

Plato: Learning-based Adaptive Streaming of 360-Degree Videos

Xiaolan Jiang*, Yi-Han Chiang[†], Yang Zhao[‡], and Yusheng Ji^{†*}

*Department of Informatics, The Graduate University for Advanced Studies, Japan.

[†]Information Systems Architecture Science Research Division, National Institute of Informatics, Japan.

[‡]The University of Tokyo, Japan.

E-mail: xljiang@nii.ac.jp, yhchiang@nii.ac.jp, zhao@is.s.u-tokyo.ac.jp, kei@nii.ac.jp

Abstract—Interactive applications that come along with 360-degree (or 360) videos have brought immersive experiences to users thanks to the elevated machine computability. In fact, the provision of such high quality of experience (QoE) hinges on the persistent delivery of 360 videos, potentially consuming an excessive need of network bandwidth. To prevent the delivery of entire 360 videos from adversely affecting QoE, tile-based viewport adaptive streaming that divides 360 video chunks into tiles and conveys streams with differentiated quality levels to viewport and non-viewport areas has been regarded as a promising solution. Existing works have been devoted to the design of 1) viewport prediction (VPP) to predict users' viewport orientation due to head movements, and 2) tile bitrate selection (TBS) to determine tile-based bitrates for viewport and non-viewport areas. Despite the heuristic solutions proposed by the existing works, there is lack of knowledge of whether QoE can be enhanced by learning from historical data. In this paper, we propose the system – Plato, to leverage machine learning to tile-based viewport adaptive streaming for 360 videos. In particular, Plato applies long short term memory (LSTM) model to VPP, and uses part of non-viewport areas to help resist prediction errors. In addition, Plato uses real-world traces to train a TBS agent based on reinforcement learning to determine tile bitrates for both viewport and non-viewport areas. Our simulation results show that Plato outperforms existing schemes in various QoE metrics.

Index Terms—Tile-based method, viewport adaptive streaming, 360-degree video, reinforcement learning.

I. INTRODUCTION

Recent advances in the computing resources of machines have prompted interactive applications (e.g., virtual reality) to be viable service types to users. The interaction of these types of applications makes use of head mounted displays (HMD) device to enable 360 video players to act immersively. To approach high-level quality of experience (QoE) for users, it is envisaged that the provisions of high-resolution and 360-degree (or 360 for brevity) videos are essential features. By the fact that conventional video providers (e.g., YouTube) deliver entire video contents to users could generate video traffic up to 40 Mbps [21], how 360 videos can be conveyed to users with high QoE under constrained network bandwidth determines the futurity of interactive applications.

For fear that the delivery of huge video traffic over the Internet leads to network congestion and QoE degradation, adaptive streaming that adjusts video qualities according to users' viewports (i.e., fields of view) is capable of adapting

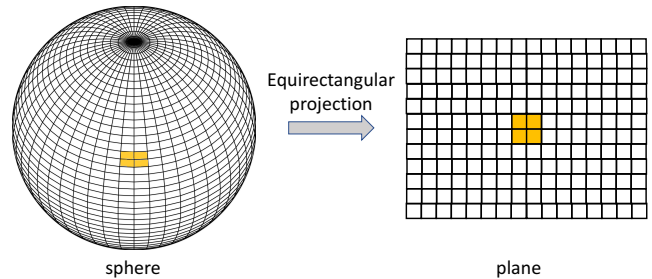


Fig. 1: The mapping of a spherical surface into a plane.

to time-varying network bandwidth. In viewport adaptive streaming, high-quality streams are delivered to viewport areas while reduced qualities are provided to non-viewport ones, since each user can only have relatively narrow fields of view of a 360 video. With the knowledge of users' viewports, the streaming qualities can then be differentiated, thereby saving video traffic from downplaying non-viewport areas [9], [14], [22].

Using tiles in viewport adaptive streaming for 360 videos offers a high level of granularity over the projected¹ plane and thus attracts much research attention in the past few years. Each tile is a spatial fragment of a video chunk, and it is typically represented by its horizontal and vertical degrees (as depicted in Fig. 1). In addition, tiles can be encoded by various quality levels according to their respective positions. For instance, we can encode tiles of viewports with high bitrates and reduce video traffic for non-viewport areas. Therefore, tile-based viewport adaptive streaming admits elastic control of the quality levels of tiles for 360 videos, thereby enhancing QoE for users.

To facilitate tile-based viewport adaptive streaming for 360 videos with high QoE, there are two key issues to be addressed: viewport prediction (VPP) and tile bitrate selection (TBS). The goal of VPP is to predict viewports for the upcoming chunks based on head movements, since the interactive applications may guide users to change their facing directions rapidly. Without high-accuracy VPP, severe QoE

¹Each 360 video surrounds a spherical surface, which can be projected onto a two-dimensional plane via a projection method [24] (e.g., equirectangular, cubemap, pyramid projections). In the projected plane, existing video coding that were designed for planar areas becomes applicable, making the design of viewport adaptive streaming for 360 videos much easier than handling spherical surfaces directly.

drops may take place. On the other hand, the mission of TBS is to determine tile bitrates, since the availability of network bandwidth is highly dynamic. Large variation among the quality levels of tiles within a viewport can also affect QoE adversely, and hence the smoothness of inter-tile quality levels is another concern. Clearly, how to tackle the VPP and TBS issues plays a vital role in the success of tile-based viewport adaptive streaming for 360 videos.

In this paper, we investigate tile-based viewport adaptive streaming for 360 videos, and propose the Plato system to leverage machine learning to address the inherent VPP and TBS issues. For VPP, Plato uses the long short term memory (LSTM) [8] based neural network model to predict users' future viewport orientation. Since the prediction errors of users' orientation can be harmful to QoE, we propose the concept of non-viewport to opportunistically cover actual areas that are slightly outside the predicted viewport areas. For TBS, Plato is equipped with a TBS agent that maps environment states (e.g., bandwidth, buffer size, tile sizes) to bitrate decisions for viewport and non-viewport areas to optimize the QoE, where the TBS agent is trained by the A3C algorithm [12] – a state-of-the-art reinforcement learning algorithm [18]. To show how Plato performs with real data sets, we employ the real-world traces of user viewport and 4G bandwidth to train the neural networks, and our simulation results demonstrate that Plato outperforms existing schemes in terms of average QoE (including bitrate in viewport areas, rebuffering time, coefficient variation and smoothness). The contribution of this paper can be summarized as follows:

- We are the first to leverage reinforcement learning for tile-based viewport adaptive streaming of 360 videos.
- We propose the system – Plato, in which VPP and TBS agents are trained to predict viewport areas and select tile bitrates, respectively.
- We use real-world traces of viewport motions and network bandwidth, and our simulation results assert that Plato achieves significant improvement on QoE.

The rest of this paper is organized as follows. In Sec. II, we overview viewport adaptive streaming for 360 videos and RL applications in regular video streaming. In Sec. III, we describe the Plato system. In Sec. IV, we show the simulation results. Finally, this paper is concluded in Sec. V.

II. RELATED WORKS

A. Tile-based Adaptive Streaming for 360 Video

Tile-based viewport adaptive 360 video streaming has been an emerging method in the research society. Many researchers have been proposing their schemes. Most recently, Graf propose three tile based schemes to compare their performance [13]. Their schemes are called full delivery basic, full delivery advanced and partial delivery. For full delivery basic, they assign the highest possible bitrate for the tiles in the viewport area, and the lowest bitrate for the other tiles. For full delivery advanced, they still try to give the viewport tiles a highest possible bitrate, give the other tiles a lower but not lowest bitrate. For partial delivery, they only download the viewport tiles at a highest bitrate, but leave all the other tiles

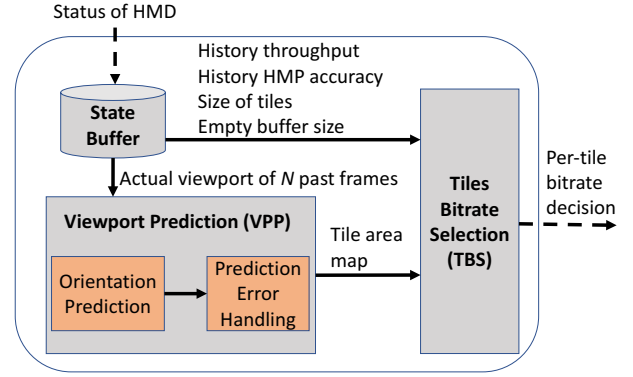


Fig. 2: The architecture of Plato.

blank. Their approaches are based on very trivial heuristic. [7] proposes a scheme which split the tiles into three areas: viewport, adjacent, and outside areas. Then try to assign the highest bitrate to the tiles at the order of viewport, adjacent, outside areas within the available bandwidth budget. However, all their schemes are based on fixed heuristics, can not get good performance under the varying network conditions.

B. Reinforcement Learning (RL) for Regular Video Streaming

A separate line of work [3]–[5], [11], [19] has proposed applying RL to adaptive video streaming. All of these schemes apply RL for regular video streaming which only need to consider the bitrate for the whole chunk of the video. However, for 360 video streaming, the problem is more complex, we need to consider not only the bandwidth variance, but also the viewport predictor accuracy. Moreover, we need to select bitrate for each tile in one chunk. This proposes more challenges for the application of RL.

III. PLATO – TOWARDS LEARNING-BASED ADAPTIVE STREAMING OF 360 VIDEOS

The architecture of Plato deployed within an HMD (worn by a 360 video player) is as depicted in Fig. 2. There are several key components: state buffer, VPP and TBS agents. The state buffer collects information from the HMD and provides them to the agents. The VPP agent predicts future viewport based on the information of historical viewports from the state buffer. Considering potential viewport prediction errors, the VPP agent divides tiles into viewport, adjacent, and outside areas. In this way, the TBS agent can select high bitrates for actual viewport areas if the viewport prediction is not accurate (e.g. user's head is moving too fast). Finally, the TBS agent selects bitrate for each tile based on the state buffer information, including historical throughput, historical viewport prediction accuracy, and current buffer size. In the following, we are going to introduce how the VPP and TBS agents work in Plato.

A. VPP – The Prediction of Viewport

In this part, we apply the LSTM based model to predict user's head orientation, and meanwhile discuss how we deal with the potential prediction errors.

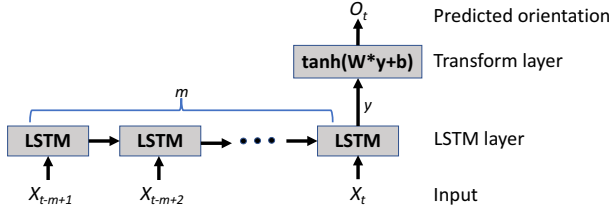


Fig. 3: LSTM based predictor.

1) *Orientation prediction*: We design a LSTM based model to predict future viewport, and illustrate its architecture in Fig. 3. The LSTM layer takes the orientations $[X_{t-m+1}, \dots, X_t]$ of m past video frames as inputs, where X_t is the array of $(yaw, pitch, roll)$ normalized to $[-1, 1]$. Then, the transform layer converts the output y of LSTM layer to normalized predicted orientation O_t . The equation of transform layer can be expressed as follows:

$$\tanh(Wy + b) = \frac{e^{Wy+b} - e^{-(Wy+b)}}{e^{Wy+b} + e^{-(Wy+b)}} \quad (1)$$

The network can predict future orientation of only one video frame now, but we can easily predict the next n orientations by making the last prediction as inputs. Then, we can calculate the viewport area according to the collected orientations in one segment length. Although the LSTM based model can predict at a relatively high accuracy, prediction errors may also take place, especially when the prediction window is long. Next, we will introduce how we handle the potential prediction error of $(yaw, pitch, roll)$.

2) *Prediction error handling*: We handle the potential prediction errors of head orientations by virtually enlarge the viewport range. In practice, the angles on a spherical surface can be defined by yaw and $pitch$ values (in degrees), both of which are discretized and mapped to pixels on a rectangular image with

$$x = \frac{width \cdot (yaw + 180)}{360}, \quad y = \frac{height \cdot (90 - pitch)}{180}. \quad (2)$$

Here, we consider the size of the equirectangular image to be $width \cdot height$, the center of the equirectangular image is at $\langle yaw = 0, pitch = 0 \rangle$, and the pitch angle increases in the upward direction.

We assume the standard vertical and horizontal FoVs of 90 and 110 degrees. We first mark the tiles covered by the actual FoV as viewport area. To handle the potential error we will mark the tiles covered by the FoVs from 90 to 90 + V degrees in vertical, and from 110 to 110 + H in horizon as the adjacent area, then all the remaining tiles will be marked as outside area. The values of H, V should depend on the accuracy of the viewport predictor and the length of the prediction interval. Fig. 4 illustrates the three areas: viewport area (yellow tiles), adjacent area (green tiles), and outside area (gray tiles). More detailed parameter settings are referred to section IV-A1. In the next part, we will introduce how we make the bitrate decisions for each area.

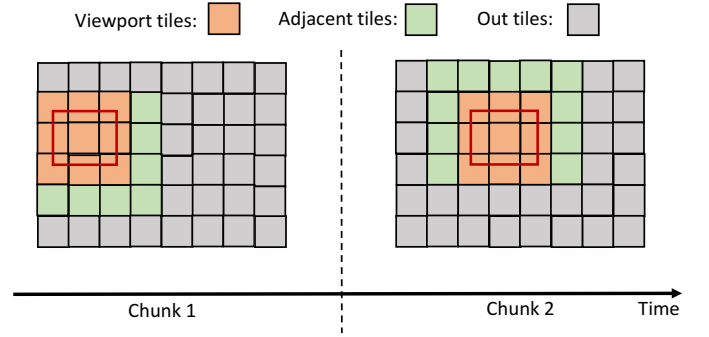


Fig. 4: Division of tile areas.

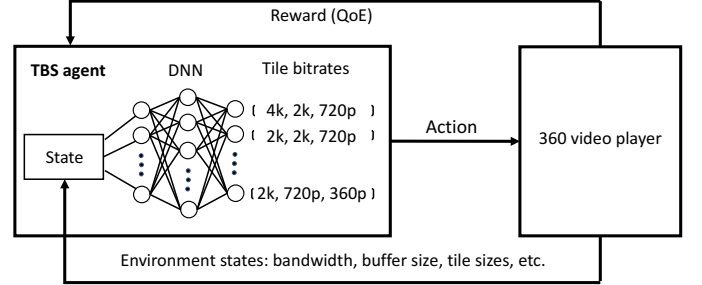


Fig. 5: The workflows of RL.

B. TBS – The Bitrate Selection of Tiles

1) *Reinforcement Learning*: In existing schemes, the TBS policy is typically generated by fixed heuristics, and our approach is to apply RL to generate it from observations. Fig. 5 illustrates how the TBS agent interacts with a 360 video player. At each time step t , the TBS agent makes an action a_t according to its observed state s_t . After the action is made, the state transitions to s_{t+1} and the TBS agent will receive a reward r_t . The objective of RL is to maximize the expected cumulative discounted reward $\mathbb{E}[\sum_{t=0}^n \gamma^t r_t]$, where $\gamma \in (0, 1]$ is a discounting factor since the values of later time steps are less influential.

In fact, the TBS policy can be interpreted as a neural network (see Fig. 5). The TBS agent first acquires a set of state information (e.g., history bandwidth, current buffer size, next video tile sizes) from a 360 video player, feeds these values to policy (neural) network, and outputs bitrate selections for each tile area of the next chunk. Then, the environment collects the resulting QoE metrics and sends them to the TBS agent as a reward. With such a reward information, the TBS agent can then train and improve the policy neural network.

2) *Training algorithm*: In this section, our aim is to show how we train the TBS agent using RL. For this, we introduce the inputs needed in the training phase, explain the neural network architecture that we use for the TBS agent, and show how A3C trains the TBS agent.

Inputs: After the download of each chunk t , the TBS agent the state s_t as its input to its neural network, where

$$s_t = (\vec{x}_t, \vec{r}_t, v\vec{p}_m, a\vec{p}_m, out_m, b_t, r_t, ratio_{vp}, ratio_{ad}, ratio_{out}). \quad (3)$$

For the past k 360 video chunks, the network throughput and

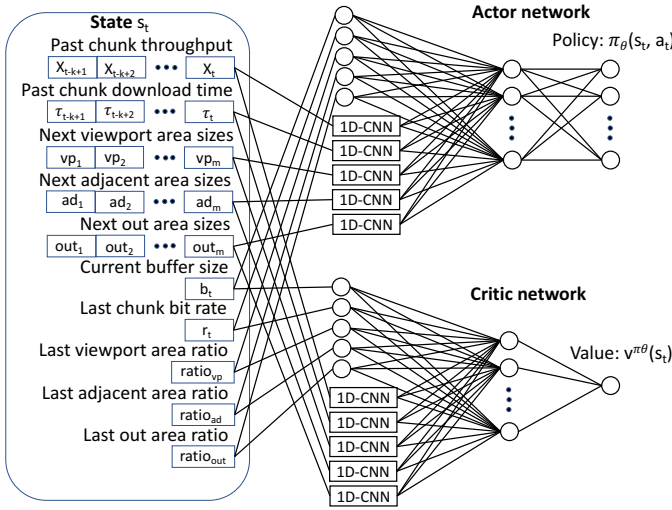


Fig. 6: An illustration of A3C algorithm.

the download time are represented by $\vec{x}_t$ and $\vec{\tau}_t$, respectively. For the next 360 video chunk, $\vec{vp}_m, \vec{ad}_m, \vec{out}_m$ are the vectors of m available sizes for viewport, adjacent, and outside areas, respectively. b_t is the current buffer size and r_t is the average bitrate of the tiles among the actual viewport in the last downloaded video chunk. $ratio_{vp}, ratio_{ad}$ and $ratio_{out}$ refer to the percentage of actual viewport tiles that are predicted as viewport, adjacent, and outside tiles, respectively.

Policy: Given the state s_t , the TBS agent takes the action a_t (i.e., bitrate selections for three tile areas) in the next video chunk. In practice, each state may have more than one action choice. Therefore, the TBS agent makes its action decision based on the policy $\pi : (s_t, a_t) \rightarrow [0, 1]$. In other words, the TBS agent in state s_t chooses the action a_t with the probability $\pi(s_t, a_t)$. Here, we use a neural network to obtain the policy $\pi_\theta(s_t, a_t)$, where θ are the parameters of the neural network. The actor network in Fig. 6 illustrates how Plato uses an NN to represent an TBS policy. The specific architecture of the neural network will be explained later. The critic network in Fig. 6 is merely used to help train the actor network, we will introduce its function in next part.

Training Algorithm: Now, we are going to introduce the state-of-the-art actor-critic method A3C [12] as our training algorithm, which involves training of two neural networks. In the following, we provide detailed explanation of how A3C works.

Whenever the TBS agent downloads a video chunk, it will receive a reward r_t from the simulated environment. Since the goal of the TBS agent is to maximize the expected cumulative discounted reward, we set the reward based on the specific QoE metrics (see Sec. IV-E) to reflect the performance of TBS policy.

To maximize the expected cumulative discounted reward, A3C first needs to estimate the policy gradient of the expected total reward by observing the history state and action pairs. Given the policy network parameters θ , the gradient can be

computed by

$$\nabla_\theta \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^n \gamma^t r_t \right] = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)], \quad (4)$$

where $A^{\pi_\theta}(s, a)$ is the advantage function, indicating the difference between the expected reward $v^{\pi_\theta}(s)$ for actions based on policy π_θ and the expected total reward when we deterministically choose an action a in state s .

In practice, the agent will empirically estimate an unbiased $A^{\pi_\theta}(s_t, a_t)$ as $A(s_t, a_t)$ by sampling a trajectory of tile bitrate selection. The actor network parameters can then be updated with the learning rate α as follows:

$$\theta = \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t). \quad (5)$$

The output of the critic network is used to estimate the expected total reward $v^{\pi_\theta}(s)$, and A3C uses $v^{\pi_\theta}(s)$ to estimate the advantage $A(s_t, a_t)$. A3C updates the parameters of the critic network as follows:

$$\theta_v = \theta_v - \alpha_v \sum (r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v))^2, \quad (6)$$

where $V^{\pi_\theta}(\cdot; \theta_v)$ is the output by the critic network, α_v is the learning rate for the critic, r_t is the reward at time step t . The pseudocode of A3C can be found in [12]. More detailed parameter settings are referred to Sec. IV-A2.

IV. PERFORMANCE EVALUATION

To evaluate the QoE performance of Plato, we develop a 360 video player simulator with real traces. We first introduce the parameter settings of Plato and real-world traces we used in the simulator and then give a detailed explanation of the 360 video player simulator. Next, we introduce the comparison schemes and QoE metrics. Finally, we show the simulation results and give some analysis.

A. Plato Parameter Settings

We implement both LSTM based predictor and RL algorithm with pytorch [15], which is a popular open source machine learning framework. The detailed parameter settings in each part are as follows.

1) *VPP*: For LSTM based viewport predictor, we use one layer LSTM with hidden size of 128. For the error handling part, we set the values of H and V to 30 and 60, respectively.

2) *TBS*: For the actor network (see Fig. 6), Plato passes $k = 8$ past throughputs, chunk download times to two identical 1D convolutional layers (CNN) with 128 filters, each of size 3 with stride 1. The sizes of next viewport, adjacent, outside areas are passed to other three 1D-CNNs with the same shape. The current buffer size, last chunk bitrate, the percentage of actual viewport tiles that are predicted as viewport, adjacent, and outside tiles are passed into five linear layers each with 128 neurons. The outputs from these layers are then passed to a linear layer with N neurons, where N is the size of action space – the number of meta selections for the three tile areas. The critic network uses the same NN architecture, but its final output is only one neuron. During training, we set the discount factor $\gamma = 0.99$. The learning rates for the actor and the critic are both set to 0.0001.

B. Datasets

1) *Video Preparation*: A 360 video with the duration of 237 sec [6] is considered. The raw 3840×2016 video is extracted from the original clip and re-encoded using the kvazaar encoder [10], which is an open source software for H.265. The video is available in a 1-sec segment version, tiled as 12x6. Bitrate is in the range of [40, 16, 8, 5, 2.5, 1] Mbps according to the recommended settings from Youtube [2].

2) *Viewport Motion*: The dataset [6] includes data collected from 59 users watching video 70-sec 360 videos on the Razer OSVR HDK2 HMD. The selected videos span a wide range of 360 contents for which different viewers involvement (e.g., navigation patterns) could be expected.

The log files in the datasets record the unit quaternion² (q_0, q_1, q_2, q_3) of the HMD device in each timestamp. The unit quaternion (q_0, q_1, q_2, q_3) represents the rotations of an object in 3D space. For each object such as the HMD device in this coordinated system, the rotation vector ($yaw, pitch, roll$) can be calculated from the unit quaternion as follows³:

$$\begin{bmatrix} yaw \\ pitch \\ roll \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_3q_0 + q_1q_2), 1 - 2(q_0q_0 + q_1q_1)) \\ \text{asin}(2(q_2q_0 - q_3q_1)) \\ \text{atan2}(2(q_3q_2 + q_0q_1), 1 - 2(q_1q_1 + q_2q_2)) \end{bmatrix}. \quad (7)$$

We preprocess all the log files according to (7), and then randomly select 80% users' processed log files for all videos as the training dataset, and the remaining users' log files for all videos as the testing dataset. It's important to note that, the tests of VPP and TBS use the different viewport motion datasets.

3) *Network Throughput*: The test bandwidth dataset we use contains the 4G networks traces along several routes in and around the city of Ghent, Belgium, from 2015-12-16 to 2016-02-04 [20]. All bandwidth logs are recorded based on the type of transportation: car, train, tram, bus, bicycle and foot. Given the high throughput observed (in some cases up to 95 Mbps) and the occasional switching to 3G, the logs have a relatively limited duration ranging from 166 to 758 sec. A total of 40 logs were collected, covering 5 hours of active monitoring.

[17] contains logs from TCP streaming session in Telenor's 3G/HSDPA mobile wireless network in Norway. In each test, video streams are downloaded at maximum speed. We reformatted the 3G dataset to have a 4G bandwidth by scaling the original bandwidth log file by a random number between [10, 25], and we use the new cooked dataset as the training dataset.

C. 360 Video Player Simulator Design

In this paper, we design a 360 video player simulator for training the TBS agent by RL and testing all the other comparison schemes. In fact, emulating the standard 360 video streaming is very slow, because the algorithm needs to wait until the chunks of a video is really downloaded before updating its neural network model. Therefore, we use

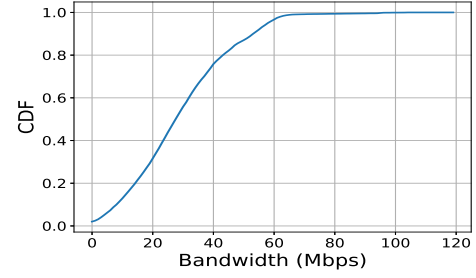


Fig. 7: CDF of Belgium Bandwidth (test set)

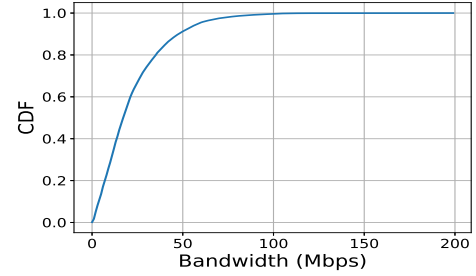


Fig. 8: CDF of Scaled Norway bandwidth (train set)

a simulator to run the training algorithm alternatively. Next, we introduce the details of the simulator design.

We assume the client has the same behavior as HTTP/2, which can request all the tiles in a video segment at one time. We set the packet payload, RTT, and buffer size to 95%, 80 msec, 3 sec, respectively. In the simulator, we try to faithfully model the dynamics of 360 video streaming with real client applications. The simulator records the playback buffer usage information while streaming the 360 video. Before downloading, we will predict the future viewport with our proposed VPP agent using the real viewport motion traces. After the bitrate for each tile is decided, the simulator begins downloading the chunk. For each downloaded chunk, the simulator calculates the download time that is solely based on the chunk's size and the real-world network bandwidth traces. The simulator will then reduce the equal time video from playback buffer. The rebuffering events are carefully recorded while the buffer occupancy status changes, i.e., situations where download time is too long. In situations where the playback buffer is full, the simulator pauses requests for 500 msec before retrying⁴. After each chunk download, the simulator will calculate the real viewport from the input viewport motion traces, calculate the average bitrate of the tiles in the real viewport, and then pass the bitrate information and several other state observations to the RL agent for processing: the current buffer size, rebuffering time, chunk download time, size of the next tiles (at all bitrates), and the number of the remaining chunks in the video. Using this chunk-level simulator, Plato can "experience" 10 hours of video downloads in only 1 minute.

D. Comparison Metrics

1) Viewport Prediction:

⁴This is the default request retry rate used by DASH players.

²https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

³<https://stackoverflow.com/questions/5782658/extracting-yaw-from-a-quaternion>

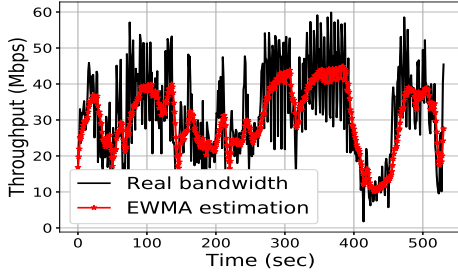


Fig. 9: sample of 4G bandwidth

- Linear Regression (LR) [23] uses least square method to predict the user's orientation by using history samples.
- AVG [16] computes the average value of the past m samples as the prediction result.

2) Tiles Bitrate Selection:

- MM [13]: video chunks are split into tiles, and all tiles are logically divided into three areas: viewport, adjacent, and outside areas. According to their algorithm, the lowest quality is assigned to all the tiles of the video first. This initial allocation guarantees that all the tiles of the video are streamed to the user. Then, the available bandwidth budget is computed as the difference between the available bandwidth and the total bitrate allocated to the tiles. Next, the highest possible quality is assigned to the tiles, given the bandwidth budget, starting from viewport tiles, then adjacent tiles, finally outside tiles.
- FDA [7]: video chunks are split into tiles, and all tiles are logically divided into two parts: viewport and outside. The strategy is similar to MM, the lowest quality is assigned to all the tiles of the video first, then update the budget. Next assign the highest possible bitrate to tiles, starting from viewport tiles.

3) *Standard throughput prediction method*: Exponentially weighted moving average (EWMA) is used in Dash player and HLS player. EWMA is an exponentially weighted mean of previous throughput, as the following equation:

$$Estimate_{t+1} = \alpha Real_t + (1 - \alpha) Estimate_t \quad (8)$$

We reimplement the EWMA predictor as the industry Dash player [1] with the same parameter settings. In Fig. 9, the red line shows the prediction performance of EWMA on one log file in the test dataset. We use EWMA for the comparison schemes MM and FDA.

E. QoE Definition

For DASH-based adaptive streaming services, the main QoE influence factors can be described as follows [23]:

- 1) Average bitrate: the average bitrate of all tiles in the users' real viewport.
- 2) Rebuffering time: the playback continuity which calculates the percentage of the duration of stall over the total video streaming.
- 3) Smoothness: average bitrates difference between the adjacent time steps.

- 4) CV: viewport spatial quality variance. If the quality of the content is not smooth, it will decrease the users' QoE, and we calculate this value according to the coefficient of variation (CV) of quality of content in the viewport.

Following [11], we define a QoE metric function of the above four metrics for 360 video streaming as:

$$QoE = \sum_{n=1}^N q(R_n) - \mu \sum_{n=1}^N T_n - \beta \sum_{n=1}^N cv - \sum_{n=1}^{N-1} |q(R_{n+1}) - q(R_n)| \quad (9)$$

For a video with N chunks, R_n is the average bitrate in the viewport of chunk n , and $q(R_n)$ maps the bitrate to the quality perceived by the users. T_n means the rebuffering time, while the third term penalizes video quality changes to favor smoothness. The final term also penalize the bitrate difference among tiles in the viewport to favor a consistent quality in a viewport. We set μ as 43, and β as 5.3. Considering giving the low bitrate with a low score and a high bitrate with a high score, we define the $q(R_n)$ as follows:

$$q(R_n) = \begin{cases} 1, & \text{for } 0 < R_n \leq 1.5 \\ 2, & \text{for } 1.5 < R_n \leq 3 \\ 3, & \text{for } 3 < R_n \leq 6 \\ 6, & \text{for } 6 < R_n \leq 10 \\ 9, & \text{for } 10 < R_n \leq 20 \\ 12, & \text{for } 20 < R_n \leq 40 \end{cases}$$

F. Result Analysis

1) *Viewport Prediction*: We compare viewport prediction accuracy with two other schemes: linear regression (LR) and AVG. Fig. 10 shows the CDF of *yaw*'s prediction error from 1 to 3-sec window length. We can find that our proposed LSTM based predictor-VPP agent can outperform other schemes in all time windows length. Fig. 11 shows the CDF of *yaw*'s prediction error from 1 to 3-sec window length. In this scenario, we find that our proposed scheme can still outperform the other two schemes while the prediction window is less than 3 sec, and achieve similar performance on AVG when prediction window is 3 sec. And the AVG model can achieve better performance than LR.

From the prediction error of *yaw* and *pitch* we can find that, users usually prefer to move a lot in horizon, but remain almost stable in the vertical direction. So we conclude that, while the user is moving fast, our proposed can have better performance than all the other schemes.

Fig. 12 shows the error correction performance of our scheme. From the figure, we can find that the viewport (*vp*) accuracy decrease from more than 90% to less than 80%. But the sum of the *vp* accuracy and adjacent accuracy (*ad*) is always higher than 90%. In this way, we can select higher bitrate for the viewport and adjacent tiles to achieve higher average bitrate in the real viewport.

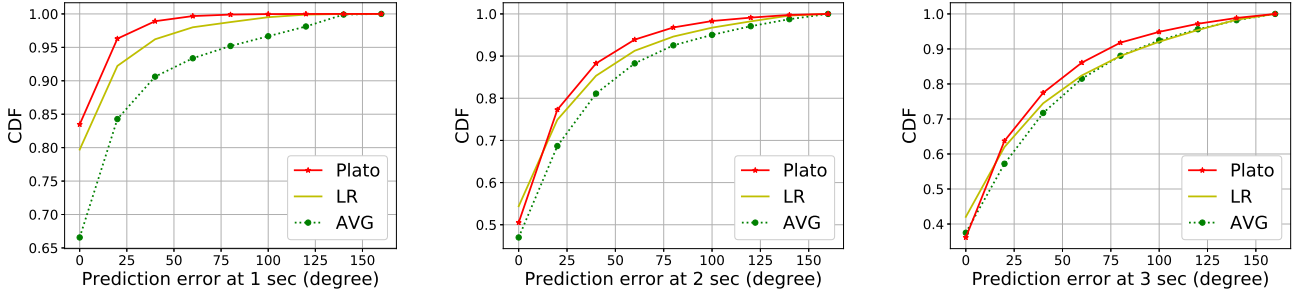
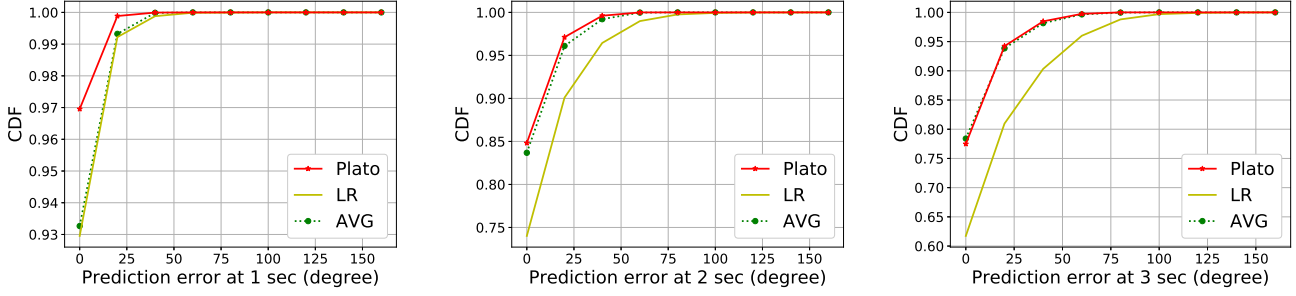
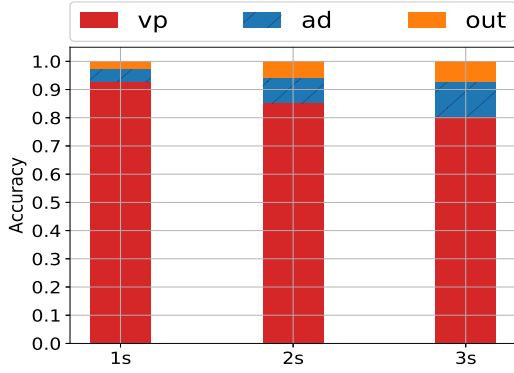
Fig. 10: CDF of *yaw* prediction error in 1, 2, 3 sec window lengthFig. 11: CDF of *pitch* prediction error in 1, 2, 3 sec window length

Fig. 12: Tile prediction accuracy of vp, ad, out areas

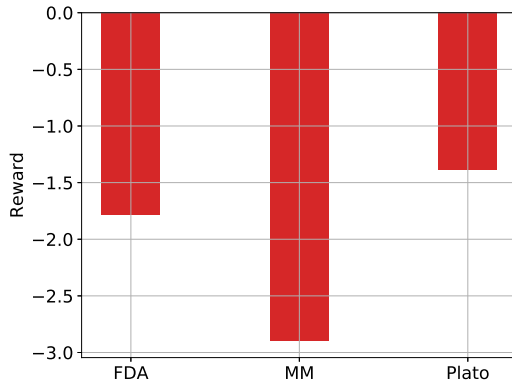


Fig. 13: Average QoE

on the QoE metric defined in IV-E. Since the penalty for rebuffering time, and CV is relatively large, the value the reward of all schemes are negative values. But we can also find that with considering all the four metrics together, our TBS agent can achieve the highest reward.

To better understand the QoE gains obtained by Plato, we analyzed Plato's performance on the individual terms in our general QoE definition. Specifically, Fig. 14 compares Plato with the comparison schemes in terms of the the average bitrate in the actual viewport, the penalty from rebuffering, the coefficient of variance (CV) and smoothness. From the figure, we find that Plato can achieve the best average bitrate among all schemes, with less rebuffering time penalty than other two schemes. The reason is that our definition of QoE function favors high quality video, assigning the highest utility to the top three bitrates available for our test video. Since the penalty of rebuffering time is also large, Plato also does not receives much rebuffering time. Here, we can say that Plato can achieve much higher video quality and similar rebuffering time than other scheme. Plato also outperform MM in term of CV and smoothness, the reason is that by learning from a trace, the TBS agent can have some foresight to achieve less smoothness and low CV. FDA is better than Plato in terms of CV, because FDA only needs to select bitrate for two areas, but Plato need to select bitrate for three areas, this makes Plato have higher CV penalty.

It's important to note that the tile-based 360 video streaming is for achieving higher quality for the user under limited bandwidth. The results show that The learning based scheme can achieve much higher performance gain in terms of video quality, with less or similar rebuffering penalty, smoothness penalty, and CV penalty.

2) *Tiles Bitrate Selection*: Fig. 13 shows the average QoE that each scheme achieves on our entire test datasets based

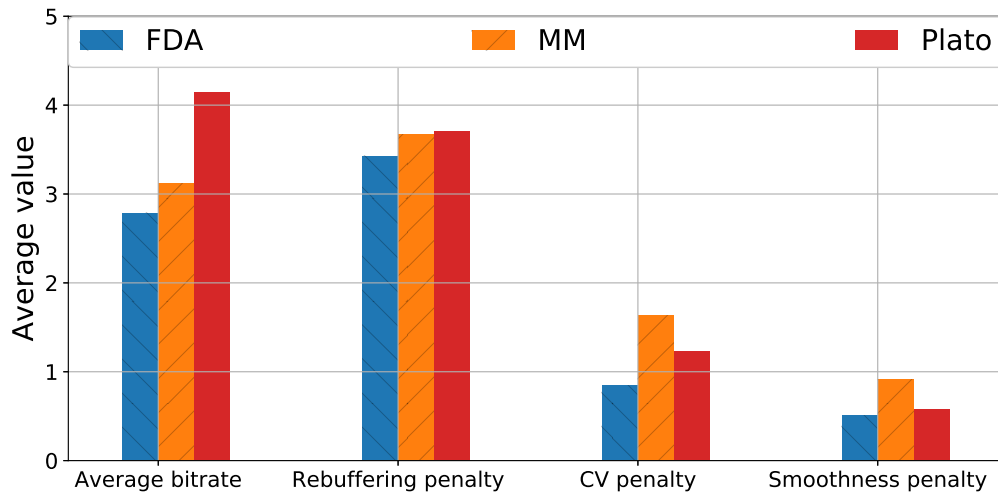


Fig. 14: Comparing Plato with existing algorithms by analyzing their performance on the individual components in the general QoE definition

V. CONCLUSION AND FUTURE WORK

In practice, delivering entire 360 videos over the Internet could be prohibitive due to the limited available network bandwidth. To resolve this problem, we propose the Plato system to facilitate tile-based viewport adaptive streaming for 360 videos. In Plato, LSTM based neural network model is applied to predict users' future viewport orientation. To be resilient to potential prediction errors, Plato delivers streams to part of non-viewport areas. In addition, Plato applies the A3C algorithm for training the TBS agent that maps environment states to bitrate decisions for both viewport and non-viewport areas. To compare Plato with existing schemes, we run simulations based on real traces of user viewport and 4G bandwidth usage, and our results demonstrate that Plato outperforms the comparison schemes in terms of various QoE metrics.

REFERENCES

- [1] "Dash player source code," <https://github.com/Dash-Industry-Forum/dash.js/blob/development/src/streaming/rules/ThroughputHistory.js>.
- [2] "Recommended upload encoding settings, youtube," <https://support.google.com/youtube/answer/1722171?hl=en>.
- [3] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard, "Online learning adaptation strategy for dash clients," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 8.
- [4] M. Claeys, S. Latré, J. Famaey, T. Wu, V. Leekwijck, D. Turck *et al.*, "Design of a q-learning-based client quality selection algorithm for http adaptive video streaming," in *Proceedings of the 2013 Workshop on Adaptive and Learning Agents (ALA)*, Saint Paul (Minn.), USA, 2013, pp. 30–37.
- [5] M. Claeys, S. Latré, J. Famaey, T. Wu, W. Van Leekwijck, and F. De Turck, "Design and optimisation of a (fa) q-learning-based http adaptive streaming client," *Connection Science*, vol. 26, no. 1, pp. 25–43, 2014.
- [6] X. Corbillon, F. De Simone, and G. Simon, "360-degree video head movement dataset," in *ACM MMSys 2017*, no. EPFL-CONF-227447, 2017.
- [7] M. Graf, C. Timmerer, and C. Mueller, "Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation," in *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 2017, pp. 261–271.
- [8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [9] M. Hosseini and V. Swaminathan, "Adaptive 360 vr video streaming: Divide and conquer," in *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 107–110.
- [10] kvazaar, "<https://github.com/ultravideo/kvazaar>," <https://github.com/ultravideo/kvazaar>.
- [11] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *ACM SIGCOMM 2017*, 2017.
- [12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.
- [13] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck, "An http/2-based adaptive streaming framework for 360 virtual reality videos," in *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 2017, pp. 306–314.
- [14] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. D. Turck, "An http/2-based adaptive streaming framework for 360 virtual reality videos," in *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017*, 2017, pp. 306–314.
- [15] Pytorch, "<https://pytorch.org/>."
- [16] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 2016, pp. 1–6.
- [17] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3g networks: analysis and applications," in *Proceedings of the 4th ACM Multimedia Systems Conference*. ACM, 2013, pp. 114–118.
- [18] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.
- [19] J. van der Hooft, S. Petrangeli, M. Claeys, J. Famaey, and F. De Turck, "A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 131–138.
- [20] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alfacc, T. Bostoen, and F. De Turck, "Http/2-based adaptive streaming of hev video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.
- [21] H. Virtual Reality/Augmented Reality White Paper, "<http://www-file.huawei.com/-/media/corporate/pdf/ilab/vr-ar-en.pdf>."
- [22] M. Xiao, C. Zhou, Y. Liu, and S. Chen, "Optile: Toward optimal tiling in 360-degree video streaming," in *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 2017, pp. 708–716.
- [23] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo, "360probdash: Improving qoe of 360 video streaming using tile-based HTTP adaptive streaming," in *Proceedings of the 2017 ACM on Multimedia Conference, MM, 2017*.
- [24] M. Yu, H. Lakshman, and B. Girod, "A framework to evaluate omnidirectional video coding schemes," in *Mixed and Augmented Reality (ISMAR), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 31–36.