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

Mattis Jeppsson, Håvard N. Espeland, Tomas Kupka, Ragnar Langseth, Andreas Petlund

ForzaSys AS, Norway

Peng Qiaoqiao, Chuansong Xue

Huawei Technologies, China

Dag Johansen

UiT - The Arctic University of Tromsø, Norway

Konstantin Pogorelov, Håkon Stensland

Simula Research Laboratory, Norway

Carsten Griwodz

University of Oslo, Norway

Michael Riegler, Pål Halvorsen

Simula Metropolitan Center for Digital Engineering, Norway

360° panorama video displayed through VR glasses or large screens offers immersive user experiences, but as such technology becomes commonplace, the need for efficient streaming methods of such high-bitrate videos arises. In this respect, the attention that 360° panorama video has received lately is huge. Many methods have already been proposed, and in this paper, we shed more light on the different trade-offs in order to save bandwidth while preserving the video quality in the user's field-of-view (FoV). Using 360° VR content delivered to a Gear VR head-mounted display with a Samsung Galaxy S7 and to a Huawei Q22 set-top-box, we have tested various tiling schemes analyzing the tile layout, the tiling and encoding overheads, mechanisms for faster quality switching beyond the DASH segment boundaries and quality selection configurations. In this paper, we present an efficient end-to-end design and real-world implementation of such a 360° streaming system. Furthermore, in addition to research an on-demand system, we als 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, and 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.

Keywords: 360° video, tiled streaming, quality switching delay, performance

1. Introduction

Today, 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. Similar, user-controlled views of panoramic video is also becoming available on set-top-boxes (STBs) like the Huawei Q22. Therefore, efficient delivery of panoramic videos to 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 [12, 14, 25, 1, 40], 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. In this respect, 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*.

Therefore, in this paper, we present a system for FoV-optimized live and on-Demand streaming for 360° video using tiling for bandwidth saving while keeping the users' perceived quality as good as possible. We have extended our work in [19], and we focus here particularly on the streaming delivery part. First, we describe our on-Demand solution before we provide multiple contributions to the *live* scenario. We use a combination of RTP streaming with DASH-based pull-patching [18] delivering the tiled HEVC-encoded video to both a Samsung Galaxy S7 with Gear VR and a Huawei 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 2. In section 3, we present our streaming systems, and we evaluate the bandwidth savings and quality switching latency in section 4. We summarize our findings and conclude the paper in section 5.

2. 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° 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 [10, 9, 4, 5, 34, 36, 37, 39, 32, 45, 47, 22] and industry [3, 38]. In each of these, the active FoV covers only part of the entire panorama, e.g., following a lecturer [34, 42] or an athlete [16, 11].

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 [35, 43], but the currently practiced options use either equirectangular projections (e.g., [16]), which can be compressed and decompressed like regular videos, or CubeMap projections (e.g., [45]), which are supported by graphics hardware.

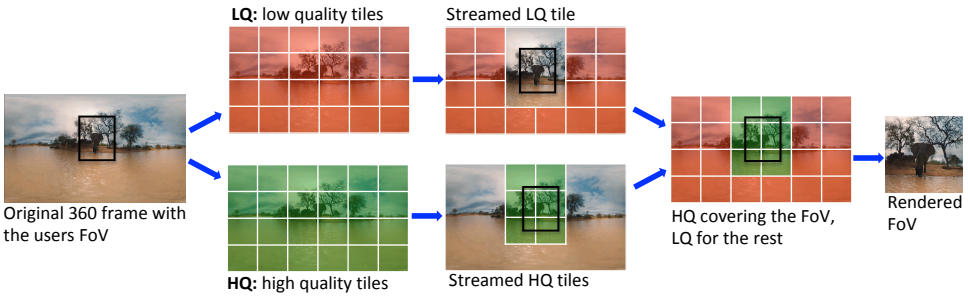
To save bandwidth, tiling is a popular approach to deliver the FoV in high quality and the rest in low quality, or not at all. Such tiled videos can be processed on the server side, to create a single, personal video stream for each viewer [21], but this approach does not scale to a large number of concurrent 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 [12, 13, 22, 21, 15, 20]. There are also many recent approaches that work exclusively on 360° systems [1, 2, 7, 14, 28, 40, 41]. For example, Graf et. al. [14] 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 [41]. Using a simplified theoretical model, Corbillon et. al. [7] inves-

tigated 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 [25, 26]. Moreover, Naik et. al. [23] 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. [46] analyzed the state of Oculus 360° video streaming. Son et. al. [33] propose an FOV-based tile method using HEVC and its scalability extension, reaching more than 47% bandwidth reduction. Finally, Zare et. al. [44] 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%.

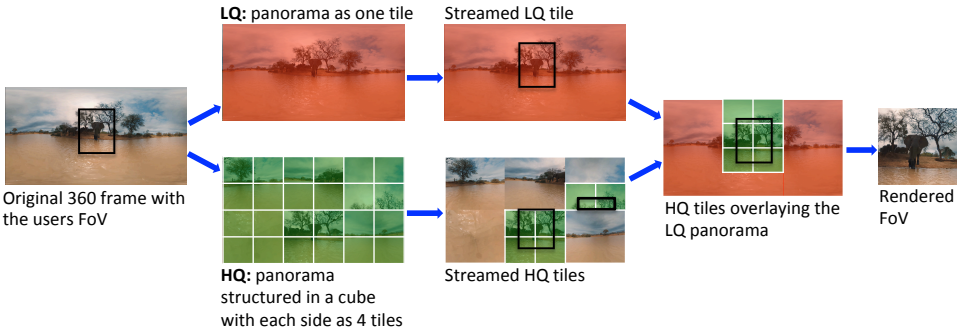
In summary, there exist several approaches for tiled streaming, many presenting a small variation or improvement of another tile retrieval algorithm. However, there seems to be little other work addressing the aspect of delivering these streams live, and the combination with on-demand, and we therefore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

3. Tiled 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. 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 [31]. 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 [13]. 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



(a) An Equirectangular panorama representation.



(b) A Cubic panorama representation

Fig. 1: Encoding and retrieval of LQ and HQ tiles according to a given FoV.

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.

Below, we present two versions of our streaming system. The first is a traditional on-Demand streaming version using DASH and a traditional *equirectangular* panorama representation streaming the different tiles as depicted in figure 1a. In our enhanced streming system, we have added support for live delivery of tiled panorama videos where we also changed the video representation to a *cubic* panorama representation as shown in figure 1b.

3.1. *On-Demand Streaming System*

HTTP adaptive streaming is the dominant technology for on-demand video streaming (VoD) in the Internet today. Adaptation is provided by segmenting the video into short, independently decodable segments that can be downloaded separately. As shown in figure 1a, clients choose quality layers based on network conditions or other preferences (e.g., conservation of bandwidth) as they see fit, and servers may be stateless web-servers or CDN nodes. We have designed and implemented an end-to-end 360 video streaming system using segmented DASH HEVC video and

playback on Android mobile phones. In this work we present techniques and trade-off considerations to achieve efficient delivery of low-latency adaptation to a user's FoV to preserve bandwidth and measure the session metrics. By utilizing client-side bitstream manipulation of the HEVC video stream, we can achieve efficient playback with a single hardware decoder supported on most modern phones.

For streaming the tiled video, we use DASH which is well defined for VoD adaptive streaming. With the recently added Spatial Relationship Descriptions (SRD) [24], the DASH manifest can specify independently decodable spatial regions, e.g., tiles, that are suitable for tiled 360° streaming. SRD is not tied specifically to HEVC tiles, and has been used for describing separate bitstreams that encompass a full frame with AVC. To support tiled HEVC streaming, the standard specifies storing tiles in separate tracks that are muxed in a fragmented MP4 - ISOBMFF (fMP4) container. A set of tracks can be stitched together to a full panorama, and for each track, segments are available in multiple qualities. Although there are many public implementations of DASH, very few support the SRD extensions and even fewer support tiled HEVC. Moreover, since the 360° video representation in DASH is work in progress, we have based our implementation on the interpretations of the standard from the GPAC library. GPAC itself is only used to generate the manifest file and the fMP4 files stored on the server.

3.1.1. *Server*

Our server is a plain nginx web-server where the different panorama tiles are DASH encoded in different qualities, and the clients can download tile-segments (the spatial regions) in qualities dependent on their FoV (figure 1a).

3.1.2. *Client*

Client application and capabilities Our client application implemented for Android on Samsung Galaxy S7 where we use the Oculus Mobile SDK for binocular rendering and sensor input natively supported on Galaxy phones. Thus, the FoV depends on the current head orientation which is calculated based on sensor information from the phone and the headsets via the Oculus SDK. Based on the orientation, a *quality selector* projects a series of sample points within the FoV to determine which tiles must be downloaded and displayed in high quality as soon as possible. The number and distribution of the FoV sample points is chosen so that all tiles within the FoV are identified. When segments are downloaded, we assembles tile segments into complete video segments containing all tiles for a segment. It then breaks the segments, which hold two seconds of a 360 degree video each, into individual frames. These are passed to the HEVC decoder for decoding and saved in a texture that is shared with Oculus Mobile SDK. The SDK renders the required part of the texture (based on the current head orientation) and displays the content.

# Decoders	480x274 (8x8)	960x540 (4x4)	1920x1080 (2x2)	3840x2180 (1x1)
1	514.72	385.96	269.09	101.91
2	213.54	170.38	104.58	55.14
4	103.87	82.49	40.33	failed
8	54.33	45.63	19.18	failed
16	27.87	22.29	failed	failed
32	failed	failed	failed	failed

Table 1: Hardware decoding performance on Galaxy S7 Exynos 8890 in fps per decoder instance using different tiling schemes.

The Samsung Galaxy S7 features an Exynos 8890 System-on-chip and has a built-in hardware HEVC decoder supporting several parallel processes, where the maximum depends on resolution. To test device performance limiting possible design schemes, we experimented using a 60-second 4k video split into up various tiling schemes. As can been seen from table 1, we were able to instantiate up to 16 decoder instances for the lowest resolution (274p) and only two instances for 2180p. Two simultaneous 2180p streams can be decoded within real-time requirements (55 fps), four simultaneous 1080p streams (40 fps) or eight 540p / 274p streams (45 / 54 fps). Thus, even though the Exynos 8890 only has a single hardware decoder, multiple video streams can be decoded in parallel on the phone within some constraints. This enables us to design schemes that can improve the performance of 360 video streaming as we will see later in the paper.

Catch up decoder - Improving quality switching latency High quality in the FoV is important, but challenging when the FoV changes. Using DASH-type of streaming, a segment quality can be changed at segment boundaries, e.g., every 2 seconds using a 2-second segment. Shorter segments are of course possible, but it affects coding efficiency. Thus, if the FoV change in the middle of a segment, lower quality tiles might be used in parts of the FoV. The average latency will be half the segment duration and the maximum the total segment duration, and a goal is to reduce the witching latency below this half segment duration. In contrast to using dependent random access points (DRAP) frames [27] or shifted instantaneous decoder refresh (IDR) frames [30] introducing additional representations, we have instead opted to trade client processing and bandwidth for improving the switching latency taking advantage of the faster-than-realtime capability of the hardware HEVC decoder on the S7, i.e., allowing time sharing of multiple simultaneous decoding contexts. A catch up decoder will try do download the required tile segment in HQ while playing back a LQ version when it enters FoV. A switching point is set at some point in the future, and when downloaded, the catchup decoder will start decoding from the segment start to reach the switching point as fast as possible. Since the catchup decoder ought to be twice as fast as the main decoder, the switching point can occur before the segment ends, and thus enable a faster switching latency compared to the end of the segment.

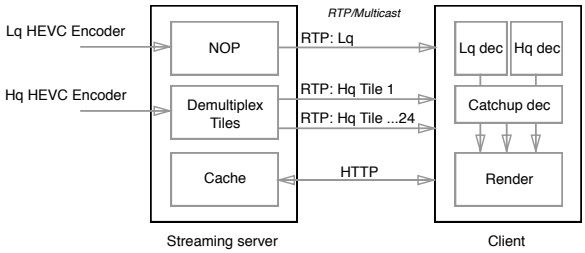


Fig. 2: System overview.

3.2. *Live and on-Demand Streaming System*

Building on the on-Demand system above, we now add support for a *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 [18].

3.2.1. *Server*

The server demultiplexes a tiled panorama into separate streams for each tile (see figure 2). The live system primarily 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.

Encoding As described above, we here use two qualities. As shown in figure 1b, 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. Furthermore, 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

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	failed	3	67.0	27.8

Table 2: HEVC hardware decoding performance in fps per decoder instance using different resolutions.

for example Harmonic[a] is able to perform such live encoding. Thus, we here focus on the live delivery.

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.

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 followed by the NAL unit.

3.2.2. *Client*

Client application and capabilities As above, we have implemented our proof-of-concept system using an Android application which runs on any Android device with HEVC hardware decoding. In table 2, we present results from our experiments with the devices' decoding capabilities using the panorama representation, 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.

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

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

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. Moreover, a separate decoder is used for decoding the LQ background. To support faster switching of HQ tiles as the FoV changes, there are as described above 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.

Catch up decoder When the FoV changes during a GOP, we again use our *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.

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.

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.

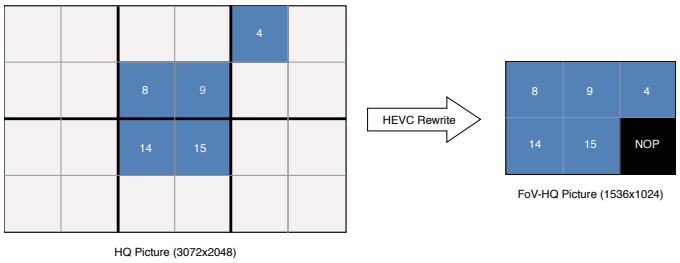


Fig. 3: HQ tiles is rewritten to a new FoV-HQ picture in the compressed domain.

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 [17] 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.

4. Experiments

Tiling by its constrained nature adds overhead to the streaming, but the benefit is that only the FoV must be transmitted in high quality. By increasing the number of tiles, the overhead increases, but the overshoot of HQ tiles outside of the FoV decreases. The experiments described in this section 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,

(a) Rollercoaster - Fast moving camera of a moving roller-coaster. Strong main focus following the rollercoaster trail.
Original bandwidth: 16075 kbps.

(b) Elephants - Fixed camera, main content along the equator line. One main azimuthal focus expected along the equator line.
Original bandwidth: 16522 kbps.

Fig. 4: Video test sequences (resolution of 3840×2048 pixels).

but it 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.

4.1. *Experimental setup*

4.1.1. *Sample videos*

To evaluate the tradeoffs with tiled streaming, we used four different videos from [6] with different content as shown in Figure 4. We have chosen to use these test videos with particular behavior since we expect the results to correlate with the type of content to some degree: We have included a fairly static video *Rhinos* with a still standing camera out in the field. There is a video *Elephants* with a static camera in the water, causing lots of motion in the picture. The video *Rollercoaster* has a moving camera causing lots of motion vectors that cannot cross the tile boundaries. A compromise between these is the *Diving* video, with fluid motion and some careful movement. The baseline for comparison is an equivalently encoded version of the videos without using tiling. We have extracted 60s clips from the same location used in the paper. Originally, these videos have a resolution of 3840×2048 pixels with a bandwidth requirement of 16.1 Mbps to 19.6 Mbps.

4.1.2. *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

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 3: Movement patterns used in the evaluation.

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 3 describes the movement patterns used.

4.1.3. *Devices and setup*

In our experiments, as mentioned above, we use the Huawei Q22 STB relevant for an IPTV scenario and the Samsung Galaxy S7 as a representative of recent smartphones. 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.

4.2. *On-Demand Scenario*

4.2.1. *Tile Encoding and Storage Costs*

There are multiple ways to encode the tile segments - trading off storage for potential bandwidth saving. The tiling overhead is introduced by the encoder constraints and represents the downside of tiled video. The measured overhead includes tiling overhead, but also DASH MP4 file overhead, which also turns out to be significant. By this, we can find the overhead compared to a naive system that would just stream the high quality untiled version. In the simplest form, two versions of the video are produced, i.e., low quality (LQ) and high quality (HQ) with a specific tiling layout. To be able to compare tiling overhead in a fair manner, we have encoded all versions of the same quality, targeting the same average Y-PSNR [29] within 1%. Since the most significant encoder constraint is restricting motion vectors to within a single

	Rollercoaster				*Elephants*			
	3x3	**5x5**	**8x8**	**9x9**	**3x3**	**5x5**	**8x8**	**9x9**
HQ (KB)	98532	105614	112718	112754	32981	33011	35282	35309
non-tiling (KB)	*98513*				*32956*			
dash % overhead	.1	.4	1.2	1.5	.4	1.5	3.8	4.9
total % overhead	.1	7.7	15.8	16.2	.5	1.6	11.2	12.4
LQ (KB)	19831	19874	22777	22824	4425	5333	7179	7230
non-tiling (KB)	*18416*				*4415*			
dash % overhead	1.0	2.8	6.2	7.9	4.6	10.5	19.8	24.9
total % overhead	8.8	10.9	31.4	33.7	4.8	33.5	94.9	104.6

Table 4: Storage requirement in KB

tile, it implies that tiling overhead depends on the level of motion in the video. As such, static cameras will typically have smaller tiling overhead than moving cameras, moving cameras higher, and so forth. It is therefore important to evaluate the system in a variety of content videos. Furthermore, for our experiments, we have used videos that have been encoded into two seconds segments (60 frames), each segment starting with an IDR frame and followed by P-frames. In addition to tile overhead, packaging the tile segments into dash compatible fMP4 tracks adds an overhead to the system, so videos that compress better get higher overhead.

The storage requirement overhead depends on the content being encoded and the tiling layout. In our videos, we can see the total tiling overhead in table 4. For example, looking at the tiling overhead for 8x8, it varies from 31.4% for Rollercoster and up to 94.9% for Elephants. The difference can be explained by the amount of motion vectors that are constrained by the tiled encoding, and the implication is that the best tiling layout may depend on the content.

4.2.2. *Bandwidth*

The main goal of using tiled streaming is to reduce the required bandwidth compared to sending the entire panorama video in high quality to the user, possibly congesting the network and overflowing the client. The reduced bandwidth strongly depends on both head movement, difference between quality layers, tile structure and layer (quality) encoding. In addition to tile overhead, packaging the tile segments into DASH-compatible fMP4 tracks adds an overhead to the system, so videos that compress better get higher overhead (in table 4). Note that this overhead is a DASH overhead and not a tiling overhead, as such, and it is a constant overhead depending on the number of tiles used. Future versions of DASH SRD might specify other containers than can support more efficient transfer of the tile segments, but this is to our knowledge not yet available although custom containers can be utilized.

In addition to the tiling overhead and the dash overhead, one must consider the overhead of transmitting tile segments that do not match expected quality, as well as tile segments that are downloaded in multiple qualities. Unexpected quality tile

	Rollercoaster				Elephants			
	3x3	5x5	8x8	9x9	3x3	5x5	8x8	9x9
DynamicMediumHorizontal	-8.0	15.2	13.2	16.1	-4.8	27.4	22.3	18.9
DynamicSlowHorizontal	21.8	36.1	36.3	36.5	25.9	42.1	39.2	36.9
DynamicMediumVertical	15.1	17.1	21.1	26.4	10.6	7.7	2.5	2.6
DynamicSlowVertical	20.0	26.2	30.8	32.9	20.7	20.7	18.5	17.4
StaticBottom	23.1	27.1	29.1	29.1	29.5	43.8	40.5	30.2
StaticTop	77.4	74.4	71.3	73.9	70.3	62.1	55.3	59.4
StaticHorizontal	46.5	40.0	45.1	46.7	31.6	31.5	33.7	30.0

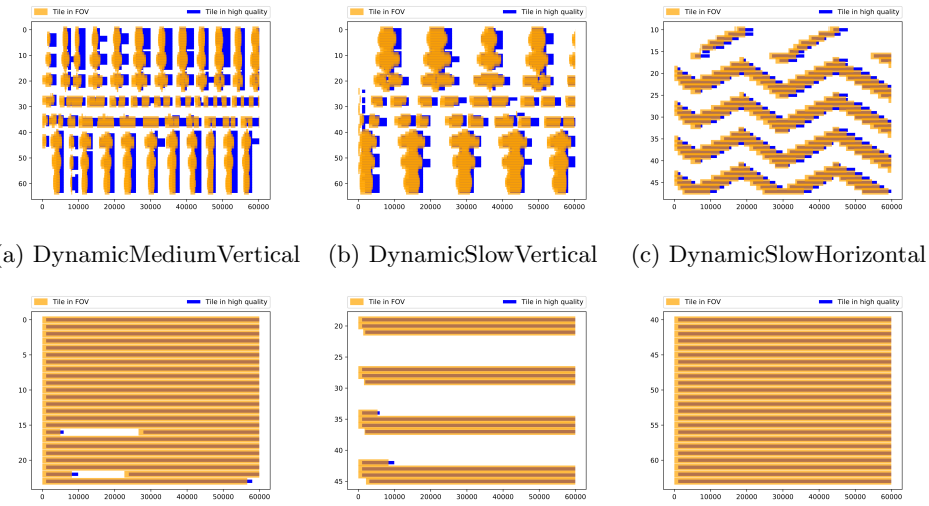
Table 5: Bandwidth savings in percent (%).

segments are sent when HQ tiles are transmitted while the FoV encapsulates the tile. Since the FoV can change quite rapidly, much faster than the segment length, it often happens that a tile is downloaded in HQ, but is displayed outside the FoV (HQ not needed). Another occurrence is when the FoV changes so a tile should be downloaded in HQ, even though it is already downloaded in LQ. To improve QoE, we re-download the tile in HQ. This is counted as double download in our system. Double downloads is a compromise between prefetching (buffering) to ensure smooth playback and bandwidth conservation. It is also required to perform intra-segment switching described earlier.

Table 5 shows the bandwidth savings for the two example videos using various tiling schemes. Depending on the tiling scheme, the FoV movement and the content, we observe a varying degree of bandwidth savings, also relating to the added tiling and coding overhead. Considering the results for the other videos in table 5, we can see that the best tile layout for bandwidth savings highly depends on content: 9x9 layout is best for the *rollercoaster*, but 5x5 performs better for *elephants*. Furthermore, a visual illustration of the quaternions evaluated over an 8x8 video with 60 second duration can be seen in figure 5. Blue bars not covered by yellow bars are tile segments downloaded and played in high quality, but at the same time out of FoV. Both switching delay (yellow before blue) is observed, but also switch out delay (blue without yellow). The latter is caused by a tile that used to be in FoV staying in HQ until the end of the segment.

4.2.3. *Switching latency*

Results from evaluating the switching latency performance with the intra-segment switching optimization can be seen in Table 6. Average switching latency for dynamic movement is in the area of 700-900 ms for all combinations. Smaller tiles achieve a slightly shorter average switching latency due to less time spent on downloading a smaller additional segment. The maximum switching latency is in the area of 1300 - 1400 ms for most dynamic cases compared to more than 2000ms which would be achieved without the intra-segment switching (download time plus switching). For the static cases presented, since the FoV does not change, there is no segment switching except for the switch from initial position to the static pose.



(a) DynamicMediumVertical (b) DynamicSlowVertical (c) DynamicSlowHorizontal

(d) StaticTop (e) StaticHorizontal (f) StaticBottom

Fig. 5: Synthetic head movement variations for 8x8. The x-axis is miliseconds. Note that the y-axes vary according which tiles that are touched by the FoV. The numbers on the y-axis represents the tile number, corresponding to the video in the same figure, starting at top left is tile 0, going to the bottom right at tile 63. Only a limited number of tiles are visible within the field of view at any time and the DynamicMediumHorizontal quaternions represent head movement horizontally in the FoV. This corresponds to a head rotation back and forth along a single axis.

4.2.4. *Discussion*

The objective of such streaming systems is to find a tradeoff between bandwidth savings, quality (blurry) transition latency and storage requirement. From experimenting with real world data instead of just simulating the environment, it is clear that these goals cannot be met in all cases, since they are highly content and context dependent. For example, varying network conditions leads to variation in tile segment download times. If we must wait 1200ms to download a complete segment on occasion, there is no way we are going to meet the 1000ms switching target. The video content also changes the tiling overhead, and this severely impacts the bandwidth savings. To give accurate results that can be expected for a production deployment, we have chosen a test environment and video dataset that tries to match reality instead of best-case results.

We have identified tile switch latency as the absolutely most important factor in producing a good tiled 360-degree streaming system. We will in the following analyze some specific results and draw an overall conclusion on this latency.

In the experiment *Elephants 8x8 DynamicMediumHorizontal*, we can observe a

Rollercoaster

	3x3	5x5	8x8	9x9
DynamicMediumHorizontal	877 (1363)	757 (1332)	714 (1362)	754 (1460)
DynamicSlowHorizontal	805 (1285)	805 (1285)	703 (1245)	804 (1288)
DynamicMediumVertical	727 (1394)	734 (1430)	713 (1412)	750 (1866)
DynamicSlowVertical	734 (1363)	760 (1304)	714 (1369)	729 (1633)
StaticBottom	926 (1091)	1080 (1080)	1101 (1101)	971 (1110)
StaticTop	1040 (1040)	1040 (1040)	1017 (1188)	1065 (1065)
StaticHorizontal	1126 (1261)	1055 (1055)	894 (1054)	1037 (1238)

Elephants

	3x3	5x5	8x8	9x9
DynamicMediumHorizontal	872 (1355)	721 (1327)	710 (1372)	743 (1365)
DynamicSlowHorizontal	811 (1300)	806 (1319)	712 (1260)	794 (1324)
DynamicMediumVertical	683 (1293)	729 (1362)	695 (1377)	718 (1915)
DynamicSlowVertical	693 (1259)	747 (1462)	719 (1353)	699 (1371)
StaticBottom	911 (1063)	1038 (1038)	1055 (1056)	957 (1072)
StaticTop	1043 (1043)	1035 (1035)	1031 (1228)	1038 (1039)
StaticHorizontal	1115 (1262)	1036 (1036)	1005 (1331)	1039 (1260)

Table 6: Average switching latency in ms (maximum) incured during 60s of video with different synthetic head movements.

22.3% total bandwidth saving compared to untiled high quality video of the same PSNR. This is not a very high saving, but it represents a 60 second continuous movement of the head (in contrast, 55.3% bandwidth is saved when the user stops to look at the horizon). The continuous movement causes some tiles within FoV to be served in LQ, but the system is able to switch to HQ within 1000ms for most of the FoV changes. As mentioned above, the switching latency is most relvant for the dynamic scenarios where we see averages below 810ms.

To give some appreciation for the experimental bandwidth savings, we sketch a comparison between what is theoretically achievable and the measured results. We do this for the Elephants video with tiling scheme of 5x5 which in the experiments had bandwidth savings ranging between 7.7% and 62.9% (Table 5). With a tiling scheme of 5x5, there are always at least 6 tiles in the FoV, that is 6 of 25 tiles are needed in HQ and 19 of 25 tiles in LQ. For the video in question, a HQ tile is on average 6.1 times larger than a LQ tile (Table 4). This implies an average best case bandwidth saving with a static view of $1-(6/25+19/25/6.1) = 63.5\%$ before tiling overhead and 53.5% worst case static average with 9 tiles visible. This seems to imply that the bandwidth saving should be 50-60% after tiling overhead.

There are two reasons for this theoretical bandwidth not being obtained: First, the observer is not static, i.e., tiles are continuously updated to match the changing FoV. Tiles that were downloaded in HQ may no longer be in the FoV at the time of playback. In addition, some tiles end up being downloaded in both HQ and LQ, what we call double downloads. This is expected and cannot be removed without reducing switching latency (disabling intrasegment switching). Second, not all tiles are equally compressible. A tile containing a moving rhinoceros requires significantly more bits to encode than a blue sky or desert floor. If movement is concentrated to

some region of the picture, then watching that region will result in lower bandwidth savings than had movement been distributed uniformly. The latter was an assumption in the theoretical calculation. The Elephants video is a good example of this. Most of the movement is concentrated to the center of the picture and viewing tiles around the center does not save much bandwidth. In comparison, tiles outside the center compress very well and viewing those instead give bigger bandwidth savings, but are unfortunately not very interesting to look at.

In general, depending on what is being viewed, the bandwidth savings vary, e.g., results range from 7.7% to 62.1% when using the Elephant video. The bandwidth savings reported are within the expected range for what is possible with this type of system. To achieve higher savings, other techniques are required. For example, it would be possible to increase the ratio of difference in bandwidth between HQ and LQ and more quality layers. Furthermore, a more efficient projection method such as Cubemap (cubic) could further improve savings.

The high fMP4 dash overhead surprised us, and is important to optimize. It is caused by the generally inefficient storage using ASCII boxes in fMP4 and depends only on the number of tiles (e.g., 9x9 tiles adds 81 files * num _ segments with all the redundancies in the fMP4 header). As such, the overhead becomes significant when the number of tiles is high and the video bitrate is medium/low. The metric was calculated by comparing the file sizes before and after fragmenting the mp4 file using MP4Box. As a side note, in later work, we built a custom container format that removed this overhead and saw a decrease in required bandwidth. However, it was not tested on the same dataset and is therefore not included in the results. The ISO committee should consider using a more efficient container format than fMP4 in their recommendation.

4.3. *Live Scenario*

In this section, we describe the results from the live streaming scenario. Again, we use the Elephants and Rollercoaster videos [7] described in section 4.1.1. The original videos are equirectangular, and we have converted them to cubic using Transform360 [8] in order to also evaluate another panorama representation. 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.

4.3.1. *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

	bw (%)	RTP (KB)	HTTP (KB)	bw (%)	RTP (KB)	HTTP (KB)
Elephants	Huawei Q22			Samsung Galaxy S7		
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		
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

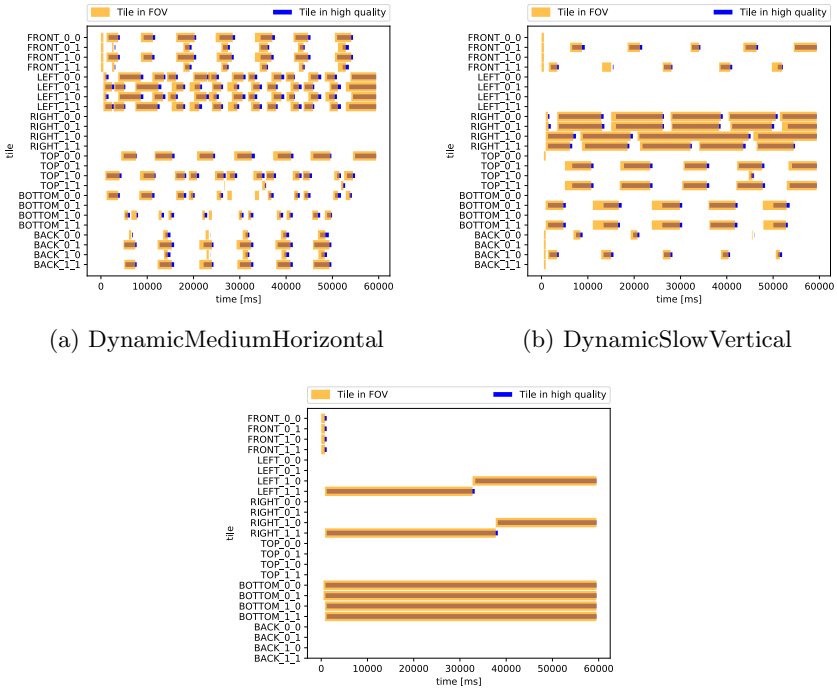
Table 7: Bandwidth (bw) savings and downloaded data using RTP and HTTP during the 60s session.

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

around 1 second. The catch up decoder introduced in this system is an attempt to reduce these delays. Table 8 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 6 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 6b). Moreover, the 95th percentile quality switching delay show that in some cases there are tiles



(a) DynamicMediumHorizontal

(b) DynamicSlowVertical

(c) StaticBottom

Fig. 6: Example S7 experiments displaying tiles needed in the FoV and available HQ 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 6. 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.

4.3.2. *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 7) 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.

4.3.3. *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.

4.3.4. *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 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.

5. 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, Francesca De Simone, and Gwendal Simon. 360 degreee video head movement dataset. In *Proc. of MMSYS*, pages 199–204, 2017.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] Mattis Jeppsson, Håvard N. Espeland, Tomas Kupka, Ragnar Langseth, Andreas Petlund, Peng Qiaoqiao, Chuansong Xue, Konstantin Pogorelov, Michael Riegler, Dag Johansen, Carsten Griwodz, and Pål Halvorsen. Efficient live and on-demand tiled hevc 360 vr video streaming. In *Proc. of ISM*, dec 2018.

[20] 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.

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

[22] 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.

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

[24] Omar A. Niamut, Emmanuel Thomas, Lucia D'Acunto, Cyril Concolato, Franck Denoual, and Seong Yong Lim. Mpeg dash srd: Spatial relationship description. In *Proc. of MMSYS*, pages 5:1–5:8, 2016.

[25] 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.

[26] 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.

[27] M. Pettersson, R. Sjöberg, and J. Samuelsson. Dependent random access point pictures in hevc. In *Proc. of IEEE ICIP*, pages 867–871, Sept 2015.

[28] 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.

[29] David Salomon. *Data Compression: The Complete Reference*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[30] Y. Sanchez, D. Podborski, C. Hellge, and T. Schierl. Shifted idr representations for low delay live dash streaming using hevc tiles. In *Proc. of IEEE ISM*, pages 87–92, Dec 2016.

[31] 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.

[32] 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.

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

[34] 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.

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

[36] 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.

[37] 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.

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

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

[40] 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.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

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

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