Scalable 360 Video Streaming using HTTP/2

Conference Paper · July 2019

DOI: 10.1109/MMSP.2019.8901805

6 authors, including:

Nguyen Duc
Tohoku Institute of Technology
45 PUBLICATIONS **317** CITATIONS
SEE PROFILE

Trung V. Hoang
Hanoi University of Science and Technology
1 PUBLICATION **6** CITATIONS
SEE PROFILE

Huong Le Dieu Hoang
Hanoi University of Science and Technology
2 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Truong Thu Huong
VNU University of Science
60 PUBLICATIONS **590** CITATIONS
SEE PROFILE

Scalable 360 Video Streaming using HTTP/2

Duc V. Nguyen[1], Hoang Van Trung[2], Hoang Le Dieu Huong[2],
Truong Thu Huong[2], Nam Pham Ngoc[3], and Truong Cong Thang[1]
[1]The University of Aizu, Japan
[2]Hanoi University of Science and Technology, Vietnam
[3] Vin-University Project

***Abstract*—360-degree video is the main content type of Virtual Reality, providing users with immersive viewing experience. In this paper, we propose a novel adaptation method for 360-degree video streaming over HTTP/2, which can provide high viewing experience to users under time-varying network conditions and time-varying user head movements. The proposed method utilizes Scalable Video Coding to solve the trade-off between network adaptivity and user adaptivity. An optimal tile layer selection algorithm is provided. To cope with sudden throughput drops, the delivery of late layers is terminated using HTTP/2's *stream termination* feature. Also, a tile layer updating scheme is proposed to deal with viewport estimation errors. Experimental results show that the proposed method can improve the average bitrate of viewport by 16-17% compared to a reference method.**

I. Introduction

360-degree video (*360 video* for short) is the main content type of Virtual Reality [1], being used in a wide range of VR applications such as VR sport [2]. 360 video has a much higher resolution and frame rate than conventional flat video. For example, to offer a true immersion to the user, 360 video is expected to have a resolution of 24K and a frame rate of 60fps [3]. As a result, 360 video requires high bandwidth when streaming over networks.

To reduce the bandwidth required for 360 video streaming, Viewport Adaptive Streaming (VAS) has been proposed. The basic idea is to deliver the viewport (i.e., video part visible to the user) at a high bitrate (quality) while delivering the other parts at a lower bitrate (quality) [4]. In the literature, Viewport Adaptive Streaming is usually implemented using the so-called tiling-based approach [4], [5]. In tiling-based VAS, a 360 video is spatially divided into multiple parts called *tiles*. Each tile is independently encoded into multiple versions of different bitrates (quality levels). Tiles overlapping (non-overlapping) the viewport are delivered at high (low) bitrate.

Since the user tends to change his/her viewing direction when watching a 360 video, the viewport is usually varying during a streaming session [6]. Hence, adaptation methods for VAS should be able to adapt the tiles' bitrates according to the time-varying viewport. In addition, the throughput in mobile networks may fluctuate significantly over time. Significant reductions in network throughput can lead to playback interruptions [7], which can greatly reduce the user viewing experience [8]. Hence, adaptation methods for VAS must also adapt the tiles' bitrates according to the time-varying network throughput.

To provide smooth playback under varying network conditions, the client should buffer some amount of video data before the playback of the video begins [7]. However, large buffer sizes can severely reduce the performances of VAS methods, as shown in [9]. Thus, there exists a trade-off between network adaptivity and viewport adaptivity in viewport adaptive streaming.

In this paper, we propose a novel adaptation method for 360 video streaming over mobile networks that can provide high viewing experience to the users with the following key features.

- Scalable Video Coding (SVC) is utilized to tackle the trade-off between network adaptivity and viewport adaptivity.
- The tile layer selection problem is formulated and an efficient algorithm is proposed.
- A late tile layer termination scheme is presented that can save network resources by terminating the delivery of late tile layers using HTTP/2's *stream termination* feature.
- A tile layer updating scheme that can effectively deal with viewport estimation errors is proposed. The scheme makes use of HTTP/2's *stream priority* feature.

Experimental results using real head movement traces and real network throughput traces show that the proposed method can improve the average viewport bitrate by 16-17% compared to a reference method. Also, our method can avoid significant reductions in the buffer level.

The remaining of this paper is organized as follows. Related work is given in Section II. The proposed method is presented in Section III. The evaluation and results are given in Section IV. Finally, the paper is concluded in Section V.

II. Related Work

Most previous studies on 360 video streaming focused on adapting the video content according to the user viewing directions [4], [5], [10]–[13]. In [9], it is found that the state-of-the-art adaptation methods perform well only for buffer sizes less than or equal to 2 seconds. The reason is that it is extremely difficult to estimate viewport positions far in the future. From network adaptivity perspective, using such small buffer sizes makes the system very vulnerable to network throughput fluctuations, causing playback interruptions [7]. There are some work that focused on dealing with network throughput variations using various HTTP/2's features such as [14].

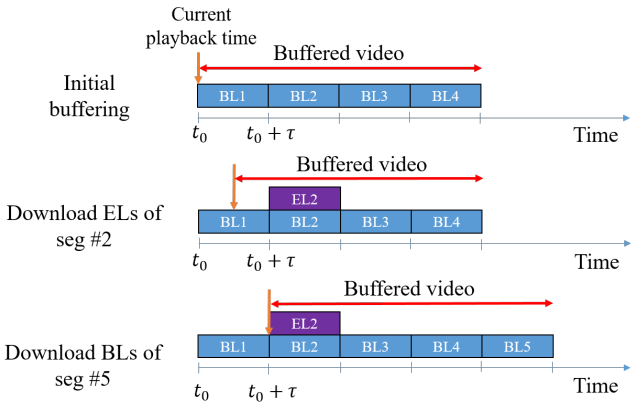


Fig. 1: Solving the trade-off between network adaptivity and viewport adaptivity using Scalable Video Coding.

In [15], the authors propose to use SVC to cope with both network variations and viewport variations in 360 video streaming. However, this method does not provide any mechanisms to deal with viewport estimation errors. In [16], a two-tier system for 360 video streaming is proposed, which can dynamically adapt the bitrates of the base tier and the enhancement tier according to the network throughput variations and FoV prediction errors. In this paper, our focus is on adapting the tiles' enhancement layer bitrates, with the base layer bitrate being fixed.

The use of HTTP/2 protocol for tile delivery has been studied in the literature. HTTP/2's *server push* feature has been used to increase network throughput under networks with long end-to-end delay [12]. In [17], HTTP/2's *stream priority* and *stream termination* features are utilized to deal with viewport estimation errors. Yet, their method is evaluated using a simulator only. In this paper, we implement and evaluate the proposed method on a real test-bed.

III. Proposed Framework

Fig. 1 shows the basic idea of using SVC to solve the trade-off between network adaptivity and viewport adaptivity in VAS. At the beginning of a streaming session, the client fills its buffer with the base layers of video segments. When the buffer level achieves a pre-defined value at time t_0, the client enters the second phase. In the second phase, for each adaptation interval, the client downloads the enhancement layers of the next playing segment and the base layer of the next segment. In the above example, from time t_0 to time $t_0 + \tau$, the enhancement layers of segment #2 is downloaded first. Then, the base layers of segment #5 will be downloaded. This will ensure that the client has enough buffered video data to cope with network throughput fluctuations. Also, as the enhancement layers of a segment are downloaded just one segment duration before the playback of the segment, high viewport quality can be achieved. In other words, the dialema between network adaptivity and viewport adaptivity can be mitigated.

When the network throughput drops significantly, it will take a longer time than expected to download layers of segments. This will cause the buffer level to decrease. In our proposed method, the client will request only the base layers to re-fill the buffer when the buffer size becomes lower than B_{min} seconds.

Because an enhancement layer cannot be used for the playback if the layer arrives at the client after its playback deadline, receiving all video data of the layer will waste network resources (i.e., bandwidth). In our proposed framework, the delivery of late layers will be terminated immediately by utilizing HTTP/2's *stream termination* feature.

As aforementioned, the enhancement layers of a segment will be downloaded one segment duration before the playback of the segment begins. In our framework, to deal with viewport estimation errors, after deciding the appropriate layers and sending the requests to the server, the client continues to monitor the viewport positions and update the estimated viewport. If the estimated viewport changes significantly, the selected enhancement layers will be updated to accommodate the difference.

A. Tile Layer Selection

In this part, the tile layer selection problem will be formulated. Then, an efficient algorithm to decide the layer for each tile will be presented.

The system needs to stream a 360 video with a duration of *VD* seconds and a frame rate of *FPS* from a server to a client over a communication network with time-varying network bandwidth. On the server, the video is divided into K segments, each has a playback duration of *SD* in seconds. Each segment is further divided into M small parts called *tiles*. Each tile is encoded into L layers with different quality levels. Layer 0 is called *base layer*, which has the lowest quality. Layers $\{l, 1 \leq l \leq L-1\}$ are called *enhancement layers*, which provide higher video quality. An enhancement layer can only be decoded and played if the base layer and all lower enhancement layers are available at the client.

Suppose that, at a given time, the system needs to adapt a segment to meet a bitrate constraint R^c. Layer l $(0 \leq l \leq L-1)$ of tile m $(1 \leq m \leq M)$ of segment k has a bitrate of $R_m^{l_m}$ and a quality $Q_m^{l_m}$. Let V_n denotes the viewport position at the n^{th} $(1 \leq n \leq N = SD \times FPS)$ frame of the segment. Typically, a position of a viewport can be defined by the longitude and the latitude of the center point of the viewport. Let l_m $(0 \leq l_m \leq L-1)$ denotes the max quality layer selected for tile m $(1 \leq m \leq M)$. The adaptation problem for segment k can be formulated as follows.

Find $\{l_1, l_2, ..., l_M\}$ to maximize a quality objective *VQ*, which is a function of the tiles' quality values $\{Q_m^{l_m}\}_{1 \leq m \leq M, 0 \leq l \leq L-1}$ and the viewport positions during the playback of the segment $\{V_n\}_{1 \leq n \leq SD \times FPS}$

$$VQ = f(Q_1^{l_1}, Q_2^{l_2}, ..., Q_M^{l_M}, V_1, V_2, ..., V_N), \quad (1)$$

and satisfy the bitrate constraint.

$$\sum_{m=1}^{M}\sum_{l=0}^{l_m-1} R_m^{l_m} \le R^c. \quad (2)$$

In our proposed method, the bitrate constraint R^c is simply set to the estimated throughput T^e, i.e., $R^c = T^e$. Here, the estimated throughput T^e is calculated as the average of the last S throughput samples as follows.

$$T^e = \frac{1}{S}\sum_{i=i_{last}-S-1}^{i_{last}} T^s(i) \quad (3)$$

Here, i_{last} denotes the last download round. The throughput sample T^s of a download round is calculated as the ratio of the downloaded data and the time taken to download that data.

The viewport position V_n $(1 \le n \le N)$ is simply estimated using the position of the viewport that the user is currently looking at the moment. Let V^{cur} be the current viewport position and V_n^e be the estimated viewport position, we have: $V_n^e = V^{cur}, 1 \le n \le N$.

Currently, the quality objective is calculated as the average viewport bitrate of the segment as follows. Let $w_m(V)$ denotes the weight of tile m given a viewport position V. The quality objective VQ is given by

$$VQ = \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}\sum_{l=0}^{l_m-1} w_m(V_n) \times R_m^{l_m} \quad (4)$$

Similar to [4], the weight $w_m(V)$ is calculated as the ratio of the visible pixels of tile m and the total number of pixels in the viewport.

Given the bitrate constraint R^c, the estimated viewport positions $\{V_n, 1 \le n \le N\}$, and the quality objective *VQ*, the optimal layers of tiles of problem (1) can be found by performing a full-search on all possible selections.

Nevertheless, doing a full-search can be so time-consuming that not suitable for real-time adaptation, especially when the number of tiles and/or the number of layers are large. Thus, we propose a heuristic to decide the layers of tiles as follows. First, the tiles are sorted according to their weights. Note that, only visible tiles are considered in the proposed algorithm since all invisible tiles have a weight of zero. The bitrate budget R^c will be allocated to the visible tiles layer-by-layer. The details of the proposed algorithm are described in Algorithm 1.

B. Throughput-aware Late Tile Termination

When the real network throughput is much lower than the estimated value, some of the requested tiles' layers will arrive at the client after their playback deadlines. Since those tiles' layers can not be used for playback, it is beneficial to terminate the delivery of those late layers as soon as possible.

In our proposed system, we propose to utilize the *stream termination* feature of HTTP/2 [18] to cancel the delivery of late tile layers. Specifically, the client will send the RST STREAM frames, which contains the stream ids of the terminated tile layers to the server. The server will discard all HTTP/2's frames belonging to the stream specified in the RST STREAM frame. Typically, at playback time $t_p(k)$ of segment k, the client will send RST STREAM frames for the tiles' layers that have not been completely received as described in Algorithm 2. In Algorithm 2, l_m^{late} is the set of late layers of tile m.

Algorithm 1: *Tile Layer Selection*

Input: $M, N, R^c, R_m^l, V_n^e, w_m(V_n^e)$
Output: $\{l_m\}_{1\le m\le M}$

```
l_m ← 0 for 1 ≤ m ≤ M;
ΔR ← R^c − Σ_{m=1}^{M} Σ_{l=0}^{l_m−1} R_m^{l_m};
sortedTile ← sort(w_m(V_n^e));
for l = 1 to L − 1 do
    foreach m ∈ sortedTile do
        if l_m < L − 1 and R_m^{l_m+1} < ΔR then
            ΔR ← ΔR − R_m^{l_m+1};
            l_m ← l_m + 1;
        end
    end
end
return {l_m}_{1≤m≤M};
```

Algorithm 2: *Late Tiles' Layers Termination*

Input: $\{l_m^{late}, 1 \le m \le M\}$

```
for m = 1 to M do
    foreach l ∈ l_m^late do
        send RST STREAM frame for the stream
         corresponding to layer l;
    end
end
```

C. Viewport-aware Tile Layer Updating

By downloading the enhancement layers of tiles of a segment approximately one segment duration before its playback, the proposed framework can reduce the viewport estimation horizon down to the duration of one segment. This will help improve the viewport estimation accuracy, especially when the segment duration is short (i.e., typically $\le 1s$). However, errors in viewport position estimation are unavoidable due to random user's head movements. Also, the segment duration can be large in several cases. Thus, it is important to deal with viewport estimation errors.

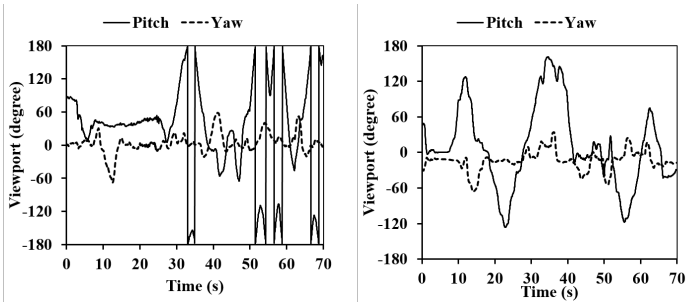
Specifically, we will continuously re-estimate the viewport position. If the newly estimated value includes some tiles that have not been initially requested, we will send additional requests for those tiles' layers. Let $\Omega(V)$ denotes the set of viewport tiles given the estimated viewport position V. Let $V^e(t)$ be the estimated viewport position at time t. At time $t + \Delta t$, we obtain the new estimated viewport position $V^e(t + \Delta t)$. Let Ω^{New} denotes the set of tiles that are in $\Omega(V^e(t + \Delta t))$ but not in $\Omega(V^e(t))$. We have:

$$\Omega^{New} = \Omega(V^e(t + \Delta t)) - \Omega(V^e(t)) \quad (5)$$

Algorithm 3: *Viewport-aware Tile Layer Updating*

Input: $\Omega(t), \Omega(t+\Delta t)$

Calculate Ω^{New} and Ω^{Old} using Eq. (5) and Eq. (6);
if $|\Omega^{New}| > 0$ **then**
  Select a tile m' in Ω^{New};
  foreach $m \in \Omega^{Old}$ **do**
    set priority $(m', 1, 0)$;
    send PRIORITY frame for tiles m;
  end
  foreach $k \in \Omega^{New}$ **do**
    send request for tile k;
  end
end



(a) Head Mov. Trace #1 (b) Head Mov. Trace #2

Fig. 2: Two head movement traces used in our experiments.

The tiles in Ω^{New} are called *new viewport tiles* and the tiles in Ω^{Old} is called *old viewport tiles*. Similarly, we calculate the set of tiles that is in $\Omega(V(t))$ but not in $\Omega(V(t+\Delta t))$.

$$\Omega^{Old} = \Omega(V^e(t)) - \Omega(V^e(t+\Delta t)) \quad (6)$$

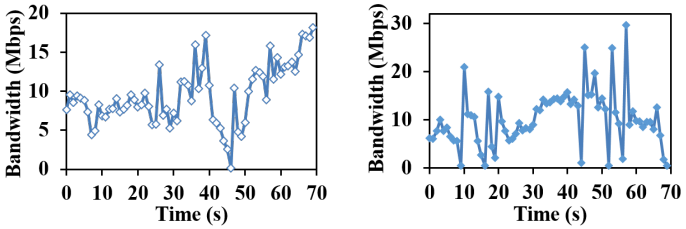
Because it is more likely that the tiles in Ω^{New} will be the actual viewport tiles than the tiles in Ω^{Old}, the delivery of tiles in Ω^{New} will be done first. The highest layer is selected for each tile in Ω^{New}. When the delivery of all tiles in Ω^{New} is completed, the delivery of tiles in Ω^{Old} will be continued.

In HTTP/2, the order in which tiles' layers are delivered can be specified using the so-called *stream priority* feature. The *stream priority* feature allows either the client or the server to define the dependence between streams. For example, if stream A is set to be dependent on the stream B, the delivery of stream B can only start after stream A has been terminated. The process of updating tiles' layers is described in Algorithm 3.

IV. Evaluation

A. Experimental Settings

To evaluate the performance of the proposed method, we build a test-bed that consists of a client connects to a server over a communication network. The proposed method is implemented on the client using nghttp2, an open-source HTTP/2 library. The network conditions including bandwidth and delay



(a) Bandwidth trace #1 (b) Bandwidth trace #2

Fig. 3: Two network bandwidth traces used in our experiments [19].

are emulated using DummyNet tool [20], which runs on the client. The end-to-end delay is set to 10ms in all experiments.

We use a 360-degree video in Cubemap projection format. The video has a duration of 70 seconds, a resolution of 2880x1920, a frame rate of 30 fps. The video is divided into 24 tiles by partitioning each face of the cube into 4 equal-size tiles. Each tile is encoded into 3 layers. The base layer has a bitrate of 125 kbps. The bitrate of the first and second enhancement layers are respectively 200 kbps and 400 kbps. The segment duration is 1 second. Two head movement traces as shown in Fig. 2 are used.

In the proposed method, the value of S in Eq. (3) is set to 3. The value of Δt in Algorithm 3 is set to a half of the download budget time. The buffer size B is set to 6 seconds. The B_{min} threshold is set to 3 seconds.

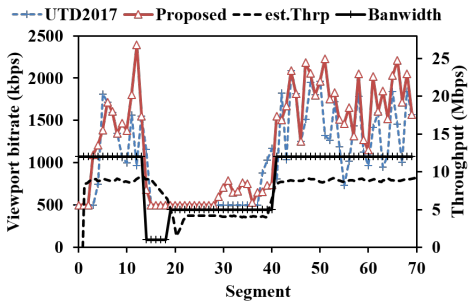
The proposed framework is compared to a reference method denoted UTD2017, which is proposed in [15]. The UTD2017 method also utilizes Scalable Video Coding to encode each tile into multiple layers. For viewport adaptation, the same layer is selected for all viewport tiles as allowed by the available network throughput.

B. Experiment Results

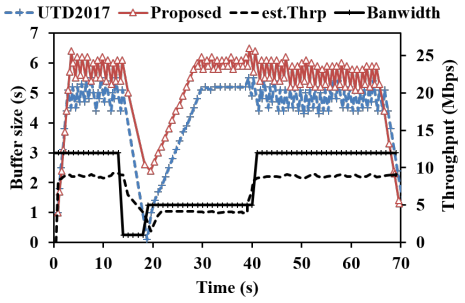
1) Simple Bandwidth Trace: In this part, we will investigate the performance of the proposed method under a simple bandwidth trace. Specifically, the bandwidth is set to 12Mbps in the first 14 seconds, then drops to 1Mbps for 5 seconds. The bandwidth increases to 5Mbps at time t=19s. At time t=40s, the bandwidth increases to 12Mbps and remains unchanged until the end of the streaming session.

Figure 4 shows the viewport bitrate and the buffer size over time of the proposed and reference methods under the simple bandwidth trace and the head movement trace #1 as shown in Fig. 2a. The number of request tiles, new viewport tiles, and canceled tiles over time of the proposed method are shown in Fig 5.

We can see that the proposed method achieves higher viewport bitrate than the UTD2017 method for most of the segments. This is because that, thanks to the proposed tile layer selection algorithm, our method is able to request higher numbers of enhancement layers than the UTD2017 method as can be seen in Fig. 5. Also, the proposed viewport-awre tile



(a) Viewport bitrate (kbps)



(b) Buffer size (s)

Fig. 4: Viewport bitrate (kbps) and buffer size (seconds) over time of the proposed and reference methods under a simple bandwidth trace (Head movement trace #1).

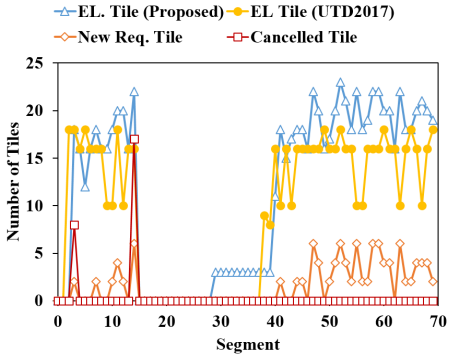
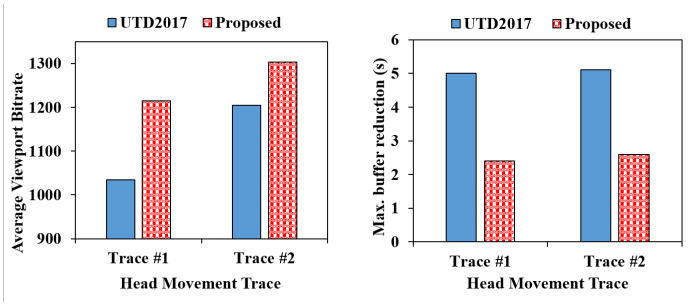


Fig. 5: Number of request tiles, new viewport tiles, and canceled tiles over time of the proposed method under a simple bandwidth trace (Head movement trace #1).

layer updating scheme frequently updates the viewport tiles, improving the viewport quality (Fig. 5)

When the throughput decreases dramatically from 12Mbps to 1Mbps at time t=14s, it can be seen that the buffer size of the UTD2017 method reduces drastically, resulting in an interruption at time t=19s. On the other hand, the proposed method can mitigate the impact of the throughput reduction on the client's buffer size by canceling late tiles' layers as can be seen in Fig 5. As a result, playback interruption can be avoided as can be seen in Fig. 4b.

It should be noted that the proposed method downloads only



(a) Average viewport bitrate (kbps) (b) Max. buffer reduction (s)

Fig. 6: Viewport bitrate (kbps) and buffer size (seconds) of the proposed and reference methods under a simple bandwidth trace.

TABLE I: Performance of the proposed and reference methods under the network bandwidth trace #1.

Metrics	Head Mov. Trace #1		Head Mov. Trace #2	
	UTD2017	Proposed	UTD2017	Proposed
Avg. viewport bitrate (kbps)	1030.1	1193.4	1135.4	1337.3
Max. buffer reduction (s)	1.3	1.3	1.5	1.4

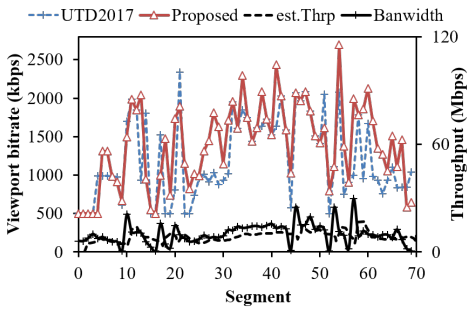
the base layer from time t=20s to t=29s. During this interval, the client re-fills the buffer until the buffer level exceeds $B = 6s$. From time t=30s to the end of the streaming session, we can see that the proposed method again achieves higher viewport bitrate than the UTD2017 method for most of the segments, thanks to the optimal tile layer selection algorithm and the user-aware tile layer updating scheme.

The average viewport bitrates and the maximum buffer reductions of the proposed and reference methods under the two head movement traces are shown in Figure 6. It can be seen that the proposed method outperforms the UTD2017 method in both terms of viewport bitrate and buffer reduction. Specifically, our method improves the average viewport bitrate by 17% and 8% for the head movement trace #1 and #2, respectively. Furthermore, the proposed method can offer more stable buffer level with lower maximum buffer reduction the UTD2017 method.

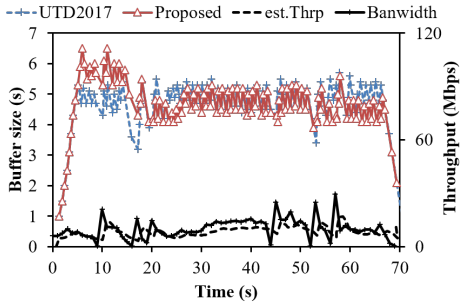
2) Real bandwidth trace: In this part, we will investigate the performance of the proposed method using a real bandwidth trace. For that purpose, we use two network bandwidth traces from 4G network provided in [19]. The used bandwidth traces are shown in Fig. 3.

Fig. 7 shows the viewport bitrate and the buffer size over time of the proposed method and the UTD2017 method under the bandwidth trace #2 and the head movement trace #2. As can be seen in Fig. 7a, the viewport bitrate of the proposed method is higher than that of the UTD2017 method for most of the segments. This implies that our method is effective in improving the viewport quality under the real bandwidth trace.

Table I and Table II show the average viewport bitrates and the maximum buffer reductions of the proposed and reference methods under the two network bandwidth traces and the two head movement traces. We can see that the proposed method



(a) Viewport bitrate (kbps)



(b) Buffer size (s)

Fig. 7: Viewport bitrate (kbps) and buffer size (seconds) over time of the proposed and reference methods under bandwidth trace #2 and head movement trace #2.

TABLE II: Performance of the proposed and reference methods under the network bandwidth trace #2.

Metrics	Head Mov. Trace #1		Head Mov. Trace #2	
	UTD2017	Proposed	UTD2017	Proposed
Avg. viewport bitrate (kbps)	1115.4	1295.8	1196.8	1396.9
Max. buffer reduction (s)	2.3	1.5	1.8	1.4

outperforms the UTD2017 method in both terms of average viewport bitrate and maximum buffer reduction. Specifically, the average viewport bitrate of the proposed method is 16-17% higher than that of the UTD2017 method for all four cases. In addition, the proposed method always has lower maximum buffer reduction than that of the UTD2017 method. This implies that the proposed late tile terminate algorithm is effective in dealing with network throughpout variations.

V. Conclusions

In this paper, we have proposed a new adaptation method for 360-degree video streaming, which can adapt the video according to both network throughput variations and user's behaviors. The trade-off between user-adaptive and network-adaptive is solved by using Scalable Video Coding. In addition, an effective tile layer selection method is proposed. To mitigate the impact of drastic throughput reductions, HTTP/2's stream termination feature is used. It is found that the proposed method can improve the average viewport bitrate by 16-17%, while providing more stable buffer level compared to a reference method.

References

[1] T. Stockhammer. MPEG Immersive Media. Accessed: 2018-09-30. [Online]. Available: https://www.itu.int/en/ITU-T/studygroups/2017-2020/16/Documents/ws/201701ILE/S2P2-1701-01-19-MPEG-Immersive-Media-Thomas-Stockhammer.pdf

[2] M. Chen, K. Hu, I. Chung, and C. Chou, "Towards VR/AR Multimedia Content Multicast over Wireless LAN," in *2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC)*, Las Vegas, US, Jan 2019, pp. 1–6.

[3] Huewei. (2017) Virtual reality/augmented reality white paper. [Online]. Available: http://www-file.huawei.com/-/media/CORPORATE/PDF/ilab/vr-ar-en.pdf

[4] D. V. Nguyen, H. T. T. Tran, A. T. Pham, and T. C. Thang, "An optimal tile-based approach for viewport-adaptive 360-degree video streaming," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 29–42, March 2019.

[5] C. Ozcinar, A. De Abreu, and A. Smolic, "Viewport-aware adaptive 360 video streaming using tiles for virtual reality," in *2017 IEEE International Conference on Image Processing (ICIP)*, Beijing, China, Sep. 2017, pp. 2174–2178.

[6] X. Corbillon, F. De Simone, and G. Simon, "360-degreee video head movement dataset," in *Proc. 8th ACM MMsys*, ser. MMSys'17. Taipei, Taiwan: ACM, 2017, pp. 199–204.

[7] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro, "An evaluation of bitrate adaptation methods for HTTP live streaming," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 4, pp. 693–705, 2014.

[8] H. T. T. Tran, N. P. Ngoc, A. T. Pham, and T. C. Thang, "A Multi-Factor QoE Model for Adaptive Streaming over Mobile Networks," in *2016 IEEE Globecom Workshops (GC Wkshps)*, Dec 2016, pp. 1–6.

[9] D. V. Nguyen and H. T. T. Tran and T. C. Thang, "Impact of delays on 360-degree video communications," in *2017 TRON Symposium (TRONSHOW)*, 12 2017, pp. 1–6.

[10] L. Xie, Z. Xu, Y. Ban, X. Zhang, and Z. Guo, "360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming," in *Proc. of 2017 ACM Multimedia Conference*, CA, USA, Oct. 2017, pp. 315–323.

[11] Y. Ban, L. Xie, Z. Xu, X. Zhang, Z. Guo, and Y. Hu, "An optimal spatial-temporal smoothness approach for tile-based 360-degree video streaming," in *2017 IEEE Visual Communications and Image Processing (VCIP)*, St. Petersburg, FL, USA, Dec 2017, pp. 1–4.

[12] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck, "An http/2-based adaptive streaming framework for 360 virtual reality videos," in *2017 ACM Multimedia (MM) Conference*, 2017, pp. 1–9.

[13] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan, "Viewport-driven rate-distortion optimized 360 video streaming," in *2018 IEEE International Conference on Communications (ICC)*, MO, USA, May 2018, pp. 1–7.

[14] D. H. Nguyen, M. Nguyen, N. P. Ngoc, and T. C. Thang, "An adaptive method for low-delay 360 vr video streaming over http/2," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, July 2018, pp. 261–266.

[15] A. T. Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash, "Adaptive 360-degree video streaming using Scalable Video Coding," in *Proc. 25th ACM Multimedia*, Mountain View, California, USA, Oct. 2017, pp. 1689–1697.

[16] L. Sun, F. Duanmu, Y. Liu, Y. Wang, Y. Ye, H. Shi, and D. Dai, "A two-tier system for on-demand streaming of 360 degree video over dynamic networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 43–57, March 2019.

[17] M. Ben Yahia, Y. Le Louedec, G. Simon, and L. Nuaymi, "Http/2-based streaming solutions for tiled omnidirectional videos," in *2018 IEEE ISM*, Dec 2018, pp. 89–96.

[18] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," Tech. Rep., 2015.

[19] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, Nov 2016.

[20] M. Carbone and L. Rizzo, "Dummynet revisited," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764876