Energy-Efficient and QoE-Aware 360-Degree Video Streaming on Mobile Devices

Xianda Chen and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {xuc23, gxc27}@psu.edu

***Abstract*—Tile-based streaming has been widely used in 360° video streaming to adapt to varying network conditions. However, downloading and processing many small tiles consumes a large amount of energy on mobile devices. To address this issue, we propose techniques to encode video by considering the viewing popularity, where the tiles requested by users of similar interests are encoded as a large tile (called Ptile). When encoding Ptiles, we propose to further save energy by reducing the insignificant frames in each video segment, i.e., reducing the frame rate to save energy while satisfying some QoE constraint. Based on real video traces, we model the impact of video features (i.e., video bitrate, frame rate) and user behavior (i.e., view switching) on QoE, and model the impact of video features on power consumption. Based on the QoE model and the power model, we formulate the energy-efficient and QoE-aware 360° video streaming problem as an optimization problem, and propose a control theory based algorithm to solve it. Through extensive evaluations based on real traces, we demonstrate that the proposed algorithm can significantly reduce the energy consumption (49.7%) and improve the QoE (7.4%).**

I. Introduction

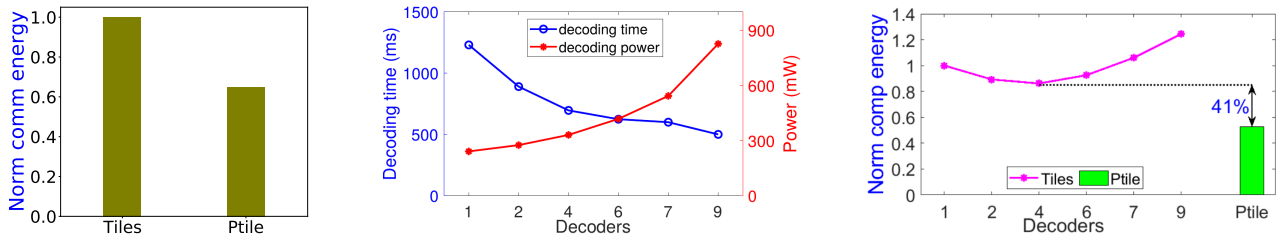
360° video, a.k.a. panoramic or immersive video, becomes increasingly popular on video platforms such as Facebook and YouTube [1, 2]. 360° video is created by capturing scenes in all directions (i.e., panoramic views), and it provides immersive experience to users by allowing them to freely change the viewing orientation during video playbacks. As 360° video is much larger than conventional video [3, 4], streaming 360° video over wireless networks with limited bandwidth usually results in very poor Quality of Experience (QoE). To address this problem, tile-based streaming [5] has been widely used in 360° video streaming. In this approach, the video is divided into a sequence of video segments. Each segment has the same time duration, and it is further divided into independently decodable *tiles*. For each tile, multiple versions are constructed by encoding the video content at different quality levels. Then, based on the network condition, the client can dynamically select tiles covering user's viewing area and determine the right quality level for each tile, such that the video can be successfully downloaded and played back to maintain good QoE.

Although tile-based 360° video streaming can improve QoE with limited wireless bandwidth, downloading and processing the tiles on mobile devices result in high energy consumption. The energy consumption for video streaming is related to the amount of video data to be downloaded. In tile-based 360° video streaming, dividing the video (i.e., the viewing area) into small sized tiles reduces the efficiency of video encoding. This is because video encoding relies on information related to other forward and backward video frames in most video compression techniques such as H.264 and H.265. If a video is divided into many small tiles, there is less opportunity for video compression (i.e., removing temporal and spatial redundancy) within each tile, and then reducing the compression efficiency. As a result, each encoded tile has more data, and more energy will be consumed to download the tiles.

During video processing, multiple independently encoded tiles covering the viewing area have to be decoded in time. To achieve this, a common method is to use multiple decoders to concurrently decode the tiles of the same video segment [3, 4]. However, using a number of decoders to concurrently decode the tiles leads to significant energy consumption. This is because applying many concurrent decoders makes the video decoding pipeline much more complex, which involves tedious CPU context switches and then high computational overhead.

To reduce energy consumption, we propose to encode the video by considering the viewing popularity, i.e., users may have similar viewing interests (i.e., viewing areas) when watching the same video. Specifically, by encoding the tiles requested by users of similar interests as a large tile (called *Ptile*), high compression efficiency can be achieved, and the total amount of downloaded data is reduced. Then, compared to existing approaches which download multiple small tiles, using Ptile can reduce the amount of data to be downloaded and reduce the energy consumption of data communication. Moreover, with Ptile, only one decoder is needed, which reduces the computational overhead and the energy consumption for video processing. To construct Ptiles, we exploit the historical viewing data from users watching the same video. Due to their common interests, the users have similar viewing areas, and their viewing centers are close to each other. We first identify these viewing centers and cluster them together, based on which we then identify and construct the Ptiles.

When encoding the Ptile, we propose to further save energy by reducing the insignificant frames in each video segment, i.e., reducing the frame rate to save energy while satisfying some QoE constraint. During fast view switching, high frame rate may not necessarily lead to high improvement on QoE, since the user has blurred vision when viewing a fast moving



(a) (b) (c)

Fig. 2: (a) The energy consumption of data transmission. (b) Time and power consumed for decoding a video segment (conventional tile-based scheme). (c) The energy consumption of video processing.

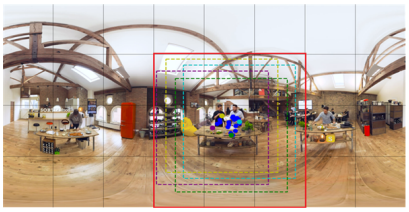


Fig. 1: The video is divided into 4 rows and 8 columns.

object [6, 7]. Thus, the frame rate is less critical to QoE during fast view switching, and can be reduced to save energy. Moreover, for Ptile with highly redundant content (i.e., consecutive frames with high similarity), dropping these redundant frames can save energy. Based on these observations, we propose to exploit user behavior and video content for frame rate adaptation. Based on real video traces, we model the impact of video features (i.e., video bitrate, frame rate) and user behavior (i.e., view switching) on QoE, and model the impact of video features on power consumption. Based on the QoE model and the power model, we formulate the energy-efficient and QoE-aware 360° video streaming problem as an optimization problem, and propose a control theory based algorithm to solve it.

The main contributions of this paper are as follows.

- Through real measurements, we identify the energy inefficiency problem of tile-based 360° video streaming and propose to encode the requested tiles as a Ptile to save energy.
- We formulate the energy-efficient and QoE-aware 360° video streaming problem as an optimization problem, and propose a control theoretic approach to solve it.
- We evaluate the performance of the proposed algorithm with real head movement data traces. Evaluation results show that the proposed algorithm can significantly reduce the energy consumption and improve the QoE.

The rest of the paper is organized as follows. We introduce the background and motivation in Section II. Section III presents the video, QoE, and power models. In Section IV, we formalize and solve the energy-efficient and QoE-aware 360° video streaming problem. In Section V, we present the evaluation results. Section VI discusses related work and Section VII concludes the paper.

II. Background and Motivation

Although tile-based 360° video streaming can improve QoE with limited wireless bandwidth, downloading and processing the tiles on mobile devices consume a large amount of energy. To identify the energy inefficiency of tile-based approach and the reason behind it, we conducted experiments based on the head movement traces of users watching a 360° video [8]. The video has 4K resolution (i.e., 3840x2160) with 30fps frame rate. The video is divided into a sequence of video segments. Each segment contains one second of video, which is further divided into 4 rows and 8 columns, a common setting used in the literature [9, 10]. As shown in Fig. 1, each dot represents the viewing center of one user, and each dashed block represents the viewing area of the corresponding user (i.e., the same color), respectively. The viewing area is determined by the viewing center and the Field-of-View (FoV) of the device, which is considered to be 100 degrees horizontally and vertically [11, 12]. In tile-based 360° video streaming, the tiles covering the viewing area are downloaded and processed in sequence on the mobile device. The experiments were performed using a Google Pixel 3 phone. The energy consumption is calculated using the power models that will be detailed in Section III-B.

The energy consumption for data transmission is related to the amount of downloaded data. Fig. 2(a) shows the energy consumption for data transmission of the Ptile scheme, normalized based on the conventional tile-based approach. Compared to conventional tile-based approach, the Ptile scheme can save the energy consumption of data transmission by 35%. This is because encoding the video area covered by the FoV tiles as one tile (i.e., the Ptile, the red block in Fig. 1) can achieve high compression efficiency, leading to less amount of data to be downloaded and less energy consumption.

The energy consumption for processing 360°video includes the energy for video decoding and view generation. In existing tile-based approach, to accelerate video decoding, multiple decoders are used to concurrently decode the tiles of the same segment. Fig. 2(b) shows the decoding time and power consumption when decoding the FoV tiles (night tiles) in existing tile-based approach. In this experiment, hardware-accelerated media codec (i.e., MediaCodec in Android) is used to support real-time decoding. The power consumption of video decoding is measured over the period of decoding a video segment, by subtracting the idle system power from the total power. As shown in Fig. 2(b), when more decoders are used, the decoding time decreases, but the power consumption significantly increases. This is because applying many concurrent decoders makes the video decoding pipeline much

complex, which leads to tedious CPU context switches and then high computational overhead on the mobile device. For example, as the number of decoders increases from 1 to 9, the total decoding time reduces from 1.3 sec to 0.5 sec (around 2.5X), but the power consumption increases from 241mW to 846mW (around 3.5X). In contrast, the Ptile uses only one decoder which can achieve both low decoding time (0.24sec) and low power consumption (287mW).

Fig. 2(c) compares the energy consumption of video processing (i.e., video decoding and view generation), normalized based on the conventional tile-based scheme using one decoder. After decoding, based on the coordinate mapping, the view is generated by drawing the pixel values onto the display. The energy consumption for view generation is calculated using the power models that will be detailed in Section III-B. As can be seen in Fig. 2(c), compared to using four concurrent decoders (the best scheme here) to process conventional tiles, the Ptile scheme can save the processing energy by 41%.

To construct Ptiles, we exploit the historical viewing data from users watching the same video. Users with similar viewing interests will most likely have similar viewing areas, and then their viewing centers will be close to each other. By identifying and clustering these viewing centers, we can construct Ptiles. The detail of Ptile construction will be presented in Section IV-A.

III. Video, Power and QoE Models

In this section, we introduce the video, power, and QoE models.

A. Video Model

On the server side, the video is broken into a sequence of video segments and each segment contains L seconds of video. Each segment is further divided into C tiles (e.g., 4 rows and 8 columns). Each tile is encoded into V copies corresponding to V different bitrates. In addition, M Ptiles are constructed, and the details will be described in Section IV-A. Each Ptile is encoded with V different bitrates and F different frame rates (i.e., indexed as 1, 2, ..., F, with F being the highest).

360° video streaming can be viewed as a sequence of data downloading tasks. Based on the network condition, the client requests the segment (i.e., the tiles) at the right quality level from the server. More specifically, the video streaming process is modeled as N data transmission tasks, corresponding to downloading N video segments. Let T_k denote the kth task which downloads the kth video segment. Let $T_k^{v,f}$ denote the kth task where the FoV tiles are encoded with bitrate level v ($v \in \{1, 2, ..., V\}$) and frame rate f ($f \in \{1, 2, ..., F\}$). In each task, the client requests the FoV tiles with the right quality and the tiles outside the FoV with the lowest quality [13]. Let B_k denote the video length of the downloaded but not yet viewed video in the buffer, in terms of seconds, when the client requests the kth segment. To avoid stall events (or rebufferings), the tiles of the segment should be completely downloaded before the buffer is drained out by the video player at the client side.

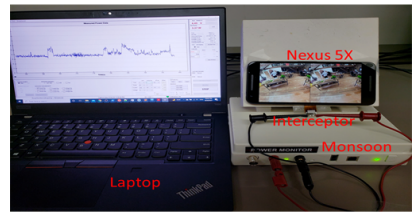


Fig. 3: Experimental setup for measuring power consumption.

B. Power Model

Based on real measurements, we model the impact of frame rate on power consumption, and we consider three parts of power consumption, namely data transmission (P_t), video decoding (P_d), and view rendering (P_r). We build the power models for various types of phones, including LG Nexus 5X, Google Pixel 3, and Samsung Galaxy S20. Since the battery connectors on the smartphone are very tiny, it is very challenging to connect them to the power monitor. To solve this problem, we design a battery interceptor based on Flex Printed Circuit Boards. The interceptor is connected to the smartphone's motherboard through the corresponding battery connector, and uses a custom designed circuit to modify the battery connection. As shown in Fig. 3, by using this interceptor, the smartphone can be connected to the Monsoon power monitor, which can directly supply power to the smartphone and accurately measure the power consumption.

To measure the power consumption of the wireless interface P_t, we conduct a set of experiments with a wget daemon running in the background (the screen is off) to download data from the server. To model P_d and P_r, we watch videos with 4K resolution (i.e., 3840x2160) based on the dataset [8] which have been locally cached in the smartphone. We consider the following three cases. In **case 1** (i.e., the baseline case), the video player is turned on but no video is played. In **case 2** (i.e., the decoding case), the video is decoded but no view is generated, i.e., the output buffer containing the decoded video is sent to the codec immediately after decoding a video frame, and the decoded data is not forwarded to the render engineer. The power difference between case 2 and case 1 represents the power consumption for video decoding (i.e., P_d). In tile-based streaming, similar to [3, 4], four decoders are used to decode the tiles in parallel. In Ptile streaming, only one decoder is needed. In **case 3** (i.e., local video playback), after decoding the video, the player retrieves the head orientation, based on which the video is rendered. The power difference between case 3 and case 2 represents the power consumption for view rendering (i.e., P_r). Note that we do not consider the power consumption of the screen (i.e., the screen is turned on for all the cases), since it depends on the screen size of the specific smartphone and the screen brightness set by the user.

The power models for different tiling schemes, which are used for evaluations in Section V-A, are summarized in Table I, where f is the frame rate and the power is in mW. Although bitrate is an important factor in vdieo streaming and it affects the amount of data to be transmitted, it is not considered in P_d and P_r. This is because the bitrate only affects the

TABLE I: The power models.

State	Nexus 5X	Pixel 3	Galaxy S20
Data trans.	$P_t = 1709.12 \pm 33.56$	$P_t = 1429.08 \pm 24.31$	$P_t = 1527.39 \pm 31.75$
Video decoding	Ctile: $P_d(f) = 1160.41 + 16.53f$ Ftile: $P_d(f) = 832.45 + 15.31f$ Nontile: $P_d(f) = 447.17 + 14.51f$ Ptile: $P_d(f) = 210.65 + 5.55f$	$P_d(f) = 574.89 + 15.46f$ $P_d(f) = 386.45 + 13.23f$ $P_d(f) = 209.92 + 10.95f$ $P_d(f) = 140.73 + 5.96f$	$P_d(f) = 798.99 + 16.49f$ $P_d(f) = 658.41 + 14.69f$ $P_d(f) = 305.55 + 11.41f$ $P_d(f) = 152.72 + 6.13f$
View rendering	$P_r(f) = 79.46 + 11.74f$	$P_r(f) = 57.76 + 4.19f$	$P_r(f) = 108.21 + 3.98f$

quantization level and thus the perceived quality, while the video processing complexity mainly depends on the resolution and the frame rate of the video, which significantly affects the power consumption of P_d and P_r [4].

Based on the power models, we can calculate the energy consumption of 360° video streaming, by summing up the energy consumption of all video segments. The energy consumed to download and process a video segment is calculated as follows

$$E(T_k^{v,f}) = E_t(T_k^{v,f}) + E_d(T_k^{v,f}) + E_r(T_k^{v,f}) \quad (1)$$

where $E_t(T_k^{v,f})$, $E_d(T_k^{v,f})$, and $E_r(T_k^{v,f})$ are the energy consumption for downloading, decoding, and rendering the kth segment with the FoV tile encoded at bitrate level v and frame rate f. $E_t(T_k^{v,f})$ can be calculated as $E_t(T_k^{v,f}) = P_t \cdot \frac{S(T_k^{v,f})}{R_k}$, where P_t is the data transmission power, as shown in Table I, $S(T_k^{v,f})$ is the segment data size, and R_k is the network bandwidth used to download video segment k. $E_d(T_k^{v,f})$ and $E_r(T_k^{v,f})$ are related to the duration of each video segment L, i.e., $E_d(T_k^{v,f}) = P_d(f) \cdot L$ and $E_r(T_k^{v,f}) = P_r(f) \cdot L$.

C. QoE Model

For each video segment k, the QoE model quantifies the user perceived quality by considering the following metrics: video quality, quality variation, and rebuffering. The QoE model is as follows:

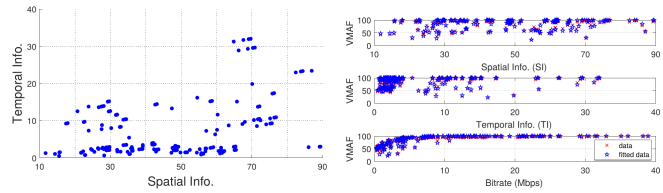
$$Q = Q_o - \omega_v I_v - \omega_r I_r \quad (2)$$

where Q_o is the "original" video quality without considering any quality impairment, I_v is the quality impairment caused by quality variation between two consecutive segments (i.e., $I_v = |Q_o^k - Q_o^{k-1}|$), I_r is the quality impairment caused by rebuffering event (i.e., $I_r = \frac{max(S_k/R_k - B_k, 0)}{B_k} Q_o^k$), where S_k is the segment data size and R is the downloading throughput), and ω_v and ω_r are the weights for quality variation and rebuffering, respectively.

To model the perceived quality Q_o, we consider both content features and video bitrate (i.e., encoding quantization). As suggested by ITU-T [14], we model Q_o with a logistic function:

$$Q_o = \frac{100}{1 + e^{-(c_1 + c_2 \cdot SI + c_3 \cdot TI + c_4 \cdot b)}} \quad (3)$$

where c_1, c_2, c_3, c_4 are the model parameters, b is the video bitrate, and SI and TI are the spatial and temporal perceptual information of the video, respectively. SI and TI capture the content complexity and motion features of the video [15].



(a) (b)

Fig. 4: (a) Spatial and temporal information of the videos. (b) The "original" quality (i.e., Eq. 3) of a video as a function of SI, TI, and bitrate.

TABLE II: Parameters in the QoE model.

Coefficient	c_1	c_2	c_3	c_4
Value	-0.2163	0.0581	-0.1578	0.7821

1) Determining the Parameters of Q_o: To determine parameters c_1, c_2, c_3, c_4 in Eq. 3, we conducted assessment experiments using a public 360° video dataset [8], which contains the head movement data traces of 48 users watching 18 videos. Each video is divided into one second segments, ten of which are uniformly selected for training the model Q_o. Fig. 4(a) shows the SI and TI values of the video segments, where a video with higher TI value has more changing scenes, and a video with higher SI has more spatial details in video frames. As can be seen, the selected videos cover a wide range of genres.

We adopt the perceptual quality metric, Video Multimethod Assessment Fusion (VMAF) [16] to model Q_o. VMAF has been widely used in the literature [17]. The VMAF scores range from 0 (lowest) to 100 (highest), thus the numerator in Eq. 3 is set to 100. Considering both spatial and temporal quality perceptions, VMAF presents a strong correlation with the subjective experiment result (i.e., mean opinion score) [18]. Recent research [17, 19] has validated that the VMAF metric can be successfully used to measure the perceived quality in 360° video streaming.

For different video segments, by changing SI, TI, and bitrate, we obtain the Q_o in terms of VMAF. Then, we can fit the model in Eq. 3 and obtain the parameters c_1, c_2, c_3, c_4. We use nonlinear least-squares regression (i.e., nlinfit in Matlab) to fit the model in Eq. 3, as shown in Fig. 4(b). The fitted model has a large Pearson correlation value (i.e., 0.9791), which implies that the model captures the training data accurately. The parameters of the Q_o model are summarized in Table II.

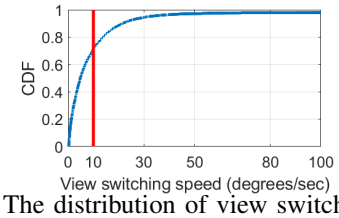


Fig. 5: The distribution of view switching speed.

2) Modeling the Impact of Frame Rate on Q_o: Although streaming 360° video at a higher frame rate can lead to better perceived video quality (i.e., Q_o in Eq. 3), more energy will be consumed for downloading and processing the video frames. We propose to reduce the frame rate to save energy while satisfying some QoE constraint. During fast view switching, high frame rate may not necessarily lead to high improvement on QoE, since the user has blurred vision when viewing a fast moving object [6, 7]. Thus, the frame rate is less critical to QoE during fast view switching, and can be reduced to save energy. Moreover, for Ptile with highly redundant content (i.e., consecutive frames with high similarity), dropping these redundant frames can save energy. Based on these observations, we propose to exploit user behavior and video content for frame rate adaptation.

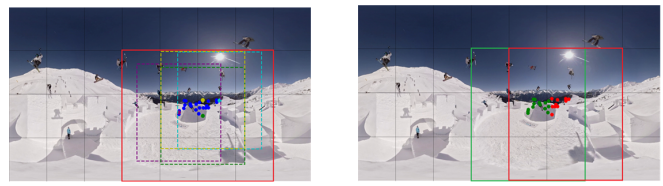
Similar to [20, 21], we model the impact of frame rate on the perceived quality using an inverted exponential function. Then, the Q_o in Eq. 3 is reduced by a factor $(1 - e^{-\alpha * f/f_m})/(1 - e^{-\alpha})$, where f is the reduced frame rate, f_m is the original maximum frame rate. Parameter α characterizes how fast the perceived quality drops as the frame rate decreases, where a larger α indicates a slower falling rate, and vice versa.

To determine the parameter α, we exploit the historical viewing data from users watching the same video. Specifically, α is related to the viewing behavior and the video content. For example, the user is more interested in the new view that he will switch to. Taking a soccer game as an example, the user would focus on a fixed view where two players are grabbing the ball from each other. When the ball is passing through a wide range, the user's attention will switch and track the ball until it reaches another view. Thus, the frame rate is less critical to QoE during fast view switching, and can be reduced to save energy. Moreover, as mentioned before, dropping the redundant frames in Ptile also reduces the energy consumption.

By considering fast view switching and video content similarity, α is defined as follows.

$$\alpha = \frac{S_{fov}}{TI} \tag{4}$$

where S_{fov} is the view switching speed (degrees per second), and TI is the motion feature of the video. With a larger S_{fov} (i.e., the user is exploring the video), α is larger and reducing the frame rate has negligible impact on the perceived quality. Similarly, with a smaller TI (i.e., the video content is more static), reducing the frame rate has insignificant impact on the perceived quality. We calculate the view switching speed based on the timestamp and the orientation vectors of the viewing centers. Specifically, the view switching speed when a user watches video from time t_{i-1} to t_i is expressed as follows.



(a) (b)

Fig. 6: Ptile construction. (a) Ptile is too large. (b) Split a large Ptile into two Ptiles (represented by two different colors).

$$S_{fov} = \frac{arccos(\frac{\vec{O}_{i-1} \cdot \vec{O}_i}{\|\vec{O}_{i-1}\| \|\vec{O}_i\|})}{t_i - t_{i-1}} \tag{5}$$

where $\vec{O}_{i-1}$ and $\vec{O}_i$ are the orientation vectors of the viewing center at time t_{i-1} to t_i, and $\| \vec{O}_{i-1} \|$ and $\| \vec{O}_i \|$ are the magnitude of $\vec{O}_{i-1}$ and $\vec{O}_i$, respectively.

Fig. 5 shows the distribution of view switching speed, when 48 users watch 18 360° videos [8]. View switching speed can affect the user's perceived quality, i.e., the user will become less sensitive to quality distortion when viewing switching speed becomes faster. Research has shown that when the view switching speed exceeds 10 degree/s, users can tolerate 50% more quality distortion than they would have if the viewpoint was static [7]. As can be seen in Fig. 5, users change their views over 10 degree/s for more than 30% of time, which means there is a large chance to reduce frame rate to save energy, with insignificant impact on the perceived quality.

IV. Energy-efficient and QoE-Aware 360° Video Streaming

In this section, we first describe how to construct Ptiles, then formalize and solve the energy-efficient and QoE-aware 360° video streaming problem.

A. Ptile Construction

When watching a 360° video, users with similar viewing interests will more likely have similar viewing areas, and then their viewing centers are close to each other. By identifying and clustering these viewing centers, we can construct Ptiles. The challenge is that we do not have any knowledge on the number of clusters (Ptiles) to construct before hand, and then many clustering algorithms such as k-means clustering are not suitable. One solution is to use non-parametric clustering algorithms like the density-based clustering algorithm [22]. Although such solution can address the challenges of not knowing the number of clusters before hand, it introduces another problem, i.e., the cluster may grow too large and then losing the benefits of Ptile such as energy efficiency. Fig. 6(a) shows an example based on a video trace (i.e., Freestyle Skiing) [8], where each dot presents the viewing center of a user. The dashed purple, cyan, green and yellow blocks represent the viewing areas of the leftmost, rightmost, down-most, and up-most users with similar viewing interests, respectively. As can be seen in the figure, clustering nearby viewing centers together can lead to a large cluster spanning

Algorithm 1: Clustering Users' Viewing Centers

Input: a set of nodes U, δ, σ
Output: a list of clusters Π

1 $N_u = \{n \mid n \in U \wedge n \neq u \wedge dist(u,n) \leq \delta\}$ *for* $u \in U$
2 **while** $U \neq \emptyset$ **do**
3 $\quad U_j, U \leftarrow ClusterFunc(U)$
4 $\quad$ **if** $\text{argmax}_{u,n \in U_j}\, dist(u,n) > \sigma$ **then**
5 $\quad\quad U_{j1}, U_{j2} \leftarrow kmeans(U_j, k=2)$
6 $\quad\quad$ add U_{j1} and U_{j2} to the list Π
7 $\quad$ **else**
8 $\quad\quad$ add U_j to the list Π
9 $\quad$ **end**
10 **end**
11 return Π
12
13 **function** `ClusterFunc`(U)**:**
14 $\quad u = \text{argmax}_{u \in U} |N_u|$
15 $\quad U_j = \emptyset$
16 $\quad U_j = U_j \cup \{u\}$ // initiate the cluster
17 $\quad Q = \emptyset$ // first-in first-out queue
18 $\quad$ ENQUEUE(Q, u) // push node into queue
19 $\quad$ **while** $Q \neq \emptyset$ **do**
20 $\quad\quad u =$ DEQUEUE(Q) // pop node from queue
21 $\quad\quad$ **foreach** $n \in N_u$ **do**
22 $\quad\quad\quad$ **if** $n \notin U_j$ **then**
23 $\quad\quad\quad\quad U_j = U_j \cup \{n\}$ // add node n to U_j
24 $\quad\quad\quad\quad U = U \setminus \{n\}$ // remove n from U
25 $\quad\quad\quad\quad$ ENQUEUE(Q, n)
26 $\quad\quad\quad$ **end**
27 $\quad\quad$ **end**
28 $\quad$ **end**
29 $\quad$ return U_j, U
30 **end**

a large area, and then the constructed Ptile (the red block in Fig. 6(a)) is too large. Then, it is better to split the large Ptile into two Ptiles (the red block and the green block), as illustrated in Fig. 6(b).

We design a new algorithm which can cluster nearby viewing centers and make sure that the constructed Ptile for each cluster is not too large. The algorithm is based on two important parameters, δ and σ. δ determines if two viewing centers should be in a cluster based on their distance, and they belong to the same cluster if their distance is less than or equal to δ. The clustering performance will be affected by δ. If δ is too small, some viewing centers may not be clustered together although they should. If δ is too large, the cluster may include viewing centers faraway, i.e., users with different viewing interests are clustered together. The size of a cluster is determined by σ, that is, the distance between any two viewing centers in the cluster should not be farther than σ. If σ is too large, the cluster may grow too large. On the other hand, if σ is too small, too many clusters may be constructed. In Section V-B, we will set up δ and σ based on experiments.

Let U denote a set of nodes, where each node ($u \in U$) represents the viewing center of a user. Let $dist(u,n)$ denote the Euclidean distance between two nodes u and n. Let $N_u = \{n \mid n \in U \wedge n \neq u \wedge dist(u,n) \leq \delta\}$ denote u's close neighbors. As detailed in Algorithm 1, the cluster is initiated with node u which has the maximum number of close neighbors (line 14), and then the cluster expands by adding nodes that are close to any node inside the cluster (lines 19-28). The cluster expanding continues until no more nodes can be added, and the nodes that have been clustered will be removed from U (line 24). Then, if the maximum distance between any two nodes within the cluster is larger than σ, the k-means clustering algorithm is applied to split the cluster (line 4-9). The algorithm repeats the above process until $U = \emptyset$.

For each cluster, a Ptile is constructed by encoding the conventional tiles that cover the viewing areas of users in this cluster. Then, downloading the Ptile will provide better QoE for most users compared to downloading conventional tiles. However, if a user is interested in a different area from the downloaded Ptile, the video may stall due to re-buffering. To address this problem, similar to existing work [13, 23], besides downloading the Ptile with high quality, the remaining video content outside of the Ptile is downloaded with low quality. Specifically, the remaining area is partitioned into large blocks along the Ptile's upper and lower horizontal lines, which can be encoded with better compression efficiency. These blocks are encoded at the lowest quality, and will be downloaded along with the Ptile. Because the video blocks are encoded with the lowest quality level, and the compression efficiency is high, the system does not sacrifice too much bandwidth for downloading them, but can minimize the QoE drop when the user viewing interest suddenly changes during video playback.

B. Energy-efficient and QoE-Aware Video Streaming

For video streaming, the client first predicts the user's viewing area for each video segment, and then prefetches the corresponding video at the right quality. The ridge regression model is applied to better predict the user's viewing area (i.e., the viewing center), since it is more robust to deal with overfitting. When a user watches 360° video, his viewing centers, represented by (x, y) coordinates, are recorded by the sensors embedded in the headset, The viewing center coordinates are recorded at a fixed sampling rate (e.g., 50 Hz), and then the recorded x and y coordinates collected at different times will form a stream of time series data. Such data can be used to train the model and predict the future. More specifically, taking x coordinates as an example, the user's most recent video watching history can be used to predict the x coordinate of the user's viewing center of the video segment that will be downloaded. Since the video player buffer is very small, the coordinates of the most recent viewed segment have strong correlation with the segment to be downloaded. Thus, the ridge regression model can better predict the viewing center of the downloading segment, and then predict the user's viewing area.

With the predicted viewing area, the client verifies if this area can be covered by a Ptile. If there is no such Ptile, the client will download conventional tiles with the best possible quality based on the current network condition. Otherwise, the client determines the right quality level and frame rate for the Ptile to minimize the energy consumption under some QoE constraints.

For a video segment k, the client determines a tuple of quality level and frame rate, i.e., (v, f), for downloading it. In 360° video streaming, video segments are fetched into a video buffer before playback. The duration of the downloaded but not yet viewed video evolves dynamically as video segments are downloaded and played. The time for downloading a segment depends on the segment data size and the network bandwidth. In the meantime, the length of the buffered video decreases as it is being played. After downloading segment k, the length of the buffered video becomes

$$B_{k+1} = max(B_k - \frac{S(T_k^{v,f})}{R_k}, 0) + L - \triangle t_k, \tag{6}$$

where $S(T_k^{v,f})$ is the data size of video segment k encoded at quality level v and frame rate f, L is the length of the video segment, and $\triangle t_k$ is the waiting time. If the buffered video data after downloading previous segment $k-1$ reaches the buffer threshold β, the player waits for $\triangle t_k$ before requesting segment k, i.e., $\triangle t_k = max(B_k - \beta, 0)$. To avoid rebuffering events, the constraint is to bound the buffered video at each downloading task, as expressed in Eq. 7.

$$0 < B_{k+1} \tag{7}$$

To formalize the energy-efficient and QoE-aware 360° video streaming problem, we introduce a binary variable $X_k^{v,f}$ for bitrate and frame rate selection, where $X_k^{v,f} = 1$ if the Ptile in task T_k is downloaded at bitrate index v and frame rate index f, i.e., $T_k^{v,f}$; otherwise, $X_k^{v,f} = 0$. Since only one quality version is selected for each downloading task, $\sum_{v=1}^{V}\sum_{f=1}^{F} X_k^{v,f} = 1$. Let v_m and f_m denote the highest possible bitrate level and frame rate for the Ptile such that the Ptile encoded at this quality version can be successfully downloaded. Let $Q(v, f)$ denote the QoE of the video segment with quality level v and frame rate f. Then, the energy-efficient and QoE-aware 360° video streaming problem can be formulated as an ϵ-constraint optimization problem as follows:

$$\textbf{min} \quad \sum_{k=1}^{N} E(T_k^{v,f}) \tag{8}$$

$$\textbf{s.t.} \quad Eq.\ 6, 7 \tag{8a}$$

$$\sum_{v=1}^{V}\sum_{f=1}^{F} X_k^{v,f} = 1 \ for\ \forall k \tag{8b}$$

$$(1-\epsilon)\cdot Q(v_m, f_m) \leq Q(v, f) \tag{8c}$$

where constraint (8a) is to ensure that the video data can be successfully downloaded before its playback, constraint (8b) states that only one quality version (i.e., specific bitrate level and frame rate) of a Ptile is downloaded, and constraint (8c) specifies the loss tolerance (i.e., ϵ = 5%) on the perceived video quality if selecting lower bitrate level and frame rate.

C. Model Predictive Control based Algorithm

Ideally, if the future bandwidth for downloading each video segment is known, the optimization problem in Eq. 8 can be solved, and the optimal (v, f) tuple can be obtained for each segment. However, it is impossible in practice to have such perfect knowledge. Instead, we can achieve a sub-optimal solution, i.e., we can predict the network bandwidth for multiple segments in the future, based on which we choose a better bitrate. Next, we propose an online solution using the Model Predictive Control (MPC) based optimization framework [24], which is widely used to optimize a complex control objective in a dynamical system with constraints. Using MPC, the algorithm is able to smooth out the bandwidth prediction error by optimizing the video adaptation over several segments, i.e., a large estimation error for one specific segment will not have significant impact on the overall performance.

We use the harmonic mean of the downloading throughput of the past several segments to estimate the network bandwidth. Since the network condition may vary widely during video streaming, some downloading throughput can be much higher or lower than others among the past several segments. The harmonic mean is used to eliminate the impacts of these fluctuations. More bandwidth estimation methods can be found in [25, 26], which is out of the scope of this paper.

For initialization, the metadata for the first H video segments are downloaded during the startup period such that the client can always know the content features of H segments in advance. For each video segment k, the MPC based algorithm works as follows.

(a) Obtain current buffer B_k and the metadata for video segments k to $k+H-1$.
(b) Predict bandwidth R_k,..., R_{k+H-1}.
(c) Solve the optimization problem in Equation 8 over video segments k to $k+H-1$.
(d) Request video segment k encoded at bitrate level v and frame rate f, obtained from the previous step.
(e) Move the optimization window forward to $[k+1, k+H]$ to determine the optimal (v, f) for segment $k+1$.

To find the solution in step (c), we propose a dynamic programming based algorithm. The algorithm simulates the process of downloading the next H video segments with index of $[k, k+H-1]$. When downloading video segment $i \in [k, k+H-1]$, the buffer size (i.e., the length of downloaded but not yet viewed video) can be in different states $B_{i-1} \in [0, \beta]$, depending on how segment $i-1$ was downloaded. In this paper, we discretize the buffer state at a granularity of 500 milliseconds, i.e., for $\beta = 3$ seconds, there are six buffer states. Based on Eq. 6, B_{i-1} is the buffer state after downloading segment $i-1$. Let $U^\star(B_{i-1}, v_{i-1}, f_{i-1})$ denote the minimum amount of energy spent on reaching buffer state B_{i-1}, i.e., the minimum amount of energy spent on downloading segments 1 to i-1. When requesting segment i with quality (v_i, f_i), $E(T_i^{v_i,f_i})$ is consumed, as calculated with Eq. 1. Then, the Bellman equation for the dynamic programming is $U^\star(B_i, v_i, f_i) = \min_{v_i,f_i}\{U^\star(B_{i-1}, v_{i-1}, f_{i-1}) + E(T_i^{v_i,f_i})\}$. That is, for each buffer state B_i, there is at most one path achieving minimum energy $U^\star(B_i, v_i, f_i)$ by picking the optimal quality version $(v_i^\star, f_i^\star)$.

After H rounds, the algorithm finds the minimum energy to reach each possible buffer state. Then, the algorithm finds the minimum energy to reach all possible buffer

states, $U^{\star}(B_{k+H-1}, v_{k+H-1}, f_{k+H-1})$, and determines the right quality level and frame rate for downloading segment k by back tracking the sequence of the optimal solution from the last segment $\{(v^{\star}_{k+H-1}, f^{\star}_{k+H-1}), ..., (v^{\star}_{k}, f^{\star}_{k})\}$. The time complexity of the algorithm is $O(HVF)$.

V. Performance Evaluations

In this section, we evaluate the performance of the proposed energy-efficient and QoE-aware 360° video streaming algorithm.

A. Experiment Setup

The evaluation is based on the head movement traces of 48 users watching eight 360° videos [8]. Table III gives the details of these videos, which cover various scenarios such as performance, sports, etc. For each video, forty users are randomly selected and their head movement traces are used to construct the video tiles (and Ptiles), and the remaining traces are used for evaluation.

There are two main parts in the 360° video streaming system: the server and the client. On the server side, similar to [3, 7], each video is divided into a sequence of video segments. Each segment contains one second of video, which is further divided into tiles (and Ptiles). The video tiles (and Ptiles) are encoded with the FFmpeg using the x264 encoder. There are five quality levels (1 to 5, with 1 being the lowest quality) for video encoding, which can be obtained by changing the constant rate factor (CRF) to different values, ranging from 38 to 18 with an interval of 5 [3]. On the client side, the FoV of the mobile device is set to be 100 degrees horizontally and vertically. For the QoE model, similar to [3, 13], the weights are set to (ω_v, ω_r) = (1, 1). The power models are built based on real measurements using three phones, including a LG Nexus 5X, a Google Pixel 3, and a Samsung Galaxy S20, as shown in Section III-B.

In the evaluation, to simulate the network traffic, we use a LTE network throughput trace [27], which has varying patterns. To consider different network conditions in the performance evaluation, the trace is linearly scaled to generate two different traces, called *trace 1* and *trace 2*, where the network throughput of trace 1 is twice that of trace 2. In trace 2, the average throughput is 3.9 Mbps and it varies between 2.3 Mbps and 8.4 Mbps.

We compare the performance of the following approaches.

- **Ctile**: The conventional tile-based 360° video streaming approach, where each video segment is divided into fixed size tiles, i.e., 4 rows and 8 columns. The client downloads video tiles for each segment in sequences and processes these tiles using multiple decoders. This approach has been used in many existing video streaming solutions [9, 10].
- **Ftile**: Each video segment is divided into a fixed number of tiles which may have different sizes. Similar to [12], each segment is first divided into 450 small blocks (i.e., 15 rows and 30 columns), which are then clustered into ten tiles based on users' views.

TABLE III: The test videos.

ID	Length	Content	ID	Length	Content
1	6:01	Basketball Match	5	4:52	Moving Rhinos
2	2:52	Showtime Boxing	6	2:44	Football Match
3	6:13	Festival Gala	7	3:25	Tahiti Surf
4	4:38	Idol Dancing	8	3:21	Freestyle Skiing

- **Nontile**: This is the 360° video streaming scheme commonly used in video platforms such as YouTube, where the video is not divided into tiles and the player always downloads the whole video.
- **Ours**: Our proposed energy-efficient and QoE-aware 360° video streaming algorithm. For each Ptile, three versions of reduced frame rates are constructed, i.e., reducing $\{10\%, 20\%, 30\%\}$ frame rate of the original video.
- **Ptile**: This is one variation of *Ours*. Different from *Ours*, each Ptile is only encoded at the frame rate of the original video. It is similar to the Ctile approach, but the tiles covering the FoV are encoded into one large tile.

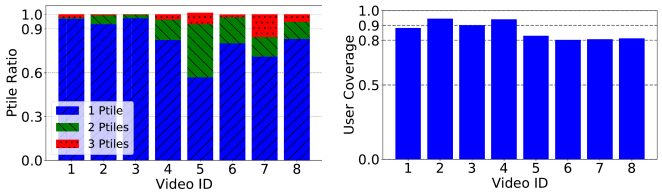
B. Performance of the Ptile Construction

We empirically set the parameter σ to be the width of a conventional tile and $\delta = \sigma/4$. To avoid constructing unnecessary Ptiles that cover too few users, a Ptile is only constructed if it covers at least five users (i.e., 10% of the users in the dataset).

1) Ptile Coverage: Fig. 7 evaluates the performance of our Ptile construction algorithm, as the users watch different videos [8] listed in Table III. These 8 videos are under different settings. Users are instructed to focus on the video content for videos 1 to 4, but not for videos 5 to 8 so that they can exhibit their unique behavior patterns. From Fig. 7(a), we can see that over 95% video segments need only one Ptile for videos 2 to 4, because users have similar viewing interests and they are instructed to focus on the video content in these videos. In video 1 (basketball match), although users' gazing directions frequently move, more than 96% video segments only require one or two Ptiles. For videos 5 to 8 where users are free to explore, more Ptiles are needed. Even under this setting, over 92% video segments only require one or two Ptiles on average.

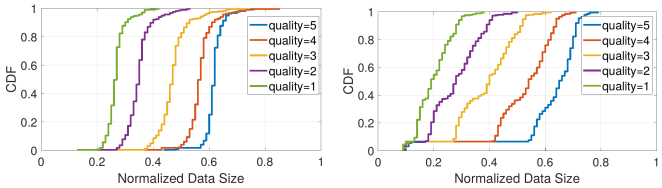
Fig. 7(b) shows the percentage of users covered by the Ptiles. For videos 1, 2, 3, and 4, 88.4%, 94.6%, 90.3%, and 94.1% users can be served by the Ptiles, respectively. Even for videos 5 to 8 where users are free to explore, more than 80% of users can be served by the Ptiles. Since the viewing areas of most users are covered by the Ptiles, most of them only need to download these Ptiles to save bandwidth and energy.

2) Ptile vs. Ctile: To see the benefit of encoding the FoV tiles as a Ptile, we compare the data size of the Ptile and that of the conventional tiles covering the same area when both are encoded at the same quality level. Fig. 8 shows the data size CDFs of using the Ptile for each video segment, normalized based on the data size of the corresponding conventional tiles. From the figure, we can see that using Ptile can significantly reduce the data size, since higher compression efficiency can be achieved by using larger Ptiles instead of smaller



(a) (b)

Fig. 7: (a) The number of Ptiles constructed in different videos. (b) The percentage of users covered by Ptiles.



(a) Video 4 (b) Video 5

Fig. 8: CDFs of the normalized data size for a Ptile. Only two representative videos are shown to save space.

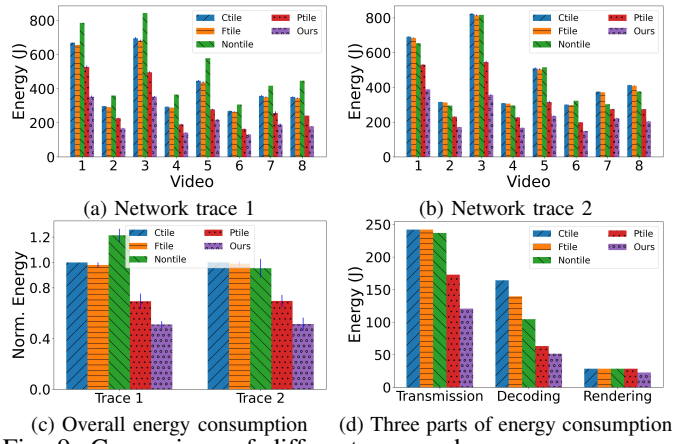
conventional tiles. As detailed in Fig. 8(a), with Ptile, the median data size is 62%, 57%, 47%, 35%, and 27% of that of conventional tiles when the video encoding quality is 5, 4, 3, 2, and 1, respectively. That is, using Ptiles can save bandwidth by 38% (i.e., 1-0.62=0.38), 43%, 53%, 65%, and 73% when the video encoding quality is 5, 4, 3, 2, and 1, respectively.

C. Performance Comparisons

We evaluate the performance of our energy-efficient and QoE-aware 360° video streaming algorithm using user's head movement data trace. Similar to [3, 7], the playback buffer is set to three seconds.

1) Energy Comparison: The energy consumption is calculated based on the power models shown in Section III-B. Fig. 9 compares the energy consumption of different approaches under different network conditions using a Pixel 3 phone. Specifically, Fig. 9 (a) and (b) show the detailed energy consumption of eight videos under network trace 1 and trace 2. Fig. 9 (c) shows the energy consumption, normalized based on *Ctile*, of different approaches. To further evaluate the proposed algorithm, different types of smartphones are also used for performance evaluations. Fig. 10 shows the energy consumption, normalized based on *Ctile*, of different approaches when using a LG Nexus 5x and Samsung Galaxy S20 phone. In general, our algorithm has the lowest energy consumption, and *Ptile* has the second lowest energy consumption, both of them significantly outperform *Ctile*, *Ftile* and *Nontile*.

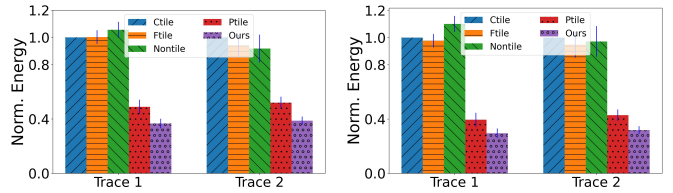
As shown in Fig. 9(c), compared to *Ctile*, the energy saving of *Ptile* and *Ours* is 30.3% and 49.7% on average. By using Ptiles, *Ours* and *Ptile* can achieve high compression efficiency and reduce the amount of data to be downloaded, and hence reducing the energy consumption of data communication. Moreover, with Ptile, only one decoder is needed, which reduces the computational overhead and the energy consumption of video processing. In contrast, for *Ctile* and *Ftile*, multiple decoders are needed to decode the video tiles in time and cause high energy consumption. By removing insignificant frames in



(a) Network trace 1 (b) Network trace 2

(c) Overall energy consumption (d) Three parts of energy consumption

Fig. 9: Comparison of different approaches on energy consumption (Pixel 3).



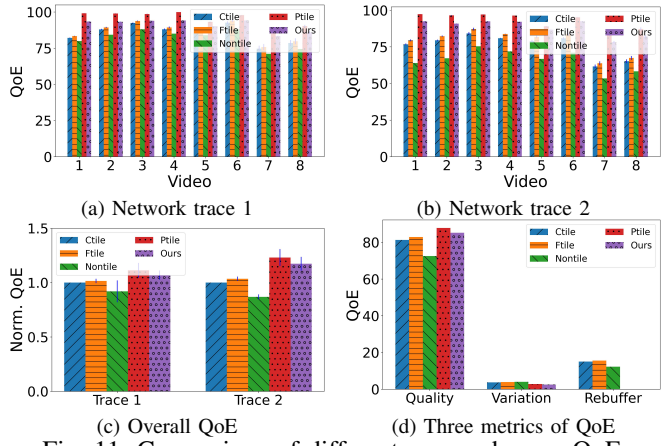
(a) Energy consumption (Nexus 5X) (b) Energy consumption (Galaxy S2)

Fig. 10: Comparison of different approaches on energy consumption for Nexus 5X (a) and Galaxy S20 (b).

the Ptiles, *Ours* can further reduce the energy consumption by around 20% compared to *Ptile*.

Taking video 8 under network trace 2 as an example, Fig. 9(d) shows the energy consumption of data transmission and the energy consumption of video processing. As can be seen, compared to *Ctile*, *Ptile* and *Ours* can save 26.1% and 47.7% energy for data transmission, respectively. The energy consumption of video processing includes the energy consumption for video decoding and view generation. As can be seen, compared to *Ctile*, *Ptile* and *Ours* can reduce the energy consumption for video decoding by 50.1% and 53.5%. Since the view generation process only involves reading the pixel values from the memory based on the coordinate mapping (i.e., projection), the energy consumption of view generation is much less than that of video downloading and video decoding in 360° video streaming. This is different from interactive virtual reality (VR) gaming applications, where new frames (i.e., video content) need to be rendered based on user interactions in real time, which imposes heavy computation overhead. In contrast, the video content in 360° video has already been recorded.

The energy consumption of *Nontile* depends on the network condition. When more bandwidth is available (trace 1), more data will be transmitted, i.e., besides the video data in FoV, video data in other areas are also transmitted in high quality. Thus, it consumes much more energy than other approaches. When less bandwidth is available (trace 2), less amount of data is downloaded, and the video quality has to be reduced.



(a) Network trace 1 (b) Network trace 2

(c) Overall QoE (d) Three metrics of QoE

Fig. 11: Comparison of different approaches on QoE.

As a result, the energy consumption is close to *Ctile*, but at a cost of lower QoE, as shown in Fig. 11.

2) QoE Comparison: Fig. 11 compares the QoE of different approaches under different network conditions. Specifically, Fig. 11 (a) and (b) show the detailed QoE of eight videos under two network traces. Fig. 11 (c) shows the QoE, normalized based on *Ctile*, of different approaches. In general, *Ours* and *Ptile* can achieve higher QoE than *Ctile*, *Ftile* and *Nontile*.

As shown in Fig. 11 (c), *Nontile* has the worst QoE because it does not differentiate videos inside the FoV and other areas, and hence has to reduce the video quality of the whole video when network bandwidth is limited. By using tiles, the other three approaches can maintain higher video quality within FoV under the same bandwidth constraint, and hence have higher QoE. Compared to *Ctile*, *Ours* can improve the QoE by 7.4% for network trace 1 and 18.4% for network trace 2. By encoding the video as Ptiles, *Ours* achieves high compression efficiency and the data size of encoded video becomes much smaller than *Ctile*. Then, under the same network condition, a user can download a Ptile encoded at higher quality, instead of Ctiles or Ftiles encoded with lower quality, and thus the QoE can be improved. In *Ftile*, it is inefficient to cluster users' views into a fixed number of clusters. This is because the video area viewed by many users (i.e., the area covered by a Ptile) is divided into unnecessary number of small tiles, reducing video compression efficiency. Moreover, dividing the video area outside of a Ptile reduces the encoding efficiency and thus causes high bandwidth demand. Compared to *Ptile*, *Ours* has a little bit lower QoE, which still satisfies the QoE constraint. By trading off a little bit of QoE, *Ours* further reduces the energy consumption compared to *Ptile*. For example, compared to *Ptile*, *Ours* reduces the energy consumption by 27.7% for network trace 2 (Fig. 9(c)), but only reduces the QoE by 4.6% (Fig. 11(c)).

Taking video 8 under network trace 2 as an example, Fig. 11 (d) shows the three metrics of the QoE (as in Eq. 2): average video quality, quality variation, and rebuffering. As can be seen, compared to *Ctile*, *Ours* and *Ptile* achieve higher average video quality, lower quality variation, and lower rebuffering effect. Since *Nontile* downloads the whole video and has to reduce the video quality when less bandwidth is available, the average video quality achieved by *Nontile* is much lower than that of *Ctile*. Due to network variations, *Ctile*, *Ftile* and *Nontile* have more rebuffering events since they request video segments encoded at high quality but the network throughput suddenly drops. In contrast, with Ptiles, *Ours* does not generate any rebuffering events.

VI. Related Work

Tile-based streaming has been widely used in 360° video streaming to adapt to varying network conditions. By cutting the video segment into tiles, only tiles covering users' viewing area are downloaded with high quality, and other tiles are downloaded with low quality to save bandwidth. In the literature, a large amount of research focuses on encoding video with fix sized tiling [4, 13, 11], while others focus on dividing the video into tiles with different sizes [12, 5]. Chen *et al.* [28] proposed technique to construct tiles to reduce the bandwidth consumption; however, they did not consider energy issues and did not consider the impact of frame rate reduction on energy consumption and QoE. Although tile-based 360° video streaming can improve QoE with limited wireless bandwidth, downloading and processing tiles consume more energy on mobile devices.

There has been considerable research on reducing the energy consumption of traditional non-360° video streaming. Some researchers proposed to reduce the power consumption of the wireless interface during video streaming [29, 30]. Hu *et al.* [29] proposed techniques to save energy based on whether the user tends to watch video for a long time, skip, or early quit. Wu *et al.* [30] designed an energy efficient video streaming scheme over heterogeneous networks. Other researchers proposed to reduce the energy consumption of video processing on mobile devices [31, 32, 33]. Yang *et al.* [31] proposed to save energy for video streaming by adaptively adjusting the CPU frequency. Chen *et al.* [34] proposed to save energy by considering the context (environment) of video streaming. However, the issue of saving energy for tile-based 360° video streaming is much more challenging, since dividing the video into tiles can make the video processing pipeline much more complex and lead to high computation overhead on mobile devices. Different from them, we identify the energy inefficiency problem of tile-based 360° video streaming and propose a Ptile approach, which can reduce the energy consumption of video downloading and the energy consumption of video processing.

VII. Conclusions

In this paper, we identified the energy inefficiency problem of tile-based 360° video streaming, and proposed an energy-efficient and QoE-aware 360° video streaming algorithm to minimize the energy consumption under some QoE constraint. We proposed to encode the video by considering the viewing popularity, where the tiles requested by users of similar interests are encoded as a large Ptile. To construct Ptiles, we exploit the historical viewing data from users watching

the same video. By considering user behavior and video content, we proposed to further save energy by reducing the insignificant frames when encoding Ptiles. Based on real video traces, we modeled the impact of video features and user behavior on QoE, and modeled the impact of video features on power consumption. Based on the QoE model and the power model, we formulated the energy-efficient and QoE-aware 360° video streaming problem as an optimization problem, and proposed a MPC based algorithm to solve it. Through extensive trace-driven experiments, we demonstrated that the proposed algorithm can reduce the energy consumption by 49.7% and improve the QoE by more than 7.4%.

VIII. Acknowledgments

This work was supported in part by the National Science Foundation under grants CCF-2125208 and CNS-1815465.

References

[1] Google. YouTube Live in 360 Degrees Encoder Settings, April 2022. https://support.google.com/youtube/answer/6396222.

[2] Facebook. Facebook 360, April 2022. https://facebook360.fb.com/.

[3] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *ACM MobiCom*, 2018.

[4] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-Degree Video Streaming for Smartphones. In *ACM MobiSys*, 2018.

[5] M. Xiao, C. Zhou, Y. Liu, and S. Chen. OpTile: Toward Optimal Tiling in 360-Degree Video Streaming. In *ACM Int'l Conf. on Multimedia*, 2017.

[6] K. Fukushima, J. Fukushima, T. Warabi, and G. R. Barnes. Cognitive Processes Involved in Smooth Pursuit Eye Movements: Behavioral Evidence, Neural Substrate and Clinical Correlation. *Frontiers in Systems Neuroscience*, 2013.

[7] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang. Pano: Optimizing 360 Video Streaming with a Better Understanding of Quality Perception. In *ACM SIGCOMM*, 2019.

[8] C. Wu, Z. Tan, Z. Wang, and S. Yang. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *ACM Multimedia Systems Conference (MMSys)*, 2017.

[9] R. I. da Costa Filho, M. C. Luizelli, M. T. Vega, J. van der Hooft, S. Petrangeli, T. Wauters, F. De Turck, and L. P. Gaspary. Predicting the Performance of Virtual Reality Video Streaming in Mobile Networks. In *ACM Multimedia Systems Conference (MMSys)*, 2018.

[10] Jeroen Van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Tile-based Adaptive Streaming for Virtual Reality Video. *ACM Trans. on Multimedia Computing, Communications, and Applications (TOMM)*, 2019.

[11] A. Mahzari, A. T. Nasrabadi, A. Samiei, and R. Prakash. FoV-Aware Edge Caching for Adaptive 360° Video Streaming. In *ACM Int'l Conf. on Multimedia*, 2018.

[12] C. Zhou, M. Xiao, and Y. Liu. ClusTile: Toward Minimizing Bandwidth in 360-Degree Video Streaming. In *IEEE INFOCOM*, 2018.

[13] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li. DRL360: 360-Degree Video Streaming with Deep Reinforcement Learning. In *IEEE INFOCOM*, 2019.

[14] ITU. ITU-T G.1070: Opinion model for video-telephony applications, April 2022. https://www.itu.int/rec/T-REC-G.1070.

[15] ITU. ITU-T P.910: Subjective Video Quality Assessment Methods for Multimedia Applications, April 2022. https://www.itu.int/rec/T-REC-P.910.

[16] Z. Li, C. Bampis, J. Novak, A. Aaron, K. Swanson, A. Moorthy, and J. D. Cock. VMAF: The Journey Continues. In *The Netflix Tech Blog*, 2018.

[17] M. Orduna, C. Díaz, L. Muñoz, P. Pérez, I. Benito, and N. García. Video Multimethod Assessment Fusion (VMAF) on 360VR Contents. *IEEE Trans. on Consumer Electronics*, 2020.

[18] D. Nandakumar, Y. Wu, H. Wei, and A. Ten-Ami. On the Accuracy of Video Quality Measurement Techniques. In *IEEE Int'l Workshop on Multimedia Signal Processing (MMSP)*, 2019.

[19] S. Croci, C. Ozcinar, E. Zerman, J. Cabrera, and A. Smolic. Voronoi-based Objective Quality Metrics for Omnidirectional Video. In *Int'l Conf. on Quality of Multimedia Experience (QoMEX)*, 2019.

[20] A. Mackin, F. Zhang, and D. Bull. A Study of High Frame Rate Video Formats. *IEEE Trans. on Multimedia*, 2018.

[21] S. Xie, Y. Xu, Q. Shen, Z. Ma, and W. Zhang. Modeling the Perceptual Quality of Viewport Adaptive Omnidirectional Video Streaming. *IEEE Trans. on Circuits and Systems for Video Technology*, 2019.

[22] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. on Database Systems*, 2017.

[23] M. Almquist, V. Almquist, V. Krishnamoorthi, N. Carlsson, and D. Eager. The Prefetch Aggressiveness Tradeoff in 360° Video Streaming. In *ACM Multimedia Systems Conference (MMSys)*, 2018.

[24] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM*, 2015.

[25] A. H. Zahran, D. Raca, and C. Sreenan. ARBITER+: Adaptive Rate-Based Intelligent HTTP Streaming Algorithm for Mobile Networks. *IEEE Trans. on Mobile Computing*, 2018.

[26] C. Yue, R. Jin, K. Suh Y. Qin, B. Wang, and W. Wei. Linkforecast: Cellular Link Bandwidth Prediction in LTE Networks. *IEEE Trans. on Mobile Computing*, 2018.

[27] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks. *IEEE Communication Letters*, 2016.

[28] X. Chen, T. Tan, and G. Cao. Popularity-Aware 360-Degree Video Streaming. In *IEEE INFOCOM*, 2021.

[29] W. Hu and G. Cao. Energy-Aware Video Streaming on Smartphones. In *IEEE INFOCOM*, 2015.

[30] J. Wu, B. Cheng, M. Wang, and J. Chen. Energy-Efficient Bandwidth Aggregation for Delay-Constrained Video over Heterogeneous Wireless Networks. *IEEE J. Selected Areas in Communications*, 2017.

[31] Y. Yang, W. Hu, X. Chen, and G. Cao. Energy-Aware CPU Frequency Scaling for Mobile Video Streaming. *IEEE Trans. on Mobile Computing*, Nov. 2019.

[32] X. Chen, T. Tan, G. Cao, and T. La Porta. Context-Aware and Energy-Aware Video Streaming on Smartphones. *IEEE Trans. on Mobile Computing*, March 2022.

[33] Z. Yan and C. W Chen. RnB: Rate and Brightness Adaptation for Rate-Distortion-Energy Tradeoff in HTTP Adaptive Streaming over Mobile Devices. In *ACM MobiCom*, 2016.

[34] X. Chen, T. Tan, and G. Cao. Energy-Aware and Context-Aware Video Streaming on Smartphones. In *IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2019.