Macrotile: Toward QoE-Aware and Energy-Efficient 360-Degree Video Streaming

Xianda Chen, *Student Member, IEEE,* Tianxiang Tan, *Student Member, IEEE,* Guohong Cao, *Fellow, IEEE,*

Abstract—Tile-based streaming techniques have been widely used to save bandwidth in 360° video streaming. However, it is a challenge to determine the right tile size which directly affects the bandwidth usage. Moreover, downloading and processing many small tiles consume a large amount of energy on mobile devices. To solve this problem, we propose to encode the video by taking into account the viewing popularity, where the popularly viewed areas are encoded as macrotiles. We propose techniques for identifying and building macrotiles, and adjusting their sizes to take into account practical issues such as head movement randomness. In some cases, the user's viewing area may not be covered by the constructed macrotiles, and then the conventional tiling scheme is used. To support macrotile based 360° video streaming, the client selects the right tiles (a macrotile or a set of conventional tiles) with the right quality level to maximize the QoE under bandwidth constraint. We formulate this problem as an optimization problem which is NP-hard, and then propose a heuristic algorithm to solve it. Through extensive evaluations based on real head movement traces, we demonstrate that the proposed algorithm can significantly improve QoE, save bandwidth usage, and reduce energy consumption.

Index Terms—360-Degree Video Streaming, Viewing Popularity, Quality of Experience (QoE), Energy Efficiency

1 Introduction

360° video is becoming more and more popular on video platforms such as YouTube and Facebook [1], [2]. Since 360° video is much larger than conventional video under the same perceived quality [3], [4], [5], streaming 360° video is much more challenging, especially over wireless (e.g., cellular) networks with limited bandwidth.

Many researchers [6], [7], [8] have addressed this challenge by only downloading part of the video. Since mobile devices have limited Field-of-View (FoV), only a portion of the downloaded video is viewed at a given time. Thus, only the video data within this FoV, instead of the whole video, should be downloaded to save bandwidth. To realize this idea, one widely used approach is the tile-based streaming [9]. In this approach, the video is broken into a sequence of video segments and each segment contains a fixed duration of video. Each segment is further divided into non-overlapping independently decodable *tiles*, each of which is encoded into multiple copies with various qualities. Based on FoV prediction and bandwidth estimation, a user can fetch a subset of tiles encoded at the right quality levels, to reduce the bandwidth usage without compromising the *Quality of Experience (QoE)*.

The tile size can significantly affect the amount of data to be downloaded. Dividing a video into small tiles reduces the efficiency of video encoding. Video codecs, such as H.264 [10] and H.265 [11], use motion compensated prediction technique for video compression, where video frames are encoded by referencing to past or future video frames. Dividing a video into small sized independently decodable tiles reduces the pool of such reference frames within each tile, and then reduces the compression efficiency. Thus, the data size of each encoded tile will be larger and more bandwidth will be consumed. On the other hand, large sized tile can improve the compression efficiency, but more data outside of the FoV will have to be downloaded, and thus consuming more bandwidth. Moreover, in tile-based 360° streaming, since more data has to be downloaded, more energy will be consumed for data transmission.

During video processing, multiple independently encoded tiles covering the viewing area have to be decoded. To reduce the decoding time, multiple decoders are used to simultaneously decode the tiles of the same video segment [3], [5]. However, starting multiple decoders to decode the tiles may lead to more energy consumption, since applying many concurrent decoders affects the video decoding pipeline which involves more CPU context switches and more computational overhead.

To address the aforementioned problems, we propose to encode video by considering the viewing popularity; i.e., users may have similar viewing interests (i.e., viewing areas) when watching the same video. By encoding these users' viewing area as large tile (called *macrotile*) instead of multiple small tiles, high compression efficiency can be achieved, Since the macrotile includes less data outside of the user's viewing area, using macrotile can reduce the amount of data to be downloaded, and then saving bandwidth and energy, compared to existing approaches that download multiple small tiles. Moreover, with macrotile, only one decoder is needed, which reduces the computational overhead and the energy consumption of video processing. To construct macrotiles, we have the following challenges: (1) How to identify the macrotiles? (2) How to determine the right macrotile size? To address these challenges, we exploit the historical viewing data from users watching the same video. Due to their common interests, they may 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 can identify the

- *The authors are with School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA 16802. E-mail: {xuc23, txt51, gxc27}@psu.edu.*

macrotiles. Due to head movement randomness, users may watch the video outside the downloaded macrotile if the macortile is too small. To address this problem, the macrotile is constructed to cover the user's viewing area plus some marginal area, which is determined based on the variations of the user's viewing centers.

In some cases, a user's viewing area may not be covered by the constructed macrotiles, and then the conventional tiling scheme (i.e., the 4x6 tiling scheme) is used. That is, the macrotiles are added to the conventional tiling solution to reduce the bandwidth usage for most users while few users have to use the conventional tiling scheme. To support macrotile based 360° video streaming, we have the following challenges: (1) How to eliminate the impact of head movement randomness? (2) How to determine the right tiles (a macrotile or a set of conventional tiles) and the right quality level such that the QoE is maximized under the network bandwidth constraint? To address these challenges, we propose a macrotile based 360° video streaming algorithm, which first predicts the user's viewing area for each video segment, and then prefetches the corresponding macrotile or conventional tiles if necessary. We formulate the problem as an optimization problem and propose an algorithm to solve it.

The paper has the following contributions.

- We encode the video by considering the viewing popularity, where the popularly viewed areas are encoded as macrotiles to save bandwidth.
- Through real measurements, we identify the energy inefficiency problem of tile-based 360° video streaming and apply macrotile based approach to save energy.
- We formulate the macrotile based 360° video streaming problem as an optimization problem. Since the problem is NP-hard, we propose a heuristic based algorithm to solve it.
- Based on real head movement traces, we evaluate the performance of the proposed algorithm. Evaluation results show that our algorithm can significantly improve QoE, save bandwidth, and save energy.

The remaining of this paper is organized as follows. In Section 2, we introduce the background and motivation. The system model and the problem formulation are presented in Section 3. Section 4 presents our macrotile based 360° video streaming algorithm. In Section 5, we present the evaluation results. Section 6 discusses related work and Section 7 concludes the paper.

2 Background and Motivation

In this section, we first introduce the background of 360° video streaming, and then give the motivation of our work.

2.1 Background

Different from conventional video streaming, 360° video streaming provides content-rich immersive user experience, i.e., a user can navigate in a virtual world by looking around to interact with the virtual world. 360° video can be viewed through dedicated head mounted display, such as Oculus [12] and HTC Vive [13], or by placing smartphones in headsets like Google Cardboard [14] and Samsung Gear VR [15]. In tile-based 360° video steaming, the device first downloads encoded 360° video (i.e., video tiles) from the video servers. During video processing, to decode multiple encoded tiles covering the viewing area in time, multiple decoders are commonly used to concurrently decode the tiles of the same video segment [3], [5]. After video decoding, the original 360° frames are retrieved and buffered in the video buffer, waiting to be rendered. Unlike the conventional 2D video processing where the decoded frames can be directly displayed on the screen, in 360° video processing, the actual viewing video content (called FoV frame) is rendered before being displayed. Based on user's head orientation, a coordinated projection is performed to map the 3D coordinates of the viewing area to 2D coordinates, and accordingly generate the FoV frames. After projection, the display processor reads the generated FoV frames from the video buffer and displays on the screen.

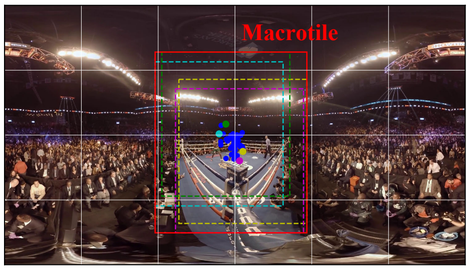
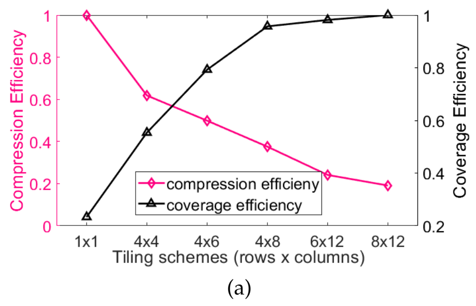


Fig. 1: A 360° video with 4x6 tiling.

The tile size can significantly affect the amount of data to be downloaded and processed. To illustrate this, we conducted experiments based on the head movement data traces of 48 users watching a 360° video [16]. The video has 4K resolution (i.e., 3840x2160) with 30 frames per second (fps). The video is divided into a series of video segments. Each segment has one second of video, which is further divided into tiles based on the commonly used tiling schemes, represented by (rows x columns); i.e., 1x1 (no tiling), 4x4 [17], 4x6 [5], [18], [19], [20], 4x8 [21], 6x12 [22], and 8x12 [21]. For fair comparison, FFmpeg [23] with encoder x264 is applied to crop and encode the tiles with the same encoding parameters. The user's viewing area is determined by the viewing center and the FoV of the device, i.e., 100 degrees horizontally and vertically [19], [24], [25], [26]. As shown in Figure 1, the video is divided into 4 rows and 6 columns, i.e., 24 tiles. Each dot represents the viewing center of one user. The dashed yellow, cyan, green, and purple blocks represent the rightmost, leftmost, up-most, and down-most viewing areas of all users (i.e., the group of users in Figure 1), respectively.

2.2 Motivation for Saving Bandwidth

We use Figure 2 to identify the bandwidth inefficiency problem of traditional tile-based approach, which provides motivation for our macrotile based approach. In Figure 2(a), we compare the effectiveness of different tiling approaches. The coverage ratio is the video data within FoV divided by all downloaded video data. The boundary area of some tiles may not be within FoV, and then the boundary area (the coverage ratio) will be different for different tiling schemes.



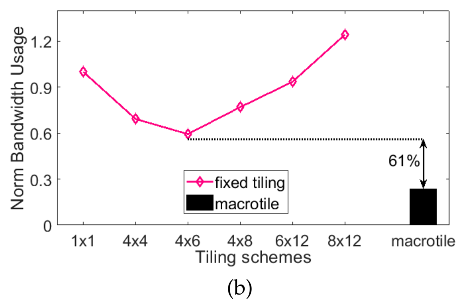


Fig. 2: (a) Compression efficiency and coverage efficiency of different tiling schemes. (b) A comparison of bandwidth usage.

To compare them, we define a metric called *coverage efficiency*, which is the coverage ratio normalized based on the largest coverage ratio of these tiling schemes. Similarly, the *compression efficiency* is the compression ratio normalized based on the largest compression ratio of these tiling approaches, where the compression ratio [26] is defined as the total amount of data needed to represent the tiles covering the FoV divided by the total encoded data. From Figure 2(a), we can see that smaller tiles (e.g., 8x12) achieves higher coverage efficiency while larger titles (e.g., 1x1) achieves higher compression efficiency. This is because the spatial/temporal redundancy with large tiles can be easily identified and removed to achieve higher compression efficiency, but such redundancy is harder to identify with small tiles.

The downloaded video data (or the bandwidth usage) is related to the coverage efficiency and compression efficiency. In Figure 2(b), we draw the bandwidth usage, normalized based on the 1x1 tiling scheme, of different tiling schemes. Compared to the 1x1 tiling scheme which delivers the entire video, data size can be reduced when the video is divided into smaller tiles and then only tiles covering the user's viewing area are downloaded. However, when the tiles are too small (smaller than 4x8), the compression efficiency drops and thus increasing the amount of downloaded data. For example, the bandwidth usage of 8x12 tiling becomes larger than that of 1x1 tiling. Thus, how to find the right tiling is a challenge.

To address this challenge, we construct a large sized tile (i.e., the red block in Figure 1), called macrotile, which covers all viewing areas (dashed blocks). This macrotile has high compression efficiency because of the large tile size. It has high coverage efficiency because only one large tile is used instead of 12 tiles, represented by the cyan blocks. On the other hand, none of the conventional tiling approaches can achieve both high compression efficiency and high coverage efficiency. As can be seen from Figure 2(b), compared to 4x6 tiling (the best scheme), macrotile further cuts the bandwidth usage by 61%.

2.3 Motivation for Saving Energy

We use Figure 3 to identify the energy inefficiency problem of existing approach, which provides motivation for our macrotile based approach. Video processing includes two parts: video decoding and view generation. In existing tile-based approach, to accelerate video decoding, multiple decoders can be applied to decode the tiles in parallel; however, this also increases the energy consumption. Figure 3(a) shows the tradeoff between decoding time and power consumption in existing tile-based approach (i.e., the 4x6 tiling scheme). Here, we use hardware-accelerated media codec (i.e., MediaCodec) to achieve real-time decoding. The power consumption of video decoding is the difference between the total power and that of the idle system. As can be seen from Figure 3(a), when the number of decoders increases, the decoding time drops but the power consumption increases. This is because applying multiple concurrent decoders makes the video decoding pipeline much complex, which leads to tedious CPU context switches and high computational overhead. For instance, when the number of decoders increases from 1 to 9, the total decoding time decreases from 1.17 sec to 0.48 sec (around 2.5X), but the power consumption increases from 229 mW to 785 mW (around 3.5X).

To illustrate the energy consumption of video processing (i.e., video decoding and view generation) in conventional tile-based approach, Figure 3(b) draws the energy consumption as a function of the number of decoders used, normalized based on that of using one decoder. After decoding, based on the coordinate mapping, the view is generated by drawing the pixel values onto the display (more details in Section 5). As shown in the figure, the energy consumption decreases as the number of decoders reduces from 1 to 4, and then increases as more decoders are used. On the other hand, the macrotile approach uses only one decoder which has low decoding time (less number of tiles) and low power consumption, i.e., it can cut the energy consumed for video processing by 44% compared to using four decoders (the best solution) in conventional tile-based approach.

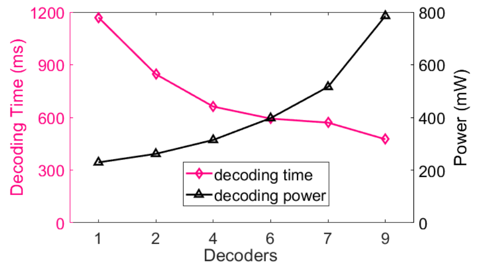
Based on the results shown in Figure 2 and Figure 3, we can see the benefits of the macrotile approach on saving bandwidth and energy. To construct macrotiles, we exploit the historical viewing data from users watching the same video. Since most users have common interests, they may have similar viewing areas and their viewing centers are close to each other. In Section 4, we present techniques to identify and cluster these viewing centers, based on which macrotiles can be constructed.

3 System Model and Problem Formulation

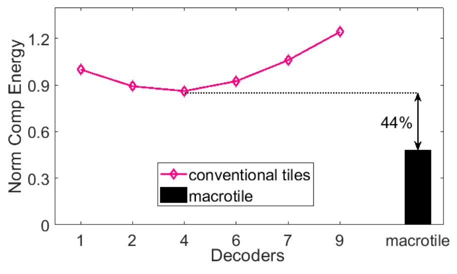
In this section, we present the video model, the QoE model, and the problem formulation for 360° video streaming.

3.1 Video Model

The video is divided into a sequence of video segments and each segment has a fixed duration of video. Each



(a)



(b)

Fig. 3: (a) Time and power consumed for decoding a video segment in conventional tile-based approach. (b) A comparison of energy consumption in video processing.

segment is further divided into C tiles using a conventional tiling scheme (e.g., 4x6). To save bandwidth and energy, M macrotiles are constructed (details in Section 4). At the server, each tile (and macrotile) is encoded into V copies corresponding to V different qualities. The 360° video streaming can be viewed as a sequence of downloading tasks. For each task, the client selects the right tiles (a macrotile or a set of conventional tiles) with the right quality. Let L denote the video length of the downloaded but not yet viewed video in the buffer, in terms of seconds, when the client requests the tiles. To avoid stall events (or rebuffering), the tiles should be completely downloaded before the buffer is drained out ($L = 0$) by the video player at the client side.

3.2 QoE Model

The user's perceived QoE for watching a video is defined as the average QoE values for all video segments. For each video segment k, similar to [5], [27], [28], [29], the QoE model quantifies the user perceived quality by considering the following metrics: average video quality, quality variation, and rebuffering. The QoE model is defined as follows:

$$Q(\mathcal{V}_k) = Q_0(\mathcal{V}_k) - \omega_v I_v(\mathcal{V}_k) - \omega_r I_r(\mathcal{V}_k) \tag{1}$$

where $\mathcal{V}_k$ represent the video qualities of the tiles being downloaded for segment k, Q_0 is the average quality, I_v is the quality impairment caused by quality variation, I_r is the quality impairment caused by rebuffering event, and ω_v and ω_r are the weights for quality variation and rebuffering, respectively. Q_0, I_v, and I_r are defined as follows.

- *Average Quality*. Because the user perceived quality is only determined by the video content within the viewing area, the average quality is calculated over all tiles in the viewing area, as shown in Eq. 2

$$Q_0(\mathcal{V}_k) = q(\overline{\mathcal{V}_k}) \tag{2}$$

  where $\overline{\mathcal{V}_k}$ represents the average video quality (i.e., video bitrate) of the tiles in the viewing area, $q(.)$ is a mapping function that maps the video quality of a segment to the user perceived quality [6].
- *Quality Variation*. The quality variation between two consecutive video segments may cause users discomfort such as dizziness, and thus should be considered in the QoE model. When a user downloads a set of tiles, the quality variation of these tiles will affect the user's perceived quality, and should also be considered in the QoE model [5]. The following equation defines the quality variation.

$$I_v(\mathcal{V}_k) = |Q_0(\mathcal{V}_k) - Q_0(\mathcal{V}_{k-1})| + \widehat{\mathcal{V}_k} \tag{3}$$

  where $|Q_0(\mathcal{V}_k) - Q_0(\mathcal{V}_{k-1})|$ represents the inter-segment temporary quality variation (i.e., the quality variation between the k^{th} and $(k-1)^{th}$ video segment), $\widehat{\mathcal{V}_k}$ represents the intra-segment spacial quality variation, which is calculated as the standard deviation of $\mathcal{V}_k$ [5].
- *Rebuffering*. Rebuffering will significantly affect the QoE since the video will freeze during rebuffering events. The rebuffering time is defined as follows.

$$I_r(\mathcal{V}_k) = (\frac{S(\mathcal{V}_k)}{R} - L, 0)_+ \tag{4}$$

  where $S(\mathcal{V}_k)$ is the segment data size, R is the downloading throughput, and $(x)_+ = max\{x, 0\}$.

3.3 Problem Formulation

Before formalizing the macrotile based 360° video streaming problem, we introduce some notations. Let β^v_m (β^v_c) represent if the corresponding macrotile (or conventional tile) will be downloaded. Specifically, $\beta^v_m = 1$ if the macrotile m encoded at quality level v is downloaded, and the bandwidth usage is B^v_m; otherwise $\beta^v_m = 0$. $\beta^v_c = 1$ if the tile c encoded at quality level v is downloaded, and the bandwidth usage is B^v_c; otherwise $\beta^v_c = 0$. A user should download the macrotile to cover his viewing area. If such macrotile does not exist, or not enough to cover his viewing area, a set of conventional tiles will be downloaded.

In our macrotile based approach, the goal is to maximize the user's perceived QoE under the network bandwidth constraints. This can be achieved by selecting the right tiles (a macrotile or a set of conventional tiles) with the right quality level for each video segment. We formalize this optimization problem as follows.

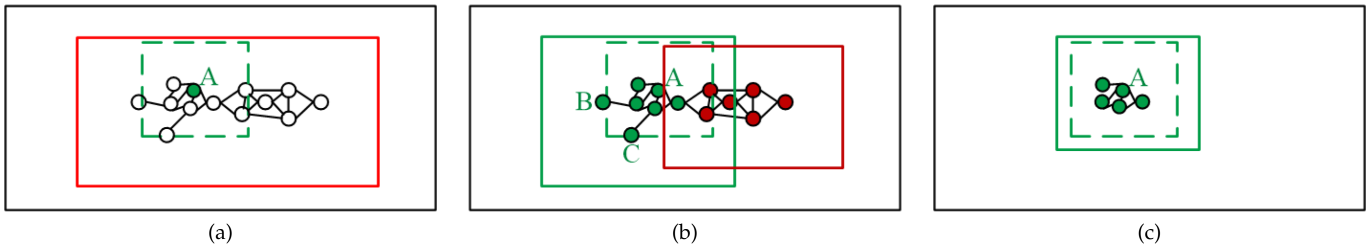


Fig. 4: Macrotile construction. (a) Macrotile is too large. (b) Splitting a large macrotile into two macrotiles. (c) Macrotile optimization (only shows the green one).

$$\max \quad Q(\{v \mid \forall_{m,v}\beta_m^v = 1\}) + Q(\{v \mid \forall_{c,v}\beta_c^v = 1\}) \tag{5}$$

$$\text{s.t.} \quad \sum_{m=1}^{M}\sum_{v=1}^{V}\beta_m^v + \mathbf{1}\Big(\sum_{c=1}^{C}\sum_{v=1}^{V}\beta_c^v\Big) = 1 \tag{5a}$$

$$\sum_{v=1}^{V}\beta_c^v \le 1, \qquad for\ c = 1, ..., C \tag{5b}$$

$$\sum_{m=1}^{M}\sum_{v=1}^{V}\beta_m^v B_m^v + \sum_{c=1}^{C}\sum_{v=1}^{V}\beta_c^v B_c^v \le R \cdot L \tag{5c}$$

where $Q(.)$ is the QoE as defined in Eq. 1, R is the network bandwidth, and $\mathbf{1}(x) = 1$ if and only if $x > 0$, otherwise $\mathbf{1}(x) = 0$. Constraint (5a) enforces that either a macrotile or a set of conventional tiles is downloaded for the viewing area. Constraint (5b) states that only one quality version of a conventional tile is downloaded. Similarly, only one quality version of a macrotile is downloaded, which can be inferred from Constraint (5a). Constraint (5c) guarantees that the video data can be successfully downloaded before its playback.

Given the user's viewing area, that is, assuming the candidate macrotile (and the candidate set of conventional tiles) covering the viewing area is known, the problem in Eq. 5 can be decomposed into two sub-problems: one is to determine the right quality level for the macrotile and the other is to determine the right quality levels for the tiles. Then, the one with better QoE will be the solution for Eq. 5. If the QoE model does not consider the quality variation of the tiles, i.e., the overall QoE will be the average quality level of the downloaded tiles, the latter sub-problem can be simplified as Eq. 6, where $\mathcal{C}$ is the set of conventional tiles covering the viewing area.

$$\max \quad \sum_{c\in\mathcal{C}}\sum_{v=1}^{V} Q(\beta_c^v v) \tag{6}$$

$$\text{s.t.} \quad \sum_{v=1}^{V}\beta_c^v = 1, \quad for\ c \in \mathcal{C} \tag{6a}$$

$$\sum_{c\in\mathcal{C}}\sum_{v=1}^{V}\beta_c^v B_c^v \le R \cdot L \tag{6b}$$

Lemma 1. *The problem in Eq. 6 is NP-hard.*

Proof. The problem can be proved to be NP-hard via a reduction from the multiple-choice knapsack (MCK) problem. In the MCK problem, there are a number of classes of items in which each item has a value and weight. Given a knapsack with a weight limit, the problem is to choose one item from each class such that the total value is maximized and the total weight is no more than the weight limit.

For any instance of the MCK problem, we can construct an instance of the problem in Eq. 6 in the following way. We construct a tile c as a class, where the quality versions (V) of this tile corresponds to the items of the class. For the v^{th} version, its quality level v is set to the value of the v^{th} item, and its bandwidth usage B_c^v is set to the weight of the v^{th} item. The network bandwidth limit RL is set to the weight limit of the knapsack.

A solution to this instance of the problem in Eq. 6 maximizes the total quality of the tiles. When the quality versions are seen as items, the solution chooses one item from each class to maximize the total value of items under the weight constraint. Therefore, the solution to this problem is also a solution to the MCK problem, which completes the reduction and hence the proof. □

Theorem 1. *The macrotile based 360° video streaming problem is NP-hard.*

Proof. The macrotile based 360° video streaming problem in Eq. 5 is much harder than the problem in Eq. 6, because the problem in Eq. 6 is a sub-problem of Eq. 5. Based on Lemma 1, the problem in Eq. 5 is NP-hard, and thus the macrotile based 360° video streaming problem is NP-hard. □

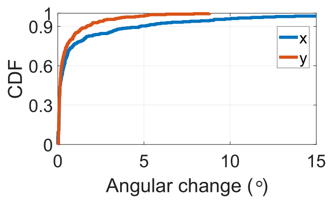
Because the macrotile based 360° video streaming problem is NP-hard, we can only propose a heuristic based algorithm.

4 Macrotile Based 360° Video Streaming

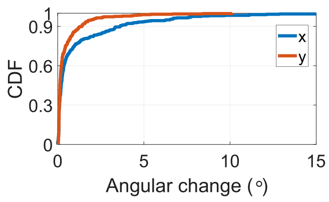
In this section, we first describe how to construct macrotiles and then present our macrotile based 360° video streaming algorithm.

4.1 Identifying Macrotiles based on Viewing Areas

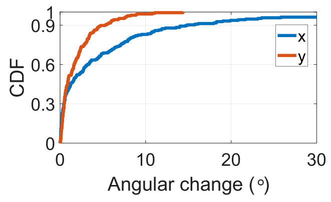
Most users have similar viewing interests when watching the same 360° video. Thus, they have similar viewing areas and their viewing centers are close to each other. To construct macrotiles, we have to first identify these viewing centers, and cluster them together. Since the number of clusters (macrotiles) is not known as a priori, many well-known clustering algorithms such as k-means clustering, cannot be directly applied. Although other non-parametric clustering algorithms such as the density-based clustering



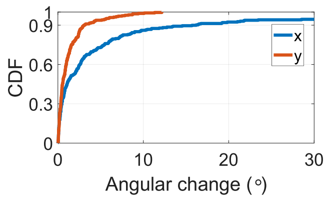
(a) Video 1



(b) Video 2



(c) Video 7



(d) Video 8

Fig. 5: CDFs of angular changes along x and y directions for a user (only showing four to save space).

algorithm (DBSCAN) [30] do not need to know the number of clusters before hand, they may lead to another problem. That is, the cluster may keep growing and become too large, losing the benefit of saving bandwidth and energy. For example, in Figure 4(a), since the viewing centers of a cluster span a large area, the constructed macrotile (the red block) becomes too large. Then, the benefits of using macrotile to save bandwidth will be lost. To address these problems, we propose the following clustering algorithm.

There are two important parameters, λ and γ in the algorithm. λ 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 λ. The clustering performance is affected by λ. If λ is too small, some viewing centers may not be clustered together even though they should. If λ is too large, the cluster may include viewing centers far away, i.e., users with different viewing interests are clustered together. The size of a cluster is determined by γ, i.e., the distance between any two viewing centers in the cluster should not be farther than γ. If γ is too large, the cluster may grow too large. By contrast, if γ is too small, too many clusters may be constructed. To determine λ and γ, we also need to consider the effects of the conventional tile size, since the macrotiles are built on top of the conventional tiling scheme to reduce the bandwidth usage. Take the 4x6 tiling scheme as an example, as shown in Figure 1. Some users sharing similar viewing interest download two columns of tiles, while others download three columns of tiles; i.e., the difference is one column of conventional tiles. In Section 5.2, we will set up λ and γ based on experiments.

Let P denote a set of points, where each point ($p \in P$) represents the viewing center of a user. Let $dist(p,q)$ denote the Euclidean distance between two points p and q. Let $N_p = \{q \mid q \in P \wedge q \neq p \wedge dist(p,q) \leq \lambda\}$ denote p's close neighbors. The clustering algorithm is as follows.

(a) Initiate the cluster with the point that has the maximum number of close points, i.e., $p = \text{argmax}_{p \in P} |N_p|$.
(b) Expand the cluster by adding points which are close to any point inside the cluster. The expanding process continues until no more close points can be found.
(c) Check if the maximum distance between any two points in the cluster is larger than γ. If so, the cluster is split into two clusters using k-means clustering algorithm.
(d) Remove the clustered points from P.
(e) Repeat step (a) to (d) until $P = \emptyset$.

4.2 Macrotile Optimization

Based on the above algorithm, we can cluster users' viewing centers. If a macrotile is constructed for each cluster to cover all viewing areas of the users, the macrotile may be too large. The problem is illustrated in Figure 4(b), where the macrotile represented by the green block is much larger than the user's viewing area (the dashed green block). On the other hand, due to head movement randomness, users may watch the video outside of the downloaded macrotile if it is too small. Therefore, it is important to find the right macrotile size based on the cluster.

To determine the right size of the macrotile, we need to decide which users' viewing areas should be included, such that the bandwidth usage of downloading the macrotile (B_m) is less than that of downloading a set of conventional tiles (B_c). B_m and B_c denote the data size of the constructed macrotile and the data size of the conventional tiles covering the same viewing area, respectively.

To address the impact of head movement randomness, the macrotile should cover the user's viewing area plus some marginal. The marginal area can be determined based on the variation of the user's viewing centers (i.e., x and y coordinates), which are recorded at a fixed sampling rate (e.g., 50 Hz) during video streaming. The variation of x (y) coordinates within a video segment is defined as the standard deviation of the x (y) coordinates. Figure 5 shows such variations when a user watches a video [16], listed in Table 1. The user's head movement data are collected under two different settings. For videos 1 to 4, users are instructed to focus on the video content. For videos 5 to 8, users are free to explore the video; i.e., the variation can be affected by the video content and the user's unique viewing behavior. As can be seen from Figure 5, the variation of x (y) coordinates within a video segment is small. The variations on x and y directions are presented in terms of degrees, which can be converted to pixels by multiplying the video resolution. Let A_x and A_y denote the variations along x and y directions, respectively. Then, the constructed macrotile should cover the user's viewing area plus $\frac{A_x}{2}$ ($\frac{A_y}{2}$) marginal area on both sides of its x (y) direction.

To formalize the problem of macrotile construction, a binary variable α_i is introduced for user i, where $\alpha_i = 1$ if the user's viewing area is a macrotile, i.e., the user is able to download the constructed macrotile; otherwise, $\alpha_i = 0$, i.e., the user downloads a set of conventional tiles. The problem of macrotile construction can be formulated with Eq. 7, where the goal is to minimize the total bandwidth usage for all users in a cluster when downloading either the constructed macrotile or a set of conventional tiles encoded at the same quality level.

$$\min_{\{\alpha_i\}} \quad \sum_{i=1}^{N_j} \alpha_i B_m + (1-\alpha_i) B_c \tag{7}$$

TABLE 1: Video traces.

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

where N_j is the number of users in the j^{th} cluster. After solving Eq. 7, we can construct the macrotile with all $\alpha_i = 1$ users' viewing areas.

Although a brute force search can find an optimal solution for Eq. 7, its computational complexity is $O(2^{N_j})$. To reduce the computation time, we propose an iterative approach, similar to the random sample consensus paradigm [31]. Each iteration has the following steps.

(a) Randomly select a subset of users' viewing areas.
(b) Encode the macrotile, and let B_m denote the bandwidth usage for the constructed macrotile.
(c) Check if user $i \in \{1, ...N_j\}$ is covered by the constructed macrotile. If so, $\alpha_i = 1$; otherwise, $\alpha_i = 0$.
(d) If the total bandwidth usage (i.e., $\sum_{i=1}^{N_j} \alpha_i B_m + (1 - \alpha_i) B_c$) is less than that of the previous iteration, update the macrotile with that constructed in the current iteration.

If the user only downloads the macrotile covering the predicted viewing area, some area may be blank when the user suddenly navigates outside of the downloaded macrotile. To address this problem, in addition to downloading the tiles (or macrotile) covering the viewing area with high quality, the remaining tiles are also downloaded, but with the lowest quality. More specifically, for each constructed macrotile, as shown in Figure 4(c), we crop the remaining area into four parts by cutting the video along the two horizontal edges of the constructed macrotile. These four parts are also downloaded, but the extra bandwidth usage is very small, since the compression efficiency is high and these videos are encoded with the lowest quality level.

4.3 Macrotile Based 360° Video Streaming

In this subsection, we propose a macrotile based 360° video streaming algorithm, which first predicts the user's viewing area for each video segment, and then prefetches the corresponding macrotile or the conventional tiles if necessary. The goal is to select the right tiles (a macrotile or a set of conventional tiles) with the right quality level such that the QoE is maximized under the network bandwidth constraint.

To predict the user's viewing area (i.e., the viewing center), we use the ridge regression model [32] since it can deal with overfitting problems. 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.

Algorithm 1: Macrotile Based 360° Video Streaming

```
Input: $r$, $L$, $V$
Output: $\beta_m^v$ or $\{\beta_c^v\}$
1  Predict the user's viewing area
2  Determine the macrotile $m$ and the set of tiles $\mathcal{C}$
3  if $m$ exists then
4      return $SelectMacrotile(m, r, L)$
5  else
6      return $SelectCtilings(\mathcal{C}, r, L)$
7  end
8
9  function SelectMacrotile($m$, $r$, $L$):
10     for $v \leftarrow V$ to 1 do
11         if $B_m^v \le r \cdot L$ then
12             return $\beta_m^v$
13         end
14     end
15     return $\beta_m^{v_1}$    // $v_1$ is the lowest quality level
16 end
17 function SelectCtilings($\mathcal{C}$, $r$, $L$):
18     for $v \leftarrow V$ to 1 do
19         if $\sum_{c \in \mathcal{C}} B_c^v \le r \cdot L$ then
20             $\beta_c^v = 1 \quad for\ c \in \mathcal{C}$
21         end
22     end
23     $r' = r \cdot L - \sum_{c \in \mathcal{C}} B_c^v$ // the remaining bandwidth
24     sort($\mathcal{C}$) // sort tiles in ascending order of distance
25     foreach $c \in \mathcal{C}$ do
26         if $r' >= (B_c^{v+1} - B_c^v)$ then
27             $\mathcal{C}' = \mathcal{C}' \cup \{c\}$
28             $r' = r' - (B_c^{v+1} - B_c^v)$
29         end
30     end
31     if $|\mathcal{C}'| > \frac{1}{2}|\mathcal{C}|$ then
32         $\beta_c^{v=v+1} = 1 \quad for\ c \in \mathcal{C}'$
33     end
34     return $\{\beta_c^v\}$
35 end
```

Based on the predicted viewing area, the algorithm determines whether there is a macrotile which can cover the predicted viewing area plus some marginal area. If such a macrotile exists, the algorithm searches from the highest quality level until finding the highest possible quality level for the macrotile such that the macrotile encoded at this quality level can be successfully downloaded (line 9-16 in Algorithm 1).

In some cases, such macrotile may not exist, and then conventional tiles will be downloaded. The algorithm will determine the quality levels of these conventional tiles (lines 17-35 in Algorithm 1), and the details are as follows. The algorithm first determines the highest possible quality level for these tiles that can be successfully downloaded under the network condition. If the remaining bandwidth is large enough, the algorithm increases the quality level to one level higher for some tiles. Since tiles closer to the viewing center may have larger impact on user's perceived quality, the algorithm increases the quality of the tiles based on the distance between the viewing center and the center of the tile, i.e., tiles are sorted in the ascending order of their distance to the viewing center. The algorithm searches from the tile closest to the viewing center, and finds the maximum number of tiles that can increase the quality level (by one) based on the amount of remaining bandwidth. As the QoE is affected by quality variations, the quality level increase is performed only when more than half of the downloaded tiles can increase their quality levels.

5 Performance Evaluations

In this section, we evaluate the performance of the proposed macrotile based 360° video streaming algorithm and compare it to existing solutions.

5.1 Experiment Setup

The performance evaluation is based on the head movement data traces of 48 users watching eight 360° videos [16]. Table 1 shows the details of the eight videos, which cover different scenarios such as performance, TC show, sports, etc. For each video, we randomly select forty users' head movement data traces to construct video tiles (and macrotiles), and the other eight data traces are used to evaluate the performance.

The 360° video streaming system, as shown in Figure 6, consists of two major components, the server and the client. The sever module constructs video tiles (and macrotiles) and handles video requests from the client. Similar to DASH, the video server delivers the meta data of video tiles (and macrotiles) for each video segment to the client, and the client makes decision on selecting the right tiles (a macrotile or a set of conventional tiles) with the right quality level based on the viewing area prediction and bandwidth estimation. During video playback, the tiles in the viewing area are first decoded and stitched, and then played by the player. The sensors embedded in the mobile device will collect the user's head movement data, which can be used for viewing area prediction.

At the server, similar to [5], [27], [33], each video is divided into a sequence of segments. Each segment lasts one second, and it is further divided spatially into tiles (and macrotiles). Then, we use FFmpeg [23] with encoder x264 to encode all tiles (and macrotiles) into five quality levels (5 to 1, with 5 being the highest quality) using different constant rate factor (crf) values from 18 to 38 with an interval of 5 [5].

At client, the FoV of the mobile device is set to be 100 degrees horizontally and vertically. Similar to [3], the harmonic mean of the downloading throughput of the past several segments is used to estimate the network bandwidth, which can eliminate the impacts of network fluctuations. More bandwidth estimation methods can be found in [34], [35], which is out of the scope of this paper. Similar to [5], we use the ridge regression model to predict the user's viewing area (i.e., the viewing center). For the QoE model, we set the weights as (ω_v, ω_r) = (0.25, 0.25), which is a used setting in [29].

In our evaluation, a LTE network throughput trace [36] is used to simulate the network traffic. This trace shows various patterns to reflect various network traffic. We also linearly scale the trace to generate two different network

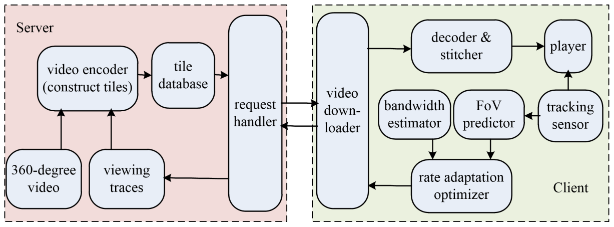


Fig. 6: The 360° video streaming system.

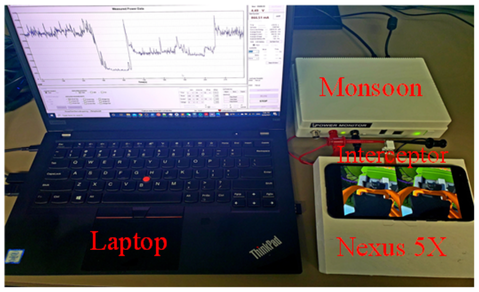


Fig. 7: Experimental setup for power measurement.

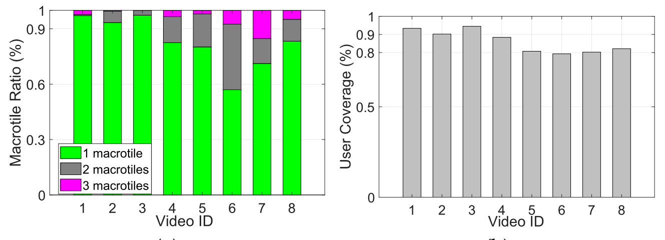
conditions, called *trace 1* and *trace 2*, where the network throughput of trace 1 is twice that of trace 2. In trace 2, the average throughput is 4.8 Mbps, varying between 1.3 Mbps and 10.9 Mbps.

In 360° video streaming, power can be consumed for data transmission, video decoding, and view rendering. We measure the power consumption using three types of phones, i.e., Google Pixel 3, LG Nexus 5X, and Samsung Galaxy S20. Because the battery connectors on the smartphone are very small, it is a challenge to connect them to the power monitor. To address this problem, we build a new battery interceptor based on Flex Printed Circuit Boards. The interceptor can be connected to the smartphone's motherboard through the corresponding battery connector, and it uses a custom designed circuit to modify the battery connection. As shown in Figure 7, with this custom designed interceptor, the smartphone can be connected to the Monsoon power monitor, which directly supplies power to the smartphone and accurately measures the power consumption.

We compare the performance of our macrotile based 360° video streaming algorithm (i.e., *Mtiling*) with the following two approaches. The first is the conventional tiling approach (*Ctiling*) which has been widely used in [5], [18], [19], [20], [37]. In this approach, each video segment is divided into files with fixed size using a conventional tiling scheme (e.g., 4x6). The second approach is the fixed tiling approach (*Ftiling*), where each video segment is divided into a fixed number of tiles which may have different sizes. Similar to [25], each segment is first divided into 450 small blocks (i.e., the 15x30 tiling), which are then clustered into ten tiles based on users' views.

5.2 Performance of Macrotile Construction

In the evaluation, we empirically set γ to be the width of a conventional tile and $\lambda = \gamma/4$. To avoid constructing unnecessary macrotiles that cover too few users, a macrotile



(a) (b)

Fig. 8: (a) Video segments with different number of macrotiles. (b) The percentage of users covered by macrotiles.

is only constructed if it covers at least five users (i.e., 10% of the users in the dataset).

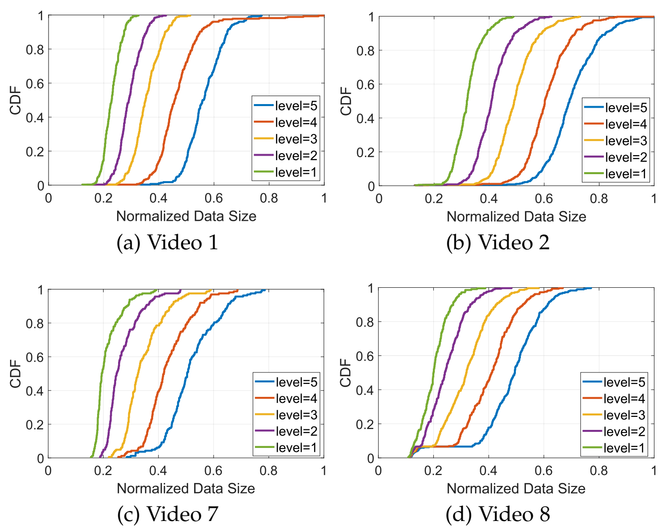
5.2.1 Macrotile Coverage

We first use the eight videos listed in Table 1 to evaluate the how our macrotile construction algorithm performs. Among these eight videos, users are more focusing on the video content in videos 1 to 4, while showing their unique behavior patterns for videos 5 to 8. As shown in Figure 8(a), more than 95% video segments need only one macrotile for videos 1 to 3, because users have similar viewing interests and they are instructed to focus on the video content in these videos. In video 4 (basketball match), although users' gazing directions frequently move, more than 96% video segments only require one or two Macrotiles. Different from videos 1 to 4, users in videos 5 to 8 are free to explore. As a result, more Macrotiles are needed. As shown in the figure, even under this setting, Only one or two Macrotiles are needed for more than 92% video segments.

From Figure 8(b), we can see that most users are covered by the Macrotiles. For example, in videos 1 to 4, about 90% of users can be served by the Macrotiles. Even for videos 5 to 8 where users are free to explore, more than 80% of users are served by the Macrotiles. Since the viewing areas of most users are covered by the Macrotiles, most of them only need to download these Macrotiles to save bandwidth and energy. Note that there are still some users not covered by macrotiles, where conventional tiles are used. This is because the viewing centers of these users are more likely to be far away from those covered by the macrotile, and adding them into the macrotile may waste bandwidth.

5.2.2 Macrotile vs. Conventional Tile

Using Macrotiles can significantly reduce the bandwidth consumption by reducing the amount of data needed for encoding the FoV area. To quantify the data size reduction, we compare the data size of both approaches covering the same area encoded with the same quality level. Figure 9 shows the cumulative distribution function (CDF) of the data size with Macrotile, normalized based on the data size of the corresponding conventional tiles. From the figure, we can see that using Macrotile can significantly reduce the data size, since higher compression efficiency can be achieved by using larger Macrotile instead of smaller conventional tiles. For example, as can be seen in Figure 9(a), with the lowest video quality level (i.e., level 1), the median data size of



(a) Video 1 (b) Video 2 (c) Video 7 (d) Video 8

Fig. 9: The data size CDFs of using macrotile, normalized based on that of the corresponding conventional tiles (Only showing four videos to save space.)

using macrotile is about 22% of that of conventional tile. That is, using macrotile can reduce the bandwidth by 78% (i.e., 1-0.22 = 0.78) compared to using conventional tiles. With similar calculations, downloading macrotiles encoded at quality level 5, 4, 3, and 2 can save bandwidth by 46%, 55%, 65%, and 71%, respectively.

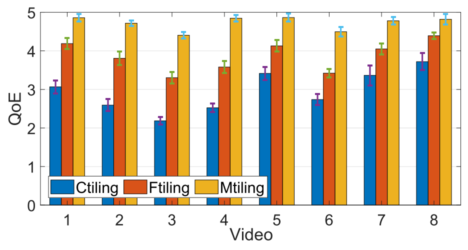
5.3 Performance of Macrotile Based 360° Video Streaming

We evaluate the performance of the macrotile based 360° video streaming based on the user's head movement data trace. Similar to [5], [33], [38], the playback buffer is set to three seconds.

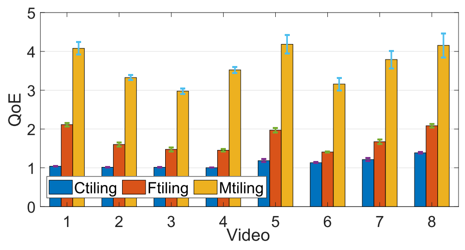
5.3.1 QoE Comparison

In Figure 10, we compare the QoE of our *Mtiling* approach with other solutions (the *Ctiling* approach and the *Ftiling* approach) using two network traces (trace 1 and trace 2) which represent different network conditions. As shown in the figure, our *Mtiling* approach significantly outperforms *Ctiling* and *Ftiling* for both traces. By exploiting the idea of macrotile, *Mtiling* can achieve high encoding efficiency and high coverage efficiency. This is in contrast to the *Ftiling* approach and the *Ctiling* approach, which have to divide the user viewing area into small tiles, and then reducing the encoding efficiency and coverage efficiency. Also, cutting the video region outside the macrotile into small tiles, will reduce the efficiency of video encoding, resulting in higher bandwidth requirements in *Ctiling* and *Ftiling*. As a result, with the same amount of network bandwidth, users using our *Mtiling* approach can download macrotiles with higher video quality, whereas users in *Ctiling* and *Ftiling* can only download small tiles with lower video quality and thus reducing the QoE.

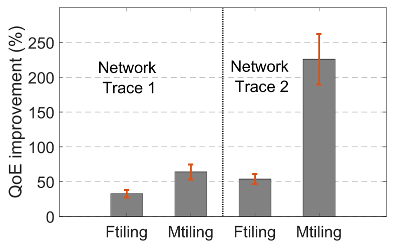
The QoE difference will be more clear when the network bandwidth becomes the bottleneck. More specifically, when the network bandwidth is cut by half, i.e., from network



(a) Network trace 1

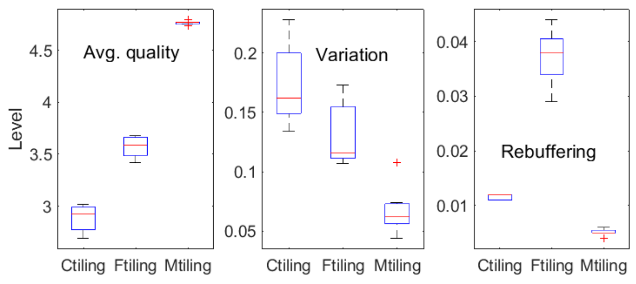


(b) Network trace 2

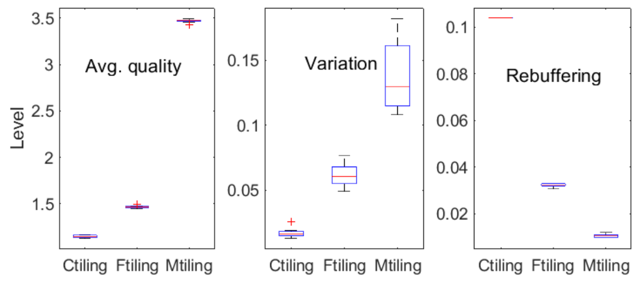


(c) Overall QoE improvement

Fig. 10: QoE comparison.

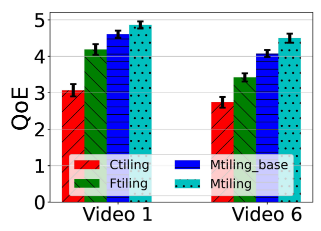


(a) Network trace 1

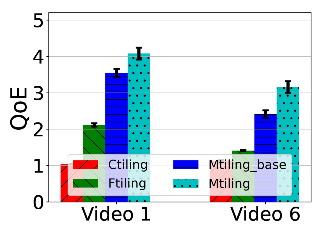


(b) Network trace 2

Fig. 11: QoE comparisons based on video quality, quality variation, and rebuffering.

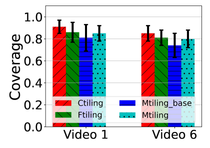


(a) Network trace 1

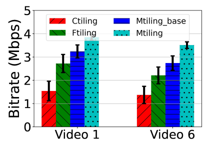


(b) Network trace 2

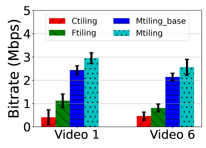
Fig. 12: Ablation study.



(a)



(b)



(c)

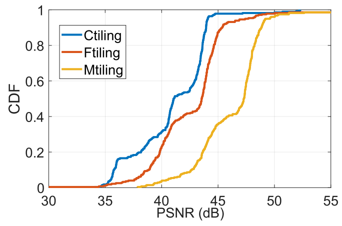
Fig. 13: Ablation study: (a) The FoV coverage ratio. (b) Consumed bitrate in network trace 1. (c) Consumed bitrate in network trace 2.

trace 1 in Figure 10(a) to network trace 2 in Figure 10(b), the QoE in *Mtiling* only drops a little bit, but the QoE in *Ctiling* and *Ftiling* significantly drops. For example, in video 1, when the network condition changes from trace 1 to trace 2, the QoE for the *Ctiling* approach drops by 66.1%, the QoE for *Ftiling* drops by 49.5%, but the QoE for our *Mtiling* only drops by 16.1%.

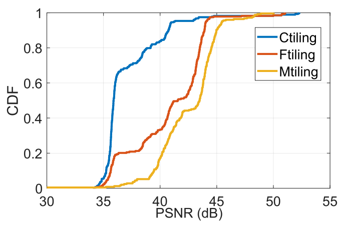
Figure 10(c) summarizes the results in Figure 10(a) and Figure 10(b) by averaging the results of all eight videos. It uses the *Ctiling* approach as the baseline, and shows the QoE improvement of our *Mtiling* approach and the *Ftiling* approach. When network trace 1 is used in the evaluation, the *Mtiling* approach can improve the QoE by 64.1%, and *Ftiling* can improve the QoE by 32.5%. When network trace 2 is applied, our *Mtiling* approach can improve the QoE by 226.1%, and the *Ftiling* approach can improve the QoE by 52.4%. As mentioned earlier, when the network bandwidth is limited (in trace 2), our *Mtiling* approach has a much higher QoE improvement because it can still successfully download macrotiles in much higher video quality, while *Ftiling* and *Ctiling* must download small tiles in lower quality level.

As presented in Section 3.2, the QoE is affected by three main factors: average video quality, quality variation, and rebuffering. To help understand the QoE results in Figure 10, we use video 6 as an example to compare these three approaches based on these three QoE factors. As shown in Eq. 1, three QoE factors are calculated for each video segment, and then the summary statistics of all segments for video 6 are plotted as a boxplot. As can be seen in Figure 11, our *Mtiling* approach achieves much higher video quality than *Ftiling* and *Ctiling* for both network trace 1 and network trace 2. Moreover, *Mtiling* has less quality variation for network track 1, where most video segments are requested at the highest video quality. For network track 2, the quality variation in our *Mtiling* approach varies, because *Mtiling* can still download video segments in high quality, but sometimes it is necessary to download low quality segments. As for rebuffering events, *Mtiling* has the least amount of rebuffering events among all three.

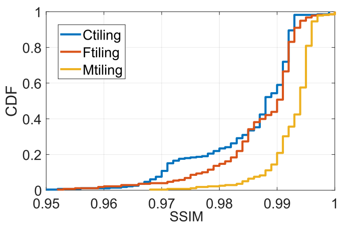
Ablation Study. To deal with the problem of users' head movement randomness, macrotiles are constructed by considering the users' viewing area plus some marginal area, which is determined based on the variations of the users' viewing centers. Fig. 12 shows the ablation study of the performance improvement when considering the marginal area in macrotile construction. The *Mtiling_base* approach is one variation of our *Mtiling* approach, where macrotiles are constructed by using only the users' viewing areas without considering the marginal area. As we can see from the figure, *Mtiling_base* outperforms *Ctiling* and *Ftiling*



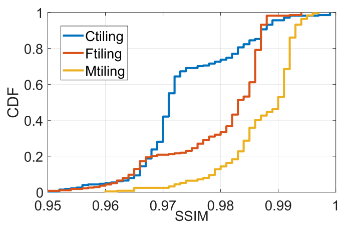
(a) PSNR (trace 1)



(b) PSNR (trace 2)

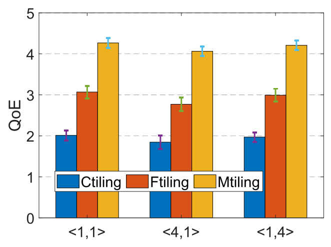


(c) SSIM (trace 1)

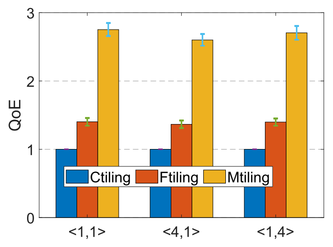


(d) SSIM (trace 2)

Fig. 15: Visual quality comparison.



(a) Network trace 1



(b) Network trace 2

Fig. 14: QoE comparisons with different weight settings.

since *Mtiling_base* encodes the popularly viewed areas as a large tile, and *Mtiling* outperforms *Mtiling_base* since *Mtiling* considers the marginal area for constructing macrotiles. Compared to *Mtiling_base*, *Mtiling* achieves significant QoE improvement especially when users experience high head movement randomness while watching 360° videos under poor network conditions. More specifically, for Video 1 where users are instructed to focus on the video content (i.e., low users' head movement randomness), compared to *Mtiling_base*, *Mtiling* enhances the QoE by 5.5% for network trace 1 and 15.1% for network trace 2. For Video 6 where users are free to explore the video (i.e., high users' head movement randomness), compared to *Mtiling_base*, *Mtiling* improves the QoE by 15.1% under good network conditions (trace 1) and 30.9% under poor network conditions (trace 2).

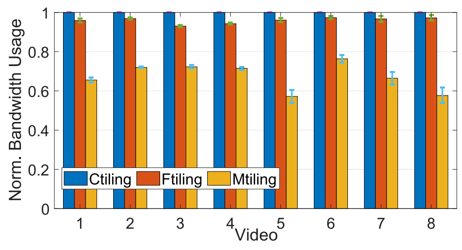
To evaluate the impacts of users' random head movement, we introduce two metrics, FoV coverage ratio (κ) and consumed bitrate (r_c). The FoV coverage ratio is defined as the percentage of the user's FoV that is covered by video tiles with high quality. The consumed bitrate (r_c) is defined as the average bitrate of the video content within the user's FoV, that is $r_c = r_h \cdot \kappa + r_l \cdot (1 - \kappa)$, where r_h is the video bitrate of high quality tiles and r_l is the lowest video bitrate if the user watches outside the predicted viewing area. Fig. 13 compares the FoV coverage ratio and consumed bitrate for different approaches. In all approaches, to address the problem of users' head movement randomness, in addition to downloading the tiles (or macrotile) covering the predicted viewing area with high quality, the remaining tiles are also downloaded with the lowest quality. The difference is that *Mtiling* encode the popularly viewed video area as a macrotile and other remaining video area as four parts to improve encoding efficiency, while *Ftiling* and *Ctiling* have to divide the user viewing area and the video region outside the macrotile into small tiles, and the *Mtiling_base* approach is one variation of *Mtiling* which uses only the users' viewing areas without considering the marginal area.

As shown in the figure, *Mtiling* achieves lower FoV coverage ratio compared to *Ctiling* (Fig. 13(a)), but *Mtiling* achieves much higher consumed bitrate than *Ctiling* (Fig. 13(b) and (c)). This is because *Mtiling* can download macrotiles encoded at much higher bitrate, but *Ctiling* needs to download much more video content (more tiles) encoded at much lower bitrate. This improvement is significant especially when the network condition becomes poor (in trace 2), where *Mtiling* can still successfully download macrotiles in much higher video bitrate, while *Ftiling* and *Ctiling* must download video tiles in lower video bitrate. As shown in Fig. 13(a), compared to *Mtiling_base* which constructs macrotiles without considering marginal area, *Mtiling* improves the FoV coverage ratio by 4.9% for video 1 where users are more likely to focus on the video content, and improves by 9.6% for video 6 where users are free to explore the video. The improvement on FoV coverage ratio leads to the improvement on QoE for *Mtiling* compared to *Mtiling_base* as shown in Fig. 12.

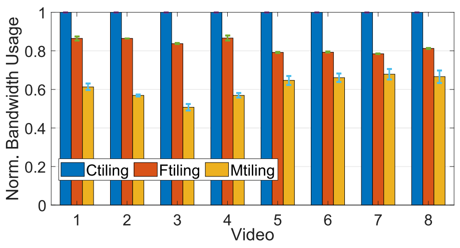
The Effect of Weight on QoE. For comprehensive comparison, we evaluate the performance of macrotile based 360° video streaming using various weight settings, similar to [29]. We set (ω_v=1, ω_r=1) to equally consider the metrics, (ω_v=4, ω_r=1) to minimize quality variation, and (ω_v=1, ω_r=4) to minimize rebuffering. Note that the QoE model in [29] does not consider the quality impairment caused by intra-segment spacial quality variation. Taking video 3 as an example, as shown in Fig. 14. As can be seen, though the QoE values for all approaches vary, the trends do not change, i.e., *Mtiling* achieve much higher QoE than *Ctiling* and *Ftiling* for different weight settings on the QoE model.

5.3.2 Visual Quality

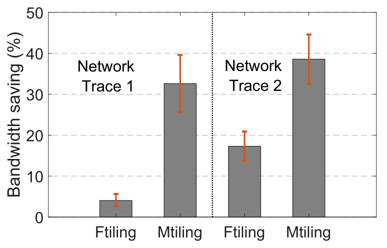
In addition to comparing the QoE, we also evaluate and compare the perception performance of the three methods in terms of two objective quality indicators: Structural Similarity (SSIM) index [39] and Peak Signal-to-Noise Ratio (PSNR) [40]. SSIM and PSNR are frequently used to quantify the quality that a user can perceive on a compressed image. In calculating these two metrics, the original frame (i.e., the uncompressed image) is used as a reference and compared with the corresponding compressed image. With a higher SSIM (PSNR) value, the visual quality a user can perceive becomes better. For the comparisons of visual quality, the actual view of a user is generated using the macrotile downloaded for *Mtiling* (or tiles downloaded for *Ctiling* and *Ftiling*), while the "reference" view is generated using the raw uncompressed video. The views are generated using lossless H.264 encoding setting (that is, crf=0 in X264 codec). SSIM and PSNR for each video segment are evaluated



(a) Network trace 1



(b) Network trace 2



(c) Overall bandwidth saving

Fig. 16: Bandwidth usage comparison (normalized based on that of *Ctiling*)

frame by frame based on the view actually watched by the user and the reference view. We use the average values of SSIM and PSNR over all video frames in a video segment to represent the SSIM and PSNR of that video segment, respectively.

We take video 1 as an example to show the comparisons of visual quality, as illustrated in Figure 15. As can be seen in the figure, the *Mtiling* approach achieves much higher PSNR and SSIM than both the *Ctiling* and *Ftiling* approaches. As shown in Figure 15(a), when network conditions are good (i.e., network trace 1), the median PSNR value for *Mtiling* is 47.3dB, and it is 41.1dB for *Ctiling*, and 43.5dB for the *Ftiling* approach. When the network condition becomes poor (i.e., network trace 2), as shown in Figure 15(b), the *Mtiling* approach still achieves much higher PSNR value (44.5dB) than the *Ctiling* approach (35.8dB) and the *Ftiling* approach (41.6dB). Figure 15(c) and Figure 15(d) present the same trend. When comparing the SSIM index, *Mtiling* significantly outperforms *Ctiling* and *Ftiling*. For network trace 1, the median SSIM is 0.994 for *Mtiling*, 0.987 for *Ctiling*, and 0.989 for *Ftiling*. For network trace 2, the median value of SSIM index for the *Mtiling* approach is 0.99. It is 0.971 for the *Ctiling* approach, and 0.983 for the *Ftiling* approach.

5.3.3 Bandwidth Usage

In this subsection, we compare the bandwidth usage of different approaches, where the bandwidth usage is normalized based on that of *Ctiling*. From Figure 16(a), we can see that the *Mtiling* approach only uses about 70% of the bandwidth required in the *Ctiling* approach when network trace 1 is used. This bandwidth usage is reduced to 60% when network trace 2 is used as shown in Figure 16(b). The *Mtiling* approach achieves high bandwidth saving due its high compression efficiency. Recall that *Mtiling* encodes the video area viewed by many users as a macrotile, and the rest video area outside the macrotile is divided into four large parts, thus achieves high compression efficiency. However, the *Ftiling* approach divides the video into large number of tiles, which reduces the video encoding efficiency and thus cost much higher bandwidth compared to *Mtiling*.

To summarize the results in Figures 16(a) and (b) by averaging eight videos, Figure 16(c) shows the overall bandwidth saving of the *Mtiling* approach and the *Ftiling* approach compared to the *Ctiling* approach. As can be seen, compared to *Ctiling*, *Mtiling* reduces the bandwidth usage by 32.6% for network trace 1, while *Ftiling* only reduces the bandwidth usage by 4.1%. For network trace 2, *Mtiling* reduces the bandwidth usage by 38.5% compared to *Ctiling*, which is much higher than *Ftiling*. Recall that the objective of the macrotile based 360° video streaming is to maximize the QoE. As shown in Figure 10(c), compared to the *Ctiling* approach, *Mtiling* improves QoE by 64.1% and 226.1% for network trace 1 and trace 2, respectively. Downloading macrotiles encoded at high quality can significantly improve the QoE and reduce the bandwidth usage, which demonstrates the effectiveness of *Mtiling*.

5.3.4 Energy Comparisons

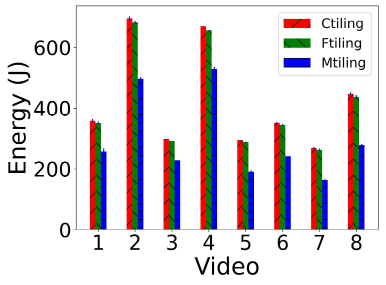
In this subsection, we compare different approaches in terms of energy consumption. We first show results measured using a Pixel 3 phone in Figure 17. More specifically, Figures 17 (a) and (b) compare different approaches when different videos are streamed under various network conditions (trace 1 or 2). To better summarize the comparison results, Figure 17 (c) shows how the *Ftiling* approach and the *Mtiling* approach perform compared to the *Ctiling* approach. That is, it shows the energy consumption of *Ftiling* and *Mtiling*, normalized based on *Ctiling*. As shown in the figure, *Mtiling* can reduce the energy consumption by about 30% compared to *Ciltting*. By using macrotiles, the *Mtiling* approach has high compression efficiency and it can reduce the amount of data to be downloaded, and thus reduce the energy consumption of data communication. Furthermore, Mtiling only uses one decoder, which reduces the computational overhead and the energy consumption of video processing. On the other hand, multiple decoders are needed in *Ctiling* and *Ftiling* in order to quickly decode the video tiles and then resulting in higher energy consumption.

We also compare the energy consumption using two different kinds of phones: LG Nexus 5x and Samsung Galaxy S20. The results are drawn in Figure 18, which shows the energy consumption of *Ftiling* and *Mtiling*, normalized based on *Ctiling*. In general, similar to the results in Figure 17 (c), our *Mtiling* approach has the lowest energy consumption and it significantly outperforms *Ctiling* and *Ftiling*.

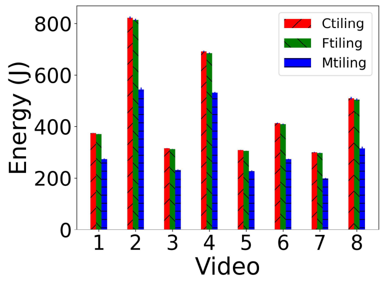
5.3.5 Energy Comparison of Different Components

To have a better understanding of the energy comparison results presented in the last section, we also measure the energy consumption of different components (i.e., data transmission, video decoding, and view rendering) in 360° video streaming.

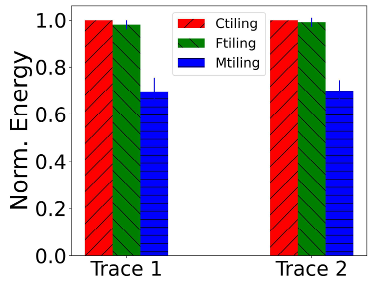
We have done some experiments to measure the power consumption of data transmission. During these experiments, a wget daemon runs in the background (with screen off) to download data from the server. Then, we cache the



(a) Network trace 1

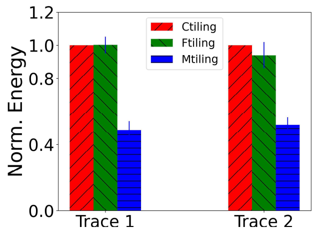


(b) Network trace 2

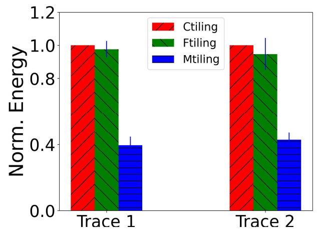


(c) Overall energy consumption

Fig. 17: Energy comparisons under different network condition (Pixel 3).



(a) Energy consumption (Nexus 5X)



(b) Energy consumption (Galaxy S2)

Fig. 18: Energy comparisons (a) Nexus 5X and (b) Galaxy S20.

dataset [16] locally on the smartphone, based on which we watch 360° videos with 4K resolution (i.e., 3840x2160), and measure the power consumption of video decoding and view rendering. There are three cases. 1) The first case is the baseline case, in which the video player is turned on but no video is played. 2) The second case is the decoding case. In this case, the video is decoded but no view is generated. That is, 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 engine. Note that the power difference between the second case and the first case will be the power consumption of video decoding. In our *Mtiling* streaming, only one decoder is needed. In other tile-based streaming such as *Ctiling* and *Ftiling*, similar to [3], [5], four decoders are used to decode the tiles in parallel. 3) The third case is for local video playback. After the video is decoded, the player retrieves the head orientation and renders the video. The power difference between the third case and the second case will be the power consumption for view rendering. Here, the power consumption of the screen is not considered, because it relies on many factors such as screen model, size, brightness, etc. Table 2 shows the power consumption for 360° video streaming when different tiling approaches are used.

Figure 19 compares different components of energy consumption when video 6 is streamed under network trace 2. As shown in the figure, our *Mtiling* approach can save energy for both data transmission and video processing when compared to the *Ctiling* approach and the *Ftiling* approach for all three types of phones. For example, with Pixel 3, as shown in Figure 19 (b), the *Mtiling* approach can reduce the energy consumed for data transmission by 26.1% when compared to the *Ctiling* approach. There are two parts of video processing: video decoding and view generation. As shown in the figure, compared to *Ctiling*, *Mtiling* can reduce the energy consumption for video decoding by 70% when Nexus 5X is used. Similar amount of energy reduction can be seen in the other two types of phones. We can also see that all three approaches have similar amount of energy consumption for view generation. This is because the view generation process only involves reading the pixel values from the memory based on the coordinate mapping (i.e., projection), which is much less than that of video downloading and video decoding. Note that the view generation in 360° video streaming is different from virtual reality (VR) applications such as gaming, where more video frames have to be rendered in real time, and thus demanding much more computations and energy.

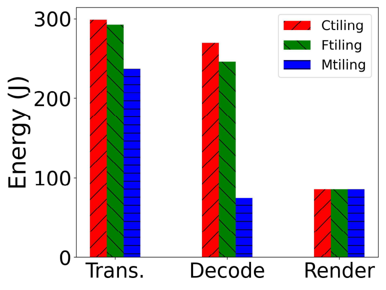
6 Related Work

In the literature, researchers have proposed two ways to save bandwidth on 360° video streaming: tile-based streaming and offset projection. In offset projections, such as pyramid projection [41] and offset cubic projection [2], the entire sphere is encoded, and more pixels are assigned to the directions that the users have a great possibility to look at. Different versions of the video are encoded, and each version focuses in a different direction on the sphere. During video streaming, the video version for which the pixel concentration most likely fit to the user's viewing direction is downloaded. However, offset projection techniques can increase the storage overhead at the server. For example, in Facebook 360° video platform, 22 versions are generated for each quality level of each video fragment, corresponding to different directions of pixel concentration [9]. In addition, offset projection techniques can incur a lot of processing burden on the client devices. For example, in the offset cubic projection, the texture color of the corners and/or edges is sampled from two different neighbouring surfaces during the rendering process, thus creating artifacts at the seams. In order to obtain better visual quality, hardware support and/or advanced software technology is needed for smooth filtering between surfaces [42].

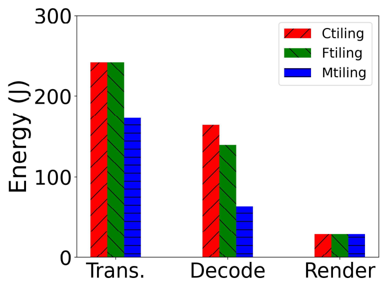
In tile-based streaming, a video is first projected to an equirectangular flat plane which has uniform pixel density on both horizontal and vertical directions, and then sliced into non-overlapping blocks (called video tiles). Only the video tiles that overlap the viewing area the user probably watches will be downloaded in high quality. Other tiles outside of the viewing area will not be downloaded or will

TABLE 2: Power consumption (mW) for 360° video streaming using different approaches.

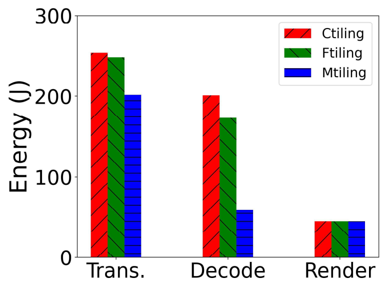
State	Nexus 5X	Pixel 3	Galaxy S20
Data trans.	1709.1 ± 33.6	1429.1 ± 24.3	1527.4 ± 31.8
Video decoding	Ctiling: 1656.3 ± 48.3 Ftiling: 1291.8 ± 41.5 Mtiling: 377.2 ± 31.4	Ctiling: 1038.7 ± 44.6 Ftiling: 783.4 ± 45.3 Mtiling: 319.5 ± 29.6	Ctiling: 1293.7 ± 53.4 Ftiling: 1099.1 ± 44.2 Mtiling: 336.6 ± 30.5
View rendering	431.7 ± 41.3	183.5 ± 32.2	227.6 ± 35.4



(a) Nexus 5X



(b) Pixel 3



(c) Galaxy S20

Fig. 19: Comparing different components of energy consumption.

be downloaded at low video quality to reduce the huge demand on network bandwidth. Tile-based streaming methods are widely utilized in 360° video streaming because of their easy implementations. Videos can be cut into fixed size tiles [3], [5], [29], [43], [44] or cut into a fixed number of video tiles of different sizes [9], [25]. However, how to set a proper value (i.e., a constant number of video tiles) for these fixed tiling methods is still an open research problem. In these methods, the video region frequently watched by a large number of users (i.e., the region that can be served by a macrotile) is cut into unnecessary small video tiles, which reduces the video encoding efficiency. In addition, cutting the video region outside a macrotile into small tiles, will also reduce the encoding efficiency and then incur more bandwidth usage. As a complement to the traditional fixed tiling methods, we propose to encode the video content frequently watched by the users into a macrotile, and cut the video region outside a macrotile into four large blocks as a compensation for the macrotile, which can obtain both high encoding efficiency and high coverage efficiency.

There are a number of studies on saving energy for video streaming in mobile environments. One way is to reduce the power consumed by the wireless interface when streaming video from the video server [45], [46]. For instance, Hu *et al.* [45] proposed techniques to reduce energy consumption based on whether users will stick on watching the video, skip video watching or abandon watching early. Wu *et al.* [46] proposed techniques to save energy for video streaming in heterogeneous networks. Other researchers studied how to save energy for video processing on the client devices [28], [47], [48]. In [28], an adaptive CPU frequency adjustment approach was proposed to reduce the energy consumption of video streaming. In [47], the environment (vibration or shaking impact) during video streaming is leveraged to design video bitrate adaptation algorithms to save energy. In [49], the authors proposed to save energy by reducing the less important video frames in each segment, and designed a control theory based algorithm to optimize QoE and energy. In [50], considering the distance between the user's eyes and the screen, a resolution dynamic scaling method is proposed to reduce energy consumption. In [48], the brightness of the display screen is dynamically adjusted to save energy. Unlike these aforementioned techniques, based on real experimental measurements, we identified the root causes of energy inefficiency of tile-based 360° video streaming and proposed a macrotile based 360° video streaming algorithm that can save energy while improving the QoE.

7 Conclusions

In this paper, we proposed a macrotile based 360° video streaming algorithm in which popularly viewed areas are encoded as macrotiles. To build macrotiles, we leverage the historical viewing data when users watch the same video. We first identify the viewing centers of these users and then cluster them together so that we can identify the macrotiles. For supporting video streaming service, the conventional tiling scheme (i.e., 4x6) is also used because the constructed macrotiles might not cover the viewing area of a small number of users who can randomly explore the video. For video streaming, the client selects the right tiles (macrotiles or a set of regular tiles) with the right quality level for each video segment in order to maximize QoE under bandwidth constraints. The macrotile based 360° video streaming problem is formulated as an optimization problem. Since the problem is NP hard, we proposed a heuristic based algorithm to solve it. Based on real head movement data traces, we evaluated and demonstrated that the proposed algorithm can significantly improve QoE and reduce energy consumption compared to existing solutions.

Acknowledgments

This work was supported in part by the National Science Foundation under grants 2215043 and 2125208.

References

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

[2] Facebook. Under the Hood: Building 360 Video, April 2022. https://code.facebook.com/posts/1638767863078802.

[3] 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.

[4] M. Dasari, A. Bhattacharya, S. Vargas, P Sahu, A. Balasubramanian, and S. R. Das. Streaming 360-Degree Videos Using Super-Resolution. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2020.

[5] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *ACM International Conference On Mobile Computing And Networking (Mobicom)*, 2018.

[6] 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.

[7] B. Chen, Z. Yan, H. Jin, and K. Nahrstedt. Event-Driven Stitching for Tile-based Live 360 Video Streaming. In *ACM Multimedia Systems Conference (MMSys)*, 2019.

[8] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun. Towards Low Latency Multi-viewpoint 360° Interactive Video: A Multimodal Deep Reinforcement Learning Approach. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[9] 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.

[10] H.264, Jan 2023. https://trac.ffmpeg.org/wiki/Encode/H.264.

[11] H.265, Jan 2023. http://x265.org/hevc-h265.

[12] Oculus, Jan 2023. https://www.oculus.com/.

[13] HTC Vive, Jan 2023. https://www.vive.com/.

[14] Google Cardboard, Jan 2023. https://arvr.google.com/cardboard.

[15] Samsung Gear VR, Jan 2023. https://www.samsung.com.

[16] 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.

[17] M. Xiao, C. Zhou, V. Swaminathan, Y. Liu, and S. Chen. BAS-360°: Exploring Spatial and Temporal Adaptability in 360-Degree Videos over HTTP/2. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[18] M. Graf, C. Timmerer, and C. Mueller. Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP: Design, Implementation, and Evaluation. In *ACM Multimedia Systems Conference (MMSys)*, 2017.

[19] 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.

[20] X. Corbillon, F. De Simone, G. Simon, and P. Frossard. Dynamic Adaptive Streaming for Multi-Viewpoint Omnidirectional Videos. In *ACM Multimedia Systems Conference (MMSys)*, 2018.

[21] 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.

[22] L. Xie, X. Zhang, and Z. Guo. CLS: A Cross-User Learning based System for Improving QoE in 360-Degree Video Adaptive Streaming. In *ACM Int'l Conf. on Multimedia*, 2018.

[23] FFmpeg. FFmpeg X264 encoding, April 2022. http://www.ffmpeg.org.

[24] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a Moving Target: Motion-Prediction-Based Transmission for 360-Degree Videos. In *IEEE Int'l Conf. on Big Data*, 2016.

[25] C. Zhou, M. Xiao, and Y. Liu. ClusTile: Toward Minimizing Bandwidth in 360-Degree Video Streaming. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[26] M. Xiao, S. Wang, C. Zhou, L. Liu, Z. Li, Y. Liu, and S. Chen. MiniView Layout for Bandwidth-Efficient 360-Degree Video. In *ACM Int'l Conf. on Multimedia*, 2018.

[27] 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.

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

[29] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li. DRL360: 360-Degree Video Streaming with Deep Reinforcement Learning. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2019.

[30] 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.

[31] Konstantinos G Derpanis. Overview of the RANSAC Algorithm. *Image Rochester NY*, 4(1):2–3, May 2010.

[32] Qshick. Ridge Regression, April 2022. https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db.

[33] 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 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2019.

[34] 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.

[35] 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.

[36] 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.

[37] J. Chen, M. Hu, Z. Luo, Z. Wang, and D. Wu. SR360: Boosting 360-Segree Video Streaming with Super-Resolution. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2020.

[38] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo. 360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming. In *ACM MM*, 2017.

[39] SSIM, Structural Similarity Index, Jan 2023. https://en.wikipedia.org/wiki/Structural_similarity.

[40] PSNR, Peak Signal-to-Noise Ratio, Jan 2023. https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

[41] Facebook. Next-Generation Video Encoding Techniques for 360 Video and VR, Jan 2023. https://code.fb.com/virtual-reality.

[42] ARM. White Paper: 360-Degree Video Rendering, Jan 2023. https://community.arm.com/developer/tools-software/graphics/b/blog/posts/white-paper-360-degree-video-rendering.

[43] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang. Flocking-based Live Streaming of 360-Degree Video. In *ACM Multimedia Systems Conference (MMSys)*, 2020.

[44] Y. Zhang, Y. Guan, K. Bian, Y. Liu H. Tuo, L. Song, and X. Li. EPASS360: QoE-aware 360-degree Video Streaming over Mobile Devices. *IEEE Trans. on Mobile Computing*, 2020.

[45] W. Hu and G. Cao. Energy-Aware Video Streaming on Smartphones. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2015.

[46] 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.

[47] 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.

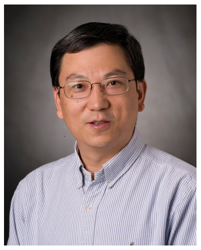
[48] Z. Yan and C. W Chen. RnB: Rate and Brightness Adaptation for Rate-Distortion-Energy Tradeoff in HTTP Adaptive Streaming over Mobile Devices. In *ACM Mobicom*, 2016.

[49] X. Chen and G. Cao. Energy-Efficient and QoE-Aware 360-Degree Video Streaming on Mobile Devices. In *IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2022.

[50] S. He, Y. Liu, and H. Zhou. Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling. In *ACM Mobicom*, 2015.

Xianda Chen received the BS degree in software engineering from Northwestern Polytechnical University, the MS degree in electrical and computer engineering from Sungkyunkwan University, and the Ph.D. degree in computer science and engineering from the Pennsylvania State University. His research interests include wireless networks, mobile computing, and mobile video. He is a student member of the IEEE.

Tianxiang Tan received the BE degree from Sun Yat-sen University, the MS degree in computer science from University of Southern California, and the PhD degree in computer science and engineering from the Pennsylvania State University. His research interests include mobile cloud computing, edge computing and deep learning. He is a student member of the IEEE.



Guohong Cao received his B.S. degree in computer science from Xi'an Jiaotong University, and his Ph.D. in computer science from the Ohio State University in 1999. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a Distinguished Professor. He has published more than 200 papers in the areas of wireless networks, mobile computing, machine learning, wireless security and privacy, and Internet of Things, which have been cited over 20000 times. He has served on the editorial board of IEEE Transactions on Mobile Computing, IEEE Transactions on Wireless Communications, and IEEE Transactions on Vehicular Technology, and has served on the organizing and technical program committees of many conferences, including the TPC Chair/Co-Chair of IEEE SRDS, MASS, and INFOCOM. He has received several best paper awards, the IEEE INFOCOM Test of Time award, and the NSF CAREER award. He is a Fellow of the AAAS and a Fellow of the IEEE.