Popularity-Aware 360-Degree Video Streaming

Xianda Chen, Tianxiang Tan and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {xuc23, txt51, gxc27}@psu.edu

***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. To address this problem, we propose to encode the video by considering the viewing popularity, where the popularly viewed areas are encoded as macrotiles to save bandwidth. We propose techniques to identify and build macrotiles, and adjust their sizes considering practical issues such as head movement randomness. In some cases, a user's viewing area may not be covered by the constructed macrotiles, and then the conventional tiling scheme is used. To support popularity-aware 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 traces, we demonstrate that the proposed algorithm can significantly improve the QoE and save the bandwidth usage.**

I. Introduction

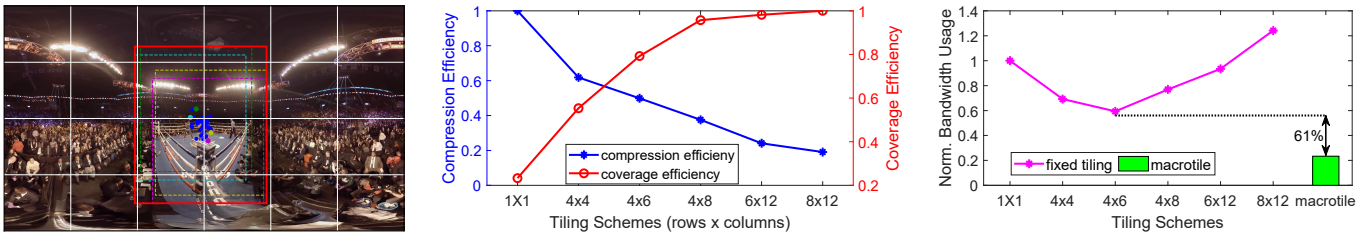
360° video is becoming more and more popular on video platforms such as YouTube and Facebook [1, 2]. As 360° video is much larger than conventional video under the same perceived quality [3, 4, 5], streaming 360° video is much more challenging, especially over wireless (e.g., cellular) networks with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

Many researchers [6, 7, 8, 9] have addressed this challenge by only downloading part of the video data. Due to the limited Field-of-View (FoV) of the mobile devices, 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 implement this idea, one widely used approach is the tile-based streaming [6]. In this approach, the video is broken into a sequence of video segments and each segment contains 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, such that the bandwidth usage can be reduced 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. In contrast, although large sized tile can improve the compression efficiency, more data outside of the FoV will have to be downloaded, and then more bandwidth will be wasted. Thus, it is important to encode the video with the right tile size.

In this paper, 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. By encoding these users' viewing areas as a tile (called *macrotile*), high compression efficiency can be achieved, since the popularly viewed video region is encoded into one macrotile instead of being divided into multiple small tiles. In addition, the macrotile includes less data outside of the user's viewing area, and then saving bandwidth. 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, these users 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 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 popularity-aware 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 popularity-aware 360° video streaming algorithm, which first predicts the user's viewing area for each video segment, and then prefetchs the corresponding macrotile or conventional tiles if necessary.



(a) (b) (c)

Fig. 1: (a) The video is divided using the 4x6 tiling scheme. (b) The comparison of compression efficiency and coverage efficiency. (c) The comparison of bandwidth usage.

We formulate the problem as an optimization problem and propose an algorithm to solve it.

The main contributions of this paper are as follows.

- We propose to encode the video by considering the viewing popularity, where the popularly viewed areas are encoded as macrotiles to save bandwidth.
- We formulate the popularity-aware 360° video streaming problem as an optimization problem which is NP-hard, and then propose a heuristic based algorithm to solve it.
- We evaluate the performance of the proposed solution using real head movement data traces. The evaluation results show that the proposed algorithm can significantly improve the QoE and save the bandwidth usage.

The rest of the paper is organized as follows. We introduce the background and motivation in Section II. Section III presents the system model and the problem formulation. Section IV presents our popularity-aware 360° video streaming algorithm. In Section V, we present the evaluation results. Section VI discusses related work and Section VII concludes the paper.

II. Background and Motivation

The tile size can significantly affect the amount of data to be downloaded in 360° video streaming. To illustrate this, we conducted some experiments based on the head movement data traces of 48 users watching one 360° video [12]. We divide the video using different tiling schemes commonly used in the literature, represented by (rows x columns); i.e., 1x1 (no tiling), 4x4 [13], 4x6 [3, 14, 15, 16], 4x8 [17], 6x12 [18], and 8x12 [17]. For fair comparison, we used FFmpeg [19] with encoder x264 to crop and encode the tiles using the same encoding parameters. The user's viewing area is determined by the viewing center and the FoV of the device, which is considered to be 100 degrees horizontally and vertically [15, 20, 21, 22, 23]. Fig. 1(a) shows a video which is divided into 4 rows and 6 columns, i.e., 24 tiles. Each dot in the figure 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 Fig. 1(a)), respectively.

Fig. 1(b) compares the effectiveness of different tiling schemes. 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. To compare these tiling schemes, we use the metric of *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 schemes, where the compression ratio [23] is defined as the total number of bytes needed to represent the tiles covering the FoV divided by the total encoded data size in bytes. As shown in Fig. 1(b), smaller tiles (e.g., 8x12) can achieve higher coverage efficiency while larger titles (e.g., 1x1) can achieve higher compression efficiency. Note that smaller tiles (e.g., 8x12) have lower compression efficiency because lots of spatial/temporal redundancy cannot be removed. For example, with large tiles, if one area has similar content with other areas, the spatial/temporal redundancy can be removed, but this will not be possible if small tiles are used.

The downloaded video data (or the bandwidth usage) is related to the coverage efficiency and compression efficiency. Fig. 1(c) shows the bandwidth usage, normalized based on the 1x1 tiling scheme, of different tiling schemes. As shown in the figure, 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 is too low and hence the downloaded data becomes larger. For example, the bandwidth usage of 8x12 tiling becomes larger than that of the 1x1 tiling scheme. Thus, it is a challenge to find the right tiling scheme to reduce the bandwidth usage.

As shown in Fig. 1(a), we construct a large sized tile (i.e., the red block), called macrotile, which covers all the viewing areas (dashed blocks). This macrotile has high compression efficiency due to its large tile size. It has high coverage efficiency since only one large tile is used instead of as many as 12 tiles, represented by the cyan blocks. In contrast, none of the conventional tiling schemes can achieve both high compression efficiency and high coverage efficiency. As shown in Fig. 1(c), compared to the 4x6 tiling scheme (the best scheme), macrotile can further reduce the bandwidth usage by 61%.

To construct macrotiles, we exploit the historical viewing data from users watching the same video. Due to their common interests, these users 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 macrotiles. We will present the detail of macrotile construction in Section IV.

III. System Model and Problem Formulation

In this section, we introduce the video model, the QoE model, and the problem formulation.

A. Video Model

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 C tiles using a conventional tiling scheme (e.g., 4x6). In addition, M macrotiles are constructed, and the details will be described in Section IV. On the server side, 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. In 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.

B. QoE Model

For each video segment k, similar to [3, 24, 25, 26], the QoE model quantifies the user perceived quality by considering the following metrics: average video quality, quality variation, and rebuffering. The QoE model is 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*. Since only video content in the viewing area contributes to user perceived quality, 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 [7].
- *Quality Variation*. The quality variation between two consecutive segments may cause users feel discomfort such as dizziness, and should be considered in the QoE model. When a user downloads a set of tiles, the quality variation of these tiles will affect user's perceived quality, and should also be considered in the QoE model [3]. The quality variation is defined as follows.
$$I_v(\mathcal{V}_k) = |Q_0(\mathcal{V}_k) - Q_0(\mathcal{V}_{k-1})| + \widehat{\mathcal{V}_k} \tag{3}$$
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 $\mathcal{V}_k$ [3].
- *Rebuffering*. Rebuffering will significantly affect the QoE since the video will freeze during rebuffering events. The rebuffering time is 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 $(x)_+ = max\{x, 0\}$.

C. Problem Formulation

In this subsection, we formalize the popularity-aware 360° video streaming problem. Let β_m^v (β_c^v) represent if the corresponding macrotile (or conventional tile) will be downloaded. Specifically, $\beta_m^v = 1$ if the macrotile m encoded at quality level v is downloaded, and the bandwidth usage is B_m^v; otherwise $\beta_m^v = 0$. $\beta_c^v = 1$ if the tile c encoded at quality level v is downloaded, and the bandwidth usage is B_c^v; otherwise $\beta_c^v = 0$. 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 popularity-aware 360° video streaming, the goal is to maximize the user's perceived QoE, and 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 to maximize the perceived quality under the network bandwidth constraint. The optimization problem is formulated as follows.

$$\textbf{max} \quad Q(\{v \mid \forall_{mv} \beta_m^v = 1\}) + Q(\{v \mid \forall_{cv} \beta_c^v = 1\}) \tag{5}$$

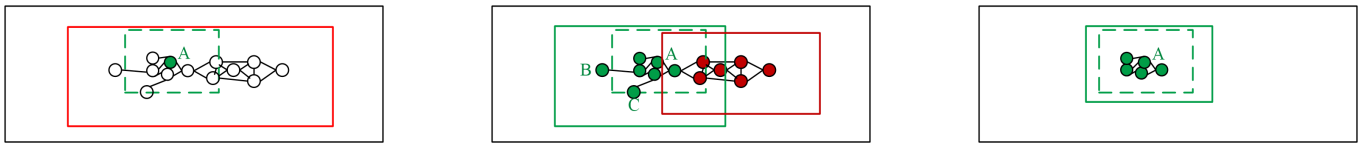
$$\textbf{s.t.} \quad \sum_{m=1}^{M}\sum_{v=1}^{V} \beta_m^v + \mathbf{1}(\sum_{c=1}^{C}\sum_{v=1}^{V} \beta_c^v) = 1 \tag{5a}$$

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

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

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, i.e., the candidate macrotile (and the candidate set of conventional tiles) that covers the viewing area is known, we can decompose the problem in Eq. 5 into two sub-problems: one is to determine the right quality level for the macrotile; the other one is to determine the right quality levels for the tiles. Then, the one achieving better QoE



(a) (b) (c)

Fig. 2: Macrotile construction. (a) Macrotile is too large. (b) Split a large macrotile into two macrotiles. (c) Macrotile optimization (only shows the green one).

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 equals to 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.

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

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

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

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, where 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 as follows. 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. Thus, the solution is also a solution to the MCK problem. This completes the reduction and hence the proof. □

Theorem 1. *The popularity-aware 360° video streaming problem is NP-hard.*

Proof. The popularity-aware 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 popularity-aware 360° video streaming problem is NP-hard. □

Since the popularity-aware 360° video streaming problem is NP-hard, we propose a heuristic based algorithm.

IV. Popularity-Aware 360° Video Streaming

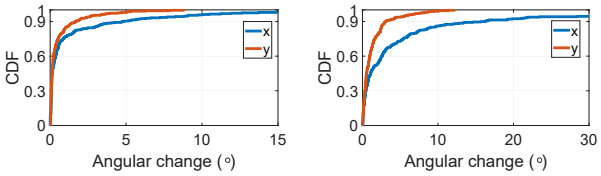
In this section, we first describe how to construct macrotiles which includes two parts: identifying macrotiles based on viewing areas, and macrotile optimization. Then, we present our popularity-aware 360° video streaming algorithm based on the idea of macrotiles.

A. Identifying Macrotiles based on Viewing Areas

Users may have similar viewing interests when watching the same 360° video. Then, they may 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. Other non-parametric clustering algorithms such as the density-based clustering algorithm (DBSCAN) [27] do not need to know the number of clusters before hand, however, they may lead to another problem; that is, the cluster may keep growing and become too large, losing the benefit of saving bandwidth. For example, as shown in Fig. 2(a), since the viewing centers of a cluster span a large area, the constructed macrotile (the red block) will be too large. Note that this is the key difference between clustering and macrotile. If the macrotile is too large, the benefits of using macrotile to save bandwidth will be lost. To address these problems, we propose the following clustering algorithm.

The algorithm uses two parameters, λ and γ, to ensure the viewing centers that are close to each other are clustered together and the macrotile constructed based on each cluster will not be too large. The parameter λ is used to determine whether two viewing centers are close enough to be in a cluster. Two viewing centers are considered to be close (i.e., two users watch similar video content) if their distance is less than or equal to λ. The parameter γ defines the maximum size of each cluster; i.e., the maximum distance of any two viewing centers of a cluster does not exceed γ.

The parameters λ and γ can affect the performance of the clustering algorithm. With a small λ, some viewing centers that should be clustered may be missed. With a large λ, the viewing centers that are far away from each other (i.e., users have different viewing interests) may be clustered together. With a large γ, the cluster becomes too large; while with a small γ, more 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 Fig. 1(a). Some



(a) Video 1 (b) Video 8

Fig. 3: CDFs of angular changes on x and y directions for a user. Only two videos are shown due to the page limit.

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. Thus, we empirically set γ to be the width of the conventional tile and set $\lambda = \gamma/4$.

Given a set of points (denoted by P), where each point represents the location of one user's viewing center, we use $N_p = \{q \,|q \in P \wedge q \neq p \wedge dist(p,q) \leq \lambda\}$ to denote the set of points that are close to point p, where $dist(p,q)$ represents the Euclidean distance between two points p and q. The clustering algorithm works 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 that 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 $P = \emptyset$.

B. Macrotile Optimization

Constructing a macrotile for each cluster by simply covering all viewing areas of the users in the cluster may build unnecessary large macrotile. For example, as shown in Fig. 2(b), the macrotile (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 the downloaded macrotile if the macortile is too small. Thus, it is important to determine the right macrotile size based on the cluster.

To determine the right macrotile size, we need to determine which users' viewing areas should be considered to construct the macrotile, such that the bandwidth usage (named B_m) when a user downloads the macrotile is less than the bandwidth usage (named B_c) for downloading a set of conventional tiles. 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 area. 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. Fig. 3 shows an example when a user watches eight 360° videos [12], as listed in Table I. The user's head movement data are collected under two different settings. When watching video 1 to 4, users are instructed to focus on the video content. When watching video 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. It can be seen in Fig. 3, 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.

TABLE I: 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

To formalize the problem of macrotile construction, we introduce a binary variable α_i for each user i, where $\alpha_i = 1$ if the user's viewing area is used to construct the 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.

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

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 guarantee an optimal solution for Eq. 7, its computational complexity is $O(2^{N_j})$. To reduce the construction time in practice, we propose an iterative approach, similar to the random sample consensus paradigm [28]. For each iteration, it works as follows.

(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 the constructed macrotile covers user $i \in \{1, ...N_j\}$. If so, $\alpha_i = 1$; otherwise, $\alpha_i = 0$.
(d) Check 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. If so, update the macrotile with the macrotile constructed in the current iteration.

If only downloading the macrotile that covers the predicted viewing area, the user may view a blank area when suddenly navigates to an area outside of the downloaded macrotile. To address this problem, in addition to downloading the tiles (or macrotile) covering the viewing area at high quality, the user

also downloads the remaining tiles at the lowest quality. More specifically, for each constructed macrotile, as shown in Fig. 2(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 encoded at the lowest quality level, and will be also downloaded along with the macrotile. The extra bandwidth usage for downloading these four parts is very small, since the compression efficiency is high and these videos are encoded at the lowest quality level.

C. Popularity-Aware 360° Video Streaming

In the last subsection, macrotiles are constructed on the server based on the historical viewing data. To support users whose viewing areas are not covered by the constructed macrotiles, the sever also encodes the video using the conventional tiling scheme (e.g., 4x6). For video streaming, the client selects 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. In this subsection, we propose a popularity-aware 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.

To predict the user's viewing area (i.e., the viewing center), we use the ridge regression model [29] due to its robustness to cope with overfitting. When watching 360° video, the user's historical viewing centers are recorded by the sensor embedded in the device, where each viewing center is represented by its (x, y) coordinates on the video frame. Since the coordinates of the viewing centers are recorded at a fixed sampling rate (e.g., 50 Hz), the recorded x and y coordinates can be considered as time series data. To predict the x coordinate of the user's viewing center, the historical data of x coordinates when the user watches the most recent video segment is used to train the model. Then, given the time stamp of the video segment to be prefetched, the trained model predicts the x coordinate of the user's viewing center when he watches that video segment. Similarly, the y coordinate of the viewing center is predicted. Based on the predicted viewing center (i.e., x and y coordinates) and the FoV, the user's viewing area is predicted.

Based on the predicted viewing area, the algorithm determines whether there is a macrotile that covers the predicted viewing area plus some marginal area, which is calculated similar to that in the last subsection. If such 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).

If such macrotile does not exist, a set of conventional tiles will be downloaded. The following shows how to determine the quality levels for the conventional tiles (line 17-35 in Algorithm 1). The algorithm first finds the highest possible quality level for these tiles such that they can be successfully downloaded under the network conditions. 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. That is, the algorithm sorts tiles in the ascending order of their distance to the viewing center. The algorithm searches from the tile most close to the viewing center, and then determines the maximum number of tiles to have one level higher quality based on the amount of remaining bandwidth. Since the QoE is affected by quality variations, the quality level increase is performed only if more than half of the downloaded tiles can increase their quality levels.

Algorithm 1: Popularity-Aware 360° Video Streaming

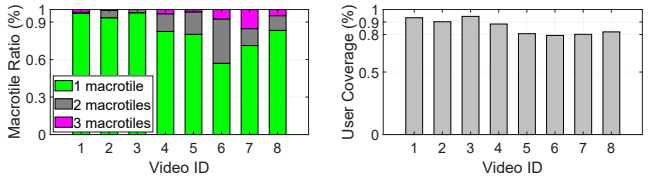
```
   Input: r, L, V
   Output: β_m^v or {β_c^v}
1  Predict the user's viewing area
2  Determine the macrotile m and the set of tiles C
3  if m exists then
4  |  return SelectMacrotile(m, r, L)
5  else
6  |  return SelectCTiles(C, r, L)
7  end
8
9  function SelectMacrotile(m, r, L):
10 |  for v ← V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

V. Performance Evaluations

A. Experiment Setup

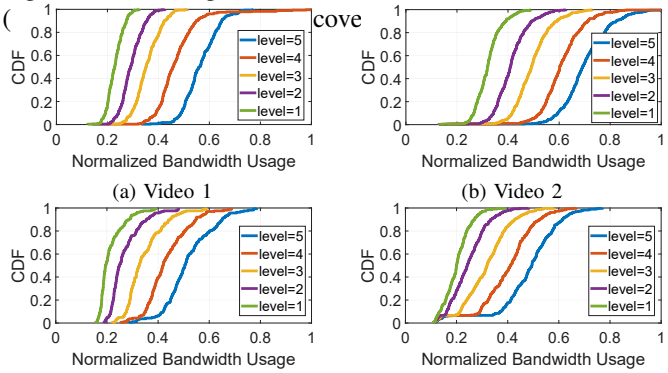
The performance evaluation is based on the head movement data traces of 48 users watching eight 360° videos [12], and the details of the video traces are shown in Table I. These videos cover different scenarios, e.g., sport, performance, TV show, etc. For each video, we randomly select forty users' head movement data traces to construct video tiles (and macrotiles), and the rest eight data traces are used to evaluate the performance of 360° video streaming.

The 360° video streaming system consists of two major components, i.e., the server and client module. On the server side, similar to [3, 30, 31], we divide each video into a sequence of one second segments. Each video segment is further divided spatially into tiles (and macrotiles). Then, FFmpeg [19] with encoder x264 is used to encode all tiles (and macrotiles) into five quality levels (5 to 1, with 5 being the



(a) (b)

Fig. 4: (a) Video segments with different number of macrotiles. (cove



(a) Video 1 (b) Video 2

(c) Video 7 (d) Video 8

Fig. 5: CDFs of the data size of a macrotile. Only four videos are shown due to the page limit.

highest quality) using different constant rate factor (crf) values from 18 to 38 with an interval of 5 [3]. On the client side, we set the FoV of the mobile device to be 100 degrees horizontally and vertically. Similar to [4], we use the harmonic mean of the downloading throughput of the past several segments to estimate the network bandwidth. For the QoE model, we set the weights as (ω_v, ω_r) = (0.25, 0.25), which is a used setting in [26].

The network traffic is simulated based on a LTE throughput trace with varying throughput patterns [32]. To evaluate the performance of video streaming under different network conditions, we linearly scale the network throughput trace into two types, named *trace 1* and *trace 2*, where the average network throughput of trace 1 is twice that of trace 2. The network throughput for trace 2 ranges from 1.3 Mbps to 10.9 Mbps, with an average of 4.8 Mbps.

We compare the performance of the following approaches.

- *Ctiling*: Each video segment is divided into fixed size tiles using a conventional tiling scheme (e.g., 4x6). The Ctiling scheme has been used in many existing streaming solutions [3, 14, 15, 16, 33].
- *Ftiling*: Each video segment is divided into a fixed number of tiles which may have different sizes. Similar to [22], 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.
- *Ptiling*: In addition to the tiles constructed with the 4x6 tiling scheme, macrotiles are also constructed with the proposed solution.

B. Performance of Macrotile Construction

We empirically set γ to be the width of a conventional tile and $\lambda = \gamma/4$. To reduce the number of unnecessary macrotiles that cover too few users, we construct the macrotile for a cluster only if it has at least 5 users (i.e., one tenth of the users in the dataset).

1) Macrotile Coverage: Fig. 4(a) shows the number of macrotiles constructed for video segments. Since users have similar viewing interests when watching the same 360° video, only limited number of macrotiles are needed. As shown in the figure, more than 95% video segments require only one macrotile for videos 1, 2, and 3. Video 4 is about a basketball game, where users' gazing directions move when the basketball players move. As a result, 82.5% segments have one macrotile, 14.1% segments have two macrotiles, and less than 4% segments have more than two macrotiles. For videos 5 to 8 where users are free to explore the video, more than 92% segments have less than three macrotiles on average.

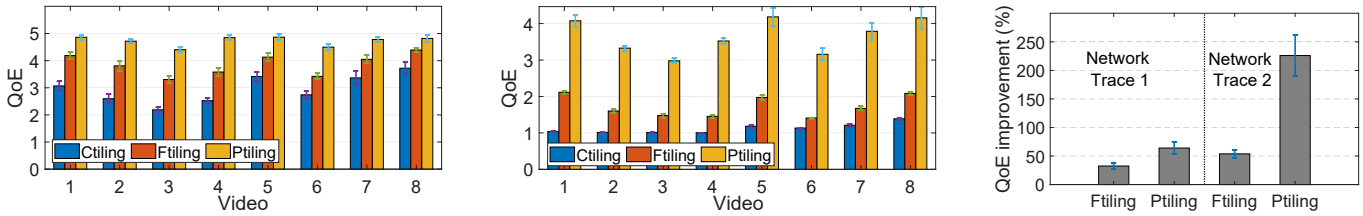
Fig. 4(b) shows the percentage of users covered by the macrotiles, and most users' viewing areas are covered by the macrotiles. Specifically, 94.1%, 90.3%, 94.6%, and 88.4% users can be served by the macrotiles when watching videos 1, 2, 3, and 4, respectively. The value is larger than 80% on average even for videos 5 to 8 where users are free to explore the video. As a result, most users only need to download these macrotiles instead of conventional tiles to save bandwidth during video streaming. For a small number of users which are not covered by the macrotiles, they have to download the conventional tiles during video streaming. Note that the viewing centers of these users are more likely to be far away from those covered by the macrotile, and it is better not to include them in the macrotile to save bandwidth.

2) Macrotile vs. Conventional Tile: A user's viewing area can be covered by a macrotile or a set of conventional tiles. To see the benefit of macrotile, we compare the data size of the macrotile and that of conventional tiles, when both are encoded at the same quality level. When more than one macrotiles exist for the video segment, the average is used. Since different users may download different sets of conventional tiles, the average is used.

Fig. 5 shows the data size CDFs of using macrotile for each video segment, normalized based on that of conventional tiles. As can be seen in the figure, the average data size of macrotile is much smaller than that of conventional tiles, because encoding the popularly viewed video as a macrotile can achieve high compression efficiency and high coverage efficiency compared to dividing the video into small tiles. Taking video 1 as an example, as shown in Fig. 5(a). The median data size of using macrotile is 54%, 45%, 35%, 29%, and 22% of that of conventional tiles when the video is encoded at quality level 5, 4, 3, 2, and 1, respectively. In other words, the bandwidth usage can be saved 46% (i.e., 1-0.54=0.46) on average when the user downloads macrotiles for all video segments encoded at the highest quality level. The bandwidth saving when downloading macrotiles at quality level 4, 3, 2, and 1 is 55%, 65%, 71%, and 78%, respectively.

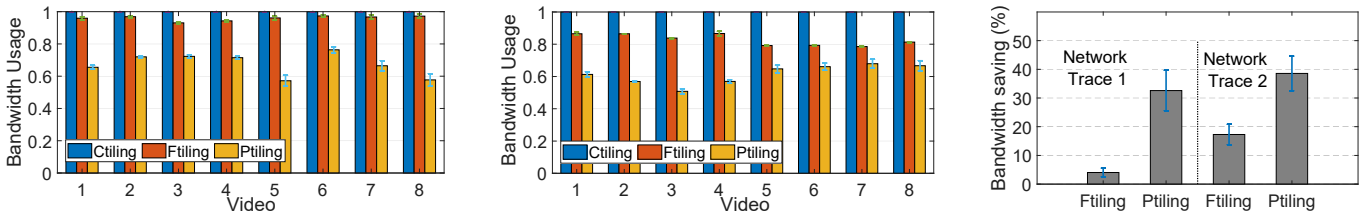
C. Performance of Popularity-Aware 360° Video Streaming

We evaluate the performance of popularity-aware 360° video streaming using user's head movement data trace. Sim-



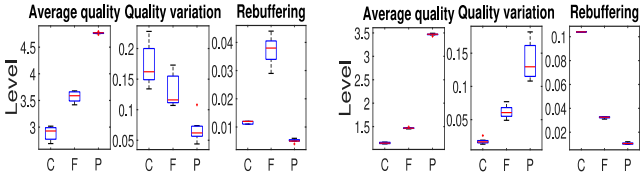
(a) Network trace 1 (b) Network trace 2 (c) Overall QoE improvement

Fig. 6: QoE comparison.



(a) Network trace 1 (b) Network trace 2 (c) Overall bandwidth saving

Fig. 8: Bandwidth usage comparison (the usage is normalized based on that of *Ctiling*)



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

Fig. 7: Comparing the three components of QoE (C:Ctiling, F: Ftiling, P: Ptiling).

ilar to [3, 34, 31], the playback buffer is set to 3 seconds.

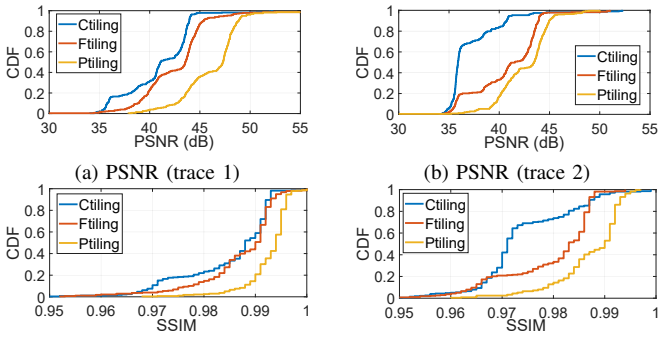
1) QoE Improvement: Fig. 6 compares the QoE of different approaches under different network conditions. As can be seen, *Ptiling* outperforms *Ctiling* and *Ftiling* for all video traces for both network throughput traces. In *Ptiling*, by using macrotiles, high compression efficiency and high coverage efficiency can be achieved. Then, under the same network condition, a user can download a macrotile encoded at higher quality level, instead of conventional tiles (or tiles constructed by *Ftiling*) encoded with lower quality level, and thus the QoE can be improved. It is inefficient for *Ftiling* 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 macrotile) is divided into unnecessary number of small tiles, reducing video compression efficiency. Moreover, dividing the video area outside of a macrotile, which is encoded together as a compensation for the macrotile, reduces the encoding efficiency and thus causes high bandwidth demand for *Ftiling*. When the network condition becomes poor (from Fig. 6(a) to Fig. 6(b)), the QoE of these approaches drops; however, the QoE of *Ptiling* drops much slower compared to that of *Ctiling* and *Ftiling*. For example, for video 1, compared to using network trace 1, when using trace 2, the QoE of *Ctiling* degrades by 66.1%, the QoE of *Ftiling* degrades by 49.5%, but the QoE of *Ptiling* only degrades by 16.1%.

Fig. 6(c) shows the overall QoE improvement of *Ptiling* and *Ftiling* compared to *Ctiling*. *Ptiling* can improve the QoE by 64.1% for network trace 1, which is around two times of that of *Ftiling* (32.5%). For network trace 2, *Ptiling* can improve the QoE by up to 226.1% compared to *Ctiling*, while the QoE improvement achieved by *Ftiling* is very small. The large QoE improvement is due to the reason that *Ptiling* can still download macrotiles encoded at much higher quality level although *Ctiling* and *Ftiling* have to use low quality video.

Fig. 7 compares the three components of the QoE (as in Eq. 1) for video 6. *Ptiling* can achieve much higher average quality than *Ctiling* and *Ftiling*, especially for network trace 2. *Ptiling* experiences less quality variation for trace 1, where most segments are downloaded at the hightest quality. For trace 2, the quality variation of *Ptiling* is high, because *Ptiling* still downloads high quality segments, but has to download low quality segments sometimes. As for rebuffering, *Ptiling* performs the best among these three approaches.

2) Bandwidth Savings: Fig. 8 compares the bandwidth usage of these three approaches, where the bandwidth usage is normalized based on that of *Ctiling*. As shown in Fig. 8(a), *Ptiling* has about 70% of the bandwidth usage of *Ctiling* for network trace 1, and it has about 60% of the bandwidth usage of *Ctiling* for network trace 2 as shown in Fig. 8(b). *Ptiling* achieves high bandwidth saving due its high compression efficiency. Recall that the *Ptiling* approach 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. For *Ftiling*, instead, dividing the video into large number of tiles, reduces the video encoding efficiency and thus cost much higher bandwidth compared to *Ptiling*.

The overall bandwidth saving of *Ptiling* and *Ftiling* is shown in Fig. 8(c). Compared to *Ctiling*, the *Ptiling* approach reduces the bandwidth usage by 32.6% for network trace 1, while that of *Ftiling* is only 4.1%. For network trace 2, the *Ptiling* approach reduces the bandwidth usage by 38.5% compared to *Ctiling*, which is much higher than *Ftiling*. Recall that the objective of the popularity-aware 360° video streaming is to maximize the QoE. As shown in Fig. 6(c), compared to *Ctiling*, the *Ptiling* approach improves QoE by 64.1% and 226.1%



(a) PSNR (trace 1) (b) PSNR (trace 2)

(c) SSIM (trace 1) (d) SSIM (trace 2)

Fig. 9: Visual quality comparison.

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 the *Ptiling* approach.

3) Visual Quality: Complementary to comparing the subjective QoE (i.e., five quality levels are defined as described in Section V-A), we also compare the performance of these three approaches using the following objective quality metrics: Peak Signal to Noise Ratio (PSNR) [35] and Structural Similarity (SSIM) index [36]. PSNR and SSIM are commonly used to quantify the perceived quality of an image after compression, where the original uncompressed image is used as reference. The higher the PSNR (SSIM) is, the better the perceived quality is. For visual quality comparisons, the actual user's view is rendered using the downloaded macrotile (or tiles), and the "reference" view is rendered from the original video using lossless H.264 encoding (i.e., setting crf=0 in x264). PSNR and SSIM between the actual user's view and the reference view are evaluated frame by frame in each video segment. We use the average PSNR and SSIM of all frames in each segment to denote the PSNR and SSIM of that segment, respectively.

Taking video 1 as an example, as shown in Fig. 9, compared to *Ctiling* and *Ftiling*, the *Ptiling* approach achieves much higher PSNR and SSIM. As shown in Fig. 9(a), for network trace 1, the median PSNR is 47.3 dB for *Ptiling*, 43.5 dB for *Ftiling*, and 41.1 dB for *Ctiling*. Fig. 9(c) and 9(d) show that the *Ptiling* approach outperforms *Ctiling* and *Ftiling* in terms of SSIM for both network traces.

VI. Related Work

There has been considerable research on saving bandwidth for 360° video streaming, which can be classified into two categories: offset projection approaches and tile-based streaming approaches. In offset projection, such as offset-cubic projection [2] and pyramidal projection [37], the full spherical surface is encoded and more pixels are devoted to a specific direction that users are more likely to view. Multiple versions of offset-projected videos are encoded, and each version with concentrating pixels in a different direction on the sphere. During video streaming, the version whose pixel concentration direction best matches the user's predicted viewing direction is downloaded. However, these approaches incur significant storage overhead on the server side. For example, Facebook encodes 22 versions corresponding to different concentration directions for each quality level for each segment [21]. In addition, these approaches incur severe processing overhead on the mobile devices. In offset-cubic projection, for example, the colors of the textures at the edges or corners are sampled from two different faces during video rendering, and thus artefacts are generated at the seams. To have better visual quality, hardware support or software techniques are required to filter smoothly between faces [38].

In tile-based streaming, the video is first projected onto an equirectangular plane with uniform pixel density and then cut into tiles. Only tiles overlapping with the user's predicted viewing area are delivered in high quality, whereas other tiles are delivered in low quality or not delivered at all. Due to its simplicity to implement, tile-based streaming approaches are widely used in 360° video streaming. Some researchers encode the video using a fixed size tiling scheme [3, 4, 26, 39, 40], while others propose to divide the video into a fixed number of tiles with different tile sizes [21, 22]. However, it is hard to set a constant value (i.e., number of tiles) for the fixed tiling schemes. In these schemes, the video area viewed by many users (i.e., the area covered by a macrotile) is divided into unnecessary number of small tiles, reducing video compression efficiency. Moreover, dividing the video area outside of a macrotile, which is encoded together as a compensation for the macrotile, reduces the encoding efficiency and thus causes high bandwidth demand. Complementary to the conventional fixed tiling schemes, we encode the popularly viewed video content as a macrotile, which can achieve both high compression efficiency and high coverage efficiency together.

VII. Conclusions

In this paper, we proposed a popularity-aware 360° video streaming algorithm to maximize the QoE under network constraints. In the proposed solution, the video is encoded by considering the viewing popularity, where the popularly viewed areas are encoded as macrotiles to save bandwidth. To construct macrotiles, we exploit the historical viewing data from users watching the same video. We first identify these users' viewing centers and cluster them together, based on which we can identify the macrotiles. Since few users' viewing areas may not be covered by the constructed macrotiles, the conventional tiling scheme (i.e., 4x6) is also used. For video streaming, the client selects the right tiles (a macrotile or a set of conventional tiles) with the right quality for each video segment, such that the QoE is maximized under bandwidth constraint. We formulated the popularity-aware 360° video streaming problem as an optimization problem which is NP-hard, and then proposed a heuristic algorithm to solve it. Through real head movement data traces and trace-driven simulations, we demonstrated that the proposed algorithm can significantly improve the QoE and save the bandwidth usage.

VIII. Acknowledgments

This work was supported in part by the National Science Foundation under grant CNS-1815465.

REFERENCES

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

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

[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. Dasari, A. Bhattacharya, S. Vargas, P Sahu, A. Balasubramanian, and S. R. Das. Streaming 360-Degree Videos Using Super-Resolution. In *IEEE INFOCOM*, 2020.

[6] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. Optimizing 360 Video Delivery over Cellular Networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, 2016.

[7] 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.

[8] 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.

[9] 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 INFOCOM*, 2019.

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

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

[12] 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.

[13] 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 INFOCOM*, 2018.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] FFmpeg. http://www.ffmpeg.org.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] K. G. Derpanis. Overview of the RANSAC Algorithm. *Technical report, York University*, 2010.

[29] Ridge Regression. https://towardsdatascience.com/ridge-regression-for-better-usage-2f19b3a202db.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

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

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

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

[39] 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.

[40] 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.