# Efficient Live and on-Demand Tiled HEVC 360 VR Video Streaming

Mattis Jeppsson[1], Håvard N. Espeland[1], Tomas Kupka[1], Ragnar Langseth[1], Andreas Petlund[1], Peng Qiaoqiao[2], Chuansong Xue[2], Konstantin Pogorelov[4,5], Michael Riegler[3,5], Dag Johansen[5], Carsten Griwodz[5], Pål Halvorsen[1,3,5]

[1]*ForzaSys AS, Norway* [2]*Huawei Technologies, China*
[3]*Simula Metropolitan Center for Digital Engineering, Norway* [4]*Simula Research Laboratory, Norway*
[5]*University of Oslo, Norway* [6]*UiT - The Arctic University of Tromsø, Norway*

***Abstract*—With $360^\circ$ panorama video technology becoming commonplace, the need for efficient streaming methods for such videos arises. We go beyond the existing on-demand solutions and present a *live* streaming system which strikes a trade-off between bandwidth usage and the video quality in the user's field-of-view. We have created an architecture that combines RTP and DASH to deliver $360^\circ$ VR content to a Huawei set-top-box and a Samsung Galaxy S7. Our system multiplexes a single HEVC hardware decoder to provide faster quality switching than at the traditional GOP boundaries. We demonstrate the performance and illustrate the trade-offs through real-world experiments where we can report comparable bandwidth savings to existing on-demand approaches, but with faster quality switches when the field-of-view changes.**

## I. Introduction

Virtual reality (VR) headsets like the Samsung's Gear VR, Facebook's Oculus Rift, HTC's Vive and Google's Daydream are an often used and owned equipment nowadays. Therefore, efficient delivery of panoramic videos for these devices becomes increasingly important. The well-known challenge in this respect is to provide the user with a good perceived experience while at the same time saving system resources. A common approach to this trade-off is to use tiling, by delivering the parts of the video frame included in the user's field-of-view (FoV) in high quality, while the rest of the frame is delivered in lower quality to save bandwidth. This means that each user interactively controls a "virtual camera" to create their view when moving their FoV, resulting in different streams for every user. In the area of on-demand over-the-top streaming type of applications, there are many solutions that use DASH (Dynamic Adaptive Streaming over HTTP) or similar solutions [11], [13], [22], [1], [34], aiming for optimized tile selection, rate adaptation and storage optimizations, etc. These have the potential to save a lot of bandwidth, but encoding the video in tiles comes with a streaming and storage penalty as more headers are required, the coding efficiency is reduced and more download requests must be sent. Furthermore, if the download strategy fails to download a tile for the FoV in high quality in time for playback, the perceived video quality suffers. For instance, in DASH-style streaming, quality changes are possible at segment boundaries, i.e., usually every two seconds. If the FoV moves to a lower quality tile in the middle of segment playback, the quality in the FoV drops.

Thus, there exist several approaches for tiled streaming, many presenting a small variation or improvement of another tile retrieval algorithm. However, there seems to be little other work addressing the aspect of delivering these streams *live*.

In this paper we present a system for FoV-optimized live (and on-Demand) streaming for $360^\circ$ video using tiling for bandwidth saving while keeping the users' perceived quality as good as possible. We focus here particularly one the streaming delivery part, and providing multiple contributions to the *live* support, we use a combination of RTP streaming with DASH-based pull-patching [17] delivering the tiled HEVC-encoded video to both a Samsung Galaxy S7 with Gear VR and a Huawei set-top box (STB). Furthermore, we demonstrate that we can save bandwidth with quality loss by transforming a panorama format from equirectangular via cubic to an "optimized" cubic tile layout, and we have successfully performed initial multicast experiments. Additionally, we present the concept of a *catch up decoder* as a mechanism to improve quality switching delay when the FoV changes by exploiting multiple decodings on a single hardware decoder. Finally, we present results that show bandwidth savings, a greatly reduced average switching delay and support for multicast, i.e., our experiments achieve more then 50% in most of the FoV-change scenarios, compared to streaming the full HQ panorama, and reducing the average switch latency from one second in a two-second segment length scenario to about 500 ms and 750 ms for the smartphone and IPTV scenarios, respectively.

The reminder of the paper is structured as follows. First, we give a brief overview of existing related work in section II. In section III, we present our streming system and evaluate the bandwidth savings and quality switching latency in section IV. A small discussion is provided in section V before we summarize our findings and conclude the paper in section VI.

## II. Related Work

Delivery and representation of panorama videos and a partial extraction of an FoV is an area that is currently receiving a lot of attention. Panoramas can be divided broadly into two groups, partial panoramas covering less than $360^\circ$

and complete panoramas spanning 360° around at least one axis. To allow users to explore these panoramas, pan-tilt-zoom (PTZ) operations of a virtual camera have been developed in both research [9], [8], [4], [5], [28], [30], [31], [33], [26], [39], [41], [20] and industry [3], [32]. In each of these, the active FoV covers only part of the entire panorama, e.g., following a lecturer [28], [36] or an athlete [15], [10].

The arrangement of panoramic video in memory requires a projection of the surrounding space into a flat 2D representation. Researchers have discussed a variety of layout options [29], [37], but the currently practiced options use either equirectangular projections (e.g., [15]), which can be compressed and decompressed like regular videos, or CubeMap projections (e.g., [39]), which are supported by graphics hardware.

To save bandwidth, tiling is a popular approach to deliver the FoV in high quality and the rest in low quality, or not at all. Such tiled videos can be processed on the server side, to create a single, personal video stream for each viewer [19], but this approach does not scale to a large number of concurrent viewers with individual control. To support individualized views, the tiles that cover each user's desired view must be delivered from the server, with subsequent processing occurring on the client side. Thus, the most common approach is to request the tiles on the client side. First, there are several approaches that evaluate strategies for tile download for partial panoramas [11], [12], [20], [19], [14], [18]. There are also many recent approaches that work exclusively on 360° systems [1], [2], [6], [13], [24], [34], [35]. For example, Graf et. al. [13] present a streaming system for omnidirectional video over HTTP and define various streaming strategies where bitrate savings from 40% (in a realistic scenario with real users) up to 65% (in an ideal scenario with a fixed viewport) are achieved. Moreover, a new data-driven probabilistic tile weighting approach and a new rate adaptation algorithm for mobile multicast environments are proposed in [1], and Aykut et. al. [2] proposed a method to compensate for the perceived delay in the horizontal direction in a stereoscopic telepresence scenario. A probabilistic tile-based adaptive streaming that applies a target-buffer-based control algorithm to ensure continuous playback and a probabilistic model to cope with the viewport prediction errors is described in [35]. Using a simplified theoretical model, Corbillon et. al. [6] investigated the fundamental trade-offs between spatial size of the quality-emphasized regions and the aggregate video bit-rate. To reduce the influence of network delays on tiled streaming with many requests from the client, the server push functionality of the HTTP/2 protocol together with a viewport prediction algorithm has been used [22], [23]. Moreover, Naik et. al. [21] assessed asymmetric video applied separately to the foreground and background views of omnidirectional sessions, i.e., applying asymmetric stereoscopic streaming delivery on the foreground view can save up to 41% bit rate, and using the same technique on the background view, one can save approximately up to 15% bit rate. Zhou et. al. [40] analyzed the state of Oculus 360° video streaming. Son et. al. [27] propose an FOV-based tile method using HEVC and its scalability extension, reaching more than 47% bandwidth reduction. Finally, Zare et. al. [38] describe a packing intended in mixed-resolution viewport-adaptive streaming of 6K resolution while complying with the standard HEVC 4K-decoding constraint, and they achieve a streaming bitrate saving of 32%.

In summary, there exist several approaches for tiled streaming, many presenting a small variation or improvement of another tile retrieval algorithm. However, there seems to be little other work addressing the aspect of delivering these streams live, and we therefore present a system that supports this. In addition to this, we also focus on fast quality switches as the individual FoV change.

## III. Live streaming system

Our tiled streaming system supports both on-demand and live FoV optimized delivery of 360° videos. Similar to many of the approaches listed in the previous section, the on-demand version uses DASH to manage qualities in each tile and delivery over HTTP. However, our focus in this paper is the *live* system. This can of course also be supported using DASH, but today's services using DASH typically have long delays and do not support multicast delivery. Therefore, we have used multicast-supported RTP streams where losses are handled using HTTP based repair streams [17].

We use the latest generation video codec, H.265 High Efficiency Video Coding (HEVC), that independently supports encodable and decodable tiles. The codec is supported on recent devices such as Huawei P10 and Samsung Galaxy S7, and STBs like the Huawei Q22. By utilizing the horizontal and vertical slicing features of HEVC, we can stitch tiles from multiple quality layers in the compressed domain, using a single hardware decoder to decode a composition of tiles from multiple quality layers. The tile support in HEVC was designed for increasing parallel codability and not for stitching tiles in the compressed domain, but by using only a limited set of the codec features, it is possible to obtain independently decodable tiles [25]. In this paper, we limit the number of qualities per tile to two: low (LQ) and high quality (HQ). This is a design choice to focus this work on the main objectives - optimize for what is in FoV and what is not, in contrast to adapting to varying network conditions. The system can easily be extended to support an arbitrary number of qualities per tile, e.g., gradually reducing the tile quality with the distance from the FoV as we did in [12].

To be able to combine tiles from multiple quality layers into a single panorama stitch, the decoding parameter context known as *video, picture and sequence parameter sets* (*VPS/PPS/SPS*) must be identical for all quality layers. Any standard compliant decoder can decode our bitstreams and as such, our approach is fully compatible with any mobile phone or client device supporting HEVC decoding.

### A. Server

The server demultiplexes a tiled panorama into separate streams for each tile (see figure 1). The live system primarily

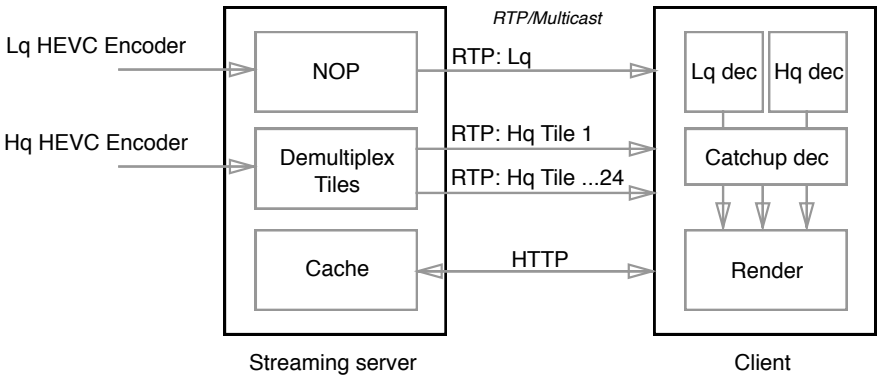



Fig. 1: System overview.

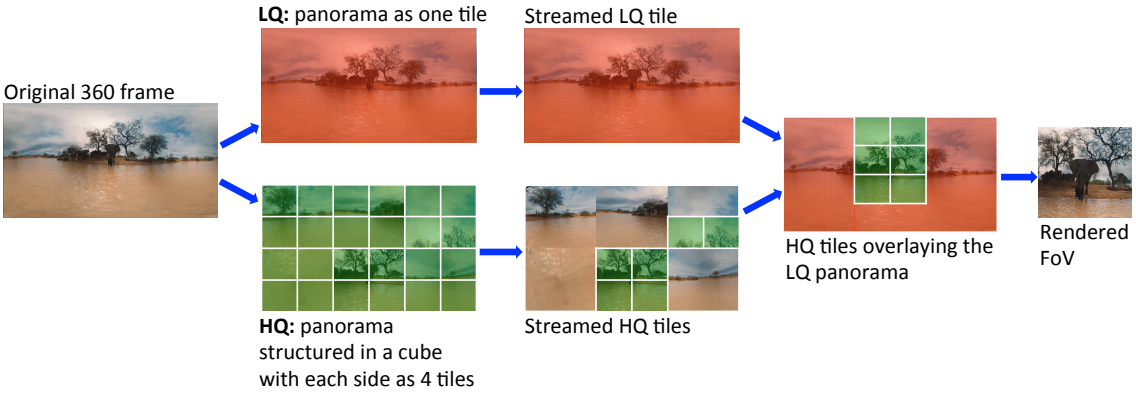



Fig. 2: Encoding and retrieval of LQ and HQ tiles

streams RTP. However, it also keeps a cache of frames which can be requested via HTTP to enable retransmits due to packet loss and to allow catch up to download HQ tiles needed for the FoV that were not received in the RTP stream, i.e., enabling faster quality switches.

*1) Encoding:* As described above, we here use two qualities. As shown in figure 2, we encode the 360 panorama into one large tile for LQ at a rate approximately 10% of the full quality panorama. For the HQ, we first organize the 360 frames in a cube where each side of the cube is tiled in a 2x2 structure.

When the FoV changes, there will be a transition period where the client does not have HQ tiles which cover the region that has just become visible. It is especially important that any motion in this region is still visible, even if the overall picture quality is reduced. In our system the LQ tile provides a fall-back mechanism for this. While HQ tiles are only downloaded if they are visible, the full LQ tile covering the entire panorama is always downloaded. Since the priority with the LQ background is to preserve motion during what should be a brief period, and in order to reduce bandwidth, the background is encoded at significantly lower bitrate than the HQ tile.

One important aspect of the encoding in a live setting is the ability to transform and encode the panorama into tiles in multiple qualities for live streaming. This is not in focus in this paper, but we have preliminary experiments setting up a large cloud instance (Amazon EC2) running the panarama transform system and the Kvazaar encoding component in real-time. Furthermore, hardware solutions from for example Harmonic[1] is able to perform such live encoding. Thus, we here focus on the live delivery.

*2) RTP:* RTP streams are initiated via RTSP. Each stream has one substream (also called a track) per tile with track 1 reserved for LQ. A client can get a list of the available tracks by issuing a DESCRIBE request. In the case that the stream is a multicast stream, the response will contain the multicast IP addresses to be used with the stream. In that case, each track will have its own multicast group. For live and VOD unicast streams, the client must go through a SETUP/PLAY workflow to start receiving RTP for a track, and similarly send PAUSE to halt it. With a multicast stream, it is enough to use IGMP to join or leave the individual tracks.

*3) HTTP:* In case of loss, ranges of frames for individual tiles can be requested from the HTTP cache by providing the stream ID, the tile ID (which coincides with the track ID of the RTP track) and starting and stopping presentation timestamps (PTS). The response body of such a range request consists of a field indicating the number of network abstraction layer (NAL) units returned, followed by the records for the individual frames which consist of PTS and number of bytes

[1]https://www.harmonicinc.com

followed by the NAL unit.

### B. Client

*1) Decoding:* Decoding of a set of tiles must always start on an I-frame. One way to reduce quality switching delays is to increase the number of I-frames, but doing so will reduce the compression rate. In our implementation, we use a GOP size of 60. A shorter GOP size will reduce quality switching delays, but will lead to worse compression rate. Similarly, increasing the GOP size will instead improve the compression rate but increase the quality switching delays. The GOP size we have selected is a trade-off between these factors. Without other mechanisms to speed up switching, this puts an upper bound on switching delay of 60 frames of at least two seconds.

A separate decoder is used for decoding the LQ background. To support faster switching of HQ tiles as the FoV changes, there are two HQ decoders, i.e., one main decoder and one *catch up decoder*. The HQ main decoder and the LQ decoder run at a fixed rate of 30 fps and are synchronised to release the next frame to rendering at the same instant.

*2) Catch up decoder:* When the FoV changes during a GOP, a *catch up* decoder, which runs in the background with rendering to the display disabled, begins decoding the new set of visible tiles starting at the previous I-frame. Any missing HQ tiles will be requested over HTTP. The catch up decoder runs at the maximum frame rate available on the hardware until it catches up with current playback, or is canceled if it was too slow and could not catch up with playback within the GOP. In case that the catch up decoder catches up to the main decoder, a decoder switch is performed with the catch up decoder and main decoder swapping roles and rendering to the display being enabled or disabled accordingly.

*3) Restricting the number of tiles:* The cubic projection used has, compared to other projections like equirectangular, much less variation in the number of visible tiles under different rotations. In order to reduce the resolution of the decoded video frames and thus increase decoding frame rate the number of HQ tiles is limited to at most six. Tiles are selected in order of their area visible in the FoV and in the case that there are more than six tiles visible some area will be covered by the LQ background. When used with horizontal FoV of 90°and aspect ratio of 16:9, the vertical FoV will be roughly 50°, and six tiles will cover the large majority of the screen for all viewing angles. Having more than 6 tiles in HQ would consume an unreasonable amount of bandwidth while yielding little or no visible improvement for the user. This strategy thus helps improve both the bandwidth savings and the quality switching delays.

*4) Tile selection and signaling:* Tile selection runs out-of-band with the decoding and rendering pipeline, and client-server signaling used to set up and tear down RTP streams for individual tiles as the FoV changes. These changes are signaled to the server via RTSP. Here, the set of FoV-visible tiles are calculated at intervals of 40 ms, and the results are communicated to both the process that manages the RTSP/RTP signaling and to the catch up decoder which may then initiate a catch up run. Additionally, the set of visible tiles is communicated to the HQ main decoder. This is done for two purposes: 1) at the start of a GOP the main decoder will select the currently visible tiles for decoding in the new GOP; and 2) in the case that a tile moved out of the FoV, unsubscribing the track, the main decoder must be notified as it will otherwise wait in vain for it to arrive on the network.

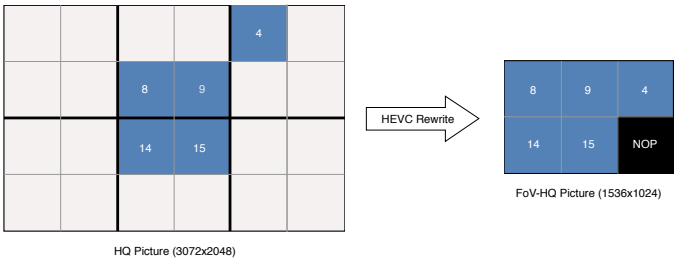



Fig. 3: Tiles from the encoded HQ picture is rewritten to a new FoV-HQ picture in the compressed domain.

*5) Bitstream rewriting:* To generate the HQ bitstream, we extract the NAL units from tiles that are within the FoV into a new bitstream. The new bitstream is decodable by a single hardware decoder and is generated on the fly by the client. Figure 3 shows the tile layout for our setup. There are 24 HQ tiles (4 tiles per face of a cube) with up to 6 HQ tiles decoded for display at any time. The HQ input resolution is 3072x2048 where each tile is 512x512 pixels in a 6x4 layout. We use the HEVC test model (HM) reference implementation running on the client [16] for doing bitstream rewriting. First, new SPS and PPS packets are generated that accommodates a FoV-HQ picture of 3x2 tiles (1536x1024). The slice headers of the individual tiles within a GOP are rewritten by updating the offset to the slice segment. A mapping of the tile position in the cube and its position in the FoV-HQ picture is created and retained for the renderer. A tile's position in FoV-HQ cannot change within the GOP since subsequent frames depend on previous frame's predictions and this is a problem if packets for a tile are lost, or if there are less than 6 tiles to decode in the FoV-HQ picture. We solve this by inserting black no-operation (NOP) tiles in the picture allowing the ordering of the tiles to stay consistent throughout the GOP.

In Figure 3, tile 4, 8, 9, 14 and 15 are within the FoV and were downloaded in HQ. The rewriter maps their bitstreams into the FoV-HQ picture and updates the slice headers. Since the FoV-HQ picture must be decodable by a standard decoder, a NOP tile is added for completeness. The tiles can be placed in any order in FoV-HQ, but the order must be retained within the GOP.

## IV. Experiments

The experiments are designed to show the characteristics of the system and in particular how well the system meets its stated design goals. To achieve this we need to show how different videos impact the bandwidth usage. An encoder will typically encode the full panorama at a fixed bitrate, but it

| | |
|---|---|
| DynamicMediumHorizontal | FoV rotating back and forth horizontally, medium speed. |
| DynamicSlowHorizontal | FoV rotating back and forth horizontally, low speed. |
| DynamicMediumVertical | Fov rotating up and down vertically, medium speed. |
| DynamicSlowVertical | Fov rotating up and down vertically, low speed. |
| StaticTop | FoV fixed looking up. |
| StaticBottom | FoV fixed looking down. |
| StaticHorizontal | FoV fixed looking straight forward at horizon. |

TABLE I: Movement patterns used in the evaluation.

| **Samsung Galaxy S7** | | | **Huawei Q22 STB** | | |
|---|---|---|---|---|---|
| # Decoders | 1920x1080 | 3840x2180 | # Decoders | 1536x1024 | 3840x2048 |
| 1 | 269.09 | 101.91 | 1 | 128.9 | 81.9 |
| 2 | 104.58 | 55.14 | 2 | 100.88 | 41.3 |
| 4 | 40.33 | N/A | 3 | 67.0 | 27.8 |

TABLE II: HEVC hardware decoding performance in fps per decoder instance using different resolutions.

will not enforce bitrates within tiles. It is thus possible that some tiles will have significantly higher bitrates than other tiles as long as the sums of bitrates of all tiles add up to the allowed total bitrate. Such uneven distribution of bitrate will not only affect the bandwidth which will increase when a tile with bitrate higher than average enters the FoV but will also cause varying download times when a catch up run is initiated on a set of tiles with a combined bitrate which deviates from the mean bitrate of the panorama.

### A. Experimental setup

*1) Sample videos:* The system has been evaluated using the Elephants and Rollercoaster videos [6]. The original videos are equirectangular, and we have converted them to cubic using Transform360 [7]. Elephants is filmed with a fixed camera, and most of the movement is concentrated to the horizon, in the center of the panorama. Rollercoaster is captured by a camera mounted on a rollercoaster, and there is movement in all directions throughout the clip. The two videos are meant to represent two extremes of a continuum ranging between movement concentrated to sub-regions on the one hand, and equidistributed movement on the other.

*2) FoV changes:* The user selects pan and tilt movements changing the FoV, and this ultimately decides which tiles are downloaded and decoded. The already mentioned uneven bitrate of tiles may correlate with the selected FoV to alter the transmitted bitrate, but more importantly, FoV changes cause catch-up runs which will download all frames for any missing tile starting at the latest I-frame. A movement pattern where tiles move in and out of the FoV at a high rate will cause a high number of downloads. To make experiments reproducible, we have recorded a set of movement sequences divided into static positions facing a single direction and dynamic with periodic movement in either horizontal or vertical direction at slow or medium speed. Each of the static positions has been selected to represent an average case when looking in that general direction, and to avoid sweet spots caused by the geometry of the projection. Table I describes the movement patterns used.

*3) Devices and setup:* We have implemented our proof-of-concept system as an Android application which runs on any Android device with HEVC hardware decoding. In our experiments, we use two different devices with a 16:9 display, i.e., the Huawei Q22 STB with a Hi3798i system-on-a-chip relevant for an IPTV scenario, and the Samsung Galaxy S7 with an Exynos 8890 SoC as a representative of recent smartphones (supporting the Gear VR, although that is not used here). In Table II, we present results from our experiments with the devices' decoding capabilities, and, one can observe that both devices are capable of decoding video faster than realtime (30 fps), allowing a secondary catch-up decoder to run in parallel. Furthermore, the live streaming experiment is performed using a traditional client-server setup where the server supports both RTP and HTTP stream delivery. Thus, we have performed experiments where the S7 is connected via WiFi on an office network with the presence of retransmissions giving transfer times relatively high variance, and the Q22 is connected to a wired network with RTTs below 10 ms and close to 0 packet loss.

### B. Quality switching delay

In our experiments, the quality switching delay is defined as the time it takes from a tile enters the FoV until it is covered by an HQ tile. With a GOP size of 60 frames playing at 30 fps, the average quality switching delay without catch up is around 1 second. The catch up decoder introduced in this system is an attempt to reduce these delays. Table IV shows the advantage of the catch up component with median and 95th percentile for switching latencies. On the Galaxy S7, our catch up effectively cuts the switching latencies in half. On the Q22, quality switching delays are not quite as good but still considerably better than 1 second. Figure 4 visualizes the quality switching delays for some movement patterns. Orange bars show when a tile was in FoV and blue bars show when a tile was available in HQ. It can be seen that tiles typically become available in HQ shortly after entering the FoV and switch back to LQ shortly after exiting the FoV, but in some cases the delay is longer, for example switching to bottom tiles in DynamicSlowVertical (Figure 4b). Moreover, the 95th percentile quality switching delay show that in some cases there are tiles which remain in LQ for more than a GOP, (for example DynamicSlowVertical on the STB). Such delays can happen because the main decoder has a buffer length around 400 ms. This buffering was necessitated by variations in networking and decoding delay. If a tile enters the FoV just as an I-frame is added to the buffer and there is no successful catch up run, playback must first consume the 400 ms buffer followed by a full 2 second GOP before the tile changes to HQ. The buffering also means that tiles that were in HQ and left the FoV will remain available in HQ for a short duration as indicated by the blue bars continuing beyond the orange bars in Figure 4. A catch up run will only be initiated following an FoV change if there is not already an ongoing catch up run. Because of this the switching delay will be longer for an FoV change that happens shortly after an earlier FoV change.

**Elephants**

| | Huawei Q22 | | | Samsung Galaxy S7 | | |
|---|---|---|---|---|---|---|
| | Bandwidth savings (%) | RTP (KB) | HTTP (KB) | Bandwidth savings (%) | RTP (KB) | HTTP (KB) |
| DynamicMediumHorizontal | 51.4 | 41753.2 | 6335.7 | 49.8 | 41665.2 | 6592.1 |
| StaticBottom | 68.4 | 31398.6 | 0 | 64.2 | 35081.2 | 0 |
| StaticHorizontal | 50.4 | 47260.5 | 0 | 52.6 | 47456.8 | 0 |
| DynamicSlowVertical | 58.4 | 37626.6 | 4224.5 | 62.0 | 37028.9 | 4115.3 |
| StaticTop | 64.6 | 33916.0 | 0 | 64.0 | 35140.7 | 0 |
| DynamicSlowHorizontal | 55.2 | 38932.3 | 1334.6 | 62.4 | 37593.7 | 1649.0 |
| DynamicMediumVertical | 53.6 | 39603.5 | 9638.8 | 46.8 | 39534.9 | 10382.1 |

**Rollercoaster**

| | Huawei Q22 | | | Samsung Galaxy S7 | | |
|---|---|---|---|---|---|---|
| | Bandwidth savings (%) | RTP (KB) | HTTP (KB) | Bandwidth savings (%) | RTP (KB) | HTTP (KB) |
| DynamicMediumHorizontal | 47.0 | 60096.0 | 10380.9 | 46.4 | 59940.1 | 11242.7 |
| StaticBottom | 63.2 | 48577.3 | 0 | 63.0 | 49003.7 | 0 |
| StaticHorizontal | 49.8 | 66922.2 | 0 | 50.6 | 66909.9 | 0 |
| DynamicSlowVertical | 59.2 | 48173.5 | 4577.9 | 59.8 | 48780.6 | 4489.8 |
| StaticTop | 77.4 | 30472.3 | 0 | 75.8 | 31838.6 | 0 |
| DynamicSlowHorizontal | 56.4 | 55845.0 | 3003.4 | 53.8 | 55881.2 | 2603.2 |
| DynamicMediumVertical | 49.4 | 53231.2 | 13965.9 | 51.6 | 52481.9 | 13631.4 |

TABLE III: Bandwidth savings. The full panorama bitrates are 13.3 Mb/s for Elephants and 20.1 Mb/s for Rollercoaster.

**Elephants**

| | Huawei Q22 | Samsung Galaxy S7 |
|---|---|---|
| DynamicMediumHorizontal | 643.0 (1542.0) | 449.0 (1253.5) |
| DynamicSlowVertical | 660.0 (2013.2) | 469.0 (2057.6) |
| DynamicSlowHorizontal | 828.5 (2152.0) | 490.5 (1498.0) |
| DynamicMediumVertical | 621.0 (1778.2) | 408.0 (1018.8) |

**Rollercoaster**

| | Huawei Q22 | Samsung Galaxy S7 |
|---|---|---|
| DynamicMediumHorizontal | 680.0 (1497.0) | 464.5 (1276.3) |
| DynamicSlowVertical | 658.0 (2235.5) | 460.0 (1473.5) |
| DynamicSlowHorizontal | 716.0 (1188.0) | 561.0 (1830.4) |
| DynamicMediumVertical | 641.0 (1430.0) | 420.0 (944.0) |

TABLE IV: Median quality switching delays in ms (95th percentile) incurred during 60s of video with different synthetic movement patterns.

### C. Bandwidth savings

In our experiments, we compare the sum of all RTP and HTTP traffic, including both HQ and LQ, and compare it to the size of the full HQ panorama. Thus, the bandwidth savings (Table III) are defined as the fraction of the full panorama not transferred. One major goal with tiled streaming is to save bandwidth. We can observe that the bandwidth savings are relatively stable over the different movement patterns with an approximate average of 57% compared to downloading everything in HQ. Limiting the number of tiles to 6 will overall restrict the total bandwidth with deviations being caused ether by unevenly distributed bitrate in tiles or by HTTP download caused by FoV changes. In the static cases, the FoV remains almost constant and the number of HTTP downloads is generally low. For the vertical static cases (StaticBottom and StaticTop), we see savings above 60%, largely because there is little movement in their FoV in both videos. In particular for StaticTop on Rollercoaster, we see savings over 75%. Furthermore, one notable case with lower savings is DynamicMediumVertical (46.8-53.6%). In this case, we see that the high rate of FoV change causes a high number of HTTP downloads explaining most of the discrepancy.

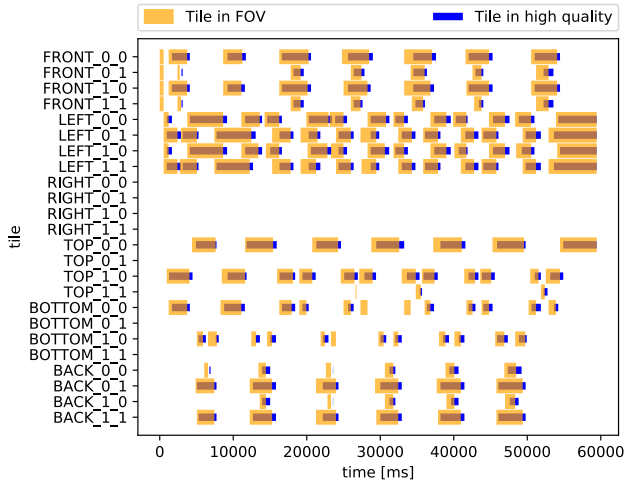
### D. Multicast streaming

The live system is also tested over multicast where our system correctly manages joining and leaving multicast groups according to a user's FoV. Over the approximately 500 km wide area network, we experienced a higher rate of packet loss, resulting in slightly more HTTP retransmits and slightly higher switching delays. Thus, our initial experiments serves as a proof of concept, but further work is needed in order to fully understand the performance of a tiled 360 multicast service.

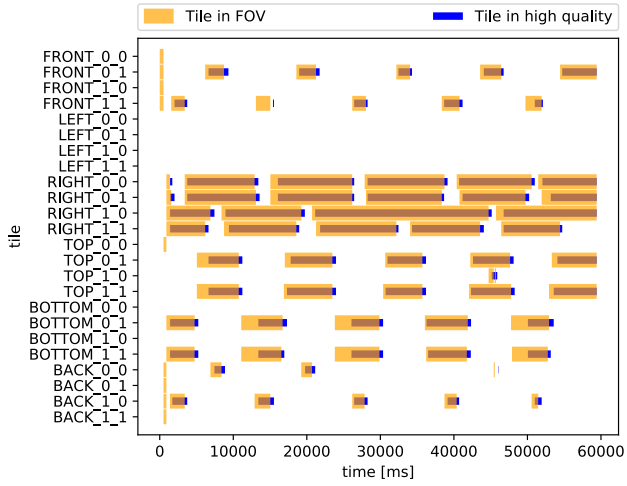
## V. Discussion

The six tile FoV puts a baseline on the bandwidth usage. With six tiles, we are downloading 25% of the tiles of the full HQ panorama, and with picking an LQ bitrate at 10% of the HQ panorama, the combined bandwidth usage adds up to 35% on average, or 65% bandwidth savings. In practice, the elephants video has a full HQ panorama bitrate of 13.3 Mbit/s with LQ at around 10% of HQ. Thus, the RTP bandwidth should end up around an average of 4.65 Mbit/s and in the experiments, we observed numbers between 3.72 and 5.45 Mbit/s. In the static cases, there is relatively little HTTP traffic, and the bandwidth savings are close to the expected 65%. Some further savings could potentially be made by increasing the number of tiles for finer granularity, but by doing so, the compression overhead and header/container overhead added by tiling may counter those savings. A further optimization could be to encode the LQ background with a longer GOP size to improve the compression rate, but since its bitrate is comparatively low, this would only give a small overall improvement and a potentially reduced quality. In the dynamic cases, we see that significant bandwidth is used to download tiles that enters the FoV but when the FoV changes at a high rate, tiles will have left the FoV before they are ready to be rendered. By better timing when catch up downloads tiles, significant savings can be made.

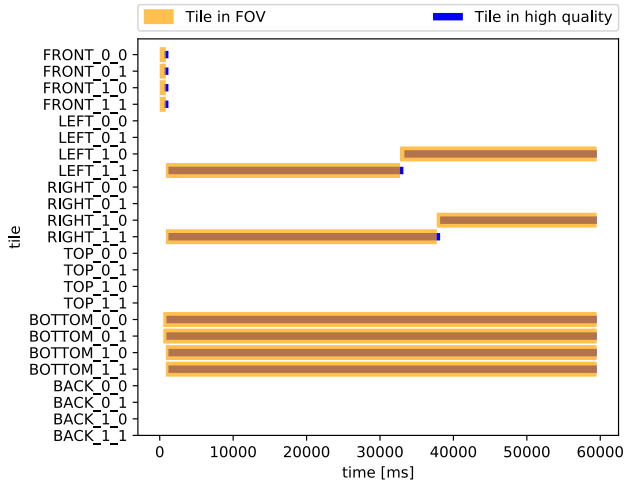
The experiments show that the catch up mechanism significantly reduces the quality switching delays, but it may still




(a) DynamicMediumHorizontal




(b) DynamicSlowVertical




(c) StaticBottom

Fig. 4: Example S7 experiments displaying tiles needed in the FoV and available HQ tiles

be noticeable. An interesting question is how much further the switching delays can be made with our approach. The catch up decoder runs in a closed loop where it sequentially and iteratively downloads, rewrites and interacts with the hardware decoder. According to measurements, we have made improvements that can be achieved by running these steps concurrently. On the Q22, it seems that slower bitstream rewriting is one source of the added delays compared to S7 and running the rewriting on a separate thread may help improve the performance. However, the ultimate limit of our approach is the decoding speed on the device, and the Q22 clock frequency is lower than the Galaxy S7 even if other overhead is removed.

Furthermore, the switching delays have significant outliers close to and above two seconds. In our implementation, the catch up decoder will start as soon as an FoV change is detected. In the event of two rapid FoV changes, the catch up decoder will already be running when the second change happens. Responding to the second change or subsequent changes until completing the first run is not possible. Thus, the delayed response to subsequent changes is increased. One way to deal with this is to only start the catch up decoder on specific frame numbers within the GOP during playback. Doing so will stop the catch up decoder from starting too early and will help cut the switching intervals more evenly.

In our PTZ system, it will be interesting to support multiple quality layers to allow further refinement, especially when increasing resolution to 8K and above. Supporting this on the client side and streaming is straightforward, but encoding of 8K panoramas cannot be made in real time on a single machine. However, the tiled H.265 encoding used in our system allows for independent encoding of tiles on multiple machines. More challenging is transforming from equirectangular to cubic in real time. Most recording systems only provide equirectangular, and this makes such conversion a requirement for live. We are not aware of any system capable of converting panoramas beyond 4K in real time. In theory, it should be possible to partition the equirectangular panorama so that different tiles on the cube can be transformed on different machines. This can be explored in future work.

## VI. Conclusion

For on-demand tiled 360 VR video streaming, a vast number of possible solutions exist. In this paper, we have addressed the challenges emerging when streaming this type of content *live* over both wired and wireless networks. For the original streams, we used RTP, but to repair for packet loss, we used HTTP pull-patching streams. Our experimental results show promising bandwidth saving potential, i.e., more than 50% in most of the FoV-change scenarios, compared to streaming the full HQ panorama. The quality catch-up solution greatly reduces the quality change latency when the FoV changes, compared to the expected one-second average using a 60 frame (2 seconds) GOP, i.e., reducing the average switch latency from one second to about 500 ms and 750 ms for the smartphone and IPTV scenarios, respectively. For future work,

there are some important directions to explore. Better live and real-time support for higher resolutions than 4K will be important, and more efficient representations and tiling of the panorama video might be important for further improvements of both bandwidth savings and quality switching latency.

## REFERENCES


[1] Hamed Ahmadi, Omar Eltobgy, and Mohamed Hefeeda. Adaptive multicast streaming of virtual reality content to mobile users. In *Proc. of Thematic Workshops of ACM MM*, pages 170–178, 2017.

[2] Tamay Aykut, Stefan Lochbrunner, Mojtaba Karimi, Burak Cizmeci, and Eckehard Steinbach. A stereoscopic vision system with delay compensation for $360^o$ remote reality. In *Proc. of Thematic Workshops of ACM MM*, pages 201–209.

[3] Camargus. Premium Stadium Video Technology Infrastructure. https://www.youtube.com/watch?v=SO32pEgCeDI.

[4] Peter Carr and Richard Hartley. Portable multi-megapixel camera with real-time recording and playback. In *Proc. of IEEE DICTA*, pages 74–80, 2009.

[5] Peter Carr, Michael Mistry, and Iain Matthews. Hybrid robotic/virtual pan-tilt-zom cameras for autonomous event recording. In *Proc. of ACM MM*, pages 193–202, 2013.

[6] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proc. of ACM MM*, pages 943–951, 2017.

[7] Facebook. Transform360. https://github.com/facebook/transform360, 2008. [Online; accessed 13-March-2018].

[8] Christoph Fehn, Christian Weissig, Ingo Feldmann, Markus Muller, Peter Eisert, Peter Kauff, and Hans Bloss. Creation of high-resolution video panoramas of sport events. In *Proc. of IEEE ISM*, pages 291 –298, December 2006.

[9] Eric Foote, Peter Carr, Patrick Lucey, Yaser Sheikh, and Iain Matthews. One-man-band: A touch screen interface for producing live multi-camera sports broadcasts. In *Proc. of ACM MM*, pages 163–172, 2013.

[10] Vamsidhar Reddy Gaddam, Ragnar Langseth, Sigurd Ljødal, Pierre Gurdjos, Vincent Charvillat, Carsten Griwodz, and Pål Halvorsen. Interactive zoom and panning from live panoramic video. In *Proc. of ACM NOSSDAV*, pages 19–24, 2014.

[11] Vamsidhar Reddy Gaddam, Hoang Bao Ngo, Ragnar Langseth, Carsten Griwodz, Dag Johansen, and Pål Halvorsen. Tiling of panorama video for interactive virtual cameras: Overheads and potential bandwidth requirement reduction. In *Proc. of Packet Video - Picture Coding Symposium (PCS)*, pages 204–209, 2015.

[12] Vamsidhar Reddy Gaddam, Michael Riegler, Ragnhild Eg, Carsten Griwodz, and Pål Halvorsen. Tiling in interactive panoramic video: Approaches and evaluation. *IEEE Transactions on Multimedia*, 18(9):1819–1831, September 2016.

[13] Mario Graf, Christian Timmerer, and Christopher Mueller. Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation. In *Proc. of MMSYS*, pages 261–271, 2017.

[14] Ravindra Guntur and Wei Tsang Ooi. On tile assignment for region-of-interest video streaming in a wireless LAN. In *Proc. of ACM NOSSDAV*, 2012.

[15] Pål Halvorsen, Simen Sægrov, Asgeir Mortensen, David K.C. Kristensen, Alexander Eichhorn, Magnus Stenhaug, Stian Dahl, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Carsten Griwodz, and Dag Johansen. Bagadus: An integrated system for arena sports analytics - a soccer case study. In *Proc. of ACM MMSys*, pages 48–59, March 2013.

[16] Fraunhofer Heinrich Hertz Institute. HEVC Test Model (HM). https://hevc.hhi.fraunhofer.de/HM-doc/, 2015. [Online; accessed 15-March-2018].

[17] Espen Jacobsen, Carsten Griwodz, and Pål Halvorsen. Pull-patching: A combination of multicast and adaptive segmented http streaming. In *Proc. of ACM MM*, pages 799–802, 2010.

[18] H. Kimata, D. Ochi, A. Kameda, H. Noto, K. Fukazawa, and A. Kojima. Mobile and multi-device interactive panorama video distribution system. In *Proc. of GCCE*, pages 574–578, Oct 2012.

[19] Feipeng Liu and Wei Tsang Ooi. Zoomable Video Playback on Mobile Devices by Selective Decoding. In *Proc. of PCM*, 2012.

[20] Aditya Mavlankar and Bernd Girod. Video streaming with interactive pan/tilt/zoom. In Marta Mrak, Mislav Grgic, and Murat Kunt, editors, *High-Quality Visual Experience*, Signals and Communication Technology, pages 431–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[21] Deepa Naik, Igor Curcio, and Henri Toukomaa. Optimized Viewport Dependent Streaming of Stereoscopic Omnidirectional Video. In *Proc. of Packet Video*, jun 2018.

[22] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. An http/2-based adaptive streaming framework for $360^o$ virtual reality videos. In *Proc. of ACM MM*, pages 306–314, 2017.

[23] Stefano Petrangeli, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. Improving virtual reality streaming using http/2. In *Proc. of MMSYS*, pages 225–228, 2017.

[24] Patrice Rondao Alface, Maarten Aerts, Donny Tytgat, Sammy Lievens, Christoph Stevens, Nico Verzijp, and Jean-Francois Macq. 16k cinematic vr streaming. In *Proc. of ACM MM*, pages 1105–1112, 2017.

[25] Y Sanchez, R Skupin, and T Schierl. Compressed domain video processing for tile based panoramic streaming using hevc. In *Proc. of ICIP*, pages 2244–2248. IEEE, 2015.

[26] Heung-Yeung Shum, King-To Ng, and Shing-Chow Chan. A virtual reality system using the concentric mosaic: construction, rendering, and data compression. *IEEE Transactions on Multimedia*, 7(1):85–95, Feb 2005.

[27] Jangwoo Son, Dongmin Jang, and Eun-Seok Ryu. Implementing Motion-Constrained Tile and Viewport Extraction for VR Streaming. In *Proc. of NOSSDAV*, jun 2018.

[28] Xinding Sun, J. Foote, D. Kimber, and B.S. Manjunath. Region of interest extraction and virtual camera control based on panoramic video capturing. *IEEE Transactions on Multimedia*, 7(5):981–990, 2005.

[29] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proc. of SIGGRAPH*, pages 251–258, 1997.

[30] Wai-Kwan Tang, Tien-Tsin Wong, and Pheng-Ann Heng. A system for real-time panorama generation and display in tele-immersive applications. *IEEE Transactions on Multimedia*, 7(2):280–292, April 2005.

[31] Wai-Kwan Tang, Tien-Tsin Wong, and Pheng-Ann Heng. A system for real-time panorama generation and display in tele-immersive applications. *IEEE Transactions on Multimedia*, 7(2):280–292, April 2005.

[32] Teamcoco. Conan 360: The New Angle on Late Night, 2014. http://teamcoco.com/360.

[33] S. Tzavidas and A.K. Katsaggelos. A multicamera setup for generating stereo panoramic video. *IEEE Transactions on Multimedia*, 7(5):880–890, Oct 2005.

[34] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. Optile: Toward optimal tiling in 360-degree video streaming. In *Proc. of ACM MM*, pages 708–716, 2017.

[35] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 360probdash: Improving qoe of 360 video streaming using tile-based http adaptive streaming. In *Proc. of ACM MM*, pages 315–323, 2017.

[36] T. Yokoi and H. Fujiyoshi. Virtual camerawork for generating lecture video from high resolution images. In *Proc. of IEEE ICME*, July 2005.

[37] Matt Yu, Haricharan Lakshman, and Bernd Girod. A Framework to Evaluate Omnidirectional Video Coding Schemes. In *Proc. of IEEE ISMAR*, pages 31–36, sep 2015.

[38] Alireza Zare, Alireza Aminlou, and Miska Hannuksela. 6K Effective Resolution with 4K HEVC Decoding Capability for OMAF-compliant 360 Video Streaming. In *Proc. of Packet Video*, jun 2018.

[39] Qiang Zhao, Liang Wan, Wei Feng, Jiawan Zhang, and Tien-Tsin Wong. Cube2video: Navigate between cubic panoramas in real-time. *IEEE Transactions on Multimedia*, 15(8):1745–1754, Dec 2013.

[40] Chao Zhou, Zhenhua Li, and Yao Liu. A measurement study of oculus 360 degree video streaming. In *Proc. of MMSYS*, pages 27–37, 2017.

[41] Goranka Zoric, Louise Barkhuus, Arvid Engström, and Elin Önnevall. Panoramic video: Design challenges and implications for content interaction. In *Proc. of EuroITV*, 2013.