DRL360 over trace-driven evaluations and real-world experiments under a wide variety of network conditions.

A. Settings

Hyper-parameters. In the experiments, the values of the hyper-parameters are fixed as shown in Tab. I. The learning rate is updated according to Adam and exponential decay [11].

TABLE I: Parameters used in this paper.

T	γ	$\lvert\theta_{\text{R}}\rvert$	$\lvert\theta_{\mathbf{v},\text{LSTM}}\rvert$	$\lvert\theta_{N,\text{LSTM}}\rvert$	ι	$\mathcal{S}$	$B_{\max}$
1s	1	128	32	8	10	1	4s

Videos and viewpoint trajectories. We collect sixteen 360-degree 25-FPS videos from the dataset [12] which includes both videos in fast pace or slow pace. In this dataset, 48 viewpoint trajectories are attached to each video in a 200-millisecond granularity. We randomly select two videos as the test set, and the rest of the videos are taken as the training set. The experiments are repeated for ten runs, and the mean values are reported. We design the videos to have five candidate rates, i.e., $|\mathcal{R}| = 5$. Each of the entire video is encoded by FFMPEG with X.264 under the average bitrate (ABR) mode, assigned by 1Mbps (360p), 5Mbps (720p), 8Mbps (1080p), 16Mbps (2K) and 35Mbps (4K). Then the videos are segmented to chunks of one second length. For each chunk, although DRL360 can be applied to any splitting scheme, we split the tiles in four rows and six columns in this experiment, which is found to have acceptable overload of encoding.

Bandwidth traces. We select 16 bandwidth traces with at least 400 seconds from the dataset HSDPA [13] with various bandwidth patterns. We randomly select two bandwidth traces to appear only in the test data. The bandwidth values are scaled up to support the playback of the 360-degree videos. To simulate the real scenarios and ensure that the user can

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

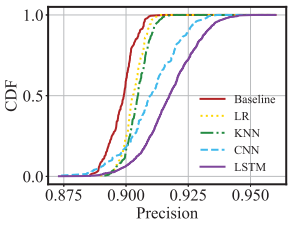


Fig. 4: The CDF of the precision of different models for predicting viewport.

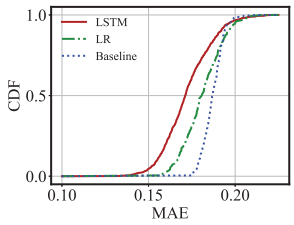


Fig. 5: The CDF of the precision of different models for predicting bandwidth.

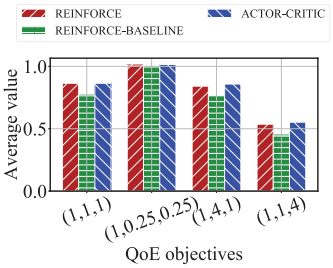


Fig. 6: Comparison over various QoE objectives produced by changing DRL-based models in DRL360.

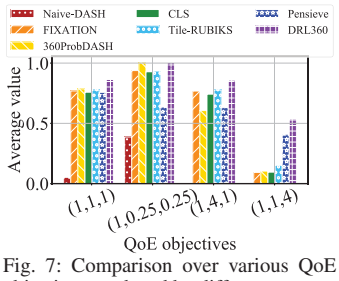
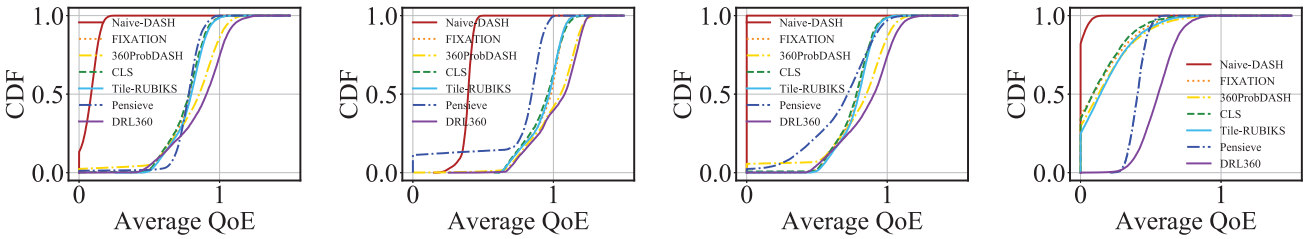


Fig. 7: Comparison over various QoE objectives produced by different streaming systems.



(a) (1,1,1) (b) (1,0.25,0.25) (c) (1,4,1) (d) (1,1,4)

Fig. 8: The CDFs of the average QoE value under four QoE objectives generated by the compared streaming systems.

watch the 360-degree videos with at least the lowest rate, the average bandwidth values fall between 2Mbps to 15Mbps.

RL algorithms for 360-degree video streaming. As there are many successful RL algorithms, we make comparisons over the following three algorithms to show that we choose the most appropriate one for this task:

- **ACTOR-CRITIC** is the utilized model in this paper which takes Eqn. (19) as loss.
- **REINFORCE** calculates the loss by Eqn. (17).
- **REINFORCE-BASELINE** calculates the loss by leveraging $\hat{V}(s_c, \mathbf{w})$ as the baseline function to reduce the variance.

We compare the performance of the proposed DRL360 system with six state-of-the-art 360-degree video streaming systems.

- **Naïve-DASH** [8] is the baseline, which dynamically allocates rates for the entire video, without the tile-based allocation. It uses linear regression (LR) [11] to predict the bandwidth, and selects the maximal rates within the bandwidth estimation.
- **FIXATION** [14] resorts to content-dependent features to predict the viewport in the future, and considers the previous bandwidth without prediction.
- **360ProbDASH** [15] leverages LR to predict the viewport, and sets up a probabilistic optimization model on average viewport rate to allocate rates for tiles.
- **CLS** [16] manually groups the historical viewpoint trajectories by DBSCAN [11], and predicts the viewport by the clusters. It constructs a Knapsack model to determine the rates for tiles.
- **360-Pensieve** is extended from Pensieve [17] which is a RL-based model for DASH over monocular video leveraging A3C model [6]. Since Pensieve does not involve a specific prediction model for bandwidth, we extend it for the 360-degree video scenario, which includes the previous viewpoint trajectories.
- **Tile-RUBIKS** is based on RUBIKS [18], which is originally designed for the videos under the Scalable Video Coding, and uses past head motion to predict future viewports. It constructs a optimization framework to allocate rates for tiles, while conservatively predicts the future network. We implement the algorithm on the tile-based 360-degree videos without the layer concepts.

QoE objectives. We select four sets of parameters for $\boldsymbol{\eta}$ to indicate four QoE objectives with various preferences, namely (1,1,1), (1,0.25,0.25), (1,4,1), (1,1,4), indicating comprehensive consideration, maximizing the rates of viewports, minimizing the rebuffer time and reducing the viewport temporal variations respectively.

Trace-driven emulation setup. The video server runs on a laptop with NIC Intel I350 (also used for the prototype of DRL360). The machine learning models are offline trained and embedded in the client. The client emulates the downloading and playing processes, and the statistics are recorded.

B. Precision of Predicting Viewport and Bandwidth

We first examine the performance of the prediction models for viewports and bandwidth values. We compare the LSTM prediction model for viewports with **LR** algorithm, Image-based **CNN** model [14] and **KNN** algorithm on clusters by DBSCAN [16]. The **Baseline** algorithm treats the latest viewport as the future viewports for each chunk. As $B_{\max}$ is set to 4 in the setting, we check the average precision of three future consecutive viewports. We leverage the precision metric to evaluate the performance, based on:

$$\text{precision} = 1 - \mathbb{E}_c[\frac{\sum_{x=1}^{3} ||\mathbf{v}(c+x) - \mathbf{v}'(c+x)||_2^2}{3\mathcal{I}\mathcal{J}}].$$

Results of predicting the viewport are plotted in Fig. 4, showing that our proposed LSTM prediction model provides the greatest precision, and LSTM is appropriate for this task.

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

Similar results can be found on predicting bandwidth. The **LR** algorithm is evaluated for comparison, and the scheme that uses the latest bandwidth to estimate the future bandwidth is seen as the **Baseline**. According to the $B_{\max}$, the predictor is expected to predict the bandwidth in the future three seconds. Regarding the performance, we focus on the Mean Absolute Error (MAE) of the prediction of bandwidth, namely:

$$\text{MAE} = \mathbb{E}_t[\frac{\sum_{x=1}^{3} \left|N'(t+x) - N(t+x)\right|}{3}].$$

We show the results of predicting bandwidth in Fig. 5, where we find that LSTM achieves a significant improvement over the compared algorithms.

C. Performance of RL Algorithms

REINFORCE algorithms with and without baseline are evaluated for comparison against DRL360 to find the best RL structure for the 360-degree video streaming task. We evaluate the performance on four QoE metrics of each RL structure by replacing the DRL-based model (in Fig. 3), and the results are shown in Fig. 6. We can observe that on average, DRL360 provides the highest precision, which implies that DRL360 can optimize various QoE objectives, and it is efficient for streaming 360-degree videos.

D. Comparison with Existing Approaches

We compare the proposed DRL360 approach with state-of-the-art algorithms on each QoE objective mentioned in the settings. Fig. 7 shows the average QoE that each streaming algorithm achieves on the entire test data, and Fig. 8 illustrates detailed results by CDFs for each algorithm under all four QoE objectives. The comparison indicates that DRL360 outperforms the best of existing algorithms on each QoE metric over the test traces, and about 20%–30% improvement on average can be observed through the results. In particular, comparing to the approaches other than the baseline, DRL360 improves the average viewport quality by 5%–20% when seeking to maximize the average quality in QoE objective, i.e., (1,0.25,0.25), and suffers from about one tenth of the rebuffer time produced by the compared approaches when focusing more on the rebuffer time in QoE objective, i.e., (1,4,1). Besides, DRL360 harmonizes the priorities among the three metrics when considering most on the viewport temporal variations in the QoE objective, i.e., (1,1,4), and outperforms the other approaches.

The detailed observations are two-fold. First, the FIXATION, 360ProbDASH and CLS algorithms employ fixed control strategies. We observe that these heuristics struggle to optimize different QoE objectives. These three algorithms tend to concentrate more on the average rate in viewport, so that they receive high performance on the first two QoE objectives, but perform relatively poor on the other two objectives. Specifically, since FIXATION leverages complex CNN network and CLS includes massive clusters along with each chunk, the computation delay limits their performance. It can be inferred that different QoE objectives require inherently different streaming strategies. Therefore, it is important to automatically learn these policies, and thus the performance of the proposed DRL360 system remains consistently high, no matter how the objective changes. Furthermore, we can infer that execution of LSTM and DRL models would not generate much overhead on the players when the parameters are trained and fixed.

Second, although the 360-Pensieve and Tile-RUBIKS algorithms have the capability to adapt to a variety QoE objectives, they do not work the best for this tile-based rate allocation task. The 360-Pensieve relies on a neural network with simple structure, which lacks of designs of submodules to predict viewpoints and bandwidth and make rate allocation scheme over tiles. Due to the limited scenarios in the training data, 360-Pensieve tends to overfit on the historical traces, losing the generalization capability. From monocular video to 360-degree video, the model would hardly converge on the optimum. Meanwhile, Tile-RUBIKS is constructed on the control theory, which tries to conservatively improve the value of the QoE objectives in several subsequent chunks. This heuristic helps the streaming system perform equally on various QoE objectives, while the optimum can hardly be reached. The proposed DRL360 system overcomes these limitations by benefiting from the predictions and automatically learning the long-term policy without complex tuning.

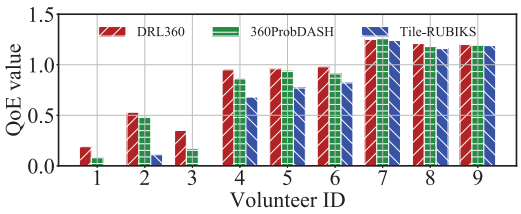


Fig. 9: Results of the system prototype, given nine volunteers with QoE objective of (1,1,1).

E. Results over System Prototype

We develop a prototype of DRL-based streaming system. We use Python-Django for the server and WebGL for the client. The server is deployed on a campus network. We also implement 360ProbDASH and Tile-RUBIKS algorithms as comparison. Three videos from the dataset are involved. We invite nine volunteers to watch the videos, and the QoE objective with $\boldsymbol{\eta} = (1, 1, 1)$ is recorded. Volunteers 1–3 are invited to watch the videos through 3G, volunteers 4–6 are asked to watch through Wi-Fi in laboratory, and the others are required to view through 4G-LTE. The volunteers watch each of the three videos, but have no idea about which videos are streamed by which streaming system. The records of the QoE objective are illustrated in Fig. 9.

We observe that under various network conditions and viewport trajectories in the real-world scenarios, the proposed DRL-based system outperforms the other two systems on the QoE objective in the Wi-Fi and 3G network, while they perform about the same in the 4G network whose network condition reaches about 25Mbps. Note that in the real-world scenarios, the viewports of the volunteers tend to be more random than the trajectories in the dataset, leading to a lower precision of the prediction of viewports by non-deep-learning methods. Hence, we can observe greater improvement of DRL360 in such scenarios.

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.

VI. Related Work

360-degree video streaming models. Streaming 360-degree videos has become a hot topic in the past few years. Regarding improving the rate of content inside the viewport, researchers have proposed tile-based [5] and layer-based [19] coding schemes to allocate various rates to different regions in a video frame. Deterministic models [1] and probabilistic models [15] are both verified to be appropriate for some scenarios of the 360-degree video streaming. To accurately predict the viewports in the future, efforts have been made based on single user's viewpoint trajectories [20] and cross users' viewpoint trajectories [16]. Meanwhile, many works focus on adapting to the network by the control theory [18] or special design over the cellular network [21].

Reinforcement learning in multimedia applications. Reinforcement learning (RL) [6] is an emerging technology which adaptively mines the relationship among the features in a dynamic system and takes the appropriate actions for maximizing a given objective. Recently, RL has shown great popularity in multimedia applications. On one hand, RL provides opportunities for efficient interaction with the video, such as video prediction [22] and visual tracking [23], so that the content sender can present specific content to the client. On the other hand, rate adaptation [17], encoder [24] and the image restoration [25] can benefit from the RL techniques, which help take full advantages of the limited resources and improve the QoE of the users.

VII. Conclusion

In this paper, we present a deep reinforcement learning based 360-degree video streaming system, with a focus on efficiently utilizing the limited bandwidth resources to improve the QoE of users when watching the viewport- and tile-based 360-degree videos. The proposed DRL360 is a video-content independent model, which can adapt to dynamically changing features in environments, and optimize various QoE objectives. The model leverages deep learning to predict the bandwidth and viewports in the future, and allocates the rates for tiles by the ACTOR-CRITIC algorithm. Trace-driven evaluations indicate that the proposed DRL360 outperforms state-of-the-art algorithms over four QoE metrics with different concerns. We develop a prototype of the DRL360 system, and reveal that it can adapt to many real-world playback conditions and provide high QoE.

Acknowledgment

This work is partially supported by the National Key Research and Development Program No. 2017YFB0803302 and the National Natural Science Foundation of China under Grant Nos. 61572051, 61632017 and 61625101.

References

[1] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan, "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices," in *Mobicom, 2018 Proceedings ACM*, 2018, pp. 1–12.

[2] Chao Zhou, Zhenhua Li, and Yao Liu, "A measurement study of oculus 360 degree video streaming," in *Proceedings of the 8th ACM MMsys*. ACM, 2017, pp. 27–37.

[3] Zhetao Li, Fei Gui, Jinkun Geng, Dan Li, Zhibo Wang, Junfeng Li, Yang Cheng, and Usama Zafar, "Dante: Enabling fov-aware adaptive fec coding for 360-degree video streaming," in *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. ACM, 2018, pp. 15–21.

[4] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen, "Optile: Toward optimal tiling in 360-degree video streaming," in *Proceedings of the 2017 ACM MM*. ACM, 2017, pp. 708–716.

[5] Mengbai Xiao, Chao Zhou, Viswanathan Swaminathan, Yao Liu, and Songqing Chen, "Bas-360: Exploring spatial and temporal adaptability in 360-degree videos over http/2," in *INFOCOM, 2018 Proceedings IEEE*, 2018, pp. 1–9.

[6] Richard S Sutton, Andrew G Barto, et al., *Reinforcement learning: An introduction*, MIT press, 1998.

[7] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] Thomas Stockhammer, "Dynamic adaptive streaming over http–: standards and design principles," in *Proceedings of the second ACM MMsys*. ACM, 2011, pp. 133–144.

[9] Chao Zhou, Mengbai Xiao, and Yao Liu, "Clustile: Toward minimizing bandwidth in 360-degree video streaming," in *INFOCOM, 2018 Proceedings IEEE*, 2018, pp. 1–9.

[10] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[12] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang, "A dataset for exploring user behaviors in vr spherical video streaming," in *Proceedings of the 8th ACM MMsys*. ACM, 2017, pp. 193–198.

[13] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM MMsys*. ACM, 2013, pp. 114–118.

[14] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu, "Fixation prediction for 360 video streaming in head-mounted virtual reality," in *Proceedings of the 27th NOSSDAV*. ACM, 2017, pp. 67–72.

[15] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo, "360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming," in *Proceedings of the 2017 ACM MM*. ACM, 2017, pp. 315–323.

[16] Lan Xie, Xinggong Zhang, and Zongming Guo, "Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming," in *Proceedings of the 2018 ACM MM*. ACM, 2018, pp. 1–9.

[17] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh, "Neural adaptive video streaming with pensieve," in *ACM SIGCOMM*. ACM, 2017, pp. 197–210.

[18] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han, "Rubiks: Practical 360-degree streaming for smartphones," in *MobiSys*, 2018, pp. 2–13.

[19] Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash, "Adaptive 360-degree video streaming using scalable video coding," in *Proceedings of the 2017 ACM MM*. ACM, 2017, pp. 1689–1697.

[20] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck, "An http/2-based adaptive streaming framework for 360 virtual reality videos," in *Proceedings of the 2017 ACM MM*. ACM, 2017, pp. 306–314.

[21] Xiufeng Xie and Xinyu Zhang, "Poi360: Panoramic mobile video telephony over lte cellular networks," in *Proceedings of the 13th CoNext*. ACM, 2017, pp. 336–349.

[22] Chelsea Finn, Ian Goodfellow, and Sergey Levine, "Unsupervised learning for physical interaction through video prediction," in *NIPS*, 2016, pp. 64–72.

[23] Sangdoo Yoo, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi, "Action-decision networks for visual tracking with deep reinforcement learning," in *CVPR*, 2018, pp. 1–10.

[24] Philipp Helle, Heiko Schwarz, Thomas Wiegand, and Klaus-Robert Müller, "Reinforcement learning for video encoder control in hevc," in *Systems, Signals and Image Processing (IWSSIP), 2017 International Conference on*. IEEE, 2017, pp. 1–5.

[25] Ke Yu, Chao Dong, Liang Lin, and Chen Change Loy, "Crafting a toolchain for image restoration by deep reinforcement learning," in *CVPR, Proceedings of the IEEE*, 2018, pp. 2443–2452.

Authorized licensed use limited to: Communication University of China. Downloaded on April 09,2021 at 00:45:05 UTC from IEEE Xplore. Restrictions apply.