Optimizing 360 Video Delivery Over Cellular Networks

Feng Qian
Indiana University
Bloomington, IN
fengqian@indiana.edu

Bo Han
AT&T Labs – Research
Bedminster, NJ
bohan@research.att.com

Lusheng Ji
AT&T Labs – Research
Bedminster, NJ
lji@research.att.com

Vijay Gopalakrishnan
AT&T Labs – Research
Bedminster, NJ
gvijay@research.att.com

Abstract

As an important component of the virtual reality (VR) technology, *360-degree videos* provide users with panoramic view and allow them to freely control their viewing direction during video playback. Usually, a player displays only the visible portion of a 360 video. Thus, fetching the entire raw video frame wastes bandwidth. In this paper, we consider the problem of optimizing 360 video delivery over cellular networks. We first conduct a measurement study on commercial 360 video platforms. We then propose a cellular-friendly streaming scheme that delivers only 360 videos' visible portion based on head movement prediction. Using viewing data collected from real users, we demonstrate the feasibility of our approach, which can reduce bandwidth consumption by up to 80% based on a trace-driven simulation.

CCS Concepts

•**Networks** → **Application layer protocols; Mobile networks;**
•**Computing methodologies** → **Virtual reality;**

Keywords

360-degree video; Head movement prediction; Virtual reality; Cellular networks

1. INTRODUCTION

The past three years have witnessed increasing commercial progress of the virtual reality (VR) technology, which has eventually stepped out of labs. It is projected to form a big market of $120 billion by 2020 [2]. Users can now experience VR capabilities on their mobile devices using affordable VR devices such as a $15 Google Cardboard [4]. 360-degree videos, also known as immersive or spherical videos, play a critical role in the VR ecosystem. They provide users with panoramic views and create a unique viewing experience in particular when used in combination with the 3D video technology. 360 videos are recorded by omnidirectional cameras or camera array systems (*e.g.,* Facebook Surround 360 [3]). They simultaneously record all 360 degrees of a scene that can be "wrapped" onto a 3D sphere, with the cameras at its center.

AllThingsCellular'16, October 03-07, 2016, New York City, NY, USA

DOI: http://dx.doi.org/10.1145/2980055.2980056

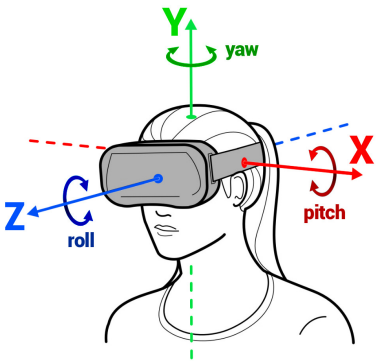

Figure 1: Adjusting 360 video viewing direction.

When watching a 360 video, a viewer at the spherical center can freely control her viewing direction, so each playback creates a unique experience. As shown in Figure 1, the user wearing a VR headset can adjust her orientation by changing the *pitch*, *yaw*, and *roll*, which correspond to rotating along the X, Y, and Z axes, respectively. Then the 360 video player computes and displays the viewing area based on the orientation and the field of view (FoV). The FoV defines the extent of the observable area, which is usually a fixed parameter of a VR headset (*e.g.,* 110° horizontally and 90° vertically).

360 videos are very popular on major video platforms such as YouTube and Facebook. Despite their popularity, the research community lacks an in-depth understanding of many of its critical aspects such as performance and resource consumption. In this paper, we fill this gap by investigating *how to optimize 360 video delivery over cellular networks*, which, we envision, form the key infrastructure that facilitates *ubiquitous* access of VR resources in the cloud. We begin with understanding the state-of-the-art of 360 video delivery by conducting measurements on two commercial 360 video platforms: YouTube and Facebook. We found that 360 video largely inherits the delivery scheme from traditional Internet videos. This simplifies the deployment, but makes 360 video streaming very cellular-unfriendly, because the video player always fetches the *entire* video including both visible and invisible portions. This leads to tremendous resource inefficiency on cellular networks with limited bandwidth, metered link, fluctuating throughput, and high device radio energy consumption.

Motivated by the above, we propose a novel cellular-friendly

streaming scheme for 360 videos. The high-level idea is the following, instead of downloading everything, the player only fetches the parts that are visible to the user in order to reduce the bandwidth consumption. This requires us to predict the viewer's head movement (to know which portion of the view to fetch). To investigate its feasibility, we collected five users' head movement traces when watching real YouTube 360 videos. Trace-driven analysis indicates that at least in the short term, viewers' head movement can be accurately predicted (with accuracy $> 90\%$) by even using simple methods such as linear regression. We also address other design challenges such as handling prediction errors and integration with DASH and HTTP.

To summarize, we make the following contributions.

• We conduct to our knowledge a first characterization study of 360 video delivery on commercial video platforms (§2).

• We presented the design of a cellular-friendly 360 video delivery scheme based on head movement prediction, whose feasibility is validated by a pilot user study (§3).

• We quantify the benefits brought by our scheme using trace-driven simulation. The results indicate that our scheme can significantly reduce the bandwidth consumption of 360 video streaming by up to 80% (§4).

2. MEASUREMENT STUDY

We conduct a measurement study to understand how 360 videos are delivered by commercial video platforms. We study the two most popular video platforms: YouTube and Facebook. We use the official YouTube app on a Samsung Galaxy S5 phone running Android 4.4.2 to watch top YouTube 360 videos. During the video playback, we capture HTTPS transactions by redirecting all traffic to a man-in-the-middle proxy (using `mitmproxy` [5]). For Facebook, we watch several popular 360 videos in a Chrome browser on a Windows 10 laptop[1], and use the Chrome debugging tool to analyze HTTPS transactions. We describe our findings as follows.

• Both YouTube and Facebook encode 360 videos into the standard H.264 format in an MP4 container. In fact, a 360 video is also playable in conventional media players, which only show the raw frames as exemplified in the large image in Figure 2. As shown, the raw frame is distorted because it was *projected* from the 3D panoramic sphere. When the viewing area is determined, the visible portion is then *reversely projected* from the raw frame to the screen, as illustrated by the two small images on the right side of Figure 2.

• The raw frames of YouTube and Facebook videos exhibit different visual "patterns". We found the reason to be their different projection algorithms. YouTube employs equirectangular projection [10] that directly uses the latitude and longitude on a sphere as the vertical and horizontal coordinates, respectively, on the raw frame. Facebook instead employs a projection scheme called Cube Map [8] that has less distortion in the polar areas of the sphere.

• Both YouTube (on Android app) and Facebook (on Chrome for Windows 10) use progressive download over HTTP, a widely used streaming technique, to deliver 360 videos. Progressive download allows a client to start playing the video before it is fully downloaded. It is realized using HTTP byte range request.

• Both video platforms support multiple encoding bitrates for 360 videos. The viewer can switch between SD and HD versions on Facebook. YouTube provides up to 8 bitrate levels from 144s to 2160s[2]. Note the video quality numbers refer to the resolution of the entire raw frame (Figure 2 left), in which the viewer only sees a small portion at any given time (Figure 2 right). Therefore, *to achieve the same user-perceived playback quality, the raw frame quality of a 360 video has to be much higher than that of a non-360 video.* Based on our viewing experience, for a decent user experience, a 360 video needs to be streamed at at least 1080s. For example, 480p is a reasonable quality for conventional videos. However, when watching the video in Figure 2 under 480s, the quality is unacceptably bad because the viewer in fact has a stretched view of a subarea of a 480s frame.

Video Scene	Length	1080s	1440s	2160s
Roller coaster	1'57"	66MB	105MB	226MB
Animals	2'49"	52MB	129MB	246MB
Aerobatic Flight	8'12"	172MB	350MB	778MB
Google IO 2016	2h8'34"	1.7GB	4.9GB	9.1GB

Table 1: Sizes of four 360 videos on YouTube.

• As a direct consequence of the above observation, for the same user perceived quality, 360 videos have very large sizes. Table 1 lists sizes of four 360 videos on YouTube, assuming 1080p is the minimum video quality for a reasonable QoE. This inevitably causes issues on cellular networks with limited bandwidth (in particular when signal strength is not good) and metered link.

• Finally, for both YouTube and Facebook, the client always downloads the *entire* raw frame regardless of user's viewing direction. This leads to tremendous waste of network bandwidth, because most areas of a raw frame are not viewed by the user. Based on our simulation in §4, such invisible areas account for up to 80% of the network bandwidth consumed by 360 video playback. We note that due to the use of a single H.264 video stream, it is inherently impossible for a 360 video client to fetch a subarea of a raw frame.

3. PROPOSED DESIGN

3.1 Problem Statement and Challenges

The measurements in §2 indicate that 360 videos largely inherit the delivery scheme from traditional Internet videos. The obvious advantage is simplicity: virtually no change is required on the server side, and a non-360 player can be easily enhanced to support 360 videos by adding projection and head movement detection. However, the negative side is, streaming 360 videos is very bandwidth consuming, because (1) under the same user perceived quality, 360 videos have much larger sizes than non-360 videos, and (2) today's 360 video players always fetch the entire raw frame including both the visible and invisible portion. This may not be a big issue for wired and high-speed WiFi networks. However, the scheme is not friendly to cellular networks where radio resources are scarce and bandwidth is limited. Also, downloading excessive data hurts mobile devices' battery life because cellular radio is known to be energy-hungry: when in active use, the LTE radio accounts for at least 50% of the entire smartphone's energy consumption [16].

Motivated by the above, we propose to improve 360 video streaming over cellular networks, with the goal of reducing the bandwidth consumption. Our high-level idea is straightforward: instead of downloading everything, the client only fetches the parts that are visible to the user. To realize this idea, we face several challenges. First, we need a mechanism that allows a client to download a subarea of a video chunk. Second, in order to determine what to fetch, the client needs to *predict* a viewer's head movement. The pre-

[1]It is difficult to conduct the experiments on a smartphone because mitmproxy does not work with the Facebook app for Android.

[2]YouTube uses suffix "s" instead of "p" for 360 video quality.

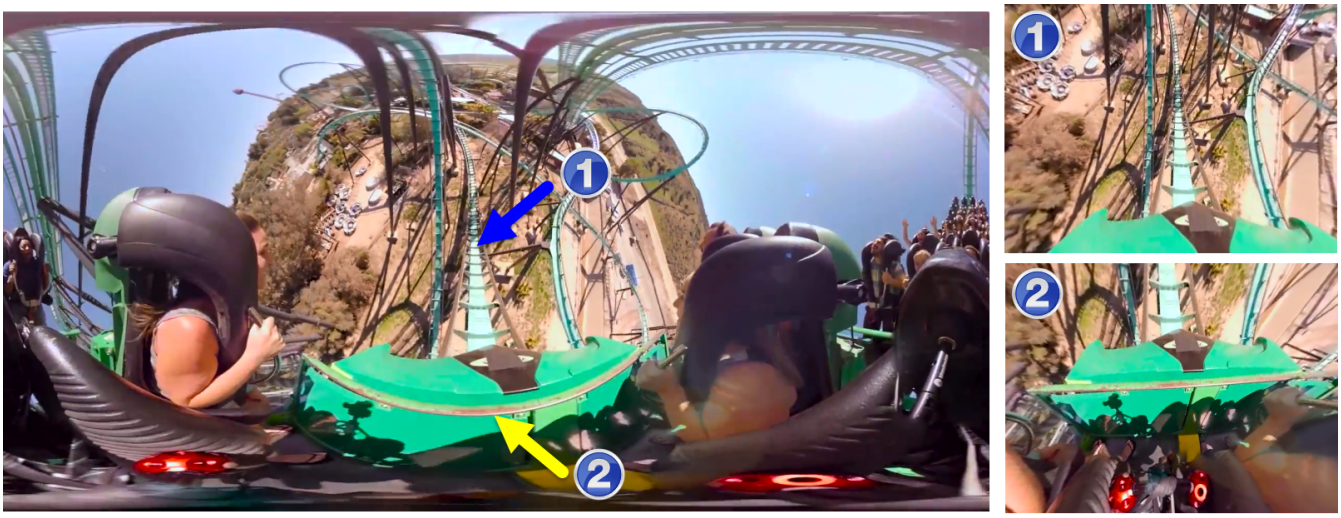

Figure 2: Left image: a raw frame of a 360 video at 1080s. Two right images: visible frames reversely projected from the raw frame when the viewer is looking at Points 1 and 2. Video source: https://www.youtube.com/watch?v=-xNN-bJQ4vI

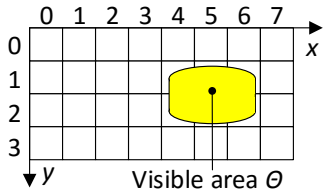

Figure 3: Spatial segmentation of a video chunk into tiles.

diction needs to be robust and efficient. We also need to tolerate inaccurate prediction by strategically sacrificing bandwidth in certain situations. Third, our approach should incur minimal changes to both the client player and in particular, the server. Next, we detail the key design aspects of our proposal.

3.2 Spatial Segmentation of the Video

For traditional videos, to support simultaneous download and playback, a video is temporally segmented into chunks or byte ranges. To support downloading a subarea of a video chunk, the video also needs to be *spatially* segmented. This can be realized in an *online* manner: the client computes the target area of a chunk, and embeds them into HTTP request parameters; the server then dynamically generates a smaller chunk containing only the target area and transmits it to the client. This approach suffers from two drawbacks. First, it greatly increases the server-side computational overhead. Second, due to projection, the target area is not a rectangle, making it hard for the client to specify the target area.

We instead propose spatially segmenting the video in an *offline* manner. Each 360 video chunk is pre-segmented into multiple smaller chunks, which we call *tiles*. A tile has the same duration as a chunk while only covering a subarea of the chunk. The easiest way to generate the tiles is to evenly divide a chunk containing projected raw frames into m*n rectangles each corresponding to a tile (we will use this approach in our simulation in §4). Suppose the projected visible area is Θ. The client only requests for the tiles that overlap with Θ. An example is illustrated in Figure 3 where m=8 and n=4, and Θ is the yellow region. The client will only request for the six tiles $(4 \leq x \leq 6, 1 \leq y \leq 2)$ overlapping with Θ. Note that due to projection, despite the viewer's FoV being fixed (§1), the size of Θ and thus the number of requested tiles may vary. For example, under equirectangular projection, as shown in Figure 2, more tiles are needed when the viewer looks downward (Point 2) compared to when she looks straight forward (Point 1).

Besides the above approach, an alternative and more complex way is to apply segmentation directly on the 3D sphere instead of on the projected 2D raw frame so that each tile covers a fixed angle. This makes the number of tiles to be requested irrespective of user's viewing direction (but their total bytes may still vary). We plan to explore both segmentation approaches.

Performing the spatial segmentation offline eliminates the server-side overhead. Multiple tiles can be requested in a single bundle to reduce network roundtrips. Tiles' meta data such as positions and URLs can be embedded in a metafile exchanged at the beginning of a video session.

3.3 Head Movement Prediction

Ideally, if a viewer's head movement during a 360 video session is known beforehand, we can generate the optimal sequence of tiles that minimizes the bandwidth consumption. To approximate this in reality, we propose to predict the head movement *i.e.,* the change of pitch, yaw, and roll.

Is a viewer's head movement indeed predictable? To get a preliminary answer to this key question, we conduct a user trial of five users. The experimental setting is as follows. Each user wears a Google Cardboard viewer [4] with a Samsung Galaxy S5 smartphone placed into the back of it[3]. The smartphone plays four short YouTube 360 videos (duration from 1'40" to 3'26") of different genres. Meanwhile, a head tracker app runs in the background and sends the raw yaw, pitch, and roll readings to a nearby laptop using UDP over WiFi (latency $<$ 1ms). We use OpenTrack [7], an open-source head tracking software, to perform preprocessing (*e.g.,* smoothing and head shaking removal) under default settings before recording the data. The sampling rate is 250Hz. During the playback, the five users can view at any direction by freely moving their heads.

Leveraging the collected traces, we use a sliding window of 1 second from $t_0 - 1$ to t_0 to predict a future head position at $t_0 + \delta$

[3]To use the Google Cardboard, we enable 3D mode of 360 videos.

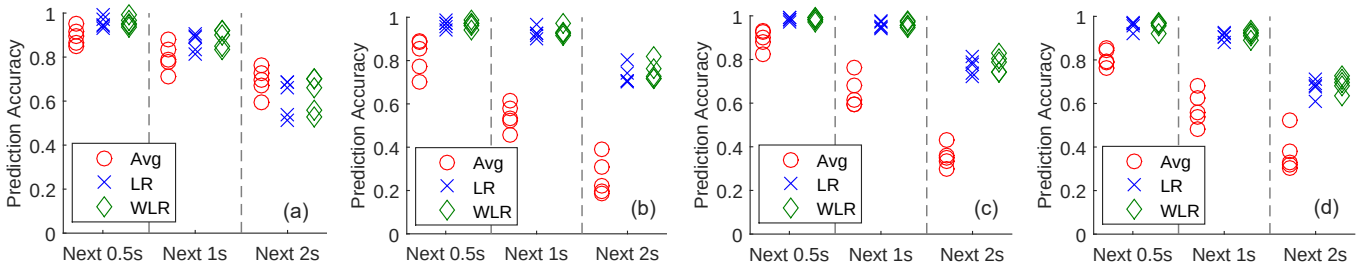

Figure 4: Head prediction results for four 360 videos: (a) roller coaster, (b) NASA Mars, (c) sailing, and (d) sports. A moving window of 1s is used to predict next δ=0.5s, 1s, and 2s using average (Avg), linear regression (LR), and weighted LR (WLR).

for each dimension of yaw, pitch, and roll. In this feasibility study, we use three simple prediction approaches.

• *Average.* We compute the average value in the window as the prediction result.

• *Linear Regression (LR).* We train an LR model using all samples in the window, and then use the model for prediction. All samples in the window have the same weight.

• *Weighted linear regression (WLR).* It is similar to LR, except that a more recent sample is considered more important: the weight of a sample at $t_0 - x$ is set to $1 - x$ ($0 \leq x \leq 1$).

For a given window $(t_0 - 1, t_0)$, the output of the prediction consists of the estimated yaw, pitch, and roll at $t_0 + \delta$. We consider the prediction to be accurate if the predicted values in *all* three dimensions differ from the true values by less than 10°, which can be easily compensated by downloading slightly more data as to be described in §3.3.1. Figure 4 shows the prediction results of $\delta \in \{0.5s, 1s, 2s\}$ for the four videos, with each point representing one video playback. The prediction accuracy (Y Axis) of a playback is defined as the number of accurate predictions over the total number of predictions.

We describe two major findings in Figure 4. First, despite the simple prediction techniques we use, we observe good short-term predictability for head movement. Using WLR, the average prediction accuracy values across all users and all videos for δ = 0.5s and 1s are 96.6%±2.0% and 92.4%±3.7%, respectively. On the other hand, prediction in the longer term is more difficult: the average accuracy drops to 71.2%±7.6% when δ = 2s. Second, the prediction methodology does matter, as linear regression significantly outperforms the naïve averaging approach. Using WLR further slightly improves the results. We plan to experiment with more sophisticated machine learning algorithms and to leverage richer training data in our future work. Also, the prediction can be further constrained by robust heuristics (*e.g.,* a user will be very unlikely to vary the roll by more than $\pm 15^\circ$).

Despite the satisfactory short-term predictability, a key concern is, *is 1 to 2 seconds too short to fetch the tiles?* We expect that under reasonable cellular network conditions, such a prediction window of 1 to 2 seconds is sufficient for network transfer. Today's LTE networks offer high bandwidth and low latency [22]. Assuming 15Mbps bandwidth, which can be achieved on today's commercial LTE networks [1], it takes only 0.53 second to download a 1-second full-frame video chunk at 1080p[4]. Since our scheme only downloads the visible portion, the required bandwidth for fetching the tiles can further be reduced by 60% to 80% (§4). This makes our scheme quite feasible on today's LTE networks and the 5G networks that offer throughput of up to 1Gbps.

[4]The average video bitrate for 1080p is about 8Mbps [9].

3.3.1 *Handling Prediction Errors*

Due to human users' randomness, prediction errors are inevitable. We realize that the head movement predictability may highly depend on the video content. For example, in Figure 4, the roller coaster video in subplot (a) has higher predictability than the NASA Mars video in subplot (b) because the former has a more clear "focal point" (the rail of the roller coaster) than the latter.

We handle prediction errors using several strategies. First, due to the online and sliding-window nature of the prediction scheme, a previous inaccurate prediction might be fixed by a more recent and accurate prediction. If the new tiles corresponding to the updated prediction can be fetched before the playback deadline, the penalty is only wasted bandwidth. We describe how to prioritize such fixes in §3.5.

Second, since most prediction errors are expected to be small, we can tolerate them by conservatively fetching more tiles covering a larger area than what is predicted. For example, in Figure 3, the client can further fetch surrounding tiles such as (3,1) and (4,0). We call these additionally fetched tiles *out-of-sight* (OOS) tiles, as they will remain invisible unless a prediction error occurs. Clearly, the number of OOS tiles incurs a tradeoff between bandwidth consumption and user experience. It can be, for example, dynamically determined by the recent prediction error e maintained by the player. The larger e is, the more OSS tiles need to be fetched.

Third, to further reduce the bandwidth consumption of OOS tiles, they can be fetched at a lower quality, which depends on their distance to the predicted area. At a high level, this is essentially a variation on Forward Error Correction (FEC), which transmits lower quality versions of alternate data in case of errors. Consider Figure 3 again. Suppose the six tiles overlapping with the predicted visible area Θ are fetched at quality level n. Then a nearby OOS tile (3,1) might be fetched at quality level $n - 1$, and an OOS tile such as (2,1) that is further away might be fetched at an even lower level. The intuition is, the likelihood that the viewer will watch a far-away OOS is low, but in case that happens, having a low-quality tile will at least ensure the smooth playback without stalling the video.

In the worst case when the user's head movement is quick and exhibits no trend, our prediction may have low accuracy, leading to potential stalls (or leaving part of the display blank if the player chooses to skip the stalls). We design a fail-safe mechanism to address this. As stalls due to wrong predictions occur more frequently, more OOS tiles will be fetched. Eventually, the player will fall back to today's simple approach of fetching all tiles. In this case, since we are fetching more tiles, their quality will be degraded accordingly if the bandwidth is limited.

3.4 Leveraging Crowd-sourced Statistics

Popular 360 videos from commercial content providers and video-

sharing websites attract a large number of viewers (*e.g.,* more than 4 million views of Figure 2's video). Also, it is known that users' viewing behaviors are often affected by the video content [14, 11]. We believe this is also true for 360 videos: at certain scenes, viewers are more likely to look at a certain spots or directions. Consider an example of a mountain climbing video. When "standing" at the peak, viewers may want to enjoy the view by looking all around.

Based on the above intuition, we propose to use crowd-sourced viewing statistics, which can be easily collected by video servers, to complement head movement prediction. In the literature, viewing statistics have been leveraged to estimate the video abandonment rate [14] and to automatically rate video contents [11]. In the context of 360 videos, for each chunk, the server records *download frequencies* of its tiles, and provides client players with such statistics through metadata exchange. A tile's download frequency is defined as the number of video sessions that fetch this tile divided by the total number of sessions accessing this video. The client can (optionally) use the statistics to guide the download strategy of OOS tiles (§3.3). For example, a simple strategy is to expand the set of OOS tiles to include tiles whose download frequencies are greater than a configurable threshold. The threshold trades off between bandwidth consumption and user experience.

3.5 Integration with DASH and HTTP

Although currently most 360 videos use progressive download, we envision they will switch to DASH soon, which is the state-of-the-art delivery mechanism for conventional videos. Extensive research has been conducted on improving the QoE of DASH video [18, 21, 13, 23, 17]. A DASH video is split into chunks encoded with multiple discrete bitrate levels; a video player can switch between different bitrate levels at a chunk boundary. In contrast, 360 videos involve more complexity, because the player needs to make decisions at both the temporal and spatial dimension.

The most important component of a DASH scheme is its rate adaptation algorithm, which determines the quality level of chunks to fetch. There are largely two categories of approaches: throughput-based and buffer-based. A throughput-based rate adaptation algorithm adjusts chunks' quality levels based on estimated throughput. The buffer-based approach, on the other hand, selects the bitrate level based on the player's buffer occupancy level, which implicitly encodes the network capacity information.

For today's 360 video delivery that downloads everything (§2), it requires no change to any DASH algorithm. But how about the interplay between our prediction-based streaming scheme and DASH? We consider the aforementioned two categories of DASH algorithms. Throughput-based DASH algorithms could work well with our scheme: when the estimated throughput decreases (increases), the quality level of tiles will decrease (increase) correspondingly. One challenge is to set the thresholds for quality level switches. Due to projection and OOS tiles, the required bandwidth in our scheme has higher variance than that for non-360 videos. Thus, the thresholds may need to be adjusted dynamically.

We now consider buffer-based DASH algorithms. One issue here is that in our scheme, the player may not want to keep a large buffer occupancy, because predicting viewer's head movement in the long term is difficult (§3.3). As a result, since the player only maintains a relatively short duration of video contents in the buffer, buffer-based DASH algorithms may interact poorly with our scheme. We plan to conduct a more in-depth study on this in our future work.

Interaction with HTTP. Similar to regular DASH, our scheme uses HTTP(S) as the underlying delivery protocol. Each tile is fetched by an HTTP request. A new observation here is that *priorities* of HTTP transactions play an important role in mitigating the user experience degradation caused by inaccurate prediction. Consider the following example. The player is in the progress of downloading tile x whose playback time is t_2. Then suddenly, the player realizes a predicted tile to be played at $t_1 < t_2$ is incorrect. To fix this issue, the player immediately issues a request for tile y whose playback time is t_1. Since the delivery of y is more urgent than x, ideally the server should pause the transmission of x, and transmit y at its full speed. This can be realized by giving y a higher priority than x. New web protocols such as HTTP/2 [12] already support fine-grained control of HTTP transactions' priorities that are very useful in our scheme.

4. TRACE-DRIVEN SIMULATION

We developed a trace-driven simulator to evaluate how much bandwidth our proposed scheme can potentially save. The input to the simulator consists of the user study traces collected in §3.3 (five users watching four YouTube 360 videos). Recall that a trace consists of a user's head positions (pitch, yaw, and roll) during a 360 video playback at a sampling rate of 250Hz. We assume the duration of each video chunk is 1 second, and the horizontal and vertical FoV are 110° and 90°, respectively (a typical setting for VR headsets). For each chunk, we consider three tile configurations: 4×8, 6×12, and 10×20. Given a particular 1-second chunk, our simulator computes the set of tiles to be fetched as follows. First, it computes the visible area Ω on the 3D sphere based on the head position trace and FoV. Second, it projects Ω to Θ on the 2D raw frame. We implemented the equirectangular projection algorithm that is used by YouTube. Third, the simulator derives the set of tiles to be fetched by calculating the overlap between Θ and all tiles as illustrated in Figure 3. The bandwidth saving of a video playback can thus be estimated as $1 - N_F/N$ where N and N_F are the total number of tiles and the number of tiles that are actually fetched, respectively, across all chunks. Note here for simplicity, we assume all tiles in a video have the same size.

We evaluate the bandwidth savings under three scenarios: (1) the player has the perfect knowledge of head positions; (2) the player fetches additional out-of-sight (OOS) tiles by virtually expanding the FoV by 10° in four directions (up, down, left, and right); (3) the player uses the same configurations as (2) except increasing the chunk duration from 1s to 4s. The results are shown in plot (a), (b), and (c) in Figure 5, respectively. We highlight several key findings below. First, in Scenario (1) where the player knows perfectly the head positions, the bandwidth saving can reach up to 80%. Second, when taking OOS tiles into account, the bandwidth saving drops as expected. However, the drop is small: when OOS tiles cover additional 10° FoV in all directions, the maximal bandwidth saving is reduced by only 5%. Third, reducing the tile size leads to higher bandwidth savings. This is because as tiles become smaller, *partial* overlaps between Θ and tiles are reduced. Due to a similar reason, the bandwidth savings decrease when the chunk duration increases to 4s. Overall, the preliminary results are promising.

The above simulation does not take prediction errors into consideration. If they are taken into account, the bandwidth consumption slightly increases by about 1.7% on average, compared to Scenario (2) described above (using Weighted Linear Regression with a 1s window to predict the head position in the next δ=1s). The penalty is small because of the high prediction accuracy as shown in Figure 4. Also the OOS tiles can already mask many prediction errors.

5. RELATED WORK

In the literature, extensive efforts have been made toward improving various aspects of mobile video [21, 13, 14]. In contrast,

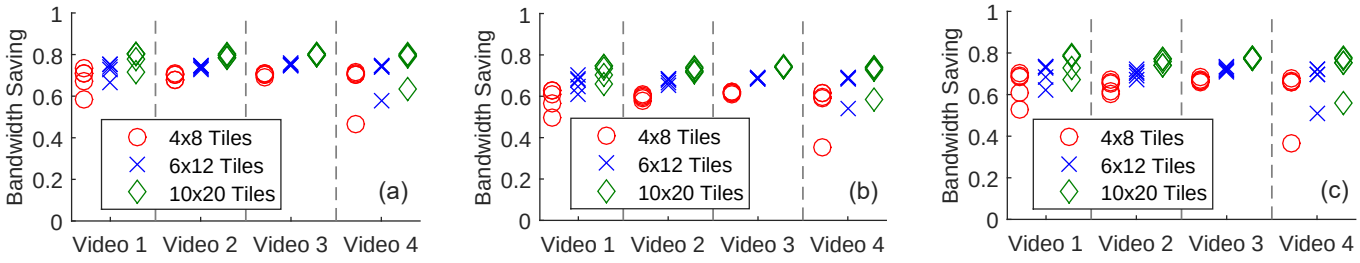

Figure 5: Bandwidth savings: (a) with perfect knowledge, (b) with a larger FoV, and (c) with 4-sec video chunk duration.

much less research has been conducted on optimizing 360 videos despite their popularity. Some early work focused on generating 360 video content from the rotation of a single camera [24] or from a camera array [20]. There are also studies on stereo and multiview videos where users can switch among two or more synchronized video streams [19, 15]. Recently, the industry has been working on improving the projection and encoding schemes for 360 videos, such as Cube Map [8] and pyramid encoding [6] described in Facebook technical blog posts. The same post [6] also envisions that when combined with the pyramid encoding, performing head orientation prediction can help reduce the bandwidth usage. Instead, we proposed the concrete design of a cellular-friendly 360 video streaming framework that does not depend on any specific projection scheme. We also proposed robust methods for tolerating prediction errors, for leveraging crowd-sourced playback statistics, and for integrating our scheme with DASH and HTTP protocols.

6. CONCLUSION

To conclude, our work sheds light on developing resource-efficient 360 video streaming schemes for LTE networks, which facilitate ubiquitous access of VR resources. The pilot user study demonstrates the feasibility of downloading only visible portion of a video for saving network bandwidth. We are currently building a full 360 video streaming system on commodity Android devices. We plan to evaluate it by large-scale user study under realistic cellular network conditions.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback. We also thank the five users who participated in our pilot user study of 360 videos.

7. REFERENCES

[1] 2015 Speedtest results for U.S. ISPs and Mobile Networks. http://www.speedtest.net/awards/us.

[2] Augmented/Virtual Reality revenue forecast revised to hit $120 billion by 2020. http://goo.gl/Lxf4Sy.

[3] Facebook Surround 360. https://facebook360.fb.com/facebook-surround-360/.

[4] Google Cardboard. https://vr.google.com/cardboard/index.html.

[5] mitmproxy. https://mitmproxy.org/.

[6] Next-generation video encoding techniques for 360 video and VR. https://goo.gl/DvYivQ.

[7] OpenTrack: head tracking software. https://github.com/opentrack/opentrack.

[8] Under the hood: Building 360 video. https://code.facebook.com/posts/1638767863078802.

[9] YouTube Live encoder settings, bitrates and resolutions. https://support.google.com/youtube/answer/2853702.

[10] YouTube live in 360 degrees encoder settings. https://support.google.com/youtube/answer/6396222.

[11] X. Bao, S. Fan, A. Varshavsky, K. A. Li, and R. R. Choudhury. Your Reactions Suggest You Liked the Movie: Automatic Content Rating via Reaction Sensing. In *UbiComp*, 2013.

[12] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, 2015.

[13] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *MobiCom*, 2013.

[14] M. A. Hoque, M. Siekkinen, and J. K. Nurminen. Using Crowd-Sourced Viewing Statistics to Save Energy in Wireless Video Streaming. In *Mobicom*, 2013.

[15] H. Huang, B. Zhang, S.-H. G. Chan, G. Cheung, and P. Frossard. Coding and Replication Co-Design for Interactive Multiview Video Streaming. In *INFOCOM*, 2012.

[16] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Mobisys*, 2012.

[17] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *SIGCOMM*, 2014.

[18] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *CoNEXT*, 2012.

[19] J.-G. Lou, H. Cai, and J. Li. A real-time interactive multi-view video system. In *ACM Multimedia*, 2005.

[20] R. Szeliski. Image Alignment and Stitching: A Tutorial. Technical Report MSR-TR-2004-92, Microsoft Research.

[21] X. Xie, X. Zhang, S. Kumar, and L. E. Li. piStream: Physical Layer Informed Adaptive Video Streaming Over LTE. In *MobiCom*, 2015.

[22] Y. Xu, Z. Wang, W. K. Leong, and B. Leong. An End-to-End Measurement Study of Modern Cellular Data Networks. In *PAM*, 2014.

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

[24] Z. Zhu, G. Xu, E. M. Riseman, and A. R. Hanson. Fast generation of dynamic and multi-resolution 360° panorama from video sequences. In *IEEE Intl. Conf. on Multimedia Computing and Systems*, 1999.