

FriSBE: Adaptive Bit Rate Streaming of Immersive Tiled Video

Saba Ahsan
Nokia Technologies
Finland
saba.ahsan@nokia.com

Ari Hourunranta
Nokia Technologies
Finland
ari.hourunranta@pp.inet.fi

Igor D.D. Curcio
Nokia Technologies
Finland
igor.curcio@nokia.com

Emre Aksu
Nokia Technologies
Finland
emre.aksu@nokia.com

ABSTRACT

DASH streaming for 360-degree content is gaining more traction in research and commercial areas. The usefulness of H.265/HEVC Motion Constrained Tile Sets (MCTS) and viewport-dependent streaming is recognized. Streaming high quality tiles for the viewport and lower quality tiles in the surrounding area provide considerable bandwidth saving. However, required bit rate estimation is difficult for a changing viewport with variable tiles, prompting research in development of new ABR algorithms. In this paper, we present Full Sphere Bit rate Estimation (FriSBE), an Adaptive Bit Rate (ABR) technique for tiled omnidirectional content that reuses existing ABR algorithms developed for 2D video. We implemented FriSBE in an OMAF-based DASH player and found that it is able to adapt to both head motion and changing network conditions with minimal stalls. Our experiments showed no stall events for stable conditions with or without head motion, and on average 1s stall duration for varying network conditions when bandwidth was sufficient for at least the lowest quality video, despite a short 3 second buffer.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Virtual reality**.

KEYWORDS

MPEG-DASH, Adaptive Streaming, Motion-constrained Tile Sets, OMAF, Immersive Video, 360-degree Video

ACM Reference format:

Saba Ahsan, Ari Hourunranta, Igor D.D. Curcio and Emre Aksu. 2020. FriSBE: Adaptive Bit Rate Streaming of Immersive Tiled Video. In *Proceedings of ACM Packet Video Workshop (PV'20)*. ACM, Istanbul, Turkey, 7 pages. <https://doi.org/10.1145/3386292.3397121>

1 Introduction

There is a growing interest in extended reality in research and industry with efforts on various fronts vested in the success of the technology. Moving Picture Experts Group (MPEG) has developed

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

PV'20, June 10–11, 2020, Istanbul, Turkey

© 2020 Association for Computing Machinery. 978-1-4503-7946-5/20/06...\$15.00
<https://doi.org/10.1145/3386292.3397121>

the first international Virtual Reality (VR) system standard, Omnidirectional Media Format (OMAF), a file format for storage and distribution of 360-degree video and audio [1]. MPEG-DASH is supported by the OMAF format, which allows adaptive streaming of VR content over HTTP, enabling the use of existing infrastructures and services to stream 360-degree videos.

Managing user experience and network bandwidth limitations pose a big challenge for 360-degree video streaming. VR content is often viewed on Head Mounted Display (HMD) devices, which project the images much closer to the eyes and block the real world. This makes users more prone to noticing visual defects in the media and exposes them to physical side effects, such as motion sickness. Hence, immersive video has more stringent requirements for high quality, uninterrupted content than 2D video. Furthermore, 360-degree video content size is much larger than 2D content, and a good experience calls for a spatial resolution of at least 4K or higher, leading to higher bandwidth requirements.

Viewport-dependent delivery offers respite to bandwidth requirements, where the viewport content is streamed at a higher quality than the surrounding non-viewport content. Since, at a given point of time, the non-viewport part of the content is not being watched, it does not have an impact on the user experience, except when the user turns his/her head. In such a case, updating the non-viewport content to viewport quality as early as possible is desirable. Extensive work has been done to show the bandwidth savings through Viewport-Dependent Streaming (VDS), where viewport tiles are requested at higher quality [2, 4, 5, 7, 10, 15, 16]. The primary challenge of using tiled VDS with ABR arises because the client receives only tile bit rates as part of the DASH manifest. At any time, depending on the viewport orientation, the number and size of the tiles in the viewport may differ, hence drastically changing the required bit rate for a particular viewport and non-viewport quality. This paper presents our work on developing an ABR solution for tiled 360-degree video streaming. We developed the Full Sphere Bit rate Estimation (FriSBE) process to estimate the bit rate for each viewport quality level, considering viewport orientation and variation in tile bit rates. We implemented FriSBE in the Nokia OMAF player and tested it with a fetch time based ABR algorithm [12], as well as the Buffer Occupancy Lyapunov Algorithm (BOLA) from dash.js [11]. In our implementation, the algorithms were able to avoid stall events and adapt to varying network conditions.

The remainder of this paper is organized as follows. Related work is presented in section 2. The design of FriSBE is discussed in section 3, followed by a description of our own implementation in section 4. Section 5 explains the test conditions used for evaluating

the implementation and finally the results are presented in section 6. Conclusions and future work are discussed in section 7.

2 Related Work

Adaptive streaming over HTTP is a well-established field with extensive research in the area and commercial solutions available for traditional video [14]. 360-degree video adaptive streaming is a relatively new area. Tile-based 360-degree content with viewport-dependent delivery has been found to be an effective solution for adaptive streaming with 30-70% savings in bandwidth consumption [2, 5, 7, 10, 15, 16]. In [8], the authors present an adaptation scheme, which assigns a weighted portion of the available bandwidth to each tile based on its distance from the viewport. An HTTP/2 based approach is provided in [22], with experiments using constant bit rate tiles, which simplifies required bit rate calculations for tile download. Variable video bit rates are more often used in practice, so our paper addresses the problems arising from variability in tile bit rates. Unlike previous work, we provide an OMAF implementation and a method for making 360-degree video compatible with existing ABR algorithms.

In addition to traditional DASH schemes, there is an effort to identify aspects that are unique to 360-degree video and how they can be manipulated for better adaptation. For instance, viewport prediction techniques can help reduce the latency of viewport updates observed for viewport-dependent streaming [6, 9]. Previous studies include work on using machine learning techniques for viewport prediction and tile bit rate selection [23, 24]. Head motion trajectory prediction can also help in reducing streaming data wastage by up to 25% [17]. An adaptive 360-degree solution for mobile devices with viewport prediction and ABR is proposed in [4]. The algorithm fetches a subset of tiles, incurring stalls when a tile in the visible region is missing of up to 0.96 s/min. The authors in [3] propose a variable-sized tiling scheme that balances video encoding efficiency with perceived quality for bandwidth savings; the quality of 360-degree video is influenced by the speed of the viewpoint, change in scene luminance and depth-of-field of the region. While these are important areas to explore, our work lays the groundwork for 360-degree adaptive video streaming that adapts to changing network conditions without prior knowledge about the content or viewport changes.

3 Design

Traditional ABR algorithms for 2D/flat video have a single bit rate value for the full picture in the DASH manifest, which is used by the algorithms when selecting the appropriate quality for the next segment. The DASH manifest for OMAF tiled videos provides an average bit rate per tile to the player, where the full sphere (FS) is composed of multiple tiles. In order to save bandwidth, VDS is used so that viewport tiles are downloaded at a high quality, whereas tiles that are outside the viewport (consequently not visible to the user) are downloaded at a lower quality. While FS bit rate is simply the sum of bit rates for the tiles, several factors make estimating FS bit rate from the manifest difficult. Firstly, the content complexity and the tile sizes are not always homogenous

for a 360-degree video; some tiles may have a much higher bit rate than others due to larger size or the encoder assigning more bits to it due to content complexity. Secondly, the viewport does not always align with tile-boundaries with some orientations requiring larger number of tiles than others. Therefore, the required FS bit rate can change drastically depending on the number of tiles in the viewport, their sizes, and the complexity of the content within those tiles. In Figure 1, the bit rate variation is shown for one of our test sequences using three vertical and 8 horizontal viewport orientations. It shows that slight head movements can create a large difference in the required bit rate values. Our motivation for FriSBE is to estimate a set of FS bit rates for different quality levels that provide the client an approximation of the required bit rate at a particular quality regardless of the current viewport. Our design consists of the following steps, which are discussed in detail later: (i) find the representative viewport V (ii) estimate the required FS bit rates for different qualities based on V , and (iii) use the calculated FS bit rates from the previous step in the ABR algorithm for network adaptation. Despite when using HEVC Motion Constrained Tile Sets it is possible to independently decode video tiles (possibly using multiple decoders), the FriSBE method treats all tiles of a single DASH segment collectively based on the assumption that a segment cannot be played until all tiles from that segment are available. This is an implementation¹ restriction which will be removed in the future.

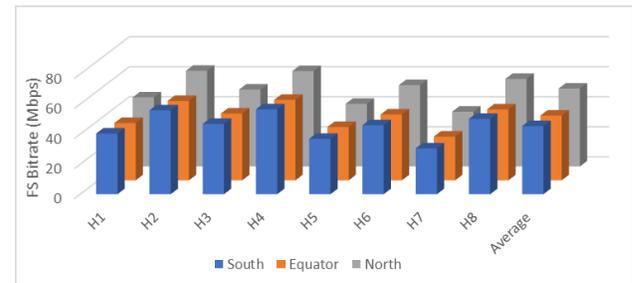


Figure 1 A chart showing variation in FS bit rate values for different viewports: 3 vertically and 8 horizontally adjacent positions. The FS bit rate is calculated using the advertised bit rate for the highest quality for viewport tiles and lowest quality for non-viewport tiles.

3.1 Representative Viewport V

As discussed previously, the FS bit rate can vary significantly based on the viewport orientation. Defining the required bit rate for several orientations for all levels is not only complicated, it also does not help in keeping a constant quality level, which is imperative for a good user experience [13]. So, we define a method for selecting a representative viewport for estimating the required bit rates. First the client identifies a multitude of viewports by moving the head orientation over the video's tile grid in granular steps, both vertically and horizontally. A viewport is selected whenever the tiles change. Once the viewports are identified, the FS bit rate is calculated using a higher quality level for the viewport tiles and a lower quality level for the non-viewport tiles. We used the highest and the lowest quality, respectively, to calculate FS bit

¹ Nokia OMAF implementation <https://github.com/nokiatech/omaf>

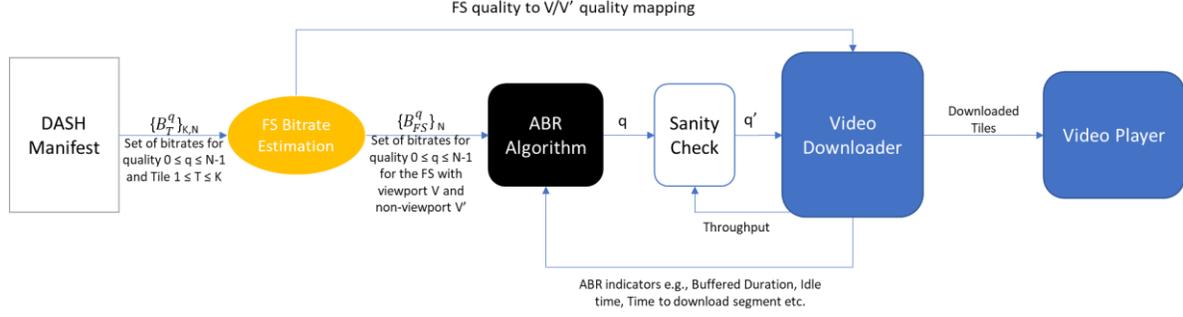


Figure 2 Flow diagram of player operation with FriSBE

rates at this stage. The viewport with the median FS bit rate was chosen as V . While we found using median to be better than using the initial viewport (azimuth and elevation are 0 for the OMAF spherical coordinates) whether using a different bit rate percentile is more efficient is left as future work.

3.2 Full sphere bit rate estimation

To compute FS bit rate we aggregate the required bit rate of all tiles at the quality at which they will be downloaded using the representative viewport. The tiles are divided in two groups; the group of tiles (partially or fully) within the viewport, V , and the group of all remaining tiles, V' . We only modify the quality for V , whereas V' is always downloaded at the lowest quality. Hence, for N quality levels (and corresponding required bit rates) advertised in the DASH manifest, we create N quality levels (and corresponding required bit rates) for FS where the required FS bit rate has a direct correlation with quality. To minimize the effects of delayed viewport update after head motion, we add a margin area to the actual viewport size of the device and use that as the viewport size. Formally, for a quality level q , the DASH manifest contains the tile bit rate B_T^q , where T is a tile that is fully or partially within the viewport area V , or it is part of the non-viewport tiles V' . Then the full sphere bit rate B_{FS}^q is:

$$B_{FS}^q = \sum_V B_T^q + \sum_{V'} B_T^0 \quad (1)$$

where q has N levels, 0 is the lowest and $N-1$ is the highest. Note that we use the term viewport/viewport area to imply the region including the margin and not just the device's viewport in the formula and also other sections of the paper, as both viewport and margin are treated equally (i.e., downloaded at the same quality). More complex schemes with variable margin sizes or higher quality for all non-viewport tiles can also be used, but require more insight into the role of margins as it may affect the relationship between quality and bit rate (a viewport with a wide margin at a lower quality may require more bit rate than one at high quality with no margin). This is left for future work.

3.3 ABR Algorithms

Having a set of FS quality levels and bit rates, the player can now use existing ABR algorithms for adaptation. Some modifications in the computation of the indicators used in ABR algorithms is

required to take into account all tiles for each segment. For instance, in our implementation, we define *buffer occupancy* as the number of segments in the buffer for which all tiles have been downloaded, and *throughput* values account for overall throughput for all tile downloads. Once the ABR has chosen the FS quality for the segment, the player uses the current viewport orientation to determine the tiles currently in viewport. Each quality for the FS is mapped to a particular V and V' quality. The player downloads the viewport tiles at quality for V and non-viewport tiles at quality for V' (lowest) based on currently chosen FS quality.

To summarize, Figure 2 illustrates the player operations described in this section. The manifest provides a $K \times N$ set of bit rates, B_T^q , for K tiles and N qualities. From this set, FriSBE creates a set of N FS bit rates, B_{FS}^q , that each represent V at quality q and V' at quality 0, which are provided to the ABR algorithm. A mapping for each FS quality and the corresponding V/V' quality is made available to the video downloader. The ABR algorithm uses the set $\{B_{FS}^q\}_N$ and the ABR indicators collected by the video downloader to choose the quality for the next segment. A further sanity check based on throughput is performed to ensure that the chosen quality by the ABR algorithm is sustainable in the current network conditions. If the sanity check passes, $q = q'$, otherwise $q > q'$. Finally, the downloader determines based on the current viewport, the tiles that qualify in V and those in V' , and downloads them according to the available quality mapping.

4 Implementation

We augmented the public sourced Nokia OMAF Player Engine with our FriSBE ABR technique. We used two ABR algorithms: a) the TIME algorithm based on work in [12], and b) the BOLA algorithm adapted from the DASH reference player [11]. Since, the scope of the paper is not to compare adaptation logic, the ABR algorithms are treated as black boxes and their detailed operation is not described in the interest of space. As mentioned previously, the indicators such as throughput and buffer occupancy consider all tiles for each segment. The TIME algorithm uses time to download as an indicator for which we use single tile downloads, but the parameters are adjusted to consider that all tiles must be downloaded within a fraction of their playout time.

For segment download the player uses a parallel segment fetching method as described in [12]. The method maintains multiple HTTP connections at the same time; one HTTP connection for each tile, i.e., one HTTP thread per tile (12 or 24 tiles for our test sequences).

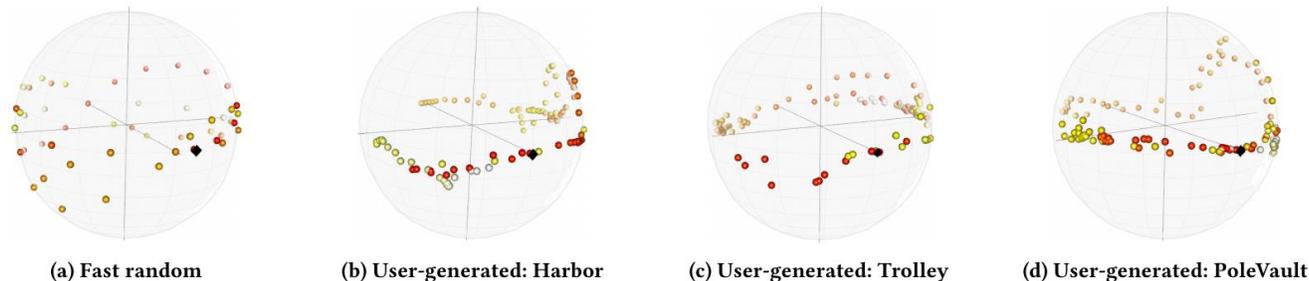


Figure 3 An illustration of the head motion; each point represents the center of viewport at a given time, the colour lightens with the passage of time changing in the order red-orange-yellow-white. The first viewport is marked with a black diamond. Note that the grid on the sphere is shown for clarity and does not match the tiling grid of the sequences.

The download is not strictly synchronized for segments; however, the HTTP thread of a tile may download up to one segment in the future if the download for a previous segment is still pending for any of the other tiles. For example, if segment i is still being downloaded for one or more tiles, the HTTP threads of the other tiles may download segment $i+1$, but not segment $i+2$; the thread must wait for all tiles to finish downloading segment i before sending a GET request for segment $i+2$. The simultaneous multiple HTTP connections for tiles that are part of the same segment can create race conditions; this is a limitation left for future work.

Finally, when the viewport changes, the player attempts to download the segments of any new tiles in the viewport at higher quality even when they are already buffered in the lowest quality. If this is done in time for the segment to be played out, the higher quality is rendered, otherwise the already buffered lower quality is rendered. We used a buffer duration of 3 seconds with a pre-buffering threshold of at least 1 second to begin playout. The short buffer was used to minimize bandwidth waste created by re-downloading viewport tiles at higher quality after head motion; the longer the buffer, the higher the number of segments that need to be downloaded again. The viewport size used was 110×110 degrees including margin area (device viewport size was 90×90 degrees).

Table 1: Test Sequences

Video	Resolution	Bit rate (Mbps)	Tiling Scheme
Trolley	7680x3840	25,20,15,10	4x3, 6x4
HarborBiking	5760x2880	35,30,25,20,10	4x3, 6x4
PoleVault	3840x2160	20,15,10,4	4x3, 6x4

5 Experiments

We evaluated the FriSBE based player using three sequences: PoleVault, Harbor Biking and Trolley [18] encoded using Kvazaar [19]. Table 1 summarizes the test sequences. Each sequence was created with two tiling schemes (4x3 and 6x4) using a single resolution, multiple quality scheme described in OMAF Annex D4.2. Smaller tiles were used in the polar regions; approximately 30 degrees high for each pole and about 120 and 60 degrees for the equator for the 4x3 and 6x4 grid respectively. All tiles had the same width. All sequences have a segment size of 566ms: a Group of Pictures (GOP) size of $16 + I$ frame at 30fps. A short segment size allowed quick viewport update after head motion. The experiments used monitor-based rendering of the viewport and the viewport information was fed using text files for the sake of automation and

reporting. However, some basic testing with HMD was conducted to validate the findings and player operation.

5.1 Head Motion

The tests were conducted with i) no head motion ii) only horizontal head motion represented with the speed of head in degrees per second (dps) iii) fast random head motion in all directions, and iv) a human generated head motion specific to the content. For the latter, we used a single test subject who explored each of the three sequences, focusing generally on interesting aspects of the video (e.g., reading texts, following the pole vault jumper, watching the approaching train etc.) while also exploring the surrounding at least once (looking towards the poles and behind). The viewport orientation over time for the three user-generated head motion files and the random fast head motion for 60 seconds is shown in Figure 3. The user generated motion has a gap at the back because a tethered HMD was used, and the user remained in the comfort zone where she did not have to readjust the cable. Also note that the points on the figure represent the centre of the viewport; the rectangular viewport can be imagined around it. The fast random head motion reaches maximum speeds of 60dps in the horizontal direction maintained over a few seconds. The human head motion has speeds of over 100dps horizontally. However, they only last for a second or less. Viewport was updated at 500ms intervals.

5.2 Network conditions

The tests were carried out in two phases. The first phase consisted of testing the sequences (duration is approximately 60s for all) under stable network conditions. Here the goal was to evaluate the performance in the presence of head-motion; therefore, the test durations were short and the network conditions were stable. We tested with no head motion, five horizontal head motion with steady speeds (5, 10, 15, 20dps), fast random and human-generated head motion. Bandwidths of 50, 35, 25 and 15 Mbps were used. Each test case was repeated 10 times for statistical significance.

In the second phase, we performed 10 minute long tests by running the sequences in a loop. In this phase we used two different varying network conditions: i) *SeeSaw*, where the bandwidth cycles between 50Mbps and 15Mbps every 30 seconds, starting at 50Mbps ii) *Slide*, where the bandwidth starts at 50Mbps and then changes every 30 seconds to 35, 20, 10, 20, 35, 50 and then repeats in that order. Since the human generated head motion does not terminate

at the initial head position, looping it would have resulted in full viewport jumps. Therefore, we used the fast random head motion for this phase of testing. In addition, we also tested for horizontal head motion (15dps) and no head motion. The 10 minute testing took significantly longer to run; hence, the testing was repeated 5 times instead of 10 as in the first phase.

5.3 Metrics

We evaluated the performance using typical adaptive streaming metrics such as stall events, stall duration, throughput, quality levels and changes. In addition, to include the 360-degree aspect, we introduced the metric for rendered viewport quality. The viewport quality is calculated at the time of rendering using the following formula, where L is the total number of tiles visible in the viewport at a given time:

$$ViewportQuality = \sum_{i=1}^L (QR[i] \cdot Coverage [i]) \quad (2)$$

QR is the Quality Ranking of the tile and Coverage is the percentage of the viewport the tile is covering. The formula is borrowed from 3GPP [20]. Since the ranking follows the OMAF specification, the highest quality has the lowest value. Hence, a low value of *ViewportQuality* indicates a better quality. Note that we used the ranking such that 1 is always the highest quality and the remaining are in uniformly descending order with a decrement of one. Since there are 5 bit rate levels for Harbor Biking (see Table 1), the lowest quality is 5, whereas it is 4 for the other two sequences. With this scheme, if the *ViewportQuality* value is not a whole number or close to a whole number, it can be deduced that the viewport is displaying tiles of two qualities at least.

6 Results

In this section, we present the results of our experiments, first under stable and then under variable network conditions.

6.1 Stable Network Conditions

Stable network conditions were used with different levels of head motion to study the effects of a changing viewport on the adaptation algorithm. The results for different metrics follow.

6.1.1 Stall Events. Stalls were generally not observed for any of the test conditions with steady horizontal head motion even at 20dps. The average stall duration for any sequence for all test cases was less than 17ms for the Time algorithm and below 2ms for BOLA. For the user-generated head motion and random head motion, the TIME algorithm showed some stalling. The total number of stall events was no more than 2, with 1 being more common. The total stall duration for any of the tests ranged from about 25ms to a little over 300ms. BOLA algorithm was more successful in avoiding stalls, with only one 25ms stall experienced for the PoleVault sequence with random fast head motion, too low to impact user experience [21].

6.1.2 Quality Variation. The average *ViewportQuality* (see Equation 2) observed for the sequences was higher for the 6x4 grid than the 4x3 grid as shown in Figure 4. Furthermore, the 6x4 grid was better at avoiding stall events as well. This is expected, since

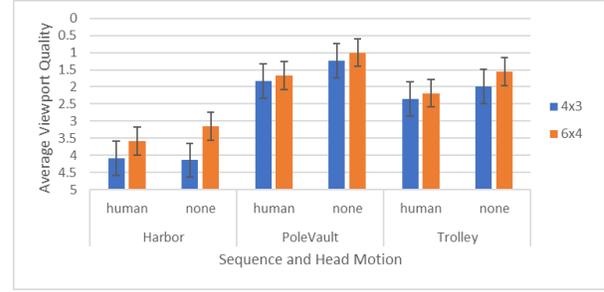
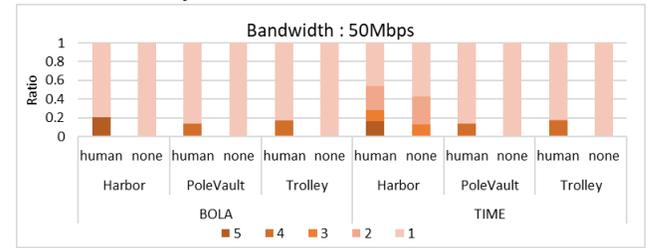


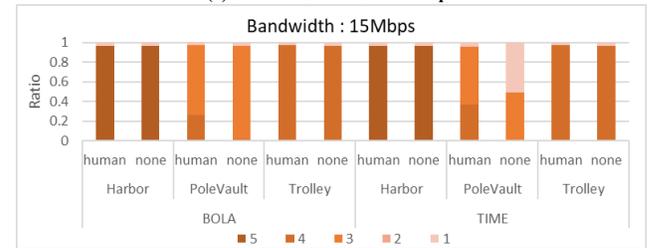
Figure 4 The 6x4 tile grid was able to not only achieve a higher quality viewport, but was able to maintain the quality during head motion better than the 4x3 grid for most cases. Results for human and no head motion (none) are shown.

the tiles are smaller and viewport changes lead to smaller overhead caused by segments download. However, the MCTS tiling implies that smaller tiles have lower encoding efficiency; so this should be considered when analysing the overall benefits.

To estimate the stability of the viewport quality, Figure 5 shows for the entire duration of the video, a stacked bar graph of the ratio of viewport tiles that were rendered at a given quality to all the viewport tiles rendered. For human head motion, the viewport is less stable than any of the horizontal head motion schemes we used.



(a) Fixed bandwidth of 50Mbps



(b) Fixed bandwidth of 15Mbps

Figure 5 Stacked graph showing ratio of the viewport tiles at different quality levels (1-5). For 50Mbps, the quality was maintained at highest level (1) for most cases. For 15Mbps, each sequence maintains a different quality.

Hence, we used that in the graphs along with no head motion for comparison. At 50Mbps, most sequences remain at the highest level of quality for most of the time. At 15Mbps, Harbor maintains the lowest quality (5) and Trolley maintains the second lowest (4) for the whole sequence. PoleVault has a lower range of required bit rates and has more alteration between QR 3 and 4. Note, that the small share of QR 1 in all 15Mbps cases is because the player always starts at the highest quality before stepping down, leading to higher startup delays (mean: 4.5s).

6.1.3 Throughput. The average throughput was calculated as the sum of the sizes of the segments downloaded over the total duration of the test. Figure 6 shows the observed throughput for the different sequences under different network bandwidths. The values are indicative of all test conditions: single viewport, horizontal, random and human head motion. We found that the bandwidth utilization factors were not as high as some 2D ABR schemes that can be found in the literature. The reason was threefold: i) the encoding bit rate levels did not always match exactly with the bandwidth, ii) the chosen quality takes into account head motion and the possibility of sudden rise in throughput, and hence is more conservative and iii) potential race conditions caused by the use of multiple HTTP connections for each tile, which can penalize the player at times. We hope to address the last of these in future work.

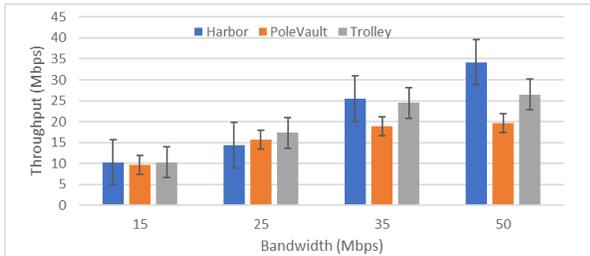


Figure 6 Average throughput for the different sequences and network bandwidth shown here with error bars.

6.2 Variable Network Conditions

The 10-minute tests show the adaptability of the algorithms to changing network conditions. Of the two configurations we used, *Slide* has one bandwidth level that is too low for two of the sequences we used, and stalls are expected. The other, *SeeSaw*, never falls to a bandwidth that is too low to sustain uninterrupted streaming but is more challenging as it sees sudden large drops in bandwidth.

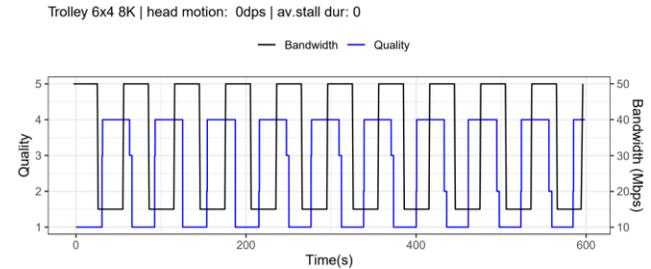
6.2.1. Stall Events. We observed no stall events for PoleVault in our testing for variable network conditions. Short stall events, 100-200ms were sometimes observed when bandwidth dropped in *SeeSaw*. For, *Slide*, more stalls were observed because the bandwidth dropped to 10Mbps, which was too low for both Trolley and Harbor at even the lowest quality as can be seen in Figure 7a. The stall event duration per stall was short due to a short buffer duration. Since several stalls are less preferred to one long stall [21], as future work we plan to experiment with buffer lengths to overcome this. Average total stall duration for the entire duration for all test conditions is summarized in Table 2.

Table 2: Total stall duration

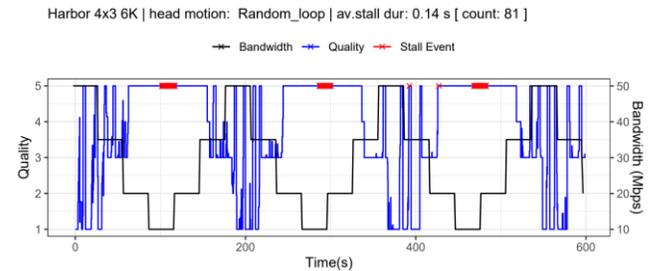
	BOLA			TIME		
	Stable	SeeSaw	Slide	Stable	SeeSaw	Slide
Mean	0.00s	0.55s	7.93s	0.01s	1.47s	8.00s
Std.Dev.	0.00s	0.78s	5.70s	0.05s	1.47s	5.87s

6.2.2. Adaptability. Our implementation was able to adapt to changing network conditions with and without head motion. For reference, we show timeline graphs in Figure 7 for the BOLA algorithm.

The first one is the 10-minute looped Trolley sequence with the *SeeSaw* network profile. Note that Quality 1 is the highest, so the algorithm is switching to highest quality when the bandwidth is 50Mbps and to lowest quality when the bandwidth is 15Mbps; there is no head motion and no stall events for this test, although there were some short stalls for bandwidth drops in some cases. The second graph is for the 10-minute looped Harbor case with *Slide* network profile with random fast head motion. Note that the stalls are mostly in the region where bandwidth is too low for the sequence and the algorithm adapts otherwise. When bandwidth is large, the head motion causes the quality to drop occasionally (calculated based on Equation 2). Note that the graph in Figure 7b shows the most challenging test sequence and test condition.



(a) SeeSaw network profile with no head motion



(b) Slide network profile with random head motion

Figure 7 A timeline showing the adaptation of quality levels

7 Conclusions

In this paper, we presented a method for calculating the required bit rates for the full sphere of a tiled 360-degree video when viewport-dependent streaming is used. It provides a practical approach for integrating the now widely deployed ABR algorithms for 2D video that have been improved over various iterations with the emerging 360-degree video use case. Using a DASH OMAF player, we showed the effectiveness of the method when used with two existing ABR algorithms to adapt based on viewport orientation as well as network conditions while minimizing the occurrence of stall events. With a buffer-based algorithm, we observed that when bandwidth was sufficient average stall duration for our 10-minute-long tests was under 1s. This was observed in the presence of head motion and drastically changing network conditions and despite a very short 3 second buffer duration. The paper highlights areas of future work to further optimize throughput utilization and stability.

REFERENCES

[1] Miska M. Hannuksela, Ye-Kui Wang and Ari Hourunranta. An Overview of the OMAF Standard for 360° Video. In *Data Compression Conference (DCC), Snowbird, UT, USA, 2019*. 418-427. <https://doi.org/10.1109/DCC.2019.00050>

- [2] Alireza Zare, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. 2016. HEVC-compliant Tile-based Streaming of Panoramic Video for Virtual Reality Applications. In *Proceedings of the 2016 ACM Conference on Multimedia Conference, MM 2016, Amsterdam, The Netherlands, October 15-19, 2016*. ACM, 601–605. <https://doi.org/10.1145/>
- [3] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*. ACM, 394–407. <https://doi.org/10.1145/3341302.3342063>
- [4] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, MobiCom 2018, New Delhi, India, October 29 - November 02, 2018*. ACM, 99–114. <https://doi.org/10.1145/3241539.3241565>
- [5] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017*. IEEE, 1–7. <https://doi.org/10.1109/ICC.2017.7996611>
- [6] Lan Xie, Xinggong Zhang, and Zongming Guo. 2018. CLS: A Cross-user Learning based System for Improving QoE in 360-degree Video Adaptive Streaming. In *2018 ACM Multimedia Conference on Multimedia Conference, MM 2018, Seoul, Republic of Korea, October 22-26, 2018*. ACM, 564–572. <https://doi.org/10.1145/3240508.3240556>
- [7] Robert Skupin, Yago Sanchez, Cornelius Hellge, and Thomas Schierl. 2016. Tile Based HEVC Video for Head Mounted Displays. In *IEEE International Symposium on Multimedia, ISM 2016, San Jose, CA, USA, December 11-13, 2016*. IEEE Computer Society, 399–400. <https://doi.org/10.1109/ISM.2016.0089>
- [8] Cagri Ozcinar, Ana De Abreu, and Aljosa Smolic. 2017. Viewport-aware adaptive 360° video streaming using tiles for virtual reality. In *2017 IEEE International Conference on Image Processing, ICIP 2017, Beijing, China, September 17-20, 2017*. IEEE, 2174–2178. <https://doi.org/10.1109/ICIP.2017.8296667>
- [9] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming. In *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, Mountain View, CA, USA, October 23-27, 2017*. ACM, 315–323. <https://doi.org/10.1145/3123266.3123291>
- [10] Mohammad Hosseini and Viswanathan Swaminathan. 2016. Adaptive 360 VR Video Streaming: Divide and Conquer. In *IEEE International Symposium on Multimedia, ISM 2016, San Jose, CA, USA, December 11-13, 2016*. IEEE Computer Society, 107–110. <https://doi.org/10.1109/ISM.2016.0028>
- [11] Kevin Spiteri, Rahul Uргаonkar, and Ramesh K. Sitaraman. 2016. BOLA: Nearoptimal bitrate adaptation for online videos. In *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 1–9. <https://doi.org/10.1109/INFOCOM.2016.7524428>
- [12] Chenghao Liu, Imed Bouazizi, Miska M. Hannuksela, and Moncef Gabbouj. 2012. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Sig. Proc.: Image Comm.* 27, 4 (2012), 288–311. <https://doi.org/10.1016/j.image.2011.10.001>
- [13] Demóstenes Zegarra Rodríguez, Zhou Wang, Renata Lopes Rosa, and Graça Bressan. 2014. The impact of video-quality-level switching on user quality of experience in dynamic adaptive streaming over HTTP. *EURASIP J. Wireless Comm. and Networking* 2014 (2014), 216. <https://doi.org/10.1186/1687-1499-2014-216>
- [14] Abdelhak Bentaleb, Bayan Taani, Ali C. Begen, Christian Timmerer, and Roger Zimmermann. 2019. A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP. *IEEE Communications Surveys and Tutorials* 21, 1 (2019), 562–585. <https://doi.org/10.1109/COMST.2018.2862938>
- [15] Igor D.D. Curcio, Henri Toukoma, Deepa Naik. Bandwidth Reduction of Omnidirectional Viewport-Dependent Video Streaming via Subjective Quality Assessment. In *ACM International Workshop on Multimedia Alternate Realities at ACM Multimedia Conference, 27 October 2017, Mountain View, CA, U.S.A.*
- [16] Deepa Naik, Igor D. D. Curcio, and Henri Toukoma. 2018. Optimized Viewport Dependent Streaming of Stereoscopic Omnidirectional Video. In *Proceedings of the 23rd Packet Video Workshop, PV@MMSys 2018, Amsterdam, Netherlands, June 12, 2018*. ACM, 37–42. <https://doi.org/10.1145/3210424.3210437>
- [17] Dmitrii Monakhov, Igor D. D. Curcio, and Sujeet Mate. 2019. On Data Wastage in Viewport-Dependent Streaming. In *21st IEEE International Workshop on Multimedia Signal Processing, MMSP 2019, Kuala Lumpur, Malaysia, September 27-29, 2019*. IEEE, 1–6. <https://doi.org/10.1109/MMSP.2019.8901701>
- [18] Virtual Reality (VR) streaming interoperability and characterization (Release 17). *3GPP Technical Specification 26.999 v 0.3.0, January, 2020*.
- [19] Marko Viitanen, Ari Koivula, Ari Lemmeti, Arttu Ylä-Outinen, Jarno Vanne, and Timo D. Hämmäläinen. 2016. Kvazaar: Open-Source HEVC/H.265 Encoder. In *Proceedings of the 24th ACM international conference on Multimedia (MM '16)*. Association for Computing Machinery, New York, NY, USA, 1179–1182. <https://doi.org/10.1145/2964284.2973796>
- [20] Virtual Reality (VR) profiles for streaming applications (Release 16). *3GPP Technical Specification 26.118 v.16.0.2, March 27, 2020, Annex D*.
- [21] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. 2017. A Quality-of-Experience Index for Streaming Video. *J. Sel. Topics Signal Processing* 11, 1 (2017), 154–166. <https://doi.org/10.1109/JSTSP.2016.2608329>
- [22] Mengbai Xiao, Chao Zhou, Viswanathan Swaminathan, Yao Liu, and Songqing Chen. 2018. BAS-360°: Exploring Spatial and Temporal Adaptability in 360-degree Videos over HTTP/2. In 2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018. IEEE, 953–961. <https://doi.org/10.1109/INFOCOM.2018.8486390>
- [23] Jun Fu, Xiaoming Chen, Zhizheng Zhang, Shilin Wu, and Zhibo Chen. 2019. 360SRL: A Sequential Reinforcement Learning Approach for ABR Tile-Based 360 Video Streaming. In IEEE International Conference on Multimedia and Expo, ICME 2019, Shanghai, China, July 8-12, 2019. IEEE, 290–295. <https://doi.org/10.1109/ICME.2019.00058>
- [24] Xiaolan Jiang, Yi-Han Chiang, Yang Zhao, and Yusheng Ji. 2018. Plato: Learning based Adaptive Streaming of 360-Degree Videos. In 43rd IEEE Conference on Local Computer Networks, LCN 2018, Chicago, IL, USA, October 1-4, 2018. IEEE, 393–400. <https://doi.org/10.1109/LCN.2018.8638092>