FoV-Aware Edge Caching for Adaptive 360° Video Streaming

Anahita Mahzari, Afshin Taghavi Nasrabadi, Aliehsan Samiei and Ravi Prakash
The University of Texas at Dallas
Richardson, Texas
{anahita.mahzari,afshin,aliehsan.samiei,ravip}@utdallas.edu

ABSTRACT

In recent years, there has been growing popularity of Virtual Reality (VR), enabled by technologies like 360° video streaming. Streaming 360° video is extremely challenging due to high bandwidth and low latency requirements. Some VR solutions employ adaptive 360° video streaming which tries to reduce bandwidth consumption by only streaming high resolution video for user's Field of View (FoV). FoV is the part of the video which is being viewed by the user at any given time. Although FoV-adaptive 360° video streaming has been helpful in reducing bandwidth requirements, streaming 360° video from distant content servers is still challenging due to network latency. Caching popular content close to the end users not only decreases network latency, but also alleviates network bandwidth demands by reducing the number of future requests that have to be sent all the way to remote content servers. In this paper, we propose a novel caching policy based on users' FoV, called FoV-aware caching policy. In FoV-aware caching policy, we learn a probabilistic model of common-FoV for each 360° video based on previous users' viewing histories to improve caching performance. Through experiments with real users' head movement dataset, we show that our proposed approach improves cache hit ratio compared to Least Frequently Used (LFU) and Least Recently Used (LRU) caching policies by at least 40% and 17%, respectively.

KEYWORDS

Adaptive 360° video streaming; Caching strategy; Field of View (FoV); Edge caching

ACM Reference Format:
Anahita Mahzari, Afshin Taghavi Nasrabadi, Aliehsan Samiei and Ravi Prakash. 2018. FoV-Aware Edge Caching for Adaptive 360° Video Streaming. In *MM '18: 2018 ACM Multimedia Conference, Oct. 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240680

1 INTRODUCTION

Several content providers are developing virtual reality (VR) technologies to provide a fully immersive user experience. This can be achieved by streaming high-resolution 360° video to users' VR headset. These videos are not limited to gaming and entertainment areas. They are employed in other scenarios such as medicine, museums, education and sports [31]. According to the prediction of Cisco Visual Networking Index [16], VR headsets are going to increase to 100 million by 2021 and about half of them are expected to be mobile VR headsets. Delivering multimedia content to such VR headsets would require high network bandwidth as these videos have much higher bitrate and require more storage space: up to 4-5 times compared to traditional videos [4]. Therefore, with the rapid increase in the number of VR headsets, the communication network can potentially become a bottleneck.

Some VR solutions employ adaptive 360° video streaming which tries to reduce bandwidth consumption by leveraging information about user's Field of View (FoV). The FoV of a user is defined as the portion of the 360° video that is in the user's line of sight. Therefore, 360° video is spatially divided into several independent tiles. Bandwidth consumption can be reduced by sending only the tiles in user's FoV at high resolution, while sending other tiles at low resolution or not at all [12, 14, 28]. Although FoV-adaptive 360° video streaming has been useful for reducing bandwidth requirements, streaming 360° video from distant content servers is still challenging due to network latency.

Deploying a proxy cache between users and content servers can significantly decrease network latency and bandwidth consumption by reducing the number of requests that have to be sent all the way to remote content servers [2]. It also reduces the load on the content servers and make them more scalable. In this regard, some works leverage caching service in Content Delivery Networks (CDNs) architecture to improve user's quality of experience (QoE) [21].

CDNs are effective in reducing the network latency and bandwidth consumption over the Internet. However, based on the growing demand on mobile data traffic, recent studies try to bring content one step closer to the user and deploy caching server at the edge of the radio access network (RAN) (e.g., base stations) to enhance mobile network performance and maintain QoE [2, 6, 24, 29, 30]. This strategy allows some of the client's request for video tiles to be satisfied without contacting CDNs which result in reducing load of CDNs, response time and RAN backhaul link consumption [2, 13]. In addition, caching at the edge of RAN can support spatial and temporal decisions in updating cache's content in near real-time [17, 18]. We consider the uplink and downlink performance of 360° video transmission over mobile network. Our proposed method can be implemented as a caching module in cache-enabled base stations (BSs) at the edge of RAN.

A brute force solution to tackle the problem of caching tile-based 360° video would be to cache all tiles of a video at all resolutions for the whole 360° video, regardless of whether the tiles belonged to FoV or not. This method would require a huge cache or, for a fixed size cache, be capable of caching only a small number of 360° videos. Another solution could be to cache the video tiles

MM '18, October 22–26, 2018, Seoul, Republic of Korea

ACM ISBN 978-1-4503-5665-7/18/10…$15.00
https://doi.org/10.1145/3240508.3240680

only at the highest resolution and apply transcoding methods for lower resolutions. But, this approach imposes excessive processing requirements on the cache servers, which may lead to wasting computing resources [24]. Therefore, the challenges in caching tile-based 360° videos are: (i) how to populate the finite cache sized with tiles of the appropriate resolution, and (ii) which tiles to evict from the cache to make space for new tiles whenever the cache capacity is exceeded.

In this paper we propose caching schema which takes tile-based 360° video streaming into account along with opportunities of caching in cache-enabled BSs.

Our contributions can be summarized as follows:

- We propose new caching policy, namely FoV-aware caching policy, for tile-based 360° video streaming. In FoV-aware caching policy we learn a probabilistic model of common-FoV for each 360° video based on previous users' watching histories. Caching module applies this knowledge to determine the priority for evicting tiles, whenever the cache capacity is exceeded.
- We adapt well-known caching policies, such as Least Recently Used (LRU) and Least Frequently Used (LFU), to tile-based 360° video streaming and analyze the performance of these policies.
- We evaluate FoV-aware caching policy with real users' head movement dataset and real 4G cellular network bandwidth traces.
- Experimental results show that FoV-aware caching policy significantly improves cache hit ratio and bandwidth saving, compared to LRU and LFU. Furthermore, in the context of the number of rebuffering events, duration of rebuffering events and number of video segments for which tiles inside user's FoV are streamed at high quality, FoV-aware caching policy considerably outperforms conventional end-to-end scenario (i.e., no caching and all video fetched from remote content server in the core network).

The rest of this paper is organized as follows. In Section 2, the background and related works are presented. System architecture is explained in Section 3. Section 4 presents the proposed method. Section 5 explains the experiment setup and section 6 evaluates the experimental results. Finally, Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORKS

The main studies related to our work can be divided into two areas. First of all, general research work about 360° video streaming is explained and how recent work tries to leverage the stringent QoE requirements of 360° video streaming. Secondly, we present studies about edge computing and caching, followed by using these technologies in 360° video.

Overview of 360° video streaming: 360° video or omnidirectional video contains view of a scene from a single point in all directions. This type of video can be modeled as a sphere. The sphere has to be projected to a 2D plane for encoding because existing encoders work on 2D rectangles. There are different projections such as equirectangular and cube-map projections. Equirectangular projection is the most common method [12, 22]. We use equirectangular projection. However, any other projection can be used.

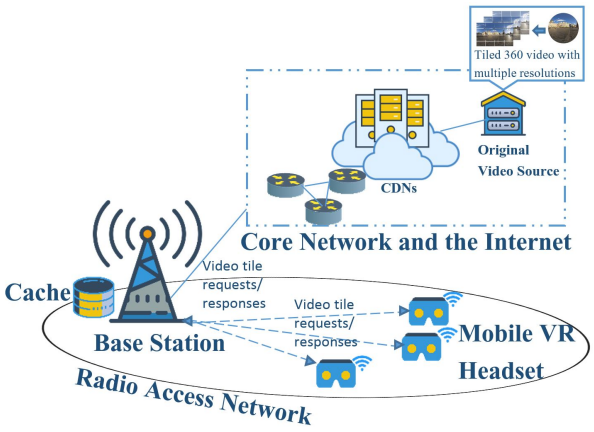

Figure 1: Architecture of edge caching

In 360° video, at each point in time, users only view a subset of video, specifically the part that lies in their FoV. In order to provide high quality video inside FoV, the actual resolution of a 360° video should be 4K or above. Therefore, 360° videos have higher bitrate than ordinary videos. To handle streaming these videos, FoV-adaptive streaming solutions are proposed. The video is divided into tiles, and tiles inside user's FoV are streamed at high quality [27]. This method enables users to watch 360° video at resolutions up to 16K with much lower bandwidth and processing power. Tiling method and FoV prediction for adaptive streaming is used in [26]. It shows that linear regression method can predict user's future FoV for up to 2 seconds, thus enabling pre-fetching of relevant FoV tiles for smooth video playback. HTTP/2 protocol was used in [25] to stream several tiles with higher performance. In [22], Scalable Video Coding (SVC) encoding method is applied to tiles, and tiles are streamed layer by layer. This method can reduce the number of video rebuffering events. Also, in order to tackle the problem of multiple decoders at the user side to decode each tile, [33] use the motion-constrained tile set (MCTS) feature of High Efficiency Video Coding (HEVC) standard to have a single hardware decoder.

Edge computing and caching: edge technology such as computing and storage at the edge of the network, close to the end users, can be utilized for streaming advanced multimedia such as 360° video. The emergence of such techniques reduce the end-to-end latency, bandwidth consumption and energy consumption for such high-quality videos[31]. Authors in [7] use edge computing to achieve real-time response for interactive multimedia and video streaming. [15] addresses the challenge of bitrate and latency requirements by proposing the rendering task to be done either at the cloud servers or at the edge server. In the context of mobile networks, [20] takes advantages of Mobile edge computing (MEC) to process and render FoV in order to optimize the bandwidth consumption and battery utilization. [8] studies the cooperation of small-cell Base Stations (BSs) which are connected through backhaul link and designs an optimization framework to maximize the reward for each BS by delivering 360° navigable videos to users.

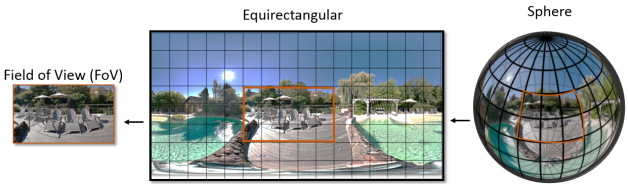

Figure 2: Mapping FoV on equirectangle projection

3 SYSTEM ARCHITECTURE

3.1 Network Model

As depicted in Figure 1, we consider a scenario that has a cache-enabled base station (BS), deployed in close proximity of clients donning mobile VR headsets. The cache-enabled BS has limited cache capacity. It needs to make space whenever the cache capacity is exceeded. Therefore, its cache replacement policy can have a significant impact on the system's utility.

Two scenarios may happen in this architecture: cache hit and cache miss. If cache has the requested content, it sends back the content immediately. Otherwise, it first downloads the requested content from the remote Internet CDNs and then sends it back to the client. Hence, a cache miss increases the download latency experienced by the client, and also increases bandwidth consumption. Therefore, increasing cache hit would benefit both the user and the network.

3.2 360° Video Model

We assume that all videos are pre-recorded[1]. For simplicity of analysis, we assume that all videos have the same duration. A video is divided into s segments, with each segment of fixed duration in playback time equal to 1 second. There are 30 frames per second of video (i.e., 30 frames per segment). Each frame is an equirectangular representation of the 360° spherical view, and is divided into t tiles. Tiles are encoded in q different qualities (i.e., video resolutions) and have different sizes: higher resolution encoding corresponds to greater size in bytes. For each 360° video, there is a Media Presentation Description (MPD) file which notifies the client to choose and locate appropriate quality for each tile to play. The Spatial Relationship Description (SRD) feature in DASH is used to define spatial relationships among tiles [12, 28].

Client has to decide about the quality of tiles based on FoV and network conditions. The goal is to stream highest possible quality for FoV that can be supported by current network throughput. We assume that FoV is $100° \times 100°$. Figure 2 shows how FoV is mapped on sphere and equirectangular videos. Client sends requests for segments whose playback times are in the future. So, future FoV has to be predicted. Moreover, future network throughput is unknown and has to be estimated. So, quality adaptation on client side has two main components: FoV prediction, and throughput estimation.

3.2.1 FoV Prediction. Adaptation algorithm takes samples from client's FoV every 100 ms, and uses 10 most recent FoV samples as input for the Weighted Linear Regression (WLR) method to predict future FoV, similar to the method proposed in [26]. This method

[1]We do not consider live streaming of 360° video.

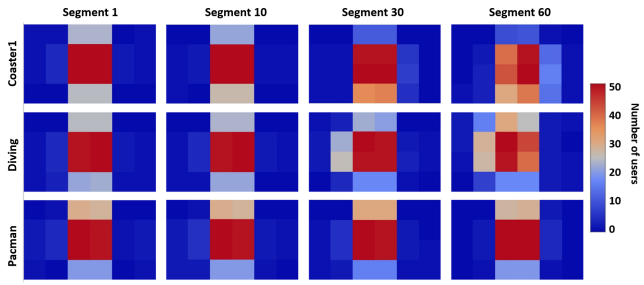

Figure 3: Heatmap of user's FoV pattern

can predict future FoV for horizon of 1 second with accuracy of 97%. Client, in its request, specifies whether the requested tile is in its FoV or not using a flag. This flag is later used in our proposed FoV-aware caching algorithm.

3.2.2 Network Throughput Estimation. For each segment, client requests high resolution for tiles within FoV and low resolution for other tiles, if estimated network throughput supports it. Otherwise, client requests low resolution for all tiles. Fetching FoV tiles at high resolution is not advisable when the network is congested because the requested tile may not finish downloading by the time it has to be displayed to the user: missing the download deadline, which could cause video playback rebuffering.

A throughput sample is the estimated throughput achieved while downloading one video segment. If we have t tiles in one segment, denoted by T_j and $j = 1, \ldots, t$, then throughput sample is calculated based on equation 1. Network throughput estimation is done by averaging last three throughput samples [22].

$$throughput\_sample = \frac{\sum_{j=1}^{t} size(T_j)}{\sum_{j=1}^{t} download\_time(T_j)} \quad (1)$$

After estimating network throughput, client determines if predicted FoV can be streamed at high quality. Assume that f tiles are covered by predicted FoV, R_{high} is the bitrate for high quality tiles, and R_{low} is the bitrate for low quality tiles. Then if network throughput estimation is higher than $[\frac{f}{t}.R_{high} + \frac{t-f}{t}.R_{low}]$, high quality will be requested for the tiles inside predicted FoV. Otherwise, low quality will be requested.

4 PROPOSED METHOD

We first describe two characteristics of adaptive 360° video streaming, users' FoV pattern and the relation between FoV's quality and videos' bitrate. Then, we present FoV-aware caching policy taking these two characteristics into account.

4.1 Users' FoV Pattern

Users' FoV adds very important variable into the adaptive 360° video streaming. Users are going to interact with multimedia stream by moving their head around. To a great extent, users' FoV would be influenced by video content. If most of the action is in front of the users, as simulated ride in a race car, the users' FoV may

be limited to scene in front of the users, and there may be a low propensity to turn around and see what is happening behind the viewer. On the other hand, 360° video tour of the US Capitol in Washington, D.C. may encourage frequent FoV changes. However, it is observed that FoV for different users follow similar pattern when watching the same 360° video [5, 10, 32]. For instance, [32] collects a head tracking dataset of real traces and shows that users in virtual environment have certain similar viewing patterns when watching the same 360° video. [5] considers these similar viewing patterns and proposes bandwidth-efficient multicast and unicast 360° video transmissions.

We analyze real traces of users' FoV from the dataset in [19]. This dataset provides users' FoV for ten different 360° videos. Each video is viewed by 50 users. We compute heatmap of users' FoV for each segment of all 360° videos. Each heatmap is an equirectangular representation of a segment of 360°. Due to space limitation, we only show the heatmap of users' FoV for segment numbers (1, 10, 30, 60) of three 360° videos in Figure 3. We chose these three videos as representatives of three categories presented in the original dataset. We chose coaster1 from (i) Natural Image-fast-paced, diving from (ii) Natural Image-slow-paced and pacman from (iii) Computer Generated-fast-paced.

Each person can potentially view a 360° video in a unique fashion. However, as we can observe in Figure 3, users' FoVs overlap with each other. By way of illustration, consider the 10th segment of coaster1, heatmap of this segment shows that most of the users' FoVs are concentrated in the red region. We refer to this region of interest as common-FoV for each segment of 360° video.

4.2 Impact of Video's Bitrate on FoV's Quality

Different 360° videos are encoded with variable bitrate, and average bitrate of a video depends on texture and motion of video content. More complex scenes, in terms of texture and motion, need higher bitrate to provide the same level of quality [1].

Let R_{high} denote the bitrate for high quality tiles, and R_{low} denote the bitrate for low quality tiles. To compute bitrate of different 360° videos, we consider two scenarios, scenario_1: effective bitrate, and scenario_2: all low. In scenario_1, estimated network throughput supports the bitrate requirement for streaming high quality tiles. Therefore, client requests high quality for tiles inside FoV and low quality for other tiles. In scenario_2, client requests low quality for all tiles. We use the notion of effective bitrate for 360° videos from [1]. They consider the fraction of the video which is covered by FoV, and compute the effective bitrate based on that. Using the definition of effective bitrate, we also use the same calculation to compute the effective bitrate for all videos. We assume FoV is 100°×100°. Therefore, the portion of video inside FoV would be $[\frac{100^\circ}{180^\circ} * \frac{100^\circ}{360^\circ} = 14\%]$. So, bitrate for scenario_1 would be $[14\%.R_{high} + 86\%.R_{low}]$ and for scenario_2 would be $[100\%.R_{low}]$.

To understand the effect of different 360° video's bitrate on receiving FoV's quality, we examine the situation that a client watches ten different 360° videos with different bitrates under the same network conditions. FoV's quality shows the fraction of segments which client receives FoV with high quality. We use ten 360° videos from dataset in [19]. We consider the client connects to the content server via BS at the cellular network and watches video_1 to

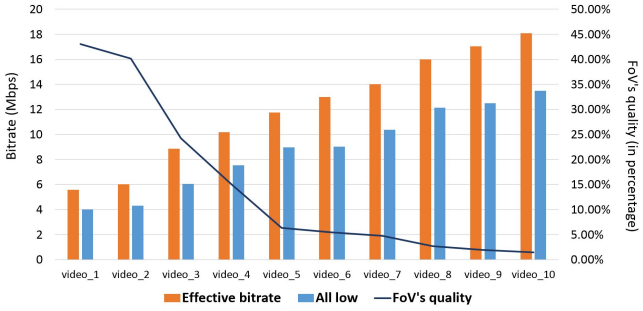

Figure 4: Impact of 360° videos' bitrate on FoV's quality

video_10 with different bitrate, under the same network conditions. The cellular network is emulated using MahiMahi emulator with a real-world 4G network bandwidth trace (with average of 26 Mbps) [23]. For the network condition between BS and content server, we assume link capacity is 1 Gbps, latency (round-trip time) and packet loss are 10 ms and 0.01, respectively, as it is used in [11].

Figure 4 demonstrates the results of relationship between FoV's quality and ten 360° videos with different bitrate in scenario_1: effective bitrate, and scenario_2: all low. We can observe that when the client watches video_1 with lower bitrate in both scenarios, it receives FoV with higher quality tiles. However, when the client watches video_10 with higher bitrate in both scenarios, it receives FoV with lower quality tiles. Therefore, video's bitrate affect receiving FoV's quality on the user side.

4.3 FoV-aware Caching Policy

As described in subsection 3.2.2, we use the adaptation logic for 360° videos in which the user always requests low quality tiles outside FoV and an appropriate quality (i.e, high or low) of the tiles inside FoV, based on current network throughput. According to this adaption logic and the two characteristics discussed in subsections 4.2 and 4.1, FoV-aware caching policy considers two parameters to prioritize tiles for removing from cache whenever cache is full. One parameter is tiles' probabilities for being inside common-FoV and the other one is probability of users requesting high quality version of tiles inside FoV for each video. FoV-aware caching policy learns these two parameters based on viewing history.

4.3.1 Learning Parameters. For each tile number i of segment s of video v, $\theta_{v,s,i}$ is the probability of tile number i being inside common-FoV. Each data sample for learning $\theta_{v,s,i}$ is a request for the corresponding tile and the only feature is whether the tile is requested as part of FoV or not. For each video v, ψ_v is the probability that a user watching video v, requests high quality version of tiles inside his/her FoV. Each data sample for learning ψ_v is a request for any tile of video v which was part of the FoV. Again there is only one feature and that is whether the high quality version or low quality of the tile is requested.

Considering this fact that there is only one feature for both $\theta_{v,s,i}$ and ψ_v and its simplicity that almost impose no extra overhead on the cache, we choose Naïve-Bayes algorithm to learn a probabilistic model for each of the parameters. By using Maximum Likelihood

Estimation (MLE), we are looking for θ and ψ that makes the observed data $\mathcal{D}$ more probable. Mathematically, we want $\theta_{v,s,i}$ and ψ_v as stated in equation 2.

$$\begin{aligned}\theta_{v,s,i} &= \arg\max_{\theta} P(\mathcal{D}_{\theta_{v,s,i}} \mid \theta) \\ &= \arg\max_{\theta} ln(\theta_{v,s,i}^{\alpha_{inFoV_{v,s,i}}} * (1-\theta_{v,s,i})^{\alpha_{outOfFoV_{v,s,i}}}) \\ \psi_v &= \arg\max_{\theta} P(\mathcal{D}_{\psi_v} \mid \psi) \\ &= \arg\max_{\psi} ln(\psi_v^{\beta_{high_v}} * (1-\psi_v)^{\beta_{low_v}})\end{aligned} \tag{2}$$

where $\alpha_{inFoV_{v,s,i}}$ and $\alpha_{outOfFoV_{v,s,i}}$ are the number of times a tile number i of segment s of video v, is requested as part of the user's FoV and outside of FoV, respectively. β_{high_v} and β_{low_v} are the number of times a high quality version and a low quality version of any tile inside FoV of video v is requested, respectively.

In order to maximize the function in equation 2 we set their derivatives to zero. Derivations for both parameters are the same. Therefore, we suffice to show the derivations only for $\theta_{v,s,i}$ which is shown in equation 3.

$$\begin{aligned}&\frac{d}{d\theta}\left[ln(\theta_{v,s,i}^{\alpha_{inFoV_{v,s,i}}} * (1-\theta_{v,s,i})^{\alpha_{outOfFoV_{v,s,i}}})\right] = \\ &\frac{d}{d\theta}\left[\alpha_{inFoV_{v,s,i}} ln(\theta_{v,s,i}) + \alpha_{outOfFoV_{v,s,i}} ln(1-\theta_{v,s,i})\right] = \\ &\alpha_{inFoV_{v,s,i}} \frac{d}{d\theta} ln(\theta_{v,s,i}) + \alpha_{outOfFoV_{v,s,i}} \frac{d}{d\theta} ln(1-\theta_{v,s,i}) = \\ &\frac{\alpha_{inFoV_{v,s,i}}}{\theta_{v,s,i}} - \frac{\alpha_{outOfFoV_{v,s,i}}}{1-\theta_{v,s,i}} = 0\end{aligned} \tag{3}$$

Setting the derivations to zero, the maximum values for $\theta_{v,s,i}$ and ψ_v can be found as in equation 4.

$$\begin{aligned}\theta_{v,s,i} &= \frac{\alpha_{inFoV_{v,s,i}}}{\alpha_{inFoV_{v,s,i}} + \alpha_{outOfFoV_{v,s,i}}} \\ \psi_v &= \frac{\beta_{high_v}}{\beta_{high_v} + \beta_{low_v}}\end{aligned} \tag{4}$$

It is reassuring to note that these equations yield intuitively obvious expressions for $\theta_{v,s,i}$ and ϕ_v. Specifically, $\theta_{v,s,i}$ is the fraction of times that a tile is being inside common-FoV and ϕ_v is the fraction of times high quality version of the tile is requested, thus far.

4.3.2 Caching Policy. The algorithm 1 shows the overview of FoV-aware caching policy in pseudocode. The algorithm describes the actions caching module takes upon receiving a request for tile number i of segment s of video v with quality q. It first calls *update_parameters* which updates the parameters $\theta_{v,s,i}$ and ψ_v and returns $\gamma_{v,s,i,q}$, which is the probability of tile $t_{v,s,i}$ being requested in future with quality q. Then, it calls *cache.contains* to check the cache and provides the tile if it has been cached before. Otherwise, caching module calls *download* to fetch the tile from the content server and add the tile to the cache with *cache.add*. Whenever the cache capacity is exceeded, *cache.remove_min* removes tiles with the lowest γ.

Algorithm 1: FoV-aware caching algorithm

```
/* • cache is a min-heap maintains cached tiles ordered
     on their γ as key.
   • remove_min() is a function removing the tile with
     lowest γ from cache.
   • update_parameters is a function that updates θ_{v,s,i}
     and ψ_v and calculates γ_{v,s,i,q}.
   • cache_capacity is the total amount of space for
     caching.
   • size() is a function returning the amount of space
     used so far in cache.
   • is_in_FoV() is a boolean function that returns
     whether the requested tile is part of the client's FoV
     or not, using the flag sent in the client's request.
*/
Function upon_request(t_{v,s,i,q}):
    γ_{v,s,i,q} ← update_parameters(t_{v,s,i,q})
    if cache.contains(t_{v,s,i,q}) then
        tile ← cache.get(t_{v,s,i,q})
        send(tile)
    else
        // downloading tile from remote content server
        tile ← download(t_{v,s,i,q})
        // adding tile to cache
        cache.add(tile, key=γ_{v,s,i,q})
        // sending tile to client
        send(tile)
        while cache.size() > cache_capacity do
            cache.remove_min()
        end
    end
Function update_parameters(t_{v,s,i,q}):
    if is_in_FoV(t_{v,s,i,q}) then
        α_{inFoV_{v,s,i}} += 1
        if q == high then
            β_{high_v} += 1
        else
            β_{low_v} += 1
        end
    else
        α_{outOfFoV_{v,s,i}} += 1
    end
    θ_{v,s,i} = α_{inFoV_{v,s,i}} / (α_{inFoV_{v,s,i}} + α_{outOfFoV_{v,s,i}})
    ψ_v = β_{high_v} /(β_{high_v} + β_{low_v})
    if q == high then
        γ_{v,s,i,q} = θ_{v,s,i} * ψ_v
    else
        γ_{v,s,i,q} = (1 - θ_{v,s,i}) + θ_{v,s,i} * (1 - ψ_v)
    end
    return γ_{v,s,i,q}
```

$\gamma_{v,s,i,q}$ can be formulated as shown in equation 5. There are two possibilities for a low quality version of a tile being requested. One is that the tile is outside of the FoV of the user which happens with probability $(1-\theta_{v,s,i})$. The other possibility is that the tile

is inside FoV of the client and due to the network throughput estimation the client prefers to request the low quality version which happens with probability $\theta_{v,s,i} * (1 - \psi_v)$. On the other hand for a high quality version of tile being requested there is only one situation in which the tile is inside FoV of the client and network throughput estimation supports streaming of high quality version of tile. This happens with probability $\theta_{v,s,i} * \psi_v$. Caching module computes $\gamma_{v,s,i,q}$ to determine the priority of tile $t_{v,s,i,q}$ with quality q. Caching module removes tiles with the lowest γ, whenever the cache is full.

$$\gamma_{v,s,i,q} = \begin{cases} \theta_{v,s,i} * \psi_v, & \text{if } q = high \\ (1 - \theta_{v,s,i}) + \theta_{v,s,i} * (1 - \psi_v), & \text{if } q = low \end{cases} \quad (5)$$

5 EXPERIMENT SETUP

We choose Mahimahi LinkShell [23] to emulate the behavior of the mobile network. Mahimahi records and replicates the behavior of real-life network, it also includes pre-recorded 4G traces. Traces shows the uplink and downlink transmission opportunity (TxOp) is given to a mobile device. Figure 5 shows the Cumulative Distribution Function (CDF) of available bandwidth traces which we used. Also, in order to emulate different network conditions between cache-enabled BSs and rest of the network, we use the same network parameters as in [11]. Link capacity is assigned to 1 Gbps. Latency (round-trip time) and packet loss is shown in Table 1. For convenience of reading, we refer to Scenario_500, Scenario_300, Scenario_200, Scenario_100 and Scenario_10 for the rest of the paper. For supporting all scenarios, we create an artificial trace with a constant bandwidth of 1 Gbps and using *mm-delay* and *mm-loss* in Mahimahi LinkShell, to emulate a delayed, lossy link for these scenarios.

In terms of cache capacity, we consider two cases: (i) case_1: 25% of total size of all 360° videos in both high and low quality representations equal to 0.61 GB, (ii) case_ 2: 50% of total size of all 360° videos in both high and low quality representations equal to 1.22 GB.

5.1 360° Videos

We used 10 video sequences from 360° video viewing dataset provided by [19]. This dataset covers a wide variety of 360° videos with different contents (all videos are available on YouTube). This dataset includes FoV traces of 50 clients watching each of these 360° videos. In total we have 500 viewing traces. In our system, we consider clients create 360° video's session based on Poisson distribution with arriving rate of $\lambda = \frac{1}{30sec}$ which makes clients to start watching 360° video every 30 seconds on average.

We divide each 360° video into 24 tiles, 6 columns and 4 rows. We have chosen 24 tiles, because it provides a good trade-off between encoding efficiency and bandwidth saving, as it was investigated in [12]. We encoded each tile using *ffmpeg*[2] H.264 encoder at two resolutions: 640×540 and 320×270 pixels, representing high and low quality tiles. So resolution of complete 360° video is 3840×2160 at high quality, and 1920×1080. We encode each tile with Constant Rate Factor (CRF) of 21 and set maximum bitrate per tile to 3Mbps

[2]https://www.ffmpeg.org/

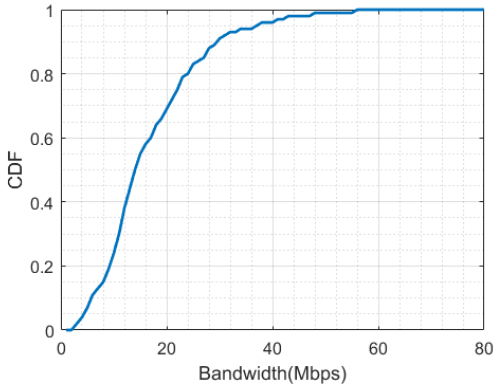

Figure 5: CDF of network bandwidth traces

Table 1: latency and packet loss

Scenario	latency	Packet Loss
Scenario_500	500 ms	0.05%
Scenario_300	300 ms	0.05%
Scenario_200	200 ms	0.04%
Scenario_100	100 ms	0.03%
Scenario_10	10 ms	0.01%

for high resolution tiles, and 1Mbps for low resolution tiles. After encoding, we divide each video into segments with duration of 1 second using GPAC MP4Box tool[3]. So each tile of a segment would be available in a separate file, and client can adapt quality at tile level for each segment. Average bitrate for high quality representation of all videos (including all tiles) is 26.3Mbps, and for low quality representation is 8.7Mbps.

6 PERFORMANCE EVALUATION

We compare FoV-aware caching policy with two well-known and extensively used caching policies including Least Frequently Used (LFU) and Least Recently Used (LRU). To the best of our knowledge, there is no related caching policy for tile-based adaptive 360° video streaming. As a consequence, we adapt LRU and LFU to tile-based adaptive 360° video streaming and compare them with FoV-aware caching policy. In addition, we consider end-to-end scenario (i.e., no caching) as baseline to evaluate and compare the performance of all three caching policies. These methods are explained as follows:

end-to-end: There is no caching schema between client and content server. Client sends request directly to the content server. Content server is responsible for delivering clients' requests.

LFU: In LFU caching policy, a tile with lowest access frequency has the highest rank for being removed from cache to make space for new tiles, when the cache is full. We keep frequency information of the tile, though the tile is not in the cache [9].

LRU: In LRU caching policy, a tile with longest time duration since the last access of the tile, has the highest rank for being

[3]https://gpac.wp.imt.fr/mp4box/

removed from cache to make space for new tiles, when the cache is full [9].

In terms of metrics, we use following metrics to verify the performance of FoV-aware with LRU, LFU and end-to-end.

- Cache hit ratio: The fraction of requested tiles that are found in the cache. This metric is referred to as *Tile hit* in results.
- Bandwidth saving (in percentage): The amount of bandwidth which is saved, when client's requests result in cache hit, over total bandwidth consumption of all requests. This metric is referred to as *Bandwidth saving* in results.
- Rebuffering percentage: When a client does not have any segment in buffer to play, video playback stalls. This is called rebuffering. This metric shows the fraction of segments for which client experiences rebuffering. This metric is referred to as *Reb* in results.
- Duration of rebufferings (in seconds): This metric shows the total duration of time that a client experiences rebufferings. This metric is referred to as *DoR* in results.
- High quality for FoV (in percentage): The number of segments for which client receives tiles inside FoV with high quality over the total number of segments of 360° video. This metric is referred to as *Qua* in results.

As described in section 5.1, we have 500 viewing traces (i.e., 10 videos, each of them is viewed by 50 users). We run the experiment for each caching policy and end-to-end over all viewing traces, three times. In following subsections, we show the average of the experiment results and compare FoV-aware caching policy with LRU, LFU and end-to-end.

6.1 Caching Policy Performance

Any caching policy's primary target is increasing cache hit ratio under limited cache capacity. Higher cache hit reduces the number of requests to the content server and that results in saving bandwidth in the core network. Therefore, if cache hit is the performance metric to be maximized, FoV-aware caching policy is the best compared to LRU and LFU. By way of illustration, consider Figure 6, where network latency is 100 ms and cache capacity is 25% of total size of all videos, LRU reaches 71.05% in terms of bandwidth saving and cache hit, respectively, and LFU reaches 49.47% for cache hit. However, FoV-aware caching policy improves caching performance significantly. Its cache hit reaches to 91.82%. Also, we show the result of bandwidth saving for the same cache capacity in Figures 7. Again, we can observe FoV-aware caching policy has better results compared to LRU and LFU.

FoV-aware caching policy capitalizes on the unique characteristics of 360° video (i.e., FoV), while LRU and LFU consider the last time and the number of times a tile was requested, respectively, to prioritize tiles and remove from the cache when the cache is full. Therefore, the best caching policy would be the one that removes tiles with least likelihood of being requested in the future. FoV-aware caching policy exploits the common-FoV phenomenon (i.e., the FoVs of most users watching the same video have significant overlap) to drive a better prediction for future requested tiles. FoV-aware caching policy applies probabilistic learning model to learn

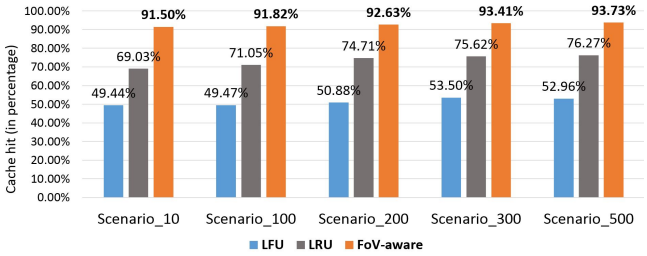

Figure 6: Results of cache hit for LRU, LFU and FoV-aware - All scenarios - Cache capacity: case_1

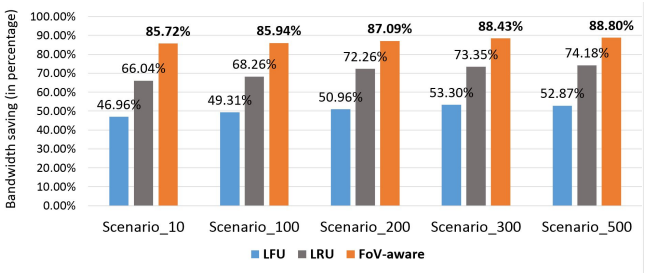

Figure 7: Results of bandwidth saving for LRU, LFU and FoV-aware - All scenarios - Cache capacity: case_1

common-FoV based on viewing history. Therefor, tiles in common-FoV that are more likely to be requested in future, are more likely to reside in the cache.

As we move from Scenario_10 to Scenario_500 (i.e., from low latency to high latency), we observe that cache hit and bandwidth saving metric for all caching polices increase slightly. The reason behind this is the fundamental idea of client adaptation algorithm which client behaves more conservatively and requests lower quality tiles when experiencing higher latency, as it is described in subsection 3.2.2. Also, low quality tiles occupy less cache space compared to high quality ones. Therefor, when client requests more low quality tiles, more low quality ones are cached resulting in higher cache hit ratio.

6.2 Streaming Performance

Caching at the edge of the network can decrease network latency by masking network's delay. Reduced network delay is interpreted as high throughput situation by a client. Therefore, the client requests more tiles inside FoV at high quality, as it is described in subsection 3.2.2. We use end-to-end scenario as the baseline to evaluate the performance of QoE's metric (i.e,. rebuffering percentage, duration of rebuffering and FoV's quality) for all three caching policies. Table 2 shows the results of QoE's metrics in end-to-end and Tables 3 demonstrates the results for the same QoE's metrics for LRU, LFU and FoV-aware caching policy under the same network latency for cache capacity of case_1.

As we move from scenario_10 to scenario_500 (i.e., from low to high latency), we observe that all caching policies improve QoE's metrics compared to end-to-end scenario. Although, the amounts of

Table 2: Number, duration of rebuffering events and high quality for FoV in End to End

Scenarios	10			100			200			300			500		
	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua
End to End	20.84%	5.75 Sec	20.65%	22.85%	6.17 Sec	11.08%	25.84%	7.02 Sec	3.12%	26.69 %	8.16 Sec	0.31%	40.19%	10.92 Sec	0.0%

Table 3: Number, duration of rebuffering events and high quality for FoV in FoV-aware, LFU and LRU - All scenarios - Cache capacity: case_1

Scenarios	10			100			200			300			500		
	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua	Reb	DoR	Qua
FoV-aware	**20.20%**	**5.71** Sec	**22.08%**	**20.36%**	**5.73** Sec	**21.45%**	**20.78%**	**5.78** Sec	**19.17%**	**20.92%**	**5.80** Sec	**16.32%**	**22.50%**	**6.10** Sec	**15.18%**
LFU	20.71%	5.75 Sec	19.24%	22.34%	6.03 Sec	14.83%	23.86%	6.47 Sec	10.88%	25.53%	6.94 Sec	8.84%	31.75%	8.46 Sec	7.50%
LRU	20.69%	5.72 Sec	21.49%	21.16%	5.79 Sec	18.77%	21.61%	5.87 Sec	14.52%	21.76%	6.07 Sec	11.61%	24.81%	6.69 Sec	10.23%

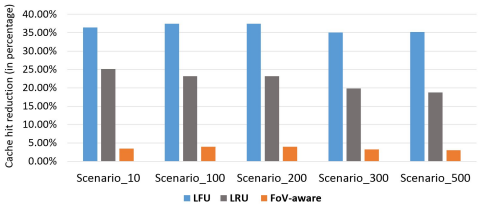

Figure 8: Cache hit sensitivity to cache size for LRU, LFU and FoV-aware - All scenarios - Reducing cache capacity from case_2 to case_1

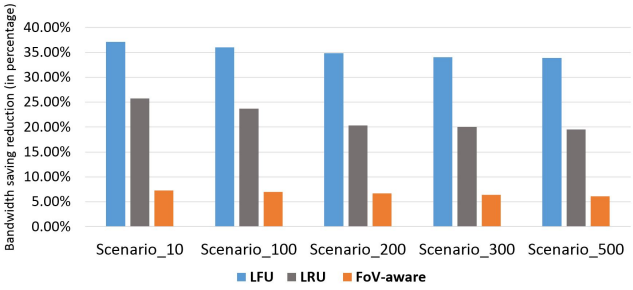

Figure 9: Bandwidth saving sensitivity to cache size for LRU, LFU and FoV-aware - All scenarios - Reducing cache capacity from case_2 to case_1

rebufferings in scenarios_10 and scenarios_100 using three caching policies are close to end to end results, under challenging network conditions such as scenario_300 and scenario_500, all caching policies achieves better results, especially FoV-aware caching policy. Because caching at the edge of mobile networks would not hide the wireless link fluctuations in mobile networks and due to low viewport prediction accuracy in 360° video, client can buffer up to 2 seconds, while traditional video can buffer up to 30 seconds[3]. Such a shallow buffer in client's side alongside with high fluctuations in wireless area can quickly deplete the buffer and cause the rebuffering. This situation is getting worse by increasing latency (i.e,. scenario_300 and scenario_500), however, caching content close to the user can be significantly reduced rebuffering and increased FoV's quality. For instance, in scenario_300 and scenario_500, FoV-aware caching policy enables clients to receive 16.32% and 15.18% of high quality tiles inside their FoV, respectively, compared to end-to-end scenario which can not deliver high quality for FoV in scenario_500 and just 0.31% in scenario_300.

6.3 Impact of Cache Capacity

In this section, we study the impact of cache capacity for all three caching polices. We double the cache capacity from case_1 to case_2, in order to observe the effect of cache capacity on cache hit ratio and bandwidth saving. The performance of all caching polices in terms of cache hit ratio and bandwidth saving, decreases when cache size reduces, as expected. However, FoV-aware caching policy performs significantly better compared to LRU and LFU in less amount of cache space. In other words, LRU and LFU performances is more sensitive to cache capacity reduction. As we can see from Figure 8, when cache capacity is halved, cache hit ratio reduced by 4% for FoV-aware caching policy while for LRU and LFU it is up to 25% and 37%, respectively. In terms of bandwidth saving, as it is shown in Figure 9, FoV-aware caching policy is reduced up to 7% while for LRU and LFU it is up to 26% and 38%, respectively.

7 CONCLUSIONS

Caching contents at the edge of mobile network can be utilized to conceal the drawbacks of duplicate downloads of popular items and reduce long network latency and bandwidth consumption. Putting all these benefits together, caching can improve 360° video streaming, especially in dynamic network condition such as mobile networks. In this paper, we proposed a caching policy for tile-based adaptive 360° video streaming by identifying common-FoV and using this knowledge to achieve better cache performance. Results show the superiority of our scheme compared to LRU and LFU not only in terms of cache hit ratio and bandwidth saving, but also increasing the QoE by reducing the number and duration of rebuffering events and providing high quality for FoV.

8 ACKNOWLEDGMENTS

This work was supported in part by a gift from Cisco Systems, Inc.

REFERENCES

[1] Shahryar Afzal, Jiasi Chen, and KK Ramakrishnan. 2017. Characterization of 360-degree Videos. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 1–6.

[2] Hasti Ahlehagh and Sujit Dey. 2012. Video caching in radio access network: Impact on delay and capacity. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*. IEEE, 2276–2281.

[3] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. 2011. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 157–168.

[4] Yanan Bao, Tianxiao Zhang, Amit Pande, Huasen Wu, and Xin Liu. 2017. Motion-Prediction-Based Multicast for 360-Degree Video Transmissions. In *Sensing, Communication, and Networking (SECON), 2017 14th Annual IEEE International Conference on*. IEEE, 1–9.

[5] Yanan Bao, Tianxiao Zhang, Amit Pande, Huasen Wu, and Xin Liu. 2017. Motion-Prediction-Based Multicast for 360-Degree Video Transmissions. In *Sensing, Communication, and Networking (SECON), 2017 14th Annual IEEE International Conference on*. IEEE, 1–9.

[6] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. 2014. Living on the edge: The role of proactive caching in 5G wireless networks. *IEEE Communications Magazine* 52, 8 (2014), 82–89.

[7] Kashif Bilal and Aiman Erbad. 2017. Edge computing for interactive media and video streaming. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*. IEEE, 68–73.

[8] Jacob Chakareski. 2017. VR/AR Immersive Communication: Caching, Edge Computing, and Transmission Trade-Offs. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 36–41.

[9] Hui Chen and Yang Xiao. 2006. Cache access and replacement for future wireless Internet. *IEEE Communications Magazine* 44, 5 (2006), 113–123.

[10] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Optimal Set of 360-Degree Videos for Viewport-Adaptive Streaming. *in Proc. of ACM Multimedia (MM)* (2017).

[11] Chang Ge, Ning Wang, Gerry Foster, and Mick Wilson. 2017. Toward QoE-Assured 4K Video-on-Demand Delivery Through Mobile Edge Virtualization With Adaptive Prefetching. *IEEE Transactions on Multimedia* 19, 10 (2017), 2222–2237.

[12] Mario Graf, Christian Timmerer, and Christopher Mueller. 2017. Towards Bandwidth Efficient Adaptive Streaming of Omnidirectional Video over HTTP: Design, Implementation, and Evaluation. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 261–271.

[13] Jingxiong Gu, Wei Wang, Aiping Huang, Hangguan Shan, and Zhaoyang Zhang. 2014. Distributed cache replacement for caching-enable base stations in cellular networks. In *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2648–2653.

[14] Mohammad Hosseini and Viswanathan Swaminathan. 2016. Adaptive 360 VR video streaming: Divide and conquer. In *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 107–110.

[15] Xueshi Hou, Yao Lu, and Sujit Dey. 2017. Wireless VR/AR with Edge/Cloud Computing. In *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 1–8.

[16] Cisco Visual Networking Index. 2016. Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper. *link: http://goo. gl/ylTuVx* (2016).

[17] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. 2016. CFA: A Practical Prediction System for Video QoE Optimization.. In *NSDI*. 137–150.

[18] Haneul Ko, Jaewook Lee, and Sangheon Pack. 2018. Spatial and Temporal Computation Offloading Decision Algorithm in Edge Cloud-Enabled Heterogeneous Networks. *IEEE Access* 6 (2018), 18920–18932.

[19] Wen-Chih Lo, Ching-Ling Fan, Jean Lee, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2017. 360 Video Viewing Dataset in Head-Mounted Virtual Reality. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 211–216.

[20] Simone Mangiante, Guenter Klas, Amit Navon, Zhuang GuanHua, Ju Ran, and Marco Dias Silva. 2017. VR is on the Edge: How to Deliver 360-degree Videos in Mobile Networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 30–35.

[21] Kianoosh Mokhtarian and Hans-Arno Jacobsen. 2017. Flexible caching algorithms for video content distribution networks. *IEEE/ACM Transactions on Networking (TON)* 25, 2 (2017), 1062–1075.

[22] Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D. Beshay, and Ravi Prakash. 2017. Adaptive 360-Degree Video Streaming Using Scalable Video Coding. In *Proceedings of the 2017 ACM on Multimedia Conference (MM '17)*. ACM, 1689–1697. https://doi.org/10.1145/3123266.3123414

[23] Ravi Netravali et al. October 2014. Mahimahi: A Lightweight Toolkit for Reproducible Web Measurement. *ACM SIGCOMM Computer Communication Review* 44, 4 (October 2014), 129–130.

[24] Hasti A Pedersen and Sujit Dey. 2016. Enhancing mobile video capacity and quality using rate adaptation, RAN caching and processing. *IEEE/ACM Transactions on Networking (TON)* 24, 2 (2016), 996–1010.

[25] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. An HTTP/2-based adaptive streaming framework for 360Âř virtual reality videos. In *2017 ACM Multimedia (MM) Conference*. 1–9.

[26] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 1–6.

[27] Patrice Rondao Alface, Maarten Aerts, Donny Tytgat, Sammy Lievens, Christoph Stevens, Nico Verzijp, and Jean-Francois Macq. 2017. 16K Cinematic VR Streaming. In *Proceedings of the 2017 ACM on Multimedia Conference*. ACM, 1105–1112.

[28] Patrice Rondao Alface, Jean-François Macq, and Nico Verzijp. 2012. Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach. *Bell Labs Technical Journal* 16, 4 (2012), 135–147.

[29] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. 2017. A Survey on Mobile Edge Networks: Convergence of Computing, Caching and Communications. *IEEE Access* 5 (2017), 6757–6779.

[30] Xiaofei Wang, Min Chen, Tarik Taleb, Adlen Ksentini, and Victor Leung. 2014. Cache in the air: exploiting content caching and delivery techniques for 5G systems. *IEEE Communications Magazine* 52, 2 (2014), 131–139.

[31] Cedric Westphal. 2017. Challenges in networking to support augmented reality and virtual reality. In *Int. Conf. on Computing, Networking and Communications (ICNC 2017)*. 26–29.

[32] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 193–198.

[33] Alireza Zare, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 601–605.