An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos

Stefano Petrangeli
Ghent University - imec
stefano.petrangeli@ugent.be

Viswanathan Swaminathan
Adobe Research
vishy@adobe.com

Mohammad Hosseini
University of Illinois at Urbana-Champaign
shossen2@illinois.edu

Filip De Turck
Ghent University - imec
filip.deturck@ugent.be

ABSTRACT

Virtual Reality (VR) devices are becoming accessible to a large public, which is going to increase the demand for 360° VR videos. VR videos are often characterized by a poor quality of experience, due to the high bandwidth required to stream the 360° video. To overcome this issue, we spatially divide the VR video into tiles, so that each temporal segment is composed of several spatial tiles. Only the tiles belonging to the viewport, the region of the video watched by the user, are streamed at the highest quality. The other tiles are instead streamed at a lower quality. We also propose an algorithm to predict the future viewport position and minimize quality transitions during viewport changes. The video is delivered using the server push feature of the HTTP/2 protocol. Instead of retrieving each tile individually, the client issues a single push request to the server, so that all the required tiles are automatically pushed back to back. This approach allows to increase the achieved throughput, especially in mobile, high RTT networks. In this paper, we detail the proposed framework and present a prototype developed to test its performance using real-world 4G bandwidth traces. Particularly, our approach can save bandwidth up to 35% without severely impacting the quality viewed by the user, when compared to a traditional non-tiled VR streaming solution. Moreover, in high RTT conditions, our HTTP/2 approach can reach 3 times the throughput of tiled streaming over HTTP/1.1, and consistently reduce freeze time. These results represent a major improvement for the efficient delivery of 360° VR videos over the Internet.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Virtual reality**; • **Networks** → *Network protocols*; *Public Internet*;

KEYWORDS

Virtual reality; HTTP Adaptive Streaming; HTTP/2; Server push; Viewport prediction; H.265; Tiling

MM '17, October 23–27, 2017, Mountain View, CA, USA

DOI: https://doi.org/10.1145/3123266.3123453

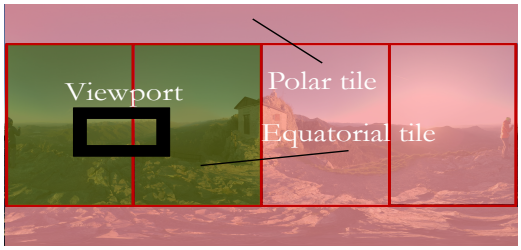


Figure 1: In tiled VR streaming, only tiles belonging to the viewport (in green) are streamed at the highest quality.

ACM Reference format:
S. Petrangeli, V. Swaminathan, M. Hosseini, F. De Turck. 2017. An HTTP/2-Based Adaptive Streaming Framework for 360° Virtual Reality Videos. In *Proceedings of MM '17, October 23–27, 2017, Mountain View, CA, USA*, 9 pages.
DOI: https://doi.org/10.1145/3123266.3123453

1 INTRODUCTION

Recent advancements in consumer electronics have made Virtual Reality (VR) devices accessible to a large public. Consequently, the demand for 360° VR video streaming is expected to grow exponentially in the near future. Streaming VR videos over the best-effort Internet is challenged today by the high bandwidth required to stream the entire 360° video. This aspect is often responsible for low video quality and buffer starvations, two of the main factors influencing users' Quality of Experience (QoE). View-dependent solutions are ideal in saving bandwidth for VR streaming, as only the *viewport*, the portion of the video watched by the user, is streamed at the highest quality, while the rest of the video is streamed at a lower quality. Despite that, current solutions require to store a different version of the video for each possible position of the viewport, entailing huge CDN costs [5]. Tiling the VR video allows to obtain similar results in terms of bandwidth savings, without additional storage compared to traditional streaming. In tiling, the VR video is divided into spatial regions, each encoded at different quality levels. To save bandwidth, only tiles inside the viewport are streamed at the highest quality (Figure 1). This approach can be combined with the bandwidth adaptation of HTTP Adaptive Streaming (HAS) techniques. In tiled HAS, the video is both temporally segmented and spatially tiled, so that each temporal segment of the video is composed of several video tiles.

Unfortunately, tiling the video causes a significant increase in the number of requests when HAS is used over HTTP/1.1. Each tile has to be requested independently from the others in order to create a complete temporal video segment, meaning that this approach is susceptible to high RTTs, typical in mobile networks. As an example, assuming the video is composed of 6 tiles and the RTT is 100 ms, it would require at least 600 ms to download each temporal segment. This behavior can consistently lower the achieved throughput and limit the practical applicability of tiled VR streaming. In this paper, we propose to overcome this drawback using the server-push functionality of the HTTP/2 protocol [18]. Only a single HTTP GET request is sent from the client; all the tiles are automatically pushed from the server. This approach allows to overcome the main drawback introduced by tiling. HTTP/2 shares the same methods, status codes and semantics with HTTP/1.1, entailing a complete backward compatibility. Server push, and more generally HTTP/2, is also completely cache- and CDN-friendly.

The main contributions of this paper are three-fold. First, we propose a viewport-dependent HTTP/2-based streaming framework, where the client decides the quality of each tile based on the user viewport. The H.265 standard is used to tile the video [10], as it allows to use a single decoder to decode the different tiles, which represents an important advantage on resource-constrained devices, as smartphones and tablets. Moreover, the HTTP/2's server push can completely eliminate the overhead introduced by tiling the video, as only one HTTP request specifying the quality of the tiles is sent from the client to the server. Based on this request, the server can push all the requested tiles back to back, consequently reducing the impact of the network RTT. This approach does not require any client status to be kept on the server. A push directive, as standardized by MPEG-DASH [11], embedded in the client request specifies the tiles qualities. By parsing this directive, the server understands which tiles to push. Second, an algorithm is proposed to predict the future viewport position, in order to minimize quality transitions during viewport changes. An estimate of the future viewport is computed based on the viewport speed, in order to anticipate the user's movements and request in advance the right portion of the video at the highest quality. This approach allows to provide a graceful viewport transition and maximize QoE. Third, extensive experimental results are collected to quantify the gains of the proposed framework using a prototype implemented on a Gear VR and Samsung Galaxy S7. Particularly, we show that our HTTP/2 solution can reach better quality and lower freeze time compared to standard tiled video over HTTP/1.1, in high RTT conditions.

The remainder of this paper is structured as follows. Section 2 presents the related work on 360° VR video streaming and HTTP/2-based adaptive streaming. Section 3 describes in detail the proposed framework, both from an architectural and algorithmic point of view, while Section 4 reports the obtained results. Section 5 concludes the paper.

2 RELATED WORK

2.1 360° Video Streaming

There is a large body of literature addressing the high bandwidth requirements of 360° videos, with tiling identified as a good candidate to alleviate this problem.

D'Acunto et al. propose an MPEG-DASH SRD client to optimize the delivery of zoomable videos, which are affected by the same bandwidth problem as 360° videos [4]. The video is spatially divided in tiles: the low resolution tile, corresponding to the whole zoomable video, is always downloaded to avoid a black screen in case of viewpoint changes. The high resolution tiles, corresponding to the zoomed part of the video, are downloaded afterwards. Wang et al. deliver the tiles of a zoomable video using multicast [17]. An algorithm is proposed to decide the resolution of the tiles to be multicasted and maximize the utility of all users. Lim et al. use tiling to efficiently deliver panoramic videos [9]. In these works, the video is tiled using H.264, which does not natively support tiling. This aspect complicates the synchronization of the tiles, as each tile has to be decoded independently. Le Feuvre et al. propose to use H.265 to spatially divide the 360° video [8]. They also propose a rate allocation algorithm to decide the quality of each tile based on the available bandwidth and the user viewport. The tiles are delivered using HTTP/1.1 and no prediction is proposed to anticipate the user's movements. Gaddam et al. use tiling and viewport prediction to stream interactive panoramic videos, where part of the panorama can be used to extract a virtual view [6]. Also in this case, the video is encoded in H.264 and transported over regular HTTP/1.1. Cuervo et al. investigate the use of panoramic stereo video and likelihood-based foveation to deliver the 360° video [3]. Any possible virtual view can be extracted from the panoramic video, whose quality gradually degrades based on the part of the video the user is most likely to watch in the future. Qian et al. propose a framework where only the portion of the 360° video watched by the user is actually transmitted, to save bandwidth [13]. The developed viewport prediction algorithm should therefore be extremely precise in order to avoid stalling when the user changes viewport. Moreover, results are only presented in a simulated environment. TaghaviNasrabadi et al. use scalable video coding and tiling to stream the VR video [15]. The video is divided in different tiles, and each tile is encoded at different scalable layers. As the video is delivered over standard HTTP/1.1, this approach is susceptible to high RTT's. While all of the above research is successful in addressing some of the problems affecting 360° video streaming, there is no single solution to address all problems. Our work is an attempt to provide a comprehensive solution for VR streaming. By using H.265 to tile the video, we can eliminate the synchronization issues at client side introduced by tiling. The tiled video is transported using HTTP/2, which allows to eliminate the significant increase of GET requests due to the spatial partitioning of the video. Finally, viewport prediction can successfully compensate the quality degradation introduced by assigning lower qualities for tiles outside the viewport, by anticipating user's movements.

Budagavi et al. mainly focus on how to optimize the encoding process to reduce the bit-rate of VR videos [1]. By gradually smoothing the quality of the bottom and top part of an equirectangular projection, they are able to reduce the bit-rate by 20%. Zare et al. propose a modified H.265 encoder to more efficiently tile a 360° video [20]. Hosseini et al. propose a new tiling structure of the 360° video, which allows to save up to 30% of the bandwidth compared to a non-tiled video [7]. The same tiling structure has also been adopted in this paper, because of its efficiency. Our approach can

be considered complimentary to these works, as in our case we mostly focus on the delivery of the video, rather than its encoding and preparation.

2.2 HTTP/2-Based Adaptive Streaming

One of the new features introduced by HTTP/2 is the possibility for the server to push resources that have not been requested directly by the client. This mechanism was originally proposed to reduce the latency in web delivery, but has also been applied in the delivery of multimedia content. Wei et al. are the first to investigate how server push can improve the delivery of HAS streams [18]. They focus on the reduction of the camera-to-display delay, which is obtained by reducing the segment duration and pushing k segments after a single HTTP GET request is issued by the client. Xiao et al. extend the k-push mechanism to optimize the battery lifetime on mobile devices, by dynamically varying the value of k based on network conditions and power efficiency [19]. van der Hooft et al. also investigate the merits of server push for H.265 videos over 4G networks [16]. Segments with a sub-second duration are continuously pushed from the server to the client, in order to reduce the live delay compared to HTTP/1.1-based solutions. Cherif et al. use server push in conjunction with WebSocket to reduce the startup delay in a DASH streaming session [2]. In this work, we exploit the server push functionality in order to reduce the network overhead introduced by spatially dividing the video into separate tiles. Instead of pushing the segments one after other, we use the k-push mechanism to push the tiles composing a single temporal segment back-to-back from the server to the client.

3 HTTP/2-BASED VR STREAMING FRAMEWORK

In this section, we describe the proposed framework for VR streaming. Our framework builds upon three components, which combined overcome the main issues affecting current VR streaming solutions, namely storage costs and bandwidth requirements. First, the VR content is encoded using the H.265 standard and divided into spatial tiles, each encoded at different quality levels (Section 3.1). Besides an encoding overhead introduced by the tiling process, this approach requires the same amount of storage as in classical video streaming. Second, the video client is equipped with an algorithm that can select the best video quality for each tile based on information as the current and predicted future viewport and the available network bandwidth (Section 3.2). By dynamically changing the viewport quality, the bandwidth required to stream the 360° video can be consistently reduced. Third, the server-push functionality of the HTTP/2 protocol allows to eliminate the significant increase of HTTP GET requests caused by tiling the video (Section 3.3), in turn increasing the achieved throughput, especially in high RTT networks.

3.1 H.265 Video Tiling

One of the innovations introduced by the H.265 standard is the possibility to spatially divide the video into regions, called tiles [10]. The tiles can be physically separated from each other and reconstructed in a common stream that can be decoded by a single decoder. This tiling process is extremely beneficial in VR streaming, where the user can only watch a fraction of the entire 360° video at any given point in time. In fact, only the tiles inside the viewport are streamed at the highest quality. This approach would still give the same feeling of immersion as if the entire 360° video was streamed at the highest quality, while requiring less bandwidth compared to full quality VR streaming and less storage compared to viewport-dependent non-tiled VR streaming. The storage savings can be quantified based on the underlying 2D projection used for the VR video. We consider an equirectangular projection with width w and height h, and viewport with width and height equal to w_p and h_p, respectively. We assume the video is composed of two quality levels, with bit-rates $b_1 \geq b_0$. In our tiled approach, the total storage required to stream a single video is given by:

$$S_{tiled} = d \times (b_1 + b_0) \times \alpha(n_t)$$

where d is the video duration and $\alpha(n_t)$ represents the encoding overhead introduced by the tiling process, which depends on the number of tiles n_t. Conversely, in a non-tiled approach, a different copy of the video has to be encoded for each desired viewport configuration:

$$S_{non-tiled} = d \times \left[\frac{w_p \times h_p}{w \times h} \times b_1 + \left(1 - \frac{w_p \times h_p}{w \times h}\right) \times b_0 \right] \times \\ \times N(w, h, w_p, h_p, \xi)$$

The term in brackets represents the bit-rate necessary to stream a single viewport configuration and is given by the percentage of the equirectangular projection occupied by the viewport at high quality (first term) plus the remaining part of the video streamed at a lower quality (second term). $N(w, h, w_p, h_p, \xi)$ indicates the number of different viewport configurations to encode, with ξ being the step separating the different viewports:

$$N(w, h, w_p, h_p, \xi) = \lfloor \frac{w - w_p}{\xi} \rfloor \times \lfloor \frac{h - h_p}{\xi} \rfloor$$

Consequently, the gain in terms of storage between a tiled and non-tiled approach can be quantified as follows:

$$G = \frac{S_{non-tiled}}{S_{tiled}} = \frac{b_1 - b_0}{b_1 + b_0} \times \frac{w_p \times h_p}{w \times h} \times \frac{N(w, h, w_p, h_p, \xi)}{\alpha(n_t)} + \\ + \frac{b_0}{b_0 + b_1} \times \frac{N(w, h, w_p, h_p, \xi)}{\alpha(n_t)}$$

The previous equation can be generalized to the case where Q quality levels are available, such that the quality is gradually degraded outside the viewport:

$$G = \frac{S_{non-tiled}}{S_{tiled}} = \frac{N(w, h, w_p, h_p, \xi)}{\alpha(n_t)} \times \\ \times \frac{\frac{1}{w \times h} \times \sum_{q=1}^{Q} b_q \times \left[w_i \times h_i - \sum_{j=1}^{q-1} w_j \times h_j \right]}{\sum_{q=1}^{Q} b_q}$$

As for the actual tiling structure, we used the same approach from Hosseini et al. [7] (see Figure 1). In total, six tiles are created: two *polar* tiles and 4 *equatorial* tiles. This tiling process allows to reduce the bandwidth needed to stream the VR video by about 30%,

when compared to a full quality non-tiled approach [7]. As in HEVC tiling all columns must have the same number of rows, we simply take the tiles belonging to the polar region and concatenate them together, so that they can be requested as a single object by the client.

3.2 Tiles Quality Selection

An important aspect of the proposed framework is the client-based heuristic in charge of deciding the quality of the tiles. While in classical HAS this decision is mainly based on network conditions, in 360° VR streaming a new dimension is added, namely the user viewport. In this work, we consider as viewport the region with center the fixation point and 60-degrees in radius, known as the mid-peripheral region. All tiles overlapping with the viewport region are considered viewport tiles. The quality of the tiles belonging to the viewport should always be maximized in order to guarantee an immersive experience to the user. The remaining tiles can be streamed at a lower quality, to save bandwidth while guaranteeing a fast transition when the viewport changes. In order to reduce the transition time and maximize the viewed quality, our heuristic can also predict the future viewport, based on the fixation point speed. This way, the video client can request in advance the tiles belonging to the predicted viewport and therefore provide a seamless transition.

As in classical HAS, a new decision about the tiles quality is made by the client after a segment has been completely downloaded. First, the client identifies tiles belonging to the current and future viewport. In order to compute the future viewport, we obtain the position p at instant k of the current fixation point on the underlying 2D projection of the VR video, for example, latitude and longitude for an equirectangular projection. The future fixation point, which defines the future viewport, is computed as:

$$p(k+\Delta) = p(k) + \Delta \times \widehat{p(k)}$$

$$\widehat{p(k)} = \frac{p(k) - p(k-\delta)}{\delta}$$

where Δ is the future viewport prediction horizon, $\widehat{p(k)}$ is the speed of the fixation point and δ is the speed measurement interval. In our work, Δ is equal to the segment duration of the video. In order to guarantee a fine-grained monitoring of the viewport speed, δ is fixed to 100 ms.

Once the future viewport is computed, the tiles are logically divided into three categories: *viewport*, for tiles belonging to the current and future viewport, *adjacent*, for tiles immediately outside the viewport tiles and *outside*, for all the remaining ones. A design choice is taken regarding the polar tiles (Figure 1). Particularly, a polar tile always belongs to the *outside* group, unless it is part of the tiles between the current and future viewport (i.e., a *viewport* tile). The actual quality of the tiles is selected based on the available bandwidth, as described in Algorithm 1. The algorithm takes as inputs the available perceived bandwidth B, the bit-rates of the video $b(-)$ and the aforementioned tiles categories. First, the lowest quality is assigned to all the tiles of the video (line 1). This initial allocation guarantees that all the tiles of the video are streamed to the user. Then, the available bandwidth budget B_{budget} is computed

Algorithm 1 Tiles quality selection heuristic.

Require:
B, available perceived bandwidth (in Mbps)
b, vector containing the bit-rates (in Mbps) of the available quality levels, from 0 (lowest) to n_q (highest)
viewport, *adjacent*, *outside* tiles groups
n_T, the total number of video tiles
Ensure:
$qt(-)$, vector of the assigned tiles quality

1: $qt(t) = 0 \quad \forall t \in \{viewport, adjacent, outside\}$
2: $B_{budget} = B - n_T \times b(0)$
3: **for** $tiles\_category$ in $\{viewport, adjacent, outside\}$ **do**
4: $\quad qt = \max_{q \in [1;n_q]} q \quad s.t. \quad b(q) \leq \frac{B_{budget}}{n_{tiles}}$
5: $\quad B_{budget} = B_{budget} - n_{tiles} \times b(qt)$
6: $\quad qt(t) = qt \quad \forall t \in tiles\_category$
7: **end for**

(line 2) as the difference between the available bandwidth and the total bit-rate allocated to the tiles. Next, the highest possible quality is assigned to the tiles, given the bandwidth budget (lines 3-7), starting from *viewport* tiles. We select the highest quality q such that the corresponding bit-rate $b(q)$ is lower than the ratio between the current bandwidth budget and number of tiles n_{tiles} belonging to the analyzed category (line 4). We then update the bandwidth budget (line 5) and repeat the allocation for the *adjacent* and *outside* tiles, till we run out of bandwidth. This way, we mitigate the edge effect between tiles at different qualities by gradually reducing the quality as we move out of the viewport.

3.3 HTTP/2 Server Push for Tiled Videos

Once the tiles quality is decided, the client issues an HTTP GET request to the server. In classical tiled streaming over HTTP/1.1, n_T HTTP GET requests have to be issued in order to retrieve a single temporal segment, with n_T equal to the number of tiles. This aspect entails that n_T RTTs are lost, which can lower the achieved throughput in mobile, high RTT networks. We propose to solve this issue by using the server push functionality of the HTTP/2 protocol and, particularly, the k-push approach proposed by Wei et al. [18], with k set to n_T. In this case, only one request is sent from the client to the server, specifying the qualities of the video tiles, decided as reported in Section 3.2. All the tiles are consequently pushed from the server to the client using the HTTP/2 protocol. This approach eliminates the request overhead due to tiling and results in a better bandwidth utilization, even in high RTT networks. It is worth stressing that the k-push mechanism does not require any client status to be kept on the server. A push directive, as standardized by part 6 of the MPEG-DASH standard [11], embedded in the client request specifies the tiles qualities. We extended the *urn:mpeg:dash:fdh:2016:push-next* push directive in a compatible way to allow the client specifying the tiles qualities. By parsing this directive, the server understands which tiles to push. HTTP/2 server push is by design cache compatible, and several CDNs are starting to deploy it[1].

[1] https://blogs.akamai.com/2016/04/are-you-ready-for-http2-server-push.html

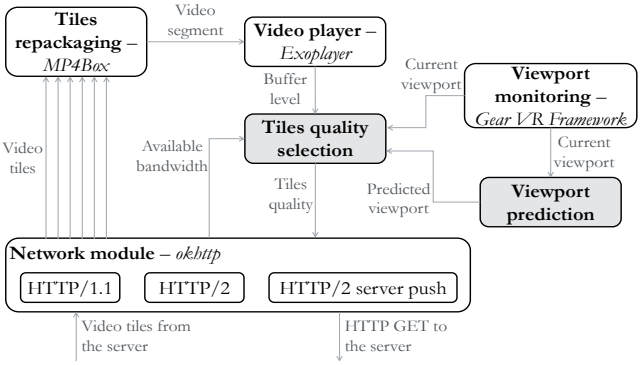


Figure 2: Illustrative diagram of the developed prototype. Gray boxes indicate algorithmic components. In italics, the names of the used libraries.

4 PERFORMANCE EVALUATION

4.1 Experimental Setup

The proposed framework has been implemented as a prototype on a Samsung Galaxy S7 and a Gear VR [12]. A high-level description of the prototype is given in Figure 2. The Gear VR Framework[2] allows to develop VR applications on Android devices and provides general VR functionalities. The framework is mainly used in the *Viewport monitoring* module, as it allows to capture where the user is watching and to enable viewport-awareness. The *Tiles quality selection* module selects the quality of the tiles and takes as input: (i) the buffer level, (ii) the available bandwidth, (iii) the current viewport and (iv) the predicted viewport, computed by the *Viewport prediction* module. The tiles quality is then communicated to the *Network module*, implemented using the okhttp[3] library, which takes care of the actual streaming of the video segments. Both regular HTTP/1.1 and HTTP/2 protocols are supported. We also extended the okhttp library to support the server push functionality of HTTP/2. Once the tiles are downloaded from the server, the *Tiles repackaging* module, realized using the MP4Box[4] library, pre-processes them before they are actually played by the *Video player*. This step is necessary because the ExoPlayer[5], which implements the video playout, is not able to directly play tiled videos. Particularly, tiles are concatenated into a single mp4 file using the *cat* command, and the raw HEVC stream is extracted using the *raw* command. Future versions of the ExoPlayer would allow to eliminate this step. Despite this process, the latency added to the system is less than 100 ms.

The Jetty server has been used as HTTP server, which was extended to implement the k-push functionality [19]. In our framework, k corresponds to the number of tiles the video is composed of. The Samsung S7 is connected to the Jetty server, hosted on a MacBook Pro Retina, via a 5GHz ad-hoc wireless network. The wireless network presents an average RTT in idle conditions of about 50ms (±33ms).

[2]https://resources.samsungdevelopers.com/Gear_VR/020_GearVR_Framework_Project
[3]http://square.github.io/okhttp/
[4]https://gpac.wp.mines-telecom.fr/mp4box/
[5]https://developer.android.com/guide/topics/media/exoplayer.html

Table 1: VR video characteristics. The nominal average bit-rate and its standard-deviation (in brackets) is reported. All values are expressed in Mbps.

	Tiled			*Non-tiled*		
	1 sec	2 sec	4 sec	1 sec	2 sec	4 sec
High	9.5(1.34)	7.1(1.18)	5.6(1.05)	8.9(1.61)	6.7(.92)	5.2(1.05)
Medium	4.8(0.45)	3.2(0.34)	2.3(0.32)	4.3(0.61)	2.9(0.34)	2.1(0.32)
Low	2.5(0.19)	1.6(0.14)	1.1(0.11)	2.3(0.26)	1.4(0.13)	0.9(0.10)

The 60 seconds *Alba 360° Timelapse*, available on YouTube, is used as video content. The raw 8K video was extracted from the original clip and re-encoded using the HM encoder (version 16.4), the reference software for H.265. Three quality levels have been encoded in variable bit-rate, corresponding to QP values equal to 30, 25 and 20. The video is available in a 1, 2, 4 seconds segment version, both non-tiled and tiled. Using shorter segments increases the adaptability to viewport and bandwidth changes, at the cost of an encoding overhead due to more frequent Instantaneous Decoding Refresh (IDR) frames at the beginning of each segment. Table 1 reports the bit-rates of the different video versions. As expected, tiling the video introduces an overhead compared to the non-tiled version, which varies between 6% and 22%.

To provide an extensive benchmark of the proposed framework (referred as to *H2P pred* in the results section), we compare its performance with that obtained using a non-tiled solution (called *Non-tiled* in the results section). Moreover, we also tested the performance of a tiled solution over HTTP/1.1 and HTTP/2 server push (referred as to *H1* and *H2P*, respectively), without the viewport prediction presented in Section 3.2. This way, it is possible to clearly identify both the gains of the prediction algorithm and those brought by server push.

To assess the performance of the viewport prediction, we recorded 10 different viewport traces from real users using our prototype, and artificially injected them during the experiments. We asked 10 users to watch the full high quality, non-tiled version of the *Alba 360° Timelapse* on the developed prototype, and recorded the viewport positions. The traces are divided in two groups, *slow* and *fast*, representing the cases where the movements are rare and slow or frequent and fast, respectively. Particularly, we characterize as *slow* the traces whose average angular speed is less than 90 deg/sec and *fast* otherwise. The tiling structure used in this paper is composed of 2 polar tiles and 4 equatorial tiles (Figure 1), each covering 90 degrees of the 360-degree video. This aspect entails that in the *slow* traces group, viewport tiles change at the same timescale as the segment duration of the video. Therefore, even non-predictive approaches should be able to adapt the tiles quality fast enough to accommodate viewport changes.

The VR client is equipped with a video buffer whose critical threshold is 2 seconds, i.e., a new segment request is issued only when the video buffer drops below 2 seconds. This choice represents a good trade-off between avoiding buffer starvations and providing a quick response to viewport changes. Each configuration in terms of segment duration, VR solution, viewport and network configuration has been repeated 10 times to guarantee statistical significance.

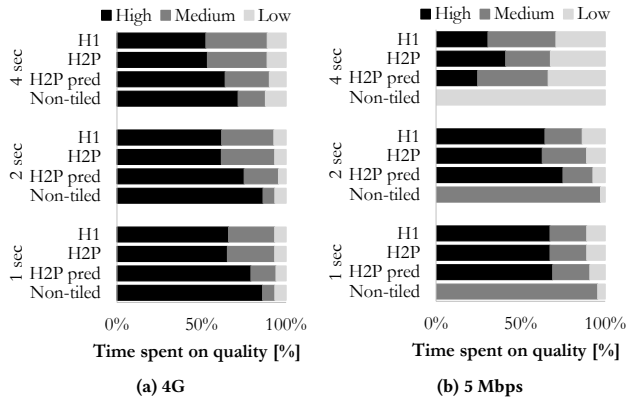


Figure 3: Viewport prediction is generally able to increase the time spent on the highest quality. Tiled solutions reach similar or better performance than the non-tiled one, when the bandwidth decreases.

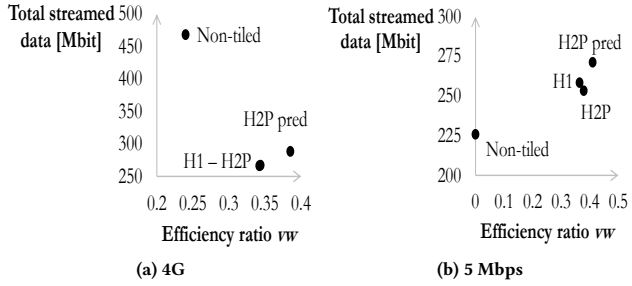


Figure 4: Tiled approaches have a higher efficiency compared to the non-tiled one, as only the viewport is streamed at the highest quality. Results are reported for the 2 seconds segments video.

4.2 Impact of Tiling and Prediction

In this section, we investigate the performance of the proposed approach. We consider two different network scenarios. In the first one, we varied the available bandwidth based on traces collected on a real 4G network [16], to assess the performance of the proposed framework under realistic network conditions. The traces present an average bandwidth equal to 21.8 Mbps (±12.3 Mbps), which is often enough to stream the highest quality. In the second scenario, the bandwidth is fixed to 5 Mbps, to clearly highlight the benefits of the proposed approach, as a classical non-tiled approach will not be able to stream the highest quality (see Table 1).

Results for the slow viewport traces are presented in Figure 3. The graphs report the percentage of time spent by the viewport on the three available quality levels, for each segment duration (1, 2, 4 seconds). The time spent on the highest quality has always to be maximized, to guarantee the best immersion to the user watching the VR video. When the viewport slowly changes, differences between a tiled and non-tiled approach are small (Figure 3a). Viewport prediction (*H2P pred*) can increase the quality by about 15% compared to tiled approaches without prediction (*H1* and *H2P*). Consequently, the difference in terms of quality spent on the highest quality is very small (about 10%) compared to a non-tiled solution. The gains of tiling approaches are evident when bandwidth is limited (Figure 3b), as the non-tiled approach cannot stream the highest quality. Conversely, tiled approaches can successfully stream the highest quality of the video, up to 70% of the time in the 1 second segments case (Figure 3b). Results for the 4 seconds segments are caused by the slightly increased amount of data needed to stream the video when prediction is used. In this case, both viewport and predicted tiles are requested at higher qualities to minimize quality transitions when the viewport changes. This aspect entails that more data is needed to stream the video (about 10%) compared to tiled non-predictive approaches. Therefore, when bandwidth is limited and segment size is bigger, the client tends to request the second highest quality instead of the highest one. Another important metric to consider is the amount of data needed to transfer the video. We therefore introduce an efficiency metric *vw*, which represents the ratio between the amount of data used to stream the viewport at the highest quality, and the total amount of streamed data, computed as in the following:

$$vw = \frac{\sum_{s=1}^{n_S} \sum_{t \in viewport} \overline{b}_{st}}{\sum_{s=1}^{n_S} \sum_{t=1}^{n_T} b_{st}}$$

where n_S and n_T are the number of segments and tiles the video is composed of, b_{st} is the bit-rate of tile t belonging to segment s and $\overline{b}_{st}$ is the bit-rate of the highest quality if tile t is at the highest quality, or zero otherwise. This metric quantifies how much bandwidth is wasted to stream tiles that are either at lower qualities or outside the viewport. Figure 4 shows the values of *vw*, for the 2 seconds segments video. The x-axis reports the *vw* ratio, while the y-axis the total amount of streamed data, to have an absolute scale to compare the different solutions. Tiled approaches reach a better efficiency when compared to the non-tiled one. Less data is needed, as only the viewport is streamed at the highest quality. Particularly, our solution is able to increase efficiency from 25% to almost 40% (Figure 4a). Despite the overhead introduced by tiling the video (see Table 1), our tiled solution uses 35% less data to stream the video than a non-tiled one. This condition entails that our solution can better redistribute the data needed to stream the video, by giving more importance to the portion of the video actually watched by the user. When the bandwidth is fixed to 5 Mbps, the efficiency of the non-tiled solution drops to zero, as the highest quality cannot be streamed.

A similar analysis can be repeated for the fast viewport (Figure 5). In this case, tiling the video reduces the amount of time spent on the highest quality. If the user viewport is moving fast, the optimal choice to provide the best immersion would be to download the entire video at the highest quality. In the 4G network configuration, the non-predictive tiling approaches can provide only 25% of the time at the highest quality, compared to 90% of the non-tiled solution (Figure 5a). Our predictive approach can consistently reduce this difference to about 30%. These results clearly show the importance of viewport prediction when the VR video is tiled. Despite these results, the non-tiled approach cannot provide the best QoE when the bandwidth is limited (Figure 5b). As for the slow viewport

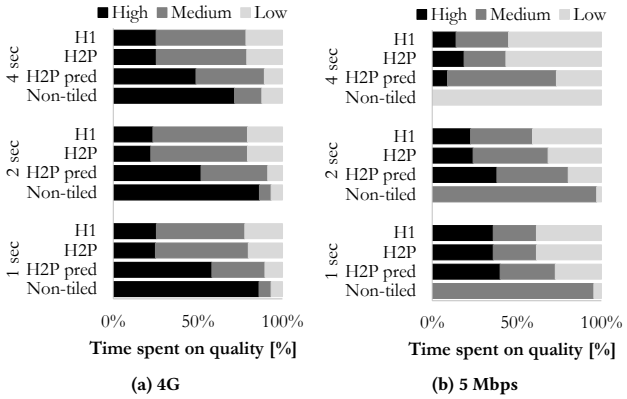


Figure 5: When the viewport is highly dynamic, a tiled approach cannot reach the same quality as the non-tiled one. Despite that, when the bandwidth drops, tiling outperforms a traditional approach.

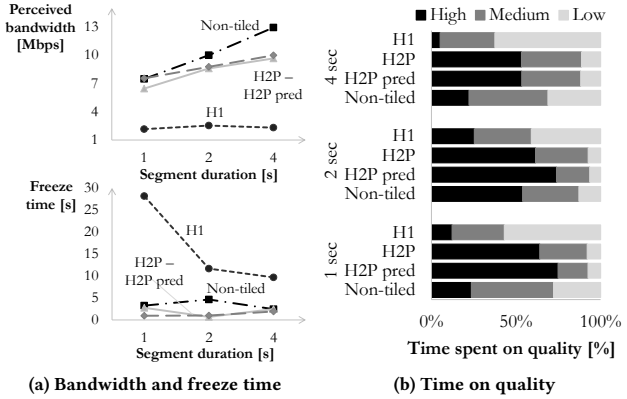


Figure 6: HTTP/1.1 cannot provide good performance when the RTT is high, as each tile has to be retrieved independently. HTTP/2 push can completely eliminate this problem.

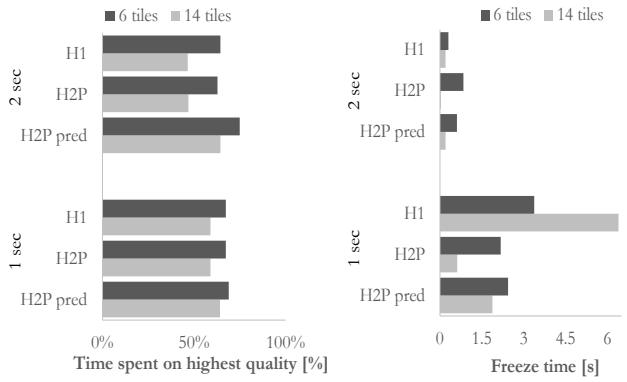
traces, streaming only the tiles watched by the user at the highest quality consistently reduces the bandwidth required to stream the VR video. Also the efficiency metric *vw* drops when the viewport is highly dynamic, as it becomes more difficult to present the right portion of the video at the highest quality. Compared to the slow viewport case, the efficiency for tiled solutions drops by about 25%. Using prediction allows to limit this drop to only 15%. Nonetheless, also in the fast viewport scenario, tiled approaches require far less data (up to 35%) compared to the non-tiled solution. Moreover, tiled approaches can still provide a better efficiency than the non-tiled one in the 5 Mbps scenario, as in this case the latter is never able to stream the highest quality.

4.3 Impact of HTTP/2 Server Push

The aim of this section is to highlight the advantages of HTTP/2 server push compared to traditional HTTP/1.1. As explained in Section 3.3, the proposed approach only needs to send a single GET request specifying the tiles qualities, which are then pushed automatically by the server. This solution is particularly beneficial when the total time to retrieve each tile individually, as with standard HTTP/1.1, is comparable to the segment duration. This problem can arise, for example, when the network RTT is high. We analyze this scenario in the remainder of this section.

To better isolate the impact of the RTT only, we increase the RTT of the wireless network connecting the server and the client to 100 ms and fix the bandwidth to 35 Mbps. Moreover, the slow viewport traces are used. Results of these experiments are presented in Figure 6. A high RTT has a negative influence on the perceived bandwidth of the HTTP/1.1 tiled approach (Figure 6a), as an RTT is lost to retrieve each single tile. This behavior has a direct impact on the total freeze time, which is extremely high in the HTTP/1.1 case (Figure 6a), for all video configurations. Pushing the tiles is extremely beneficial in this scenario. Particularly, we can still obtain similar performance to a non-tiled approach (Figure 6a), while keeping all the advantages described in Section 4.2. The perceived bandwidth increases by more than 3 times compared to HTTP/1.1, which allows to consistently reduce video freezes. From a network point of view, retrieving a single non-tiled video segment is the same as pushing the tiles using HTTP/2, as only one request has to be sent in both cases. As expected, the perceived bandwidth increases with the segment duration, as the impact of the RTT on the total download time diminishes. These results have a direct impact on the viewport quality (Figure 6b). Due to the low perceived bandwidth, in HTTP/1.1 most of the time is spent on the lowest quality. HTTP/2 solutions are instead able to provide a good viewport quality. The quality drop in the non-tiled case is mainly due to the varying bandwidth caused by a high RTT. As stated in Section 4.1, the wireless network presents an average RTT of about 50ms (±33ms). Increasing the RTT to 100 ms causes the effective available bandwidth to fluctuate. As explained in Section 4.2, a non-tiled approach needs a higher bandwidth to stream the same quality as for tiled approaches, and is therefore more susceptible to varying bandwidth conditions.

Another situation where retrieving each tile individually instead of pushing them can have a negative impact on the overall streaming performance, is when the number of tiles increases. In all the previous experiments, we used a 6 tiles configuration, shown in Figure 1, which is composed of 2 polar tiles and 4 equatorial tiles. We re-encoded the content in order to have 14 tiles in total, 12 equatorial and 2 polar. Increasing the number of tiles provides a better granularity in terms of viewport quality adaptation, as it is possible to better match the portion of the video actually watched by the user with the video tiles, but has a negative impact in terms of network overhead. Figure 7 reports the results of this experiment, for network bandwidth fixed to 5 Mbps and 1 and 2 seconds segments video. As in the previous set of experiments, the slow viewport traces have been used. In the HTTP/1.1 case, increasing the number of tiles to 14 causes the bandwidth to drop by about 13% compared to server push, due to the increased idle time between subsequent HTTP GET requests. Despite this drop, bandwidth is sufficiently high to provide both a good overall quality and few freezes in the 2



(a) Time on highest quality **(b) Freeze time**

Figure 7: In the 14 tiles video, HTTP/1.1 results in high freeze time when the idle time due to the increased number of GET requests approaches the segment duration (i.e., 1 second segments video).

seconds segments video configuration (Figures 7a and 7b). In the 1 second segment version instead, the decreased bandwidth and the increased idle time introduced by the subsequent GET requests, causes the HTTP/1.1 client to freeze (Figure 7b). HTTP/2 push solutions are much less affected by the increased number of tiles, which is actually beneficial in terms of total freeze time, compared to the 6 tiles video. This behavior can be explained by looking at the total amount of streamed data, which decreases with about 15% and 5% on average, in the 2 seconds and 1 second segments video. In the 14 tiles scenario, it is possible to better match the user viewport with the available tiles and stream a smaller portion of the video at the highest quality, compared to the 6 tiles video. This aspect also entails a disadvantage in terms of time spent on the highest quality (Figure 7a), since it is more likely for the user to watch parts of the video at lower qualities in case of viewport changes. Predicting the user viewport becomes even more prominent in this scenario, as it allows to limit the drop from 15%-20% in the non-predictive cases to only 5%-10%.

5 CONCLUSIONS

In this paper, we presented a novel framework for the efficient streaming of VR videos over the Internet, which aims to reduce the high bandwidth requirements and storage costs of current VR streaming solutions. In our framework the video is spatially divided in tiles using H.265, and only tiles belonging to the user viewport are streamed at the highest quality. A viewport prediction algorithm has been proposed to anticipate the user's movements and download in advance the part of the video that is likely going to be watched in the future. To reduce the influence of the network RTT on tiled streaming, our framework uses the server push functionality of the HTTP/2 protocol. In the evaluated streaming scenarios and in presence of slow viewport movements, our framework is able to obtain similar quality as for a non-tiled solution, by using up to 35% less data to stream the video. The gains brought by the proposed approach represent an important step toward the efficient streaming of VR videos with consistent quality.

Future work will focus on a more extensive comparison of the proposed approach with existing VR tiling solutions. Moreover, we will investigate the applicability of video quality metrics, similarly to the approach used by Alface et al. [14], to quantify the impact of tiling on user perception. To this extent, a subjective user study will be carried out to further analyze the trade-off between bandwidth savings and user experience in tiled VR streaming.

REFERENCES

[1] M. Budagavi, J. Furton, G. Jin, A. Saxena, J. Wilkinson, and A. Dickerson. 2015. 360 degrees video coding using region adaptive smoothing. In *2015 IEEE International Conference on Image Processing (ICIP)*. 750–754. https://doi.org/10.1109/ICIP.2015.7350899

[2] Wael Cherif, Youenn Fablet, Eric Nassor, Jonathan Taquet, and Yuki Fujimori. 2015. DASH Fast Start Using HTTP/2. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '15)*. ACM, New York, NY, USA, 25–30. https://doi.org/10.1145/2736084.2736088

[3] Eduardo Cuervo and David Chu. 2016. Poster: Mobile Virtual Reality for Head-mounted Displays With Interactive Streaming Video and Likelihood-based Foveation. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services Companion (MobiSys '16 Companion)*. ACM, New York, NY, USA, 130–130. https://doi.org/10.1145/2938559.2938608

[4] Lucia D'Acunto, Jorrit van den Berg, Emmanuel Thomas, and Omar Niamut. 2016. Using MPEG DASH SRD for Zoomable and Navigable Video. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, New York, NY, USA, Article 34, 4 pages. https://doi.org/10.1145/2910017.2910634

[5] Facebook. [n. d.]. Next-generation video encoding techniques for 360 video and VR. https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/. ([n. d.]).

[6] V. R. Gaddam, M. Riegler, R. Eg, C. Griwodz, and P. Halvorsen. 2016. Tiling in Interactive Panoramic Video: Approaches and Evaluation. *IEEE Transactions on Multimedia* 18, 9 (Sept 2016), 1819–1831. https://doi.org/10.1109/TMM.2016.2586304

[7] Mohammad Hosseini and Viswanathan Swaminathan. 2016. Adaptive 360 VR Video Streaming: Divide and Conquer!. In *Proceedings of the IEEE International Symposium on Multimedia (IEEE ISM 2016)*. 6.

[8] Jean Le Feuvre and Cyril Concolato. 2016. Tiled-based Adaptive Streaming Using MPEG-DASH. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, New York, NY, USA, Article 41, 3 pages. https://doi.org/10.1145/2910017.2910641

[9] S. Y. Lim, J. M. Seok, J. Seo, and T. G. Kim. 2015. Tiled panoramic video transmission system based on MPEG-DASH. In *2015 International Conference on Information and Communication Technology Convergence (ICTC)*. 719–721. https://doi.org/10.1109/ICTC.2015.7354646

[10] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou. 2013. An Overview of Tiles in HEVC. *IEEE Journal of Selected Topics in Signal Processing* 7, 6 (Dec 2013), 969–977. https://doi.org/10.1109/JSTSP.2013.2271451

[11] MPEG-DASH. [n. d.]. Dynamic adaptive streaming over HTTP (DASH) – Part 6: DASH with server push and websockets. https://www.iso.org/standard/71072.html. ([n. d.]).

[12] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. 2017. Improving Virtual Reality Streaming Using HTTP/2. In *Proceedings of the 8th ACM on Multimedia Systems Conference (MMSys'17)*. ACM, New York, NY, USA, 225–228. https://doi.org/10.1145/3083187.3083224

[13] 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 (ATC '16)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/2980055.2980056

[14] Patrice Rondao Alface, Jean-Francois Macq, and Nico Verzijp. 2012. Interactive Omnidirectional Video Delivery: A Bandwidth-Effective Approach. *Bell Labs Technical Journal* 16, 4 (2012), 135–147. https://doi.org/10.1002/bltj.20538

[15] A. TaghaviNasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash. 2017. Adaptive 360-degree video streaming using layered video coding. In *2017 IEEE Virtual Reality (VR)*. 347–348.

[16] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters* 20, 11 (Nov 2016), 2177–2180. https://doi.org/10.1109/LCOMM.2016.2601087

[17] Hui Wang, Mun Choon Chan, and Wei Tsang Ooi. 2015. Wireless Multicast for Zoomable Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 12, 1, Article 5 (Aug. 2015), 23 pages. https://doi.org/10.1145/2801123

[18] Sheng Wei and Viswanathan Swaminathan. 2014. Low Latency Live Video Streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop (NOSSDAV '14)*. ACM, New York, NY, USA, Article 37, 6 pages. https://doi.org/10.1145/2578260.2578277

[19] Mengbai Xiao, Viswanathan Swaminathan, Sheng Wei, and Songqing Chen. 2016. DASH2M: Exploring HTTP/2 for Internet Streaming to Mobile Devices. In *Proceedings of the 2016 ACM on Multimedia Conference (MM '16)*. ACM, New York, NY, USA, 22–31. https://doi.org/10.1145/2964284.2964313

[20] 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 (MM '16)*. ACM, New York, NY, USA, 601–605. https://doi.org/10.1145/2964284.2967292