TBRA: Tiling and Bitrate Adaptation for Mobile 360-Degree Video Streaming

Lei Zhang
Shenzhen University
leizhang@szu.edu.cn

Yanyan Suo
Shenzhen University
suoyanyan2019@email.szu.edu.cn

Ximing Wu
Shenzhen University
wuximing2019@email.szu.edu.cn

Feng Wang
The University of Mississippi
fwang@cs.olemiss.edu

Yuchi Chen
Simon Fraser University
yuchi_chen@sfu.ca

Laizhong Cui*
Shenzhen University
cuilz@szu.edu.cn

Jiangchuan Liu
Simon Fraser University
jcliu@cs.sfu.ca

Zhong Ming
Shenzhen University
mingz@szu.edu.cn

ABSTRACT

Tile-based approach is widely adopted in adaptive 360° video streaming systems. Existing QoE-driven streaming approaches usually obtain the tile selection and adjust the bitrate based on the viewport prediction with a fixed tiling, which fail to consider the unstable prediction performance. However, varying the tiling of the video can produce different number of tiles with different sizes, and thus can have distinct impacts on error tolerance for viewport prediction and on decoding complexity for resource-constrained mobile client. In this work, we introduce adaptive tiling into the conventional bitrate adaptation for mobile 360° video streaming. We first analyze the impacts of tilings on tile selection and decoding time, which verify the benefit of tiling adaptation in various practical aspects. We then formulate the QoE optimization problem for adaptive tiling and bitrate streaming and discuss the design details of our adaptation algorithm, which can adapt to the performance of viewport prediction and the decoding capabilities of mobile clients in addition to the conventional influencing factors. Finally, the superiority of our proposed approach compared with the state-of-the-art methods is evaluated through extensive trace-driven simulations.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → *Application layer protocols*; • **Computer systems organization** → Real-time systems.

KEYWORDS

360° video; viewport prediction; tile-based streaming; rate adaptation; mobile

*Corresponding author

MM '21, October 20–24, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8651-7/21/10…$15.00
https://doi.org/10.1145/3474085.3475590

ACM Reference Format:
Lei Zhang, Yanyan Suo, Ximing Wu, Feng Wang, Yuchi Chen, Laizhong Cui, Jiangchuan Liu, and Zhong Ming. 2021. TBRA: Tiling and Bitrate Adaptation for Mobile 360-Degree Video Streaming. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21), October 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475590

1 INTRODUCTION

360° video, which provides panoramic views to give users an immersive experience, is now becoming popular on major video sharing websites and social media channels. Streaming 360° videos is challenging. First, due to the panoramic nature, 360° videos are much larger (4x to 6x) than conventional videos under the same perceived quality [19]. Compared to a regular video, the transmission of a 360° video consumes much higher bandwidth, which can be scarce especially in wireless and mobile networks. Second, streaming 360° videos introduces higher computation and energy overhead for mobile end devices [13], which have limited CPU, GPU, storage, and battery capacities. Commodity 360° video streaming systems (e.g., YouTube and Oculus) stream entire 360° frames to clients [37], which directly employ the conventional approach for regular videos. However, streaming all the pixels of 360° videos is wasteful. When watching a 360° video, the user views a limited portion of the whole spherical image, which is often determined by the user's Field-of-View (FoV) (also referred to as viewport). To this end, tile-based approach [30] is proposed for 360° video streaming, which divides each panoramic frame into smaller-sized non-overlapping rectangular regions called *tiles*. As each tile is independently decodable, the clients can only request the tiles that are predicted to be in the user viewport. In general, tile-based streaming exploits the trade off between transmission efficiency and user experience. On one hand, only a subset of tiles are transmitted, which can significantly reduce bandwidth consumption; on the other hand, this subset may not necessarily cover the actual user viewport, and thus the user experience is greatly affected by the quality of tile selection.

Tile-based viewport-adaptive streaming approaches [8, 9, 19] that integrate tile selection with rate adaptation have been recently studied and adopted in the literature. Generally, the tiles are selected by assuming the knowledge of user viewport during the video

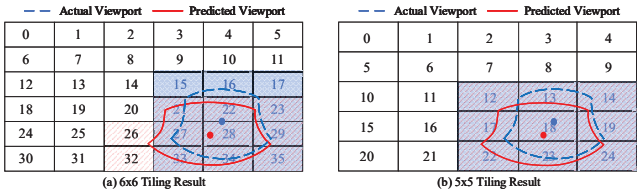


Figure 1: Tile Selections with Different Tilings

playback, which is usually obtained from predicting future user behaviors (e.g., head/gaze movement) based on historical data [6]. The existing works mostly target to select appropriate tiles and their bitrates given a fixed tiling of the video, which makes the tile selection results sensitive to the prediction accuracy. Rather than selecting tiles with a fixed tiling, in this work we take different tiling options into account for the adaptation of 360° video streaming.

Since predicting user behavior is difficult by nature, the performance of viewport prediction algorithm can vary dramatically. In the case of low prediction accuracy, tile selection with a fixed tiling can result in severe user experience degradation. Consider the example in Figure 1, which shows the tile selection with the same prediction error (the same predicted and actual viewports) using two different tiling settings (6x6 and 5x5). In the case of 6x6 tiling, three tiles in the actual viewport (tile 15, 16, and 17) are missing in the predicted viewport and will be given low priorities or even skipped for download in tile-based streaming, which will cause video quality drop or rebuffering. Moreover, as tile 26 and 27 are in the predicted viewport but not in the actual viewport, they will be given high priorities for download, which may cause unnecessary bandwidth waste. However, if 5x5 tiling is applied, the tiles in the actual viewport are exactly the same as the tiles selected based on the prediction. This example shows the deficiency of fixed tiling in viewport-adaptive streaming. Another impact of tiling 360° videos is that tiling introduces extra decoding complexity, which can no longer be ignored even for modern mobile hardware models [9]. Splitting frames into tiles reduces video encoding efficiency, as the referencing information between and within frames is reduced. Meanwhile, more tiles imply more decoding tasks on mobile clients, which are equipped with limited capabilities and may have difficulties in supporting real-time tile-based decoding for high-resolution 360° videos [5, 9]. Therefore, we advocate that flexible tiling setting should be adopted during the streaming.

In this paper, besides the common factors such as bandwidth fluctuation and buffer occupancy, we propose to adapt both tiling and bitrate to the varying viewport prediction performance and the limited mobile client's decoding capability. We first analyze typical viewport prediction algorithms using real-world traces and identify their performance instability. We then discuss the impacts of different tilings on tile selection and decoding complexity, which justifies that adaptive tiling should be integrated into the conventional bitrate adaptation framework for mobile 360° video streaming. We show that adaptive tiling can accommodate viewport prediction errors and help ensure the quality of tile selection. To support real-time decoding on mobile devices, we build the analytical model for decoding time, which will be used for choosing appropriate tiling and bitrate given limited computation resources. Further, we formulate the optimization problem in our featured problem context and discuss the design of our adaptation algorithm in detail. Finally, we evaluate our practical and efficient solution and demonstrate its superior performance through extensive trace-driven simulations.

2 BACKGROUND AND RELATED WORK

2.1 Adaptive Streaming for 360° Video

Adaptive streaming has long been the research interest for conventional videos [4, 17, 24, 35]. Since the emergence of 360° videos, how to stream them effectively and efficiently has attracted considerable attention from researches. Existing studies focus on various aspects such as adaptation algorithm design, practical system implementation, and user perceptive experience optimization. Xie et al. [31] adopted a data-driven approach to predict future viewport and make download decisions, which estimates the viewing probability by statistically studying the similarity of user fixations for multiple viewing events on the same video. As deep reinforcement learning (DRL) has been successfully applied to develop the representative adaptive streaming framework, Pensieve [17], this technique has also been adopted in 360° video streaming. Zhang et al. [36] proposed a DRL-based framework, which leverages deep learning to predict the bandwidth and the viewports, and allocates the rates for the tiles by an actor-critic algorithm. From the system perspective, Flare [19] and Rubiks [9] are two representatives of the state-of-the-art 360° video streaming systems. Flare integrates and implements all the necessary components for a real-world streaming system, while Rubiks develops a tile-based layered streaming framework that splits 360° video chunks into spatial tiles and temporal layers. As the perceptive experience varies significantly in 360° video streaming [21], Guan et al. [8] built a visual quality model that captures the impact of three 360° video-specific factors and used it to strike a balance between the perceived quality and the encoding efficiency. To better support 360° video processing in real-time, Liu et al. proposed to adaptively offload expensive transformation operations to GPUs according to the video contents [26] and adjust frame rate based on the user behaviors [12].

2.2 Viewport Prediction Algorithm

As 360° videos require intensive user interactions, user behaviors and user perceptive experience have been the key concerns. Multiple datasets [7, 14, 27]) have been collected for user behaviors during video watching. Viewport prediction is a must-be component for viewport-adaptive 360° video streaming systems [8, 9, 19, 31]. For a viewport prediction scheme, the key design is to choose a proper machine learning (ML) algorithm as the basis of the prediction. In general, there are two types of choices: (1) simple but potentially less accurate algorithms (e.g., regression-based) and (2) more accurate but complicated algorithms (e.g., neural network-based).

Type-2 designs (e.g., deep/reinforcement learning) are usually favored by the CV community to predict user interests in different objects based on the video contents [10, 15, 28, 32]. Regardless of its accuracy, this content-based approach introduces substantial computation workloads and significant training time. Another downside of Type-2 algorithms is that the learning-based approach requires large-scale datasets and long training periods. This approach's life cycle (from data collection on clients, to model training on server,

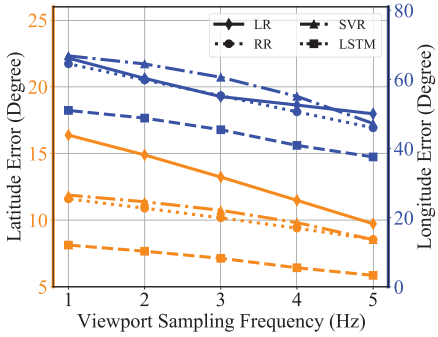


Figure 2: LonError *vs.* Sampling Frequency

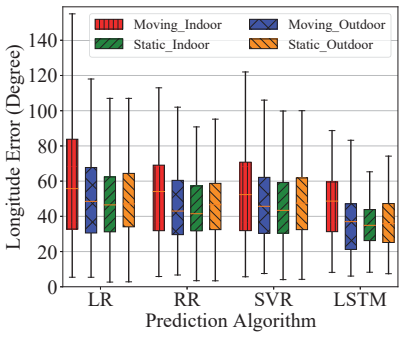


Figure 3: Longitude Error *vs.* Video Type

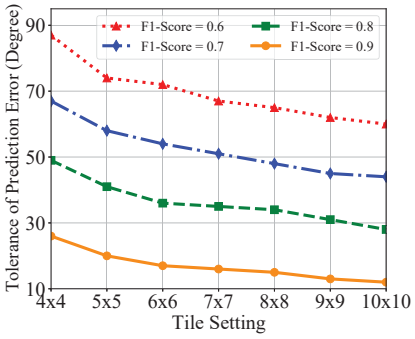


Figure 4: Prediction Error Tolerance *vs.* Tiling

to model update on clients) implies huge implementation overhead and makes it less practical for real-time systems.

Given the above reasons, Type-1 designs are usually preferred for developing user-based prediction approaches [9, 18, 19], which takes historical user behaviors to predict future behaviors. The rationale of the preference for lightweight ML algorithms is twofold: first, the prediction algorithm should run fast enough on mobile clients to meet the underlying real-time requirements; second, given the short history window and the short prediction window in the streaming scenario, lightweight ML algorithms can have comparable performance to complicated algorithms. In particular, He et al. [9] compared the linear regression with a neural network that has 3 hidden layers with 30 neurons in each layer, and observed negligible performance difference for short prediction windows.

3 MOTIVATION AND ANALYSIS

Mobile 360° video streaming faces two challenges: predicting the right set of tiles for download and decoding multiple tiles fast enough. We next present our observations on tiling's impacts on these two aspects (tile selection and decoding complexity), which motivate us to integrate tiling adaptation into 360° video streaming.

3.1 Impact of Tiling on Tile Selection

In tile-based 360° video streaming, the tiles for download are selected based on the predicted viewport given the specific tiling setting. We will show in this subsection: first, the performance of viewport prediction is unstable due to various factors; second, the adaptation of tiling setting is able to accommodate the prediction inaccuracy. We implement four lightweight (Type 1) ML algorithms (the reasons are as previously discussed) to predict the longitude and the latitude coordinates of user viewport center (user heading direction): Linear Regression (LR), Ridge Regression (RR), Supported Vector Regression (SVR), Long Short-Term Memory (LSTM). We set the history window and the prediction window to be 2 seconds and 1 second respectively, and set the width and the height of the viewport to be 100° and 90° respectively. We use 6x6 as the default tiling. The head movement traces are adopted from an open dataset [7], which contains the user behaviors for 57 users watching 19 videos.

3.1.1 Inaccuracy of Viewport Prediction. After the prediction algorithm is designed and the ML model is trained, the viewport prediction performance can still be easily affected. Previous studies observe that user head movements occur mostly in the horizontal direction and much less frequently in the vertical direction [7, 23], which implies that predicting longitude coordinates is the major challenge. Our result is consistent with this observation. As shown in Figure 2, compared to latitude prediction error, longitude prediction error dominates. As all the four prediction algorithms show reasonably good performance in predicting latitude, we only present the analysis for longitude prediction hereafter.

As the user behaviors are captured on the client, one practical factor that can affect the prediction performance is the capability of the mobile end device. Given the hardware diversity of commodity devices, mobile clients are likely to have limited sensing and processing capabilities and cannot support high frequency of viewport sampling. We vary the viewport sampling frequency for viewport prediction, which directly affects how much information can be used for prediction. As all the four prediction algorithms show reasonably good performance in predicting latitude, we only present the analysis for longitude prediction hereafter. We can see from Figure 2 that, as viewport sampling frequency drops from 5Hz to 1Hz, the longitude prediction errors of LR, RR, SVR, and LSTM increase significantly by 32.2%, 40.3%, 41.4%, and 35.8%, respectively.

We next check the prediction performance for different video type. We classify the videos based on the shooting environment (indoor or outdoor) and the camera status (static or moving). As shown in Figure 3, compared to indoor/outdoor environment, whether the camera is static or moving has a greater impact on prediction accuracy. For all the four algorithms, the average prediction errors for the camera-moving videos are at least 9% larger than those for the camera-static videos. Moreover, the variances of the prediction errors for the camera-moving videos are significantly larger, which implies the pre-trained ML models do not have stable performance for this video type.

Remark. *After the prediction algorithm designing stage, the viewport prediction performance is still vulnerable due to various factors.*

3.1.2 Tolerating Prediction Error by Tiling. As a tile is selected as long as any portion of it intersects with the predicted viewport, tiling can naturally accommodate the inaccuracy of viewport prediction: if the prediction error does not cross a tile, it causes no change on the tile selection result. Therefore, different tiling settings will produce tiles with different sizes, which can tolerate prediction errors to different degrees. Increasing the size of each tile (while reducing the number of tiles in each frame) can further help absorb larger errors from the viewport prediction algorithms.

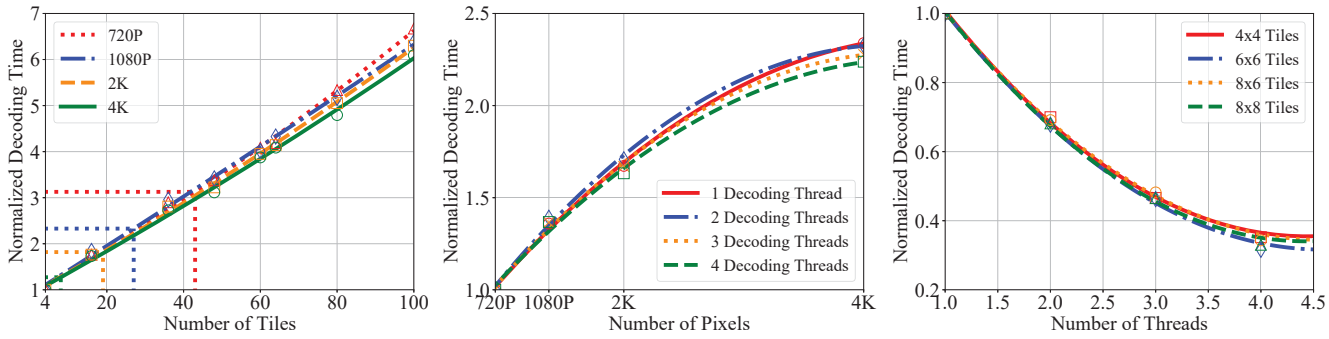


Figure 5: Decoding Time *vs.* Number of Tiles **Figure 6: Decoding Time *vs.* Video Resolution** **Figure 7: Decoding Time *vs.* Thread Number**

To justify our discussion, we vary the tiling setting from 4x4 to 10x10 and apply different prediction errors to check the largest error that a tiling setting can accommodate while keeping the same quality of the resulting tile selection. We use F_1 score to assess the tile selection quality, which is defined as $F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$. Figure 4 plots the tolerances for the prediction error to achieve different F_1 scores. As expected, tilings with larger sizes of tiles have higher prediction error tolerances. The result suggests that, with inaccurate viewport predictions, it is feasible to change the frame tiling to achieve a better tile selection.

Remark. *With inaccurate viewport predictions, tiling adaptation can be applied to obtain better tile selection results.*

3.2 Impact of Tiling on Decoding Complexity

Although today's mobile hardware is rapidly developing, decoding and processing high-resolution 360° video contents in real-time is unfortunately still challenging. Previous research efforts [9, 19, 20] have confirmed that the decoding overhead for tiled videos cannot be ignored for mobile devices. Even for the recent work [22] that claims achieving low-latency ultra-high resolution 360° video decoding, it actually avoids the problem by downscaling the effective resolution for different viewport sizes and screen sizes.

Tiling the 360° frames changes not only the number of decoding tasks but also the video encoding efficiency. As the number of tiles increases, the size of each tile decreases, which implies lower encoding efficiency because the reference area (the redundant information that can be compressed) between and within frames is reduced, e.g., motion vectors that reference the best block matches can be cut by tile boundaries. To investigate the impact of tiling on decoding complexity at the mobile client side, we conduct real-world measurements. We use FFmpeg [1] and the open-source HEVC encoder, Kvazaar [2], to prepare the test videos and encode them into HEVC streams with motion constrained tiling, which ensures each tile is independently decodable. We use Huawei Mate 30 as the test device with Kirin 990 CPU and 8GB RAM. We implement a parallel decoding scheme, which uses multiple decoding instances to decode tiles asynchronously and a decoding buffer to cache the portions of the decoded frame. To construct the decoding instances, we call *MediaCodec API* [3] for Android to initialize the hardware codec *OMX.hisi.video.decoder.hevc* on our test device.

We consider three factors that influence the decoding complexity directly. First, the decoding time is affected by the number of decoded tiles. Second, the video resolution implying the total number of decoded pixels reflects the amount of computation. Third, the decoding time further depends on how much resource is utilized for decoding. We keep one factor fixed and check the influences of the other two. We plot the normalized decoding time versus the number of tiles, the video resolution, and the number of decoding threads in Figure 5, 6, 7, respectively. The results are normalized over the corresponding baseline in each case (with 2x2 tiling/720p/1 thread). We use the dotted line in Figure 5 to denote the playback time length for the given resolution and the corresponding tile count. It indicates the real-time decoding limit, exceeding which the decoding cannot keep up with the playback. It is not surprising that the decoding time increases as the number of the decoded tiles gets larger or the video resolution gets higher, and decreases as the number of the decoding threads grows.

Another observation from the figures is that the three key influencing factors (number of tiles, video resolution, number of threads) are empirically mutually independent, as the curves in each of these three figures are nearly identical. We use $F_n(x)$ ($F_r(x)$ or $F_c(x)$) to denote the ratio between the decoding time when tile number (resolution or thread number) is x and that of the baseline case. We take $F_n(x)$, $F_r(x)$, $F_c(x)$) as three multipliers for different factors and build the analytical model of as $D = D_0 \cdot F_n(x_1) \cdot F_r(x_2) \cdot F_c(x_3)$ to calculate the overall decoding time for decoding a resolution x_2 video of x_1 tiles with x_3 threads, where D_0 is the decoding time of the baseline case. This model will be utilized in later sections to help make tiling and bitrate decisions. It is worth noting that, although using more decoding threads can reduce the decoding time, the gain can be marginal when the resource limit is hit (for our test device the thread number should be no more than 4), which suggests us to decide the appropriate thread number according to the current available computation resources on the mobile device.

Remark. *The analytical model of decoding time can be built by quantifying the impacts of tile number, video resolution, and decoding thread number.*

4 DESIGN OF TBRA

In this section, we present the design of our tiling and bitrate adaptation (TBRA) for mobile 360° video streaming. TBRA attempts to select appropriate tiling and bitrate by considering the viewport

prediction performance and the decoding capability, in addition to the conventional factors such as bandwidth and buffer.

Let $S = \{s_1, s_2, ...\}$ be the set of the tiling options for a 360° video, and for tiling s_i let $|s_i|$ denote its number of tiles. Without loss of generality, we assume $|s_i| < |s_j|$ if $i < j$. Given a tiling s, let $b_{i,j}$ denote the bitrate for tile j in chunk i, where $i \leq$ total number of chunks and $j \leq |s|$. Our goal is to determine appropriate tiling s and bitrate $b_{i,j}$ for each tile in the streamed video.

4.1 Tiling Adaptation

4.1.1 Idea of Adaptation. To accommodate the viewport prediction error, existing works usually naively expand the predicted viewport. We also take this general direction but in a calibrated way by taking advantage of the underlying tile-based transmission, which is essentially the tradeoff between user experience and transmission efficiency. As discussed earlier, different tiling options can tolerate viewport prediction errors to different degrees. When the frame is split into more tiles, each tile has smaller size, and thus the viewport can be better fitted but more difficult to predict. In an extreme tiling case where the frame is cut into single-pixel tiles, the viewport can be perfectly fitted by the tiles and transmitted with the least unnecessary pixels, which however has to be predicted into pixel level. On the contrary, if the tiles get bigger, the viewport can only be fitted in a coarse-grain, which thus tolerates higher prediction errors but waste more bandwidth. The extreme case is letting the whole frame be one tile, which turns into streaming all the pixels regardless of the predicted viewport. Therefore, our idea of tiling adaptation is to increase the size of tiles when the viewport prediction has poor performance, and vice versa. The key question to answer is which tiling setting is the optimal choice to strike the best balance between user experience and transmission efficiency.

4.1.2 Tiling Selection. During a video chunk's playback, the client usually makes several viewport predictions and selects the tiles based on the area covering the viewport trace. We extend this prediction area by $\bar{r_d}$ in direction $d \in \{\text{left, right, up, down}\}$, which is the running average of the prediction error in this direction e_d for the last n seconds and is updated as $\bar{r_d} = (1-\alpha) \cdot \bar{r_d} + \alpha \cdot e_d$. This prediction area extension is further used for tile selection, which is dynamic and affected by the recent prediction accuracy.

The next step is to check across different tiling settings to find the best tradeoff between user experience and transmission efficiency. For each tilling setting, we compare the tile selection based on the expanded prediction area with that based on the viewport ground-truth. We calculate two ratios as the metrics for user experience and transmission efficiency, respectively:

$$\text{Miss Ratio} = \frac{\text{\# of missed pixels in expanded prediction}}{\text{\# of viewed pixels}},$$

$$\text{Waste Ratio} = \frac{\text{\# of unnecessary pixels in expanded prediction}}{\text{\# of viewed pixels}}.$$

The tradeoff is clear between the two ratios as observed in Figure 8. We evaluate the goodness of a tiling setting using the penalty calculated as a weighted sum of these two ratios,

$$\text{Tiling } i_{\text{penalty}} = \beta \cdot \text{ Miss Ratio} + |1/\cos(\phi_i)| \cdot \text{ Waste Ratio}, \quad (1)$$

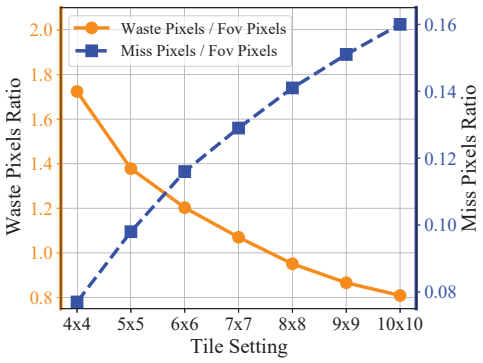


Figure 8: Tradeoff between Miss and Waste

where ϕ_i is the latitude of viewport i's center. It suggests that the weight of Waste Ratio changes as the viewport moves vertically. Most 360° video systems adopt Equirectangular projection (ERP), which maps meridians to vertical straight lines of constant spacing and circles of latitude to horizontal straight lines of constant spacing. ERP is known to have distortions, which is exaggerated along with the vertical direction. Hence, we design the weight to be automatically tuned to the level of distortion. After checking all the tiling settings, we select the one with the smallest penalty.

4.2 Rate Adaptation

After selecting the tiling setting and classifying the tiles accordingly, the next step is allocating bitrate. We next formulate and solve the optimization problem for bitrate adaptation.

4.2.1 Video Quality. Let $w_{i,j}$ denote the weight of tile j during chunk i's playback. Although in our current scheme $w_{i,j}$ is 0/1 depending on whether the tile is in the predicted viewport, we still leave $w_{i,j}$ as an open setting to keep our model generic, as in certain cases the tiles may be ranked into more tiers with different weights. Ideally, $w_{i,j}$ is determined based on the tile class at a given frame, and thus there may be multiple weights for a tile in one chunk, in which case we set $w_{i,j}$ as the maximal weight over all the predictions. Let $q(b_{i,j})$ be the non-decreasing mapping function between the tile bitrate selection $b_{i,j}$ and the user perceived quality. The video quality level of chuck i is defined as

$$Q_i^{(1)} = \sum_{j=1}^{n} w_{i,j} q(b_{i,j}). \quad (2)$$

As recent studies[8, 12] have show that the user perceived quality should be the quality metric for 360° videos. We adopt the latest subjective video quality model [16] as $q(\cdot)$,

$$\text{subjective PSNR: } q_i = PSNR_i \cdot [M(v_i)]^{\gamma} [R(v_i)]^{\delta}, \quad (3)$$

where $M(v_i)$ and $R(v_i)$ are the detection threshold and retinal slip rate, respectively, v_i is the viewport moving speed for chunk i, $\gamma = 0.172$, and $\delta = -0.267$.

4.2.2 Quality Churn. Significant quality change between the consecutive video chunks can hurt user QoE. We define the quality churn as the change between two consecutive chunks

$$Q_i^{(2)} = |Q_i^{(1)} - Q_{i-1}^{(1)}|, \ i \in [2, m]. \quad (4)$$

4.2.3 Rebuffering Time. We denote C_i as the predicted throughput for downloading chunk i, and B_i as the buffer occupancy when the client starts to download chunk i. Note that $B_1 = B_{default}$, where $B_{default}$ is the default buffer level filled at the start-up stage. Downloading chunk i needs time $\sum_{j=1}^{n} b_{i,j}/C_i$. Let L be the length of each chunk. The buffer status should be updated every round. The buffer occupancy for the next chunk $i+1$ can be calculated as

$$B_{i+1} = \max(B_i - \sum_{j=1}^{n} b_{i,j}/C_i, 0) + L. \quad (5)$$

We have the rebuffering time during the download of chunk i as

$$Q_i^{(3)} = \max(\sum_{j=1}^{n} b_{i,j}/C_i - B_i, 0) + t_{miss}. \quad (6)$$

The first part is the rebuffering time of the case that the download takes too long and the buffer runs out of video to play. The second term t_{miss} indicates the time for downloading the missing tiles that are previously assigned with zero bitrate but actually watched in this chunk. There are two implicit constraints related to rebuffering for our problem. We will illustrate the details in the later subsection.

4.2.4 Optimization Objective. The overall QoE of chunk i can be defined as a weighted sum of the aforementioned QoE metrics

$$Q_i = pQ_i^{(1)} - qQ_i^{(2)} - rQ_i^{(3)}. \quad (7)$$

The first term is positive and the latter two terms are negative since we want to have the maximal video quality, the minimal quality change, and the least rebuffering time. Traditionally, we use the mean QoE for all chunks as the optimization objective. However, the perfect future information over the entire horizon from chunk 1 to m is difficult to obtain in practice. To handle the hardness of predicting the long-term throughput and user behavior, we apply the MPC-based framework [35] and optimize the QoE of multiple chunks over a limited horizon. Fortunately, the streaming scenario is friendly for collecting related information for short-term analysis or prediction. The objective function can be formulated as

$$\max_{b_{i,j}, i\in[t,t+k-1], j\in[1,n]} \sum_{i=t}^{t+k-1} Q_i, \quad (8)$$

where k denotes the size of the optimization window.

Since viewport prediction performance and network conditions are easy to monitor on short timescales, the QoE optimization can be done using the predicted information in window $[t, t+k-1]$. We then move the horizon forward to $[t+1, t+k]$, update the information in the new optimization window, and run the QoE optimization for the next chunk, so on and so forth. The benefit to use the MPC-based formulation is that each optimization problem is practically solvable due to the limited size of the problem instance.

4.2.5 Efficient Solution. The proposed formulation by nature suits for online execution, as the QoE optimization can be periodically solved through exhaustive search because of the small instance size for short windows. Since the optimization should be invoked at a high frequency, it is still challenging due to the large search space. To support real-time optimization, we need to efficiently prune the search space. To this end, we identify the following important constraints and opportunities for boosting the solution's efficiency.

Algorithm 1: TBRA: Tiling and BitRate Adaptation

```
Initialization: default buffer level B_default, optimization
  window size k, bitrate set, the objective weights p, q, and r
for i ← 1 to m do
    if client is in startup phase then
        given a fine-grain tiling, download all the tiles at
        highest bitrate;
        measure the throughput C̄_i;
        start playback when buffer reaches B_default;
    else
        C_i = ThroughputPrediction(C̄_[1,i−1]);
        update r̄_d from the recent prediction error;
        get tiling s̄ with the smallest penalty in Eq. 1;
        set w_{i,j} for s̄ based on the prediction;
        get the feasible bitrate range by comparing the
        decoding time with the playback duration;
        b_{i,j} = arg max Σ_{t=i}^{i+k−1} Q_t;
        download the tiles at corresponding b_{i,j};
    end
end
```

First, the decoding time should be constrained. Besides buffer draining out under low bandwidth, another cause of rebuffering can be that decoding cannot catch up with playback. Therefore, our first constraint is that the decoding time should be less than the playback length. Given the available computation resource on the mobile device, we can obtain the largest number of parallel decoding threads it can support. Further, based on the analytical model of decoding time, taking that built in Section 3.2 as an example, due to the monotony of decoding complexity versus resolution, we can find the maximum quality level that the device is capable to decode in time, which limits the bitrate selection to a bounded search space.

Second, the bitrate selection should consider the constraint of throughput: $\sum_{j=1}^{n} b_{i,j} \le LC_i$. In other words, we do not actively drain the buffer and leave it to handle the throughput fluctuation. The only exception is when the buffer hits its maximal limit B_{max}.

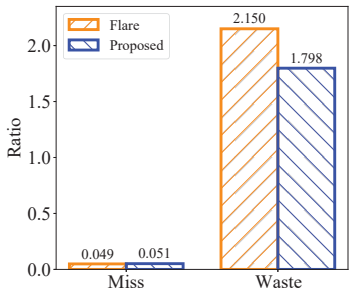
Third, the bitrate selection should consider the class of the tiles. The bitrate of a tile should not be lower than that of any other tile in the same chunk with a lower weight: $b_{i,j} \ge b_{i,j'} \forall w_{i,j} > w_{i,j'}$.

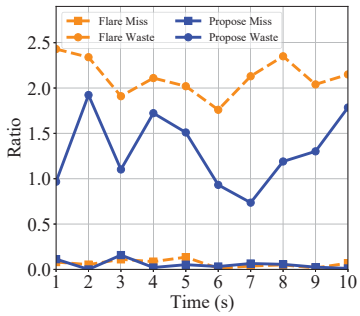
Fourth, all tiles belonging to the same class should have the bitrate selected at the same quality level. This constraint allows us to perform rate adaptation on a per-class basis instead of on a per-tile basis, which significantly reduces the search space.

Finally, in the cases where the throughput and the user behavior are stable over the optimization window (i.e., the throughput does not vary much and the number of tiles in each class stays the same), all the chunks in the same window should have the same result.

4.3 TBRA Workflow

The complete TBRA workflow for our MPC-based optimization is summarized in Algorithm 1. As a high-level overview, the algorithm has two stages. At the startup stage, the client downloads all the tiles to fill up the buffer and measures the effective throughput. Once the buffer reaches the default setting, it iteratively executes





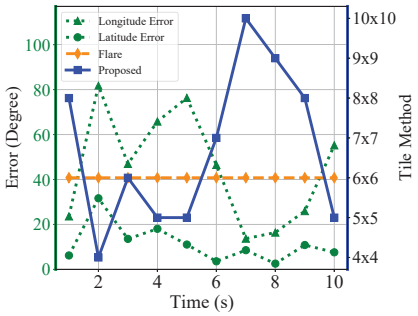


Figure 9: Tiling adaptation performance **Figure 10: Miss and Waste in a watch event** **Figure 11: Tiling setting in a watch event**

three steps (i.e., throughput prediction, tiling and bitrate selection, and download) in each optimization window. For throughput prediction, we rely on the existing approaches (e.g. the harmonic-mean predictor [11], the moving average predictor [34]). For tiling and bitrate selection, we first decide the tiling setting based on the recent viewport prediction performance and assign the tile weights accordingly. We then identify the feasible range for bitrate selection. Given the decoding time constraint, the thread number, and the tile number, we can find the highest resolution that can be decoded in time without causing a playback stall. The rate adaptation problem can be solved in a reduced and limited solution space.

Admittedly, as multiple versions of encoded chunks with different tilings are needed, our approach introduces certain transcoding and storage overhead on the server side, which should be further examined. However, we want to emphasize that our work's focus is the mobile client side. Running our algorithm only introduces negligible overhead on the client. The tiling adaptation problem is observed to have a nice feature that the global optimal is usually the local optimal, and thus the computation can be greatly reduced. The rate adaptation problem can also be solved efficiently using the optimized MPC-based workflow.

5 PERFORMANCE EVALUATION

5.1 Methodology

5.1.1 Data Traces. To emulate throughput fluctuations, we replay the bandwidth traces from a 4G/LTE dataset captured during mobility [25]. Since the bandwidth in some traces is extremely high, we linearly scale each trace to emulate difference network conditions. The 360° video viewing traces are from the open dataset [7] analyzed in Section 3.1. We replay all the 360° video watching events and for each video watching event randomly select 10 bandwidth traces from the 4G/LTE dataset.

5.1.2 Algorithms for Comparison. We implement our proposed TBRA algorithm as well as other two algorithms: (1) Flare [19]: is an enhanced viewport-adaptive scheme that first selects consumed tiles according to the viewport prediction and then include more tiles considering the prediction inaccuracy. (2) Knapsack [33]: is a tile bitrate adaptation method, which considers the bitrate per tile as well as the buffer-level per tile, integrating them into the profit function of a multiclass knapsack problem. Both of them do not adopt tiling adaptation, which only try to transmit more tiles or in different bitrates with a fixed tiling. (3) OpTile [29]: like TBRA, is also an adaptive tiling method but focuses on encoding the tiles with the optimal size and shape to achieve low storage and bandwidth cost, which works more like a video preparation scheme rather than a streaming algorithm.

5.1.3 Settings and Metrics. Empirically, we set the size of the optimization window in our MPC-based workflow to be 2 seconds. We set the size of viewport to be 100° wide and 90° high and set $B_{\text{default}} = 3s$ and $B_{max} = 10s$. The viewport sampling frequency is set to be 5Hz. For the weights in the QoE objective, we set $p = 0.1$, $q = 0.2$ and $r = 0.4$, which is similar to other commonly used settings in the existing work [9, 19, 35]. By testing the performance of different settings, we empirically set the parameters for tiling adaptation as $\alpha = 0.9$ and $\beta = 50$. For throughput prediction, we adopt a moving average predictor [34] based on the past 5 samples. We consider five quality levels with different bitrate settings: (1) 480p (2.5Mbps), (2) 720p (5Mbps), (3) 1080p (8Mbps), (4) 2K (16Mbps), and (5) 4K (40Mbps) according to the recommendation of upload encoding bitrates from YouTube[1]. The default setting for Flare and Knapsack is 6x6 tiling. To evaluate the performance, we examine subjective PSNR, quality level change, rebuffering time during the video playback, and total bandwidth consumption.

5.2 Result and Analysis

5.2.1 Performance Gain from Tiling Adaptation. We first validate our design of tiling adaptation scheme, by comparing its performance to that of Flare without the rate adaptation part. We use Flare for comparison and omit Knapsack and OpTile here because Flare, although uses fixed tiling, can adaptively expand the prediction area based on the prediction accuracy. Knapsack is fully a fixed tiling approach and OpTile works before the watch event with the knowledge of viewport's distribution, both of which cannot adapt to viewport prediction error. Figure 9 shows the average missed and wasted pixels for running TBRA's and Flare's tile selection process for the whole dataset, which shows the clear performance improvement from our tiling adaptation and justifies our idea of design. Flare transfers 0.352 viewport-size (19.6%) more unnecessary pixels than our approach to avoid 0.002 viewport-size (4.0%) less pixels missing, which shows that we trade negligible user experience degradation for considerable transmission efficiency increase. Note that we can always tune the weight β in Eq. 1 to strike another balance point to put more weight on user experience. We further

[1]https://support.google.com/youtube/answer/1722171

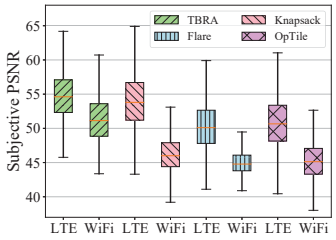


Figure 12: Subjective quality with varying bandwidth (LTE > WiFi)

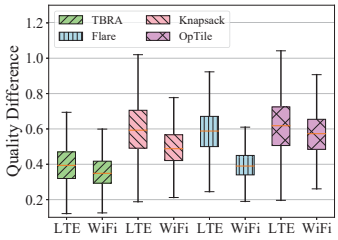


Figure 13: Quality change with varying bandwidth (LTE > WiFi)

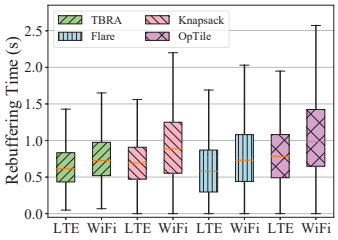


Figure 14: Rebuffering time with varying bandwidth (LTE > WiFi)

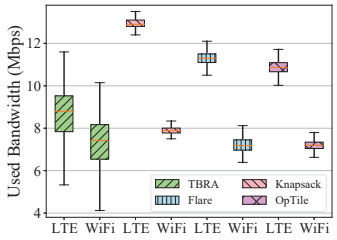


Figure 15: Total bandwidth consumption with varying bandwidth (LTE > WiFi)

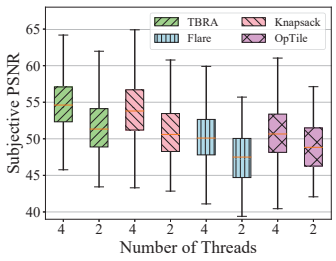


Figure 16: Subjective quality with different decoding capabilities

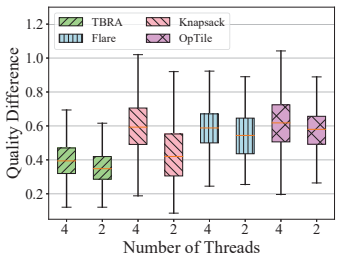


Figure 17: Quality change with different decoding capabilities

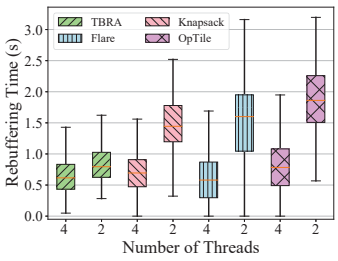


Figure 18: Rebuffering time with different decoding capabilities

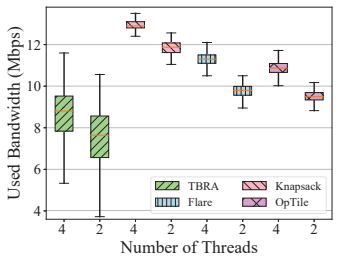


Figure 19: Total bandwidth consumption with different decoding capabilities

check a representative video watching event (User 19 watching Video 17) in Figure 10 and 11. Figure 10 confirms that TBRA can significantly reduce the bandwidth waste for unnecessary pixels while keep the same level of missed pixels as Flare. In Figure 11, we can clearly observe how TBRA timely reacts to the prediction error increase and decrease.

5.2.2 Performance with Varying Bandwidth. We next check the performance of TBRA, Flare, Knapsack, and OpTile (with 4 decoding threads) using bandwidth traces for different networks. As WiFi networks are simulated to have connection interferences, its average bandwidth is lower than that of 4G. We plot the average subjective PSNR, quality change, rebuffering time, and total bandwidth consumption in Figure 12, 13, 14, and 15, respectively. As we can see from the figures, TBRA significantly outperforms Flare, Knapsack, and OpTile in most cases. TBRA achieves 14.8%, 11.6%, 13.8% (9.1%,1.7%, 8.1%) higher subjective PSNR and 9.1%, 27.7%, 37.4% (31.2%, 33.7%, 35.5%) less quality change than Flare, Knapsack, and OpTile, respectively under WiFi (LTE) connections. The only exceptions are that (1) Flare experiences less rebuffering with LTE; (2) Flare and OpTile use less bandwidth with WiFi. Another observation is that varying network conditions have less impacts on TBRA's performance, while the other three methods are more unstable under two types of network connections.

5.2.3 Performance with Different Decoding Capabilities. We further vary the number of the decoding threads to simulate different decoding capabilities for mobile devices (using the LTE network traces). The corresponding results are shown in Figure 16, 17, 18, and 19. For all four approaches, reducing the number of decoding threads leads to clear performance degradation except for quality change and bandwidth consumption. This is because when there is insufficient decoding capability, more tiles with lower resolutions are likely to be selected. Similar to the previous results, TBRA has the best performance in almost all the cases for the four metrics. Moreover, reducing decoding capability has clearly the least impacts (5.8% higher subjective PSNR, 10.8% lower quality change, 35.7% more rebuffering time, 12.2% less bandwidth consumption) on TBRA. It implies that our proposed approach can better adapt to the varying amount of computation resources on the mobile client.

6 CONCLUSION

In this paper, we proposed the design of tiling and bitrate adaptation for mobile 360° video streaming. We first identified that, for mobile 360° video streaming, the viewport prediction algorithms have unstable performance and the mobile clients may not have enough capabilities for real-time decoding. To cope with these two challenges, we analyzed the impact of tiling setting and justified the benefit of tiling adaptation. We further presented the detailed design of our TBRA algorithm, including tiling adaptation scheme, the QoE optimization problem formulation, and the MPC-based solution. The results from the trace-driven simulations demonstrated that our proposed TBRA outperforms the three representative state-of-the-art algorithms.

ACKNOWLEDGMENTS

This work was partially supported by National Key R&D Program of China under Grant No.2018YFB1800302 and No.2018YFB1800805, National Natural Science Foundation of China under Grant No.61902257 and No.61772345, Natural Science Foundation of Guangdong Province under Grant No.2021A1515012633, Shenzhen Science and Technology Program under Grant No. RCYX20200714114645048, No. JCYJ20190808142207420 and No. GJHZ20190822095416463.

REFERENCES

[1] Accessed: 2021. FFmpeg. https://ffmpeg.org/.

[2] Accessed: 2021. Kvazaar endocer. https://github.com/ultravideo/kvazaar.

[3] Accessed: 2021. MediaCodec API. https://developer.android.com/reference/android/media/MediaCodec.

[4] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 44–58.

[5] Ghufran Baig, Jian He, Mubashir Adnan Qureshi, Lili Qiu, Guohai Chen, Peng Chen, and Yinliang Hu. 2019. Jigsaw: Robust live 4k video streaming. In *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*. ACM, 1–16.

[6] Lovish Chopra, Sarthak Chakraborty, Abhijit Mondal, and Sandip Chakraborty. 2021. PARIMA: Viewport Adaptive 360-Degree Video Streaming. In *Proceedings of the Web Conference 2021*. 2379–2391.

[7] Erwan J David, Jesús Gutiérrez, Antoine Coutrot, Matthieu Perreira Da Silva, and Patrick Le Callet. 2018. A dataset of head and eye movements for 360° videos. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 432–437.

[8] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 394–407.

[9] Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. 2018. Rubiks: Practical 360-Degree Streaming for Smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 482–494.

[10] Yifei Huang, Minjie Cai, Zhenqiang Li, and Yoichi Sato. 2018. Predicting gaze in egocentric video by learning task-dependent attention transition. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 754–769.

[11] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2014. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (ToN)* 22, 1 (2014), 326–340.

[12] Nan Jiang, Yao Liu, Tian Guo, Wenyao Xu, Viswanathan Swaminathan, Lisong Xu, and Sheng Wei. 2020. QuRate: power-efficient mobile immersive video streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*. ACM, 99–111.

[13] Nan Jiang, Viswanathan Swaminathan, and Sheng Wei. 2017. Power Evaluation of 360 VR Video Streaming on Head Mounted Display Devices. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 55–60.

[14] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. 360 video viewing dataset in head-mounted virtual reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 211–216.

[15] Haoran Lv, Qin Yang, Chenglin Li, Wenrui Dai, Junni Zou, and Hongkai Xiong. 2020. SalGCN: Saliency Prediction for 360-Degree Images Based on Spherical Graph Convolutional Networks. In *Proceedings of the 28th ACM International Conference on Multimedia*. ACM, 682–690.

[16] Saeed Mahmoudpour and Peter Schelkens. 2020. Omnidirectional Video Quality Index Accounting for Judder. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 1 (2020), 61–75.

[17] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[18] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. An http/2-based adaptive streaming framework for 360 virtual reality videos. In *Proceedings of the 25th ACM International Conference on Multimedia*. ACM, 306–314.

[19] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. ACM, 99–114.

[20] Giuseppe Ribezzo, Luca De Cicco, Vittorio Palmisano, and Saverio Mascolo. 2020. TAPAS-360: A Tool for the Design and Experimental Evaluation of 360 Video Streaming Systems. In *Proceedings of the 28th ACM International Conference on Multimedia*. ACM, 4477–4480.

[21] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R Das, and Gregory Zelinsky. 2016. Design and evaluation of a foveated video streaming service for commodity client devices. In *Proceedings of the 7th ACM Multimedia Systems Conference*. ACM, 1–11.

[22] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. 2019. Mobile VR on edge cloud: a latency-driven design. In *Proceedings of the 10th ACM Multimedia Systems Conference*. ACM, 222–231.

[23] Vincent Sitzmann, Ana Serrano, Amy Pavel, Maneesh Agrawala, Diego Gutierrez, Belen Masia, and Gordon Wetzstein. 2018. Saliency in VR: How do people explore virtual environments? *IEEE Transactions on Visualization and Computer Graphics* 24, 4 (2018), 1633–1642.

[24] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 272–285.

[25] Jeroen Van Der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. 2016. HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180.

[26] Shuoqian Wang, Xiaoyang Zhang, Mengbai Xiao, Kenneth Chiu, and Yao Liu. 2020. SphericRTC: A System for Content-Adaptive Real-Time 360-Degree Video Communication. In *Proceedings of the 28th ACM International Conference on Multimedia*. ACM, 3595–3603.

[27] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A dataset for exploring user behaviors in VR spherical video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 193–198.

[28] Chenglei Wu, Ruixiao Zhang, Zhi Wang, and Lifeng Sun. 2020. A Spherical Convolution Approach for Learning Long Term Viewport Prediction in 360 Immersive Video. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 14003–14040.

[29] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. 2017. Optile: Toward optimal tiling in 360-degree video streaming. In *Proceedings of the 25th ACM International Conference on Multimedia*. ACM, 708–716.

[30] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Proceedings of the 25th ACM International Conference on Multimedia*. ACM, 315–323.

[31] Lan Xie, Xinggong Zhang, and Zongming Guo. 2018. Cls: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming. In *Proceedings of the 26th ACM International Conference on Multimedia*. ACM, 564–572.

[32] Yanyu Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, Jingyi Yu, and Shenghua Gao. 2018. Gaze prediction in dynamic 360 immersive videos. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5333–5342.

[33] Praveen Kumar Yadav and Wei Tsang Ooi. 2020. Tile rate allocation for 360-degree tiled adaptive video streaming. In *Proceedings of the 28th ACM International Conference on Multimedia*. ACM, 3724–3733.

[34] Jun Yao, Salil S Kanhere, and Mahbub Hassan. 2008. An empirical study of bandwidth predictability in mobile computing. In *Proceedings of the third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. ACM, 11–18.

[35] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 325–338.

[36] Yuanxing Zhang, Pengyu Zhao, Kaigui Bian, Yunxin Liu, Lingyang Song, and Xiaoming Li. 2019. DRL360: 360-degree video streaming with deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1252–1260.

[37] Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 27–37.