Adaptive 360-Degree Video Streaming using Scalable Video Coding

Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D. Beshay, Ravi Prakash
The University of Texas at Dallas
800 W Campbell Rd.
Richardson, Texas 75080
{afshin,anahita.mahzari,joseph.beshay,ravip}@utdallas.edu

ABSTRACT

Virtual reality and 360-degree video streaming are growing rapidly, yet, streaming high-quality 360-degree video is still challenging due to high bandwidth requirements. Existing solutions reduce bandwidth consumption by streaming high-quality video only for the user's viewport. However, adding the spatial domain (viewport) to the video adaptation space prevents the existing solutions from buffering future video chunks for a duration longer than the interval that user's viewport is predictable. This makes playback more prone to video freezes due to rebuffering, which severely degrades the user's Quality of Experience especially under challenging network conditions. We propose a new method that alleviates the restrictions on buffer duration by utilizing scalable video coding. Our method significantly reduces the occurrence of rebuffering on links with varying bandwidth without compromising playback quality or bandwidth efficiency compared to the existing solutions. We demonstrate the efficiency of our proposed method using experimental results with real world cellular network bandwidth traces.

KEYWORDS

360-degree video; Adaptive video streaming; Scalable Video Coding; Quality of Experience; Rebuffering

1 INTRODUCTION

Multimedia technology is evolving from providing users with a simple viewing experience to a fully immersive virtual environment. Providing this experience requires various components; hardware, content, and delivery. At the hardware level, Virtual Reality (VR) Head Mounted Displays (HMD) are being developed. VR HMDs utilize 360-degree video, also known as spherical video, which involves a 360-degree view of the scene captured from a single point. The captured video is mapped to the internal surface of a sphere. An HMD views a limited portion of the video as seen from the center of the sphere, referred to as viewport. The area covered by the viewport is limited by the HMD's Field of View (FoV) and its coordinates are based on the orientation of the user's head.

MM'17, October 23–27, 2017, Mountain View, CA, USA.
 ISBN 978-1-4503-4906-2/17/10…$15.00
DOI: https://doi.org/10.1145/3123266.3123414

For the content and its delivery, video platforms and network providers are working on hosting and delivering 360-degree videos. Presenting high video quality within the user's viewport requires a complete high-quality 360-degree frame. This places much higher demands on the network in terms of throughput and latency. For example, Oculus Rift viewing resolution is 1080x1200 per eyes with a FoV of 110 degrees; thus, the complete 360-degree frame should have 6K resolution to exploit the highest quality that the device can offer. This would require around 40 Mb/s of bandwidth. This bandwidth requirement is too high for most end-users. Also a significant portion of the downloaded video content goes unused, since only a subset, specifically the viewport, is displayed by the HMD.

One solution is adaptive 360-degree video streaming. The scheme is similar to HTTP Adaptive Streaming (HAS), with the spatial dimension added to the adaptation space. The video is segmented into chunks of fixed duration. The server stores a representation set for each time-domain chunk that enables clients to receive only the area of a chunk covered by user's viewport. Clients choose which areas to download at which representation based on available bandwidth and the user's viewport. When a client requests a video chunk, the elapsed time between request dispatch and chunk arrival is a function of network latency, throughput, and chunk size. This elapsed time is not deterministic. Instead, it fluctuates based on varying network condition and variable bit-rate (VBR) encoding of the video. To tide over these changes, clients prefetch future chunks. Prefetching chunks requires knowledge about the user's future viewport. Attempts have been made to predict the user's future viewport based on viewport history. The mismatch between the user's predicted and actual viewport is likely [12]. In order to tackle this potential mismatch, parts of the video outside the predicted viewport are also streamed, but at lower quality [19].

Viewport prediction accuracy decreases with time. It has been shown that the viewport can be predicted accurately for one second [12], but if we increase the prediction horizon, the mismatch between actual and predicted viewport also increases. If the prediction is not accurate, the downloaded chunk cannot provide high-quality video for the user's viewport, resulting in a degraded Quality of Experience (QoE). The client's other option is to prefetch more chunks and replace them with new ones every time the predicted viewport changes, which would result in bandwidth waste. To strike a balance between degraded QoE and bandwidth waste, the client does not prefetch chunks for more than the high-accuracy viewport prediction interval, currently around 1 to 2 seconds. While regular video players (other than 360-degree video) buffer 20 to 30 seconds of video to deal with network variations [1]. Such a shallow buffer

makes it impossible for the client to survive even short-lived variations in network conditions. A throughput drop can quickly deplete the buffer and cause the video playback to stall until new chunks are fetched (an occurrence referred to as rebuffering).

Rebuffering has a large negative impact on the user's QoE [9]. In this paper, we propose an adaptation method for 360-degree video using scalable video coding (SVC) [3][18]. Our method is both viewport and bandwidth adaptive. It aims at maintaining the continuity of video playback at the highest possible quality under challenging network conditions. This is made possible by utilizing the SVC base and enhancement layers. In SVC, the base layer is always required for decoding the video. Irrespective of the user's viewport, the base layer can be prefetched and buffered at the client for a long duration, significantly reducing rebuffering. Enhancement layers are then prefetched for a shorter duration to increase the quality of the parts of the video within the user's viewport. Besides that, our method inherits other SVC benefits such as reducing the storage requirements on servers, and improving the performance of intermediate cache servers.

The rest of this paper is organized as follows. In Section 2, the background and related work are presented. Our method is explained in details in Section 3. Section 4 presents and evaluates the experimental results. Finally, Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORK

Streaming 360-degree video is different from regular video. We start by presenting a quick overview of how 360-degree video is projected and encoded, followed by a short summary of the existing solutions for adaptive streaming.

2.1 Projection

Encoding of spherical videos are not supported by existing video coding standards directly. The video is first projected to a two-dimensional plane using a projection method, such as equirectangular, cubemap or pyramid projection [16]. Figure 1 shows a sphere mapped to a plane using these projections. The most common projection is the equirectangular projection which maps a sphere into a rectangle. This projection introduces severe stretching at the north and south poles of the sphere which reduces the efficiency of encoding. The pyramid projection maps the user's viewport to the base of the pyramid, and the remaining parts to the sides. Severe quality degradation at the sides and high GPU processing requirements are the two main issues with pyramid projection [8]. Cubemap projection maps 90° FoVs to the sides of a cube. This projection results in less quality degradation across the video, and has lower bandwidth overhead than equirectangular projection [7]. Moreover, the cubemap projection needs lower rendering processing power. So, in terms of quality and required processing for rendering, cubemap projection provides a desirable trade-off [8].

2.2 Adaptive streaming

To stream a video sequence, it is segmented in the time domain into chunks of fixed duration. For each chunk, the client can adapt the quality based on the user's viewport in the spatial domain. Existing adaptation methods can be classified according to their representation of video into two groups: tile-based and monolithic methods.

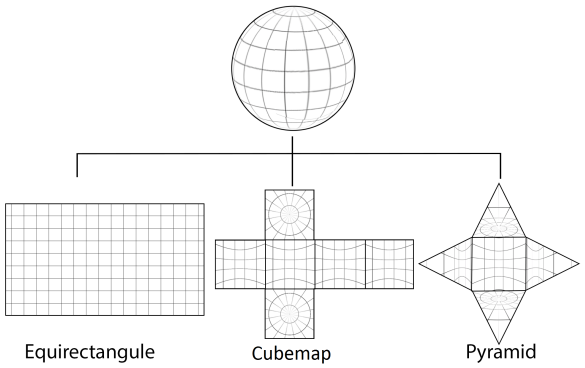


Figure 1: Different projections

2.2.1 Tile-based streaming. Tile-based streaming divides the video into tiles. Each tile is made available in a set of quality levels. The client tries to prefetch higher quality tiles for the area covered by the user's viewport as far as the available bandwidth allows. A number of of tile-based solutions tailored for 360-degree video are presented in [5][13][12][19].

One of the earliest solutions for adaptive 360 video streaming is proposed in [13]. The authors formulate and solve a bandwidth-optimal system for selecting the quality of tiles that maximizes the quality of viewport depending on available bandwidth. Another tile-based method that focuses on a cellular-friendly streaming method for 360-degree videos is proposed in [12]. It examines three prediction approaches to predict the user's future viewport, and selects the optimal sequence of tiles which minimizes bandwidth consumption. The presented results show that viewport can be predicted rather accurately for the next second.

The main difficulty with tile-based methods is that it requires multiple decoders at the client side to decode each tile. To overcome this issue, motion-constrained tile sets (MCTS) feature of HEVC is used in [19]. MCTS enable a client to use a single hardware decoder to decode a set of selected tiles based on the user's viewport. The authors of [19] examine the effect of three tiling schemes on compression performance and the streaming bitrate. They show that it is possible to reduce the bitrate by 30% to 40% compared to streaming the entire 360-degree video at the same quality level.

2.2.2 Monolithic streaming. An alternative to tile-based streaming is to create the viewport-adaptive representations for each chunk independently. First, a set of overlapping viewports is defined that can be of interest to the users. Then, for each pre-defined viewport a representation is created that contains the entire spherical frame providing higher quality within the pre-defined viewport and gradually lower quality in the rest of the frame. Facebook [8] and Pixvana [11] employ this method for adaptive 360-degree video streaming. Facebook creates representations for 30 different viewports covering the whole 360-degree video. These viewports are separated by about 30 degrees angles. Five quality levels are defined for each pre-defined viewport. The main benefit of this method is that the video can be decoded using a single decoder. However,

monolithic streaming produces high redundancy between the different representations compared to tile-based methods. The reason is that each chunk has to be encoded multiple times, i.e., for the number of pre-defined viewports, while quality of different areas of each chunk does not change among all representations. This redundancy results in higher storage requirements.

3 PROPOSED METHOD

Our method is a tile-based streaming method that provides both viewport and bandwidth adaptation. We start by explaining the tiling and encoding scheme followed by the adaptation mechanism.

3.1 Tiling Scheme

We assume that the video source uses cubemap projection, however, our proposed method can be modified for any other projection. Given a video sequence of duration D seconds, we propose segmenting it in time domain into chunks of s seconds each. So the video will have $N = D/s$ chunks. C_t denotes the t^{th} chunk, where $t \in \{1, ..., N\}$. Each chunk contains the six faces of the cube-mapped 360-degree video. We divide each face of the cube into a grid of tiles of equal size. We denote the number of tiles vertically/horizontally by m. Tiles are encoded in L different quality levels according to the layered scheme of scalable video coding [3]. Scalable coding is a hierarchical layered encoding technique that encodes a video sequence into multiple layers. The lowest quality layer is the base layer, and other layers are enhancement layers that can be increasingly added to the lower layers to enhance quality. So, each enhancement layer only contains the difference signal of that quality level from all lower layers.

We denote each tile with $T_{c,f,i,j,l}$, where c is the chunk number, $f \in \{1, .., 6\}$ is the face number, i and $j \in \{1, .., m\}$ are the latitude and longitude of each tile on a face, and $l \in \{0, ..., L-1\}$ is the quality level. To achieve quality level l' for a tile, all co-located tiles from layer 0 up to l' are required: $\sum_{l=0}^{l=l'} T_{c,f,i,j,l}$ ($\sum$ shows adding layers on top of each other). We show the bitrate for decodable complete 360-degree video at quality level l' with variable $R_{l'}$. If function $size()$ shows the size of a tile, then we have:

$$R_{l'} = \frac{\sum_{c=1}^{N} \sum_{f=1}^{6} \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{l=0}^{l'} size(T_{c,f,i,j,l})}{D} \quad (1)$$

Figure 2 shows a frame of spherical video mapped into a cube, with each face sliced into four tiles ($m = 2$), and encoded in three layered quality levels. Each color represents a face of the cube.

In order to adapt to the user's viewport, tiles should be decodable independently. Therefore, the client can download any combination of tiles that cover user's viewport. In our proposed method, we assume that base layer tiles are always downloaded to enable the client to show any part of the video to the user in case of a mismatch between predicted and actual viewport. Although base layer is always available during playback time, a subset of enhancement layer tiles is available based on the user's viewport. So, they should be decodable when neighboring enhancement layer tiles are not available. At the encoding stage, an enhancement layer tile $T_{c',f',i',j',l'}$

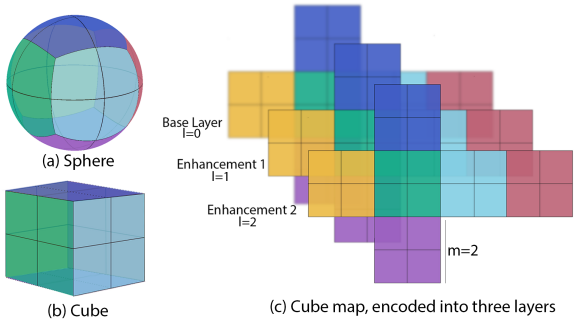


Figure 2: A sphere mapped into cube, and scalable encoded

can get encoding predictions (for intra predictions and motion estimation) from all base layer tiles: $\{\forall T_{c',f,i,j,0} | 1 \le i, j \le m\}$, and only from other co-located enhancement layers at lower layers: $\{\forall T_{c',f,i,j,l} | f = f', i = i', j = j', l < l'\}$. This limitation will result in slightly higher encoding overhead, but enables the client to fetch and decode any subset of enhancement layer tiles as long as the lower layer tiles are available. We call our proposed method *scalable-tiled adaptation*.

As previously mentioned, tile-based streaming requires multiple parallel decoders to decode tiles, which increases decoding complexity. A solution similar to the proposed one in [19] can be used to address this issue. In this approach, although tiles are encoded independently, when the client requests each tile of the whole sphere at different quality levels, they are rewritten into a single bitstream which is decodable by a single hardware decoder.

3.2 Viewport and Bandwidth Adaptation Method

In scalable video coding, the base layer is always needed for decoding the video. This turns out to be the main advantage of using scalable coding in 360-degree video. It allows buffering low-quality base layer chunks for a longer duration than the viewport prediction high-accuracy interval. A longer buffer enables the client to continue video playback at the lowest quality during network throughput drops instead of interrupting video playback.

An optimal solution for adaptive 360-degree video streaming *should provide highest possible QoE for user's viewport with minimum bandwidth usage.* First, the user's viewport should be predicted. Then, QoE should be maximized for that viewport given the existing network conditions. The three components of QoE we focus on are:

- Average video quality.
- Duration and frequency of rebufferings.
- Number and Frequency of quality switches.

If the client always requests the higher quality for the user's viewport while network throughput cannot support the traffic load, video chunks cannot be downloaded before playback deadline and rebuffering will happen. So the goal is to maximize the quality at viewport and minimize rebuffering and switches.

Figure 3 depicts the internal components of a 360-degree video streaming client. Adaptation is performed by the adaptation logic

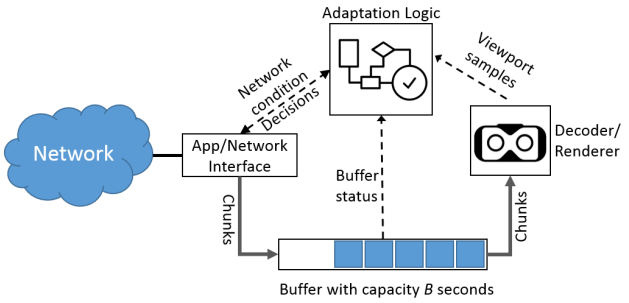


Figure 3: Client components

unit. This unit utilizes information from other components to adapt the playback quality. The inputs to this unit are:

- Video information: number of chunks (N), quality layers (L) and their bitrates(R_l), and tiling configuration (m) which are sent to the client at the beginning of a session.
- Viewport: Samples from the user's previous viewports.
- Network throughput: Measured throughput samples from previously downloaded chunks.
- Buffer status: Buffer occupancy in seconds.

The details of viewport prediction and adaptation logic are explained in the remaining part of this section.

3.2.1 Viewport Prediction. Because at each point of time, the client requests chunks which would be played in the future, adapting to viewport requires having information about the future viewport. To achieve this, we use Weighted Linear Regression (WLR) as in [12]. This method uses the last 10 viewport samples (with a frequency of 10 samples/sec) and applies WLR on them. This method achieves an average prediction accuracy of 96.6% within 10 degrees deviation from the actual viewport, when it predicts viewport for 1 second in future. Increasing the prediction interval to 2 seconds causes the accuracy to drop down to 71.2%. Therefore, we limit our prediction interval to 1 second, i.e. at time t, we predict the user's viewport at time $t + 1$.

3.2.2 Adaptation Logic. Adaptation to network conditions can be done based on throughput estimation (rate-based), buffer occupancy (buffer-based), or a combination of both. Throughput estimation is obtained by smoothing throughput samples from previous chunk downloads. However, dynamic network conditions introduce unpredictability, e.g., in the case of wireless and cellular networks. An over-estimation of network throughput can result in rebuffering, especially for existing adaptive solutions for 360-degree video that use shallow buffers. Buffer-based adaptation methods are proven to perform better in dynamic network conditions[15][6]. Since our proposed method can buffer base layer for longer durations, contrary to existing solutions, we design a buffer-based adaptation method.

The first priority in our adaptation algorithm is to reduce the probability of rebuffering. With scalable video coding, this is achieved by buffering the base layer for a long duration even beyond the viewport prediction high-accuracy interval. This approach enables the client to continue the playback even during bandwidth drops. Algorithm 1 shows the details of proposed adaptation algorithm. The algorithm has two main parts:

- Base layer buffering: Prefetching and buffering base layer chunks for a longer duration.
- Viewport quality enhancement: Enhancing the quality of tiles within the user's viewport.

3.2.3 Base layer buffering. Assume that the client has a buffer that can store B seconds of video, and the length of chunks is 1 second, $s = 1$. The client tries to always buffer B seconds worth of full 360-degree base layer video. So, unless the buffer has B seconds of video, the client prefetches all base layer tiles for the chunks sequentially. Since the client always requires the base layer of a chunk completely, we provide it in a single file that can be prefetched using one request. Enhancement layer tiles are available in different files.

3.2.4 Viewport quality enhancement. If the client had B seconds of base layer chunks in the buffer, it tries to improve the quality of the user's viewport. But to avoid quality switches, the client first considers the network throughput. If the network throughput is less than that required for streaming the higher quality, then that quality level cannot be sustained. Were the client to switch to the higher quality, the buffer would shrink because the arrival rate for tiles will fall behind the playback rate. As a result, the client would switch back to lower quality, which would, in turn, fill the buffer, and lead the client to switch to the higher quality once again. This cycle would keep repeating causing periodic quality switches [15]. To avoid these switches, we consider the throughput estimate when we want to improve the quality within user's viewport.

For estimating network throughput, we sample the available throughput after each iteration of the proposed algorithm. In each iteration, the client either downloads a full 360-degree base layer of a chunk, or enhancement layer for a subset of tiles that are covered by the viewport (VP), denoted by VP_{tiles}. Each throughput sample is calculated according to equation (2).

$$throughput_{sample} = \begin{cases} \dfrac{\sum_{\text{for all } f,i,j} size(T_{c,f,i,j,0})}{T_{fetch}} & \text{if base layer} \\ \dfrac{\sum_{f,i,j \in VP_{tiles}} \sum_{l=1}^{l'} size(T_{c,f,i,j,l})}{T_{fetch}} & \text{if enh. layer} \end{cases} \quad (2)$$

$Size()$ returns the size of tiles in bits. T_{fetch} is the download time for all requested tiles in each iteration of the algorithm. The average of last three samples is calculated as the estimation denoted by $throughput_{est}$. The next step is to select the highest quality level for the tiles within the viewport that can be supported by estimated throughput. For smooth playback, the network throughput should be more than the bitrate of selected quality level. The client always downloads full 360-degree base layer, with a bitrate of R_0, and potentially enhancement layers up to quality level l, with bitrate $R_l - R_0$, for a subset of tiles. The client determines what percentage of tiles are covered by the predicted viewport, denoted by p. Then,

Algorithm 1: Adaptation algorithm

```
/*  •A video consists of N chunks
    •Each chunk consists of tiles T_{c,f,i,j,l} as defined in Sec. 4.1.
    •headNext() is a function returning:
    ·index of the next chunk within 1 second of playback head.
    •predictViewPort() is a function returning:
    · set of viewport-covered tiles in the format of (f,i,j)
    · the percentage of covered tiles to all tiles (p)
    • buff is a buffer of a fixed size with capacity of B seconds.   */
curr_chunk ← 1
while curr_chunk ≤ N do
    if length of buff is less than B − 1 then
        for all f, i, j prefetch T_{curr_chunk,f,i,j,0}
        curr_chunk++
    else
        next ← headNext()
        {VP_tiles, p} ← predictViewPort()
        choose highest l s.t. R_0 + (R_l − R_0) × p < throughput_est
        for each 3-tuple (f, i, j) ∈ VP_tiles and l' ∈ {1, ..., l}
          download T_{next,f,i,j,l'}
    end
end
// All the base layers are downloaded but playback is not finished
next ← headNext()
while next is not null do
    {VP_tiles, p} ← predictViewPort()
    choose highest l s.t. R_0 + (R_l − R_0) × p < throughput_est
    for each 3-tuple (f, i, j) ∈ VP_tiles and l' ∈ {1, ..., l} download
      T_{next,f,i,j,l'}
    next ← headNext()
end
```

the adaptation logic chooses the highest quality level l for the tiles within the predicted viewport such that:

$$R_0 + (R_l - R_0) \times p < throughput_{est}\,^{1} \quad (3)$$

So, the client receives enhancement layer tiles at quality level l, if the estimated throughput is higher than the bitrate of the covered portion of the video.

Figure 4 shows an example of viewport with FoV of 100 degree. The area covered by the viewport is denoted by yellow color. By mapping the viewport to a cube, the client can determine which tiles are covered by the viewport. First, the base layer for all tiles is prefetched and buffered. Later, based on estimated network throughput, the client downloads enhancement layer tiles for the covered tiles.

4 PERFORMANCE EVALUATION

In this section, we evaluate performance of proposed tiling scheme and adaptation algorithm.

4.1 Encoding performance

We have selected four video sequences that cover various spatiotemporal complexities, We refer to as Seq1-4. Seq1[2] and Seq2[3] have

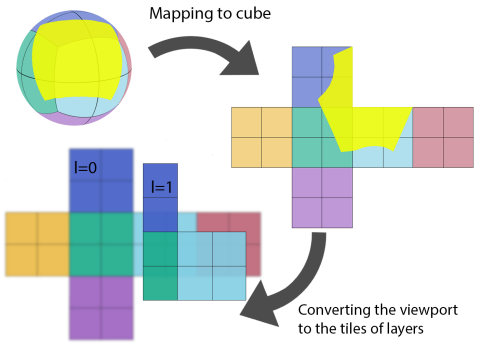


Figure 4: Viewport mapping and adaptation

fast motion content. Seq3[4]has low motion but contains high detail textures, and Seq4[5] has both low motion and low texture. All video sequences were available in the equirectangular projection with 4K resolution and frame rate of 30 frames per second.

For studying encoding performance, we selected 100 frames from each of the sequences. First, we map the equirectangular video to cubemap projection. The dimensions of each face of the cube are set to 1024 × 1024 pixels. Then, we evaluate the performance for two sets of tiling, 1 or 4 tiles per face ($m = 1$ and $m = 2$). For each configuration, we encode the videos with non-scalable (HEVC) and scalable (SHVC) version of H.265/HEVC standard [17]. HM reference software version 16.7 [4] and SHM reference software version 12 [14] are used for this purpose. Based on the performance evaluation done in [3], the encoding overhead of quality (SNR) scalability is less than spatial scalability. Moreover, using SNR scalability, no up-sampling is required to render the video on cube map. So we decided to encode the scalable version with quality scalability. For scalable encoding, we encode video into two layers. Quantization Parameters (QP) of 20, 24, 28, and 32 are used for enhancement layer, and the QP for the base layer is set to 4 or 6 steps higher than the enhancement layer QP. The encoding GOP size is 16, and the interval for IDR frames is 32. So, we can divide the video into chunks with duration of 32 frames.

The compression performance is reported based on Bjøntegaard Delta-rate (BD-rate) metric [2], which shows the average difference between bitrates of two encoded streams at the same quality level. Table 1 shows the compression penalty of dividing each face of the cube into 4 tiles vs. 1 tile. In this table, HEVC shows the result for the non-scalable version, and SHVC-SNR4 and SHVC-SNR6 are for the scalable version with the QP difference of 4 and 6, respectively. It can be seen that dividing the video into more tiles increases overhead on bitrate. With smaller tiles, objects in the scene are divided among more tiles. So encoding the tiles independently will result in encoding inefficiency.

Table 2 shows the overhead of scalable encoding. The overhead is between 10% to 21%. However, this overhead is for the case that both layers of all tiles are considered. In our streaming system, enhancement layer tiles are sent only for the viewport. So the encoding overhead in bitrate will be smaller.

[1] Assuming that the bitrate of a video is uniformly distributed among tiles
[2] https://www.youtube.com/watch?v=yVLfEHXQk08 (starting at 0s)
[3] https://www.youtube.com/watch?v=5fyVCocJoks (starting at 0s)
[4] https://www.youtube.com/watch?v=bz2Ea2KtFIU (starting at 34s)
[5] https://www.youtube.com/watch?v=11ovodjozLI (starting at 0s)

Table 1: BD-rate for penalty of encoding with 4 tiles vs. 1 tile per face

Encoding	Seq1	Seq2	Seq3	Seq4
HEVC	+4.24%	+8.38%	+8.55%	+7.67%
SHVC-SNR4	+8.18%	+6.86%	+7.06%	+7.51%
SHVC-SNR6	+3.2%	+6.84%	+7.20%	+6.53%

Table 2: BD-rate for overhead of SHVC encoding

Tiling	Encoding	Seq1	Seq2	Seq3	Seq4
1	SHVC-SNR4	+16.89%	+17.72%	+13.52%	+14.28%
	SHVC-SNR6	+15.75%	+19.23%	+12.65%	+15.67%
4	SHVC-SNR4	+9.69%	+19.6%	+15.49%	15.87%
	SHVC-SNR6	+16.7%	+21.01%	+14.36%	16.02%

Table 3: BD-rate for only enhancement layer compared to equivalent quality HEVC encoded video

Tiling	Encoding	Seq1	Seq2	Seq3	Seq4
1	SHVC-SNR4	-42.73%	-46.13%	-54.58%	-53.21%
	SHVC-SNR6	-28.52%	-29.95%	-41.58%	-42.47%
4	SHVC-SNR4	-46.40%	-45.33%	-46.12%	-48.64%
	SHVC-SNR6	-28.32%	-29.07%	-40.69%	-38.52%

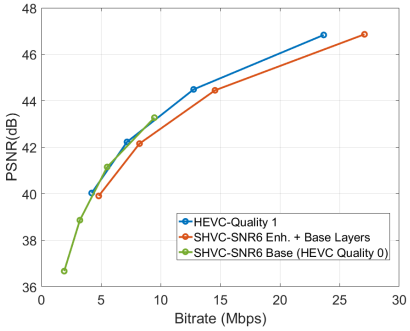


Figure 5: Rate-Distortion curve for Seq1

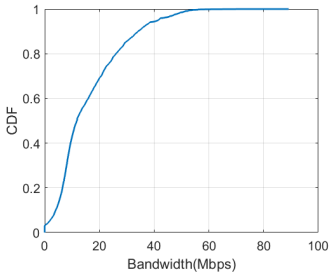


Figure 6: CDF of five 4G traces

Table 3 shows BD-rate between the enhancement layer of SHVC and the highest quality bitstream of HEVC. When the QP difference between the two layers is higher, the enhancement layer bitrate increases. This is an important consideration for streaming scalable 360 video. By increasing QP difference, the base layer can be of lower bitrate and quality which results in more bandwidth saving. But on the other hand, giving a higher share of bitrate to enhancement layer makes video streaming more challenging for clients. Because enhancement layer tiles are streamed close to their playback deadline, and they are more sensitive to network conditions variation.

One of the benefits of using scalable encoding is that the redundancy between the layers is reduced, and for higher quality layers, only the differences with lower layers are encoded. While in non-scalable encoding, each quality level is encoded independently. This results in lower storage requirements for scalable encoding. Encoding seq1-seq4 with SHVC and HEVC at two quality levels shows that storage requirement for the base layer and the enhancement layer is about 20% lower than storing low and high videos independently encoded by HEVC.

4.2 Streaming Performance

The next set of experiments evaluates the performance of our proposed method in terms of bandwidth consumption and user's QoE.

4.2.1 Experimental Setup. Our setup consists of one client connected over a cellular link to a web server hosting the video chunks. So the connection between client and server is over HTTP. We chose Mahimahi LinkShell [10] to emulate the behavior of the cellular link. It accurately replicates the behavior of real networks by replaying pre-recorded traces of network packet transmissions. We use five traces provided by the authors of Mahimahi which record the behavior of 4G networks in the U.S. under varying channel conditions. Figure 6 shows the Cumulative Distribution Function (CDF) of available bandwidth in all traces. Cellular networks suffer most from high variability in network conditions, so it was a natural choice for our evaluation environment. In addition, we use an artificial trace with a constant bandwidth of 50 Mb/s as a control. This control scenario is to show how adaptation methods perform in bandwidth saving when they have no limitation on bandwidth.

In all experiments, the client plays 120 seconds of Seq1 (the video is repeated to cover 120 seconds). We use HEVC and SHVC-SNR6 for non-scalable and scalable videos. The rate-distortion of this video sequence is shown in Figure 5. We select the bitstreams encoded with QP 34 for quality level 0 and QP 28 for quality level 1. The tested video has a bitrate of 3230Kbps for quality level 0 (and base layer). The bitrate for HEVC quality level 1 is 7148Kbps, and bitrate for SHVC-SNR6 enhancement+base layer is 8229Kbps.

The client runs our player application which has the proposed adaptation algorithm as its adaptation logic unit. The components of the player are the same as Figure 3. The player application emulates user's viewport changes. It is capable of emulating various vertical and horizontal FoVs, and for the user head movements, it takes real viewport traces as input. These traces have samples of head orientation recorded every 0.1 second. In all experiments, we use vertical and horizontal FoVs of 90 and 100 degrees, respectively.

The player supports streaming of both non-scalable and scalable-tiled videos. For scalable-tiled streaming, which is our proposed method, a buffer of $B = 10s$ for the base layer is considered. For non-scalable tiled video, client acts like the existing tile-based methods. It predicts viewport for next 1 second, and based on bandwidth estimation decides to get either all tiles in quality 0 or the viewport covered tiles at quality level 1 and other tiles in quality level 0. The player plays a video chunk only after receiving all of its tiles.

The client records following measurements:

- Number of bytes downloaded
- Average quality: Average quality level of downloaded tiles within user's viewport. We consider the viewport sample at the start of chunk playback to calculate the average over all tiles. This metric is between 0 and 1.
- Number of playback quality switches: If the difference of average quality between two consecutive chunks is more than 0.5 we consider it as a quality switch.
- Number and duration of rebuffering events.

We run two sets of experiments. In the first set, in order to avoid the dependency of experiments on human viewport traces, we run each experiment twice with the best- and worst-case static viewports. The best-case viewport places the least demand on the network by always looking at one face of the cube-mapped video while the worst-case viewport places the most demand on the network by looking at the intersection point between three faces of the cube. In summary, each experiment can be uniquely identified by the choice of four parameters: network trace (out of 6 traces), viewport (best/worst-case), streaming method (non-scalable/scalable), and a number of tiles per face (1 or 4).

In the next set of experiments, we use the same 6 network traces with changing viewport scenarios. We use 5 viewport traces gathered by viewing Seq1 to five users. Each user watched Seq1 using an HMD. We recorded the head orientation samples every 0.1 second during playback.

4.2.2 Bandwidth Saving Results. Figure 7 shows the results for bandwidth saving obtained by using scalable and non-scalable adaptive streaming methods compared to the baseline streaming which streams the whole video at the highest quality. These results are taken from the control bandwidth scenario that allows streaming of highest quality without any interruption. In best-case scenario, only one face of the cube is enough to cover viewport, and the results show similar bandwidth saving for scalable and non-scalable methods. Our method has slightly lower bandwidth saving which is due to the overhead of scalable encoding. However, because in adaptive methods only the viewport is streamed at high quality, the penalty in bandwidth saving is 2% for the scalable method, while the overhead in encoding is around 15% (Table 2).

In worst-case viewport scenario, the difference between 1 and 4 tile per face is more visible. The reason is that the 4 tiles case represents the worst-case scenario with 7 tiles of size 512×512 pixels, while for the 1 tile case, the viewport will be covered with 3 tiles of 1024×1024 size. According to the results, dividing each face of the cube into 4 tiles results in better bandwidth saving.

4.2.3 Effect of Buffer Length. Figure 8 shows the average duration and average number of rebufferings for three buffer sizes:

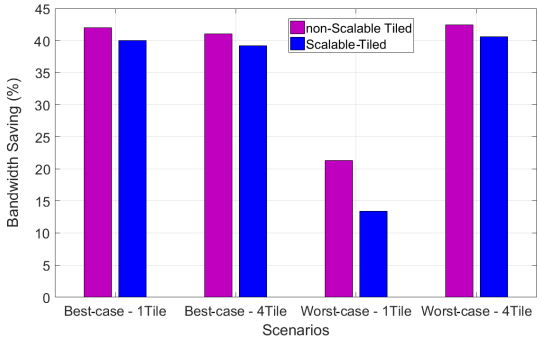
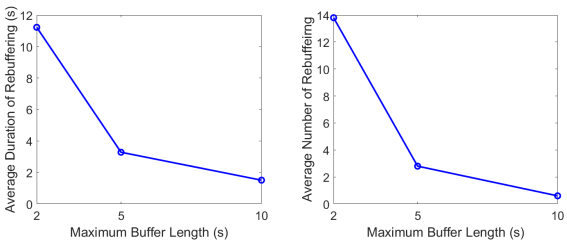


Figure 7: Bandwidth saving



(a) Duration of rebufferings **(b) Number of rebufferings**

Figure 8: Effect of buffer length on rebuffering

$B = 2$, 5 and 10 seconds. We can see that with a 2-second buffer, the client cannot provide smooth playback under varying network conditions, and it experiences a lot of rebufferings. This experiment supports our argument about the problem of buffering with existing solutions with shallow buffers. As buffer length increases, rebufferings decreases drastically. For all other experiments, we use buffer length of 10 seconds.

4.2.4 Quality of Experience results. Figure 9 shows detailed result for playback in experiments with fixed viewport and 4 tiles per face.[6] Due to rebufferings, the playback may take longer time than 120 seconds, and we show the playback of first 120 seconds of experiments. The right column is for scalable streaming, and left column shows non-scalable streaming. Blue parts show playback at highest possible quality, and red ones show low-quality playback. Rebufferings are depicted with white spaces. It can be seen that our proposed method outperforms non-scalable tiled streaming method. The main advantage is reduced rebuffering. Moreover, in each scenario, number of quality switches and average quality is also improved with using scalable-tiled streaming. The main reason is the buffering of the base layer enables the proposed adaptation algorithm to react better to network changes. The details for rebufferings are shown in Table 4 for all test cases.

Table 5 shows the results for measured QoE metrics for the experiments with dynamic real traces of viewport. We only used the configuration with 4 tiles per face because the performance for

[6]The results for 1 tile per face are similar, but the number of rebufferings is slightly higher.

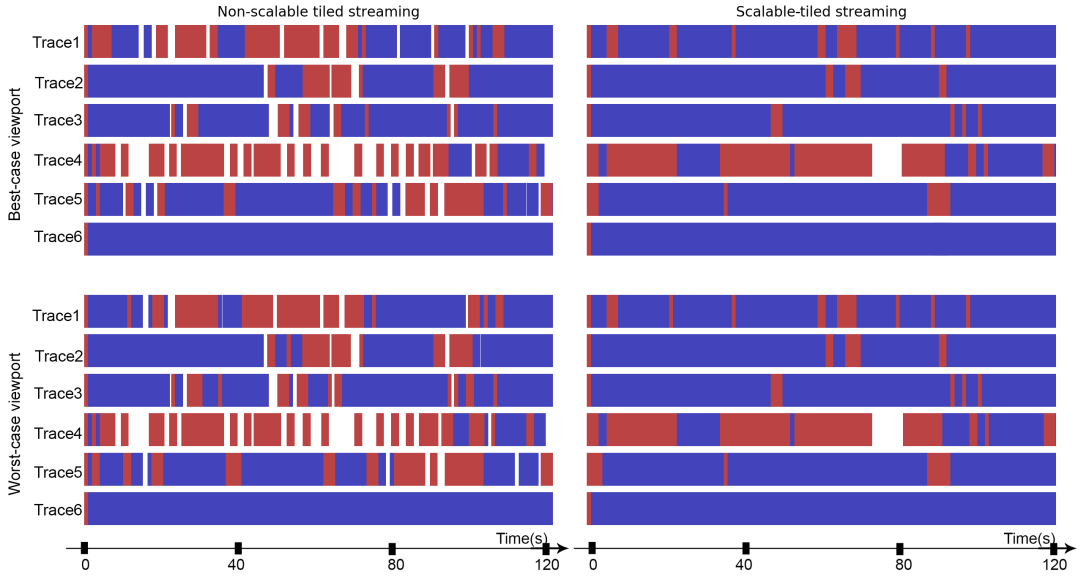


Figure 9: First 120 seconds of experiments with best and worst case viewport, and 4 tiles per face (white=rebuffering, red: low quality playback, blue: high quality playback)

Table 4: Number and duration of rebuffering events

	Best-case viewport				Worst-case viewport			
Trace	1 Tile		4 Tile		1 Tile		4 Tile	
	Non-scalable	Scalable	Non-scalable	Scalable	Non-scalable	Scalable	Non-scalable	Scalable
Trace1	12 (13.0s)	0	11 (12.2s)	0	17 (16.5s)	0	6 (7.3s)	0
Trace2	6 (7.8s)	0	4 (4.5s)	0	8 (11.8s)	0	4 (4.4s)	0
Trace3	4 (2.8s)	0	7 (7.6s)	0	9 (10.8s)	0	7 (6.7s)	0
Trace4	24 (54.4s)	2 (5.2s)	26 (5.6s)	1 (7.5s)	27 (57.2s)	2 (5.8s)	25 (54.7s)	1 (7.9s)
Trace5	12 (15.5s)	0	11 (10.2s)	0	19 (25.2s)	0	7 (8.1s)	0
Average	11.6 (18.7s)	0.8 (1.0s)	11.8 (8.0s)	0.6 (1.5s)	16 (24.3s)	0.8 (1.2s)	9.8 (16.2s)	0.6 (1.6s)

Table 5: Results for QoE metrics with dynamic viewport

Adaptation method	Average quality	No. of switches	Rebuffering duration	No. of rebufferings
Non-scalable tiled	0.66	14.1	17.62s	11.2
Scalable-tiled	0.88	10.2	1.5s	0.6

bandwidth saving is better. It can be seen that in all metrics, our proposed method based on scalable video encoding outperforms non-scalable tiled streaming.

5 CONCLUSION AND FUTURE WORK

Streaming 360-degree video is very challenging under bandwidth limited and dynamic network conditions. In this paper, we proposed a tile-based adaptive 360-degree video streaming method based on scalable video coding. The proposed method adapts to both viewport and network throughput. Our experimental results using real-world 4G traces and viewport traces show that this method mitigates the rebuffering problem of existing adaptive streaming solutions, while it provides bandwidth saving by streaming high quality only for the viewport. This improvement is achieved by buffering base layer of video at the client, which makes client robust to network condition changes. The client can continue playback of at least a low-quality version of the video when the download time of chunks varies. Moreover, the proposed scalable-tiled method provides saving of storage space at the server side, and it has the potential for improving the in-network caching as multiple users watch the same content that require the same base layer.

As future work, we plan to extend our work to design a monolithic version of scalable streaming, and compare the performance of tiled-based and monolithic methods in terms of encoding and streaming performance.

REFERENCES

[1] 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.

[2] Gisle Bjontegaard. 2001. Calcuation of average PSNR differences between RD-curves. *Doc. VCEG-M33 ITU-T Q6/16, Austin, TX, USA, 2-4 April 2001* (2001).

[3] Jill M Boyce, Yan Ye, Jianle Chen, and Adarsh K Ramasubramonian. 2016. Overview of SHVC: scalable extensions of the high efficiency video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 1 (2016), 20–34.

[4] HM. 2016. High Efficiency Video Coding (HEVC) reference software. https://hevc.hhi.fraunhofer.de/. (2016). Access date: November 2016.

[5] M. Hosseini and V. Swaminathan. 2016. Adaptive 360 VR Video Streaming: Divide and Conquer. In *2016 IEEE International Symposium on Multimedia (ISM)*. 107–110. DOI: http://dx.doi.org/10.1109/ISM.2016.0028

[6] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[7] Evgeny Kuzyakov and David Pio. 2015. Under the hood: Building 360 video. https://code.facebook.com/posts/1638767863078802/under-the-hood-building-360-video/. (2015). Access date: April 2017.

[8] Evgeny Kuzyakov and David Pio. 2016. Next-generation video encoding techniques for 360 video and VR. https://code.facebook.com/posts/1126354007399553/next-generatio-video-encoding-techniques-for-360-video-and-vr/. (2016). Access date: April 2017.

[9] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. 2011. Measuring the quality of experience of HTTP video streaming. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 485–492.

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

[11] Pixvana. 2017. An Intro to FOVAS: Field of View Adaptive Streaming for Virtual Reality. http://blog.pixvana.com/intro-to-field-of-view-adaptive-streaming-for-vr. (2017). Access date: April 2017.

[12] 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*. 1–6.

[13] 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.

[14] SHVC. 2016. HEVC Scalability Extension reference software. (2016). https://hevc.hhi.fraunhofer.de/shvc.

[15] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[16] Kashyap Kammachi Sreedhar, Alireza Aminlou, Miska M Hannuksela, and Moncef Gabbouj. 2016. Viewport-Adaptive Encoding and Streaming of 360-Degree Video for Virtual Reality Applications. In *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 583–586.

[17] Gary J Sullivan, Jens Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology* 22, 12 (2012), 1649–1668.

[18] Afshin TaghaviNasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash. 2017. Adaptive 360-degree video streaming using layered video coding. In *Virtual Reality (VR), 2017 IEEE*. IEEE, 347–348.

[19] 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.