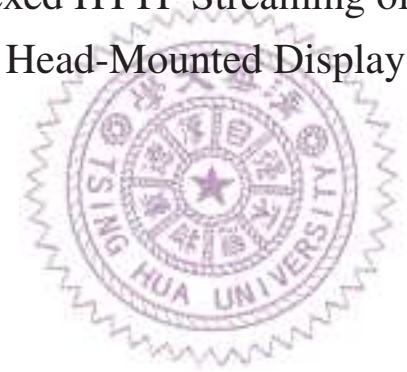
國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

搶先式多工HTTP串流360方塊影片至頭戴式顯示器

Preemptive Multiplexed HTTP Streaming of 360° Tiled Videos to Head-Mounted Displays



嚴守成
Shou-Cheng Yen

學號：107062569
Student ID:107062569

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 108 年 5 月
May, 2019


TSING HUA UNIVERSITY

Acknowledgments

I would like to express my gratitude toward my advisor: Dr. Cheng-Hsin Hsu. He spent a lot of time providing me suggestions and directions to help me solve problems when I was having problems or being confused. Through the guidance of him, I learned to solve the problems in a structured way and gradually completed my thesis. Besides, thank the labmates in Networking and Multimedia Systems Laboratory, epsecially Ching-Ling Fan, for assisting me in solving problems that I am not familiar with. I have learned a lot through cooperation with her. In addition, she often helped me to clarify and solve many problems of my research that bothered me. Without her help, I could not finish my thesis as scheduled.

致謝

在此我要特別感謝我的指導教授: 徐正炘教授，在我研究碰到難題或者感到迷茫時花很多時間給我 建議與方向幫助我解決問題。透過教授的指導，我學會有條理的解決所面臨的問題，並且逐步的完成我的論文。 感謝網路與多媒體系統實驗室的同學，特別是樊慶玲，協助我解決我不熟悉的領域的問題。 透過與她的合作我學到了許多。在研究過程中， 她也常常幫助我釐清與解決許多在研究上困擾我的問題。 如果沒有她的幫忙，我沒辦法如期的完成我的論文。

Abstract

We design, implement, and evaluate a tiled DASH streaming system for 360° videos using QUIC/UDP protocol, in which multiplexed and prioritized streams are leveraged for sending urgent tiles that are about to miss their playout time. In particular, we develop a new architecture to concurrently request for regular tiled segments at lower priorities and urgent tiled segments at higher priorities as multiple streams over a single QUIC connection. Several core components, including the fixation prediction algorithm, fast tile selector, and Adaptive Bit Rate (ABR) algorithm are designed for this new architecture. Compared to streaming 2D planar videos, streaming 360° tiled videos using DASH to head-mounted displays is much more challenging. To the best of our knowledge, most existing DASH ABR algorithms are not designed for multiple and concurrent streams. Therefore, we design an ABR algorithm tailored for preemptive multiplexed DASH streams carrying 360° tiled videos. A suite of design decisions are made for three design objectives: (i) preventing buffer under-run, (ii) avoiding large quality jumps, and (iii) maximizing average quality. Capitalizing a few open-source projects, we implement our proposed solutions in a real end-to-end Linux system. Our experiment results show that compared to the baseline algorithms, our algorithm: (i) averagely reduces the rebuffering counts by up to 3.2 and rebuffering time by up to 2.54 s when the bandwidth is limited, (ii) achieves at most 40.02% higher bandwidth utilization, and (iii) delivers good average V-PSNR at 39–49 dB under 5–15 Mbps bandwidth.

中文摘要

我們使用QUIC / UDP協議設計，設計、實作、評估用於360度視頻的方塊DASH串流媒體系統， 其中利用多路復用和優先串流來發送即將錯過其播出時間的緊急方塊視頻。 我們設計了一個新架構， 可以同時向伺服器端利用較低優的順位索取常規方塊視頻和 較高順位索取的緊急方塊視頻，讓單個QUIC連線上有多個串流。 我們為這個新架構設計了幾個核心組件， 包括眼球注視預測演算法、快速方塊視頻挑選器、動態調整視頻畫質(ABR)的演算法。 與串流2D平面視頻相比， 使用DASH將360度方塊視頻傳輸到頭戴式顯示器更具挑戰性。 據我們所知，大多數現有的DASH的ABR演算法不是針對多個串流而設計的。 因此我們設計了一種針對串流360度方塊視頻的搶占式多路復用DASH串流的ABR算法。 我們的演算法設計基於三個目標：（i）防止影片緩衝區無內容、（ii）避免影片間畫質的差異太大、以及（iii）最大化平均影片畫質。 我們利用一些開源專案在Linux系統中實現我們提出的解決方案。 我們的實驗結果顯示，與其他基準演算法相比，我們的算法：（i）當網路資源有限時，平均將緩衝計數減少3.2次，緩衝時間最多減少2.54秒，（ii）最多達到網路資源利用率提高40.02%，以及（iii）在5-15 Mbps頻寬下提供39-49 dB的良好平均V-PSNR。

Contents

List of Figures

List of Tables


TSING HUA UNIVERSITY

Chapter 1

Introduction

360° videos enable interactive experience of watching video content in arbitrary orientations determined by viewers, which have become very popular nowadays. Viewers no longer sit still in front of laptops and TV sets, but freely turn their heads to experience 360° videos in Head-Mounted Displays (HMDs) [46]. Because of the immersive experience provided by HMDs, HMDs have become the dominating displays/devices to consume 360° videos. To cope with the extremely high bandwidth consumption of streaming the complete 360° videos encoded at very high resolutions, most 360° DASH (Dynamically Adaptive Streaming over HTTP) streaming systems employ *tiled streaming* [15, 52, 27] to selectively send the tiles in viewers' viewports (typically $67° \times 67°$–$100° \times 100°$ [15]).

Achieving high streaming quality in such systems is very challenging, because conventional DASH streaming suffers from long response time as: (i) each segment typically lasts for a few seconds, and (ii) large buffer is required to absorb the network dynamics. Therefore, 360° tiled DASH video streaming clients *must* predict each viewer's future viewports, so that the required tiles can be requested *in advance*. Such prediction is done by some *fixation prediction algorithms* [19, 8, 31] proposed in the literature, which calculates the viewer's expected viewport centers in the future. To our best knowledge, these prediction algorithms are not 100% accurate. Therefore, when *predicted* viewport centers are different from the *actual* viewport centers, the DASH client has to either stall the playout or omit some tiles in the actual viewports referred to as *missing tiles*. This in turn results in *playout freezes* or *black holes*: both dramatically degrade the viewing experience.

The *missing-tile* issue is further *amplified* by the dominating streaming protocol: Dynamic Adaptive Streaming over HTTP (DASH) [43], which employs HTTP over TCP for video streaming. While DASH is quite suitable for *presentational* or unidirectional video streaming, it is not suitable for 360° tiled video streaming that is more interactive, because of its inherent extra delay. As illustrated in Fig. 1.1, the HMD orientations or

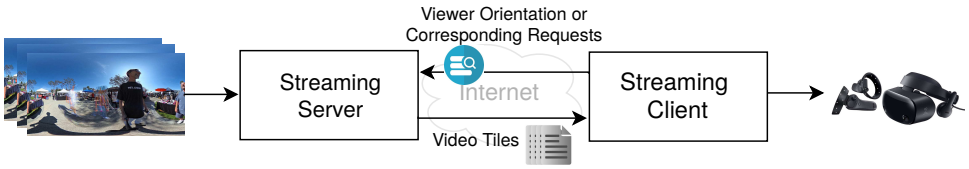


Figure 1.1: 360° video streaming needs high bandwidth, low latency.

sensor data determine the tiles that will fall in the viewports, and the corresponding requests *must* be sent to the streaming server in time. *Therefore, naively applying DASH for 360° video streaming may result in suboptimal streaming quality.* Based on this observation, we may switch to another extreme design and adopt Real-time Transport Protocol (RTP) for high responsiveness. Doing so however complicates the 360° video streaming systems, because of the UDP-related details, such as flow/congestion control and reliable transmission.

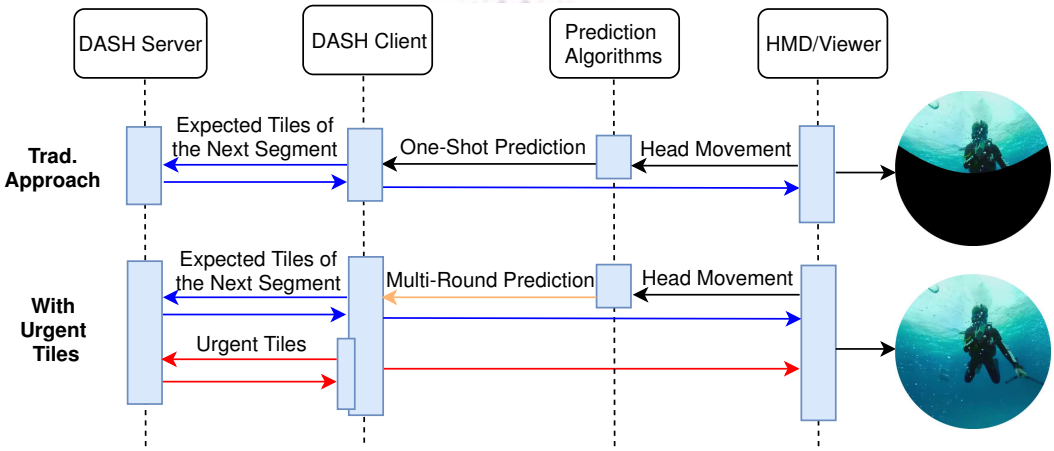


Figure 1.2: 360° tiled streaming systems with/without urgent tiles.

A better approach to cope with the long response time is to have the DASH client urgently request the missing tiles (due to imperfect fixation predictions) at a higher priority than the regular requests. By doing so, the problematic playout freezes and black holes might be mitigated. We adopt the QUIC protocol [23] to optimize 360° video DASH streaming to HMDs. QUIC was initially created by Google, and has been adopted as an IETF standard. QUIC runs on UDP, and was designed to replace the HTTP/2, TLS, and TCP protocols in the HTTPS stack. The emerging QUIC has three main features: (i) secured communications, (ii) multiplexed streams with prioritized schedulers, and (iii) low latency at the time of writing. We make a critical observation: *stream multiplexing* and *low latency* are the key enablers to optimize 360° tiled video DASH streaming. This is because *when a viewer suddenly rotates his/her head, QUIC allows the system to*

quickly send urgent tiles at high priority to mitigate the negative impacts due to missing tiles. This approach provides higher *interactivity* while retaining the *simplicity* of DASH streaming. More specifically, we request the tiles in predicted viewports in low-priority QUIC streams. These tiles are referred to as the *regular tiles*. We then frequently search for any required tiles that have not been requested due to imperfect fixation prediction and request those tiles in high-priority QUIC streams. These tiles are referred to as *urgent tiles*.

The illustrations of the 360° tiled streaming with/without urgent tiles are shown in Fig. 1.2. With the help of urgent tiles, black holes (or playout freezes) may be avoided in the viewports. Several DASH components, however, need to be optimized for capitalizing the unique QUIC features. For example, existing Adaptive Bit Rate (ABR) algorithms [22] are designed for unprioritized/sequential video streaming, which may not work well with QUIC protocol. We first slightly augments the DASH ABR algorithms that were not designed for multiplexed QUIC sessions [22, 42, 41] for our proposed system. Such an approach, however, may suffer from suboptimal performance (as we will show later). Therefore, we further design and evaluate a new ABR algorithm for *multiplexed* DASH streaming[1] of 360° tiled videos to HMDs. This is not an easy task because of the following challenges:

- **Viewer dynamics and imperfect fixation prediction algorithms**. HMD viewers may change their viewports any time, leading to *viewport mismatches*, where the earlier predicted viewports are different from the actual viewports. *How to quickly adapt to the viewer dynamics and minimize their negative impacts become a unique challenge for 360° tiled DASH streaming.*

- **Network dynamics and varying workloads.** Unlike traditional ABR algorithms, 360° tiled video streaming suffers from huge workload variations due to factors like: (i) non-uniform pixel density of projected tiles and (ii) sudden viewer head movements in diverse degrees and frequency. *How to faithfully measure the throughput for available bandwidth estimation under varying workloads becomes more difficult in 360° tiled video streaming.*

- **Complex human visual systems.** Traditional ABR algorithms trade off absolute visual quality and its variation across consecutive video frames, as human eyes are sensitive to visual quality jumps [7]. Tiled DASH streaming amplifies this challenge in both temporal (across adjacent video frames) and spatial (across adjacent tiles in

[1]While we often use QUIC protocol as a sample protocol for concrete discussion in this thesis, we envision that the *multiplexed* DASH streaming may be realized by other protocols, including those developed in the future.

the same video frame) domains. In addition, human eyes are more sensitive to the details at the vision center [16], which renders a uniform quality level for all tiles of a video frame less efficient. *How to properly assign quality levels of individual tiles to avoid impairments such as large quality jumps, low vision-center quality, and missing tiles, being noticed by human visual systems becomes the key of intelligently allocating precious resources in 360° tiled DASH streaming.*

Our ABR algorithm aims to address the challenges mentioned above through a few key design decisions:

- **Organize the DASH requests into multiple prioritized streams in a single DASH session.** This is to: (i) allow us better estimate the available throughput, (ii) give the ABR algorithm finer controls over individual DASH requests.

- **Employ the *preemptive DASH* requests to facilitate urgent DASH requests with significantly closer playout time.** By *preemptive*, we refer to pausing the ongoing transfers on low-priority streams, so as to temporarily allocate all available bandwidth to the urgent tiles on high-priority streams. This allows us to better adapt to viewer dynamics and imperfect fixation predictions.

- **Assign tile quality levels based on the properties of human visual systems.** For example, through careful buffer management, we avoid the playout freezes and black holes. We also cap the temporal/spatial quality jumps and assign higher quality levels to tiles closer to the vision center.

The resulting ABR algorithms are implemented on top of a real preemptive multiplexed DASH streaming system for 360° tiled videos. Our open-source implementation not only allows us to evaluate and fine-tune our ABR algorithms using real experiments, but also enables researchers and engineers to reduce their prototyping efforts when evaluating their multiplexed 360° DASH streaming solutions.

1.1 Contributions

We tackle the aforementioned challenges and make the following contributions:

- **We design and implement a QUIC-based DASH streaming system** on a few open-source projects [39, 11, 6]. We then evaluate our system through real experiments driven by a public HMD viewer dataset [26]. To our best knowledge, 360° tiled video DASH streaming over QUIC has not been experimentally evaluated.

- **We optimize the proposed system by realizing a few key components**: (i) *fixation predictor* that predicts the user viewports in the future, (ii) *tile selector* that maps (future) viewports to tiles for DASH requests, and (iii) *ABR algorithms* that decide video quality level and control prioritized streams.

- **We design an ABR algorithm that is specific for 360° tiled video DASH streaming over preemptive multiplexed streams.** Our experiment results reveal the merits of our proposed ABR algorithm in preemptive multiplexed DASH streaming of 360° tiled videos to HMDs. For example, compared to the state-of-the-art algorithms, our ABR algorithm: (i) averagely reduces the rebuffering counts by up to 3.2 and rebuffering time by up to 2.54 s when the bandwidth is limited, (ii) achieves at most 40.02% higher bandwidth utilization, and (iii) delivers good average V-PSNR at 39–49 dB under 5–15 Mbps bandwidth.

1.2 Thesis Organization

This thesis is structured as follows. We give an introduction and list down the challenges of 360° video streaming in Chapter 1. Chapter 2 provides the background knowledge of 360° videos. Our proposed system architecture is given in Chapter 3. Chapter 4 presents our design decisions and the proposed ABR algorithms. We evaluate our proposed solutions using real experiments in Chapter 5. The related works of streaming protocols and 360° videos are surveyed in Chapter 6. Chapter 7 concludes the contributions of the thesis and discusses future works.

Chapter 2

Background

2.1 360° Videos Pre-processing

360° videos can be generated based on camera arrays or a single camera with multiple lens to cover the entire scenes surrounded by the cameras. For example, 360° images can be acquired by a fisheye camera with a curved mirror. Some algorithms are then further required to stitch two or more images from the cameras into a seamless panoramic image. As a result of the combination of multiple frames, 360 videos become high resolution and high frame rates. However, this makes the video transmission a challenging work due to the high bandwidth requirement and sensitivity to latency to prevent sickness. Therefore, video pre-processing including projecting, encoding, and tiling are required in order to fluently stream 360° videos on the modern Internet.

360° video encoding and projection. To cope with the large file size of 360° videos, some efficient compression algorithms need to be employed to mitigate the network traffic. There are several coding standards for 360° videos such as Advanced Video Coding (H.264/AVC), Scalable Video Coding (H.264/SVC), High Efficient Video Coding (H.265/HEVC), and Scalable High-Efficient Video Coding (H.265/SHVC). Among the coding standards, HEVC is widely used in the literature. Due to the property of 360° videos, viewers are able to view surrounding scenes in arbitrary orientations at any time known as omnidirectional videos. Therefore, 360° videos need to be projected from spherical fields to two-dimensional (2D) image planes for omnidirectional video compression. The coding techniques of omnidirectional videos leveraging different projection strategies are widely studied in the literature [2, 10, 21, 30]. There are two projection strategies commonly used in existing 360° video system, including equirectangular projection (ERP) and cubemap projection (CMP). Besides, in 2016, Facebook detailed a novel projection technique, pyramid projection (PRP). Among the projection strategies, equirectangular projection (ERP) is the most popular one due to its properties of being

both rectangular and straightforward to visualize. Moreover, its relative ease of employment makes it widely supported by both hardware and software. Hence, based on the properties mentioned above, we employ equirectangular projection (ERP) as our default projection.

Tile-based 360° videos. Although 360° videos allow users to view video content in arbitrary orientations, users can only see a small part of the entire videos in the viewport (about one-third of the whole videos). To avoid wasting unnecessary network bandwidth resources, a 360 video can be split into several equal-size pieces known as tiles. Only tiles overlapping with the viewports are sent to viewers, where each tile is independently decodable and a basic transmission unit. The illustration of the tile-based video is shown in Fig. 2.1. The selected tiles are transmitted from the server to the client, and then the player decodes the tiles and render the content to HMD viewers. In this way, more resources can be applied to viewport tiles which significantly improve the video quality and enhance the user viewing experience.

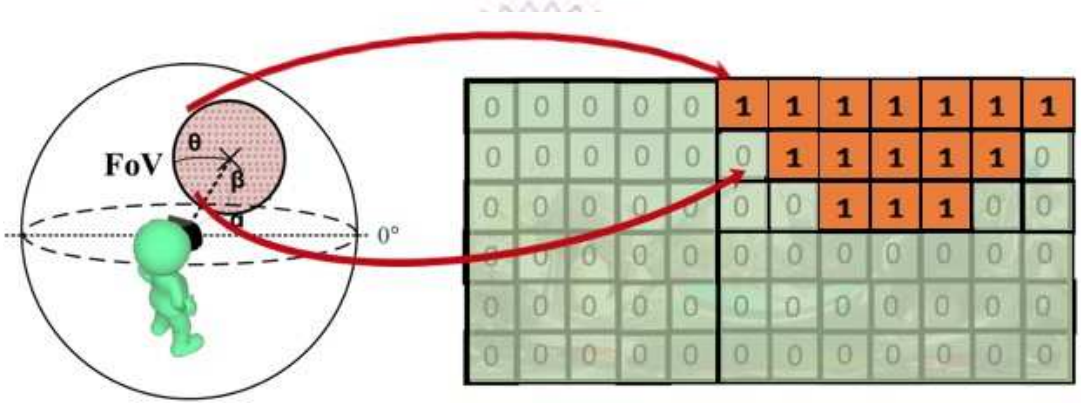


Figure 2.1: Illustration of tile-based 360° videos. A video is cut into 12×6 tiles. A viewer watches 360° videos with HMDs, and only part of the content (mark with 1) is shown inside the viewport.

2.2 360° Video Streaming

There are two main protocols streaming media over the modern Internet. One is Real-Rime Transport Protocol (RTP) over UDP and the other HyperText Transport Protocol (HTTP) over Transmission Control Protocol (TCP). TCP guarantees *data* delivery from the server to users and vice versa. In contrast, UDP is implemented to confirm the *time* delivery from the server to users. Compared to HTTP, RTP is intended for the use in latency-critical scenarios. RTP (RTCP) is a stateful protocol which means that once the connection between the server and the user is built, the information and status of the

user will be continuously checked by the server until the connection is interrupted, in order to make corresponding interactions. Moreover, RTP is a push-based protocol which means that once the session between the user and the server is established, the server can proactively transmit the data to the client without any requests from the user. Although RTP provides streaming over TCP, it is usually streaming over UDP to ensure timely delivery. However, RTP over UDP does not guarantee the data delivery to users. On the other hand, HTTP is a stateless and pull-based protocol. Each request is handled as a unique and independent one-time transaction. For instance, if a user requests the video content from the server over the HTTP protocol, the server transmits corresponding data based on the request. Then the transaction is terminated, and the server is unable to retain any information of the user.

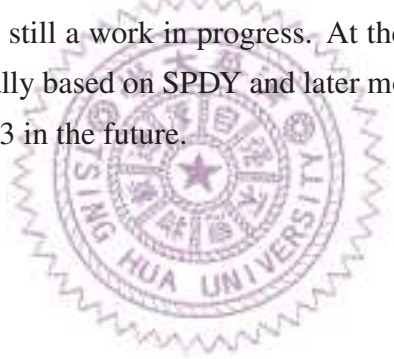
Due to the unreliable transmission of RTP over UDP, most existing systems employ HTTP over TCP for 360° video streaming to make sure that users can receive video content. Moreover, HTTP does not suffer from NAT traversal issue, and media can be sent over Content Distribution Networks (CDNs). To adapt to the dynamic network, Dynamic Adaptive Streaming over HTTP (DASH) is widely adopted. DASH works by cutting the entire video content into multiple segments and then a few segments are sent each time over HTTP instead of the entire video, where each segment is a few seconds long. In general, each segment is encoded with multiple bitrates (video quality) and stored in the server. The information of the segments with available bitrates are then described in the Media Presentation Description (MPD) file. Based on the network bandwidth and the MPD file, the client can select the most suitable video quality level of the required segment to prevent buffer under-run and playout stall.

There are two main versions of the HTTP protocol, including HTTP/1.1 and HTTP/2. HTTP/1.1 is widely used in traditional video streaming. In HTTP/1.1, only one packet can be transmitted in one connection at any time, which means that a packet cannot be delivered until its preceding packet finishes sending. Compared to HTTP/1.1, HTTP/2 supports additional three main features: (i) multiplexing, (ii) stream termination, and (iii) server push. In HTTP/2, each packet is delivered by a stream and one connection can contain multiple streams at the same time referred to as multiplexing. Besides, users can decide the priority of each stream. The streaming order/speed of each stream depends on the weight given to the stream, and the weight can be dynamically updated during the video streaming. HTTP/2 also allows users to terminate the ongoing streams. In terms of server push, only one request is required to request multiple files. In this way, only one round trip time (RTT) is needed for multiple requests, which reduces the network overhead. Due to the popularity of tile-based 360° videos, HTTP/2 is widely used to optimize tile-based 360° video streaming systems.

2.3 QUIC Protocol

In addition to HTTP, there is an emerging experimental transport layer protocol, QUIC, which provides encrypted, multiplexing, and low-latency data transfer. The goal of QUIC is to provide the same service as a TCP connection, but reduce network latency compared to that of TCP. QUIC allows multiple streams of data to reach all the endpoints independently, and thus independent of packet losses do not involve other streams. In contrast, since HTTP/2 runs on TCP, it can suffer head-of-line (HOL) blocking delays during the video transmission if any of the TCP packets are delayed or lost.

QUIC runs on UDP, however, QUIC guarantees the data delivery, where loss packets are retransmitted at the level of QUIC instead of UDP. QUIC was initially created by Google, and made available as part of Chromium project [11]. Since then, it has moved to IETF for standardization. Although most of the concepts of QUIC are decided, all the details are changed in IETF QUIC. Google's implementation is in the process of updating to be IETF compliant. Besides, each stream can also be prioritized in QUIC protocol, but the area of stream priority is still a work in progress. At the wire format level, Google's QUIC priorities were originally based on SPDY and later moved to HTTP/2. They aim to replace HTTP/2 with HTTP/3 in the future.



Chapter 3

Proposed Preemptive Multiplexed System

In this section, we present our proposed architecture. Fig. 3.1 summarizes the components in our proposed streaming system. We classify these components into two parts including DASH server and DASH client, and briefly introduce the individual components below.

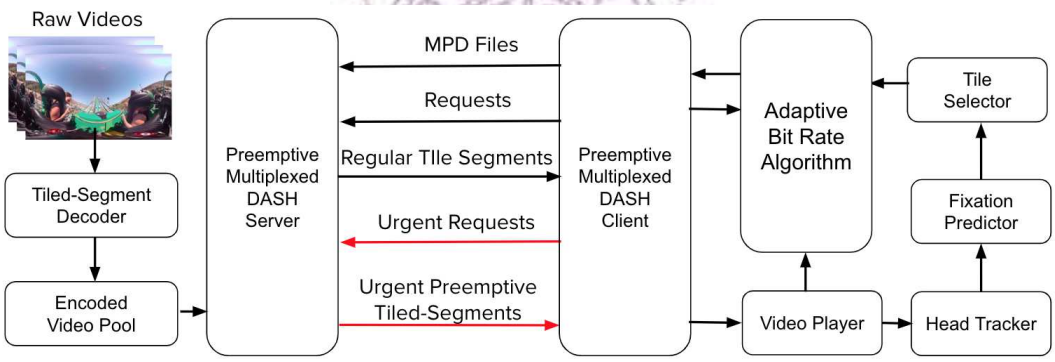


Figure 3.1: The architecture of the 360° video streaming using preemptive multiplexed streams over each DASH session.

3.1 DASH Server

- **Tiled-segment encoder** contains an HEVC encoder and an MPEG DASH content generator, which first cuts each 360° video into tiles, and then compresses them into *tiled segments*. Each tiled segment is: (i) a few seconds long, (ii) independently decodable, and (iii) a basic transmission unit.

- **Encoded video pool** stores the encoded tiled segments of each video at different quality levels, which are determined by control parameters like Quantization Pa-

rameters (QPs), resolutions, or bitrates.

- **Preemptive multiplexed DASH server** is an enhanced web server, which also supports HTTPS/QUIC and schedules the transfer of *tiled segments* based on their priority levels. By a *tiled segment*, we refer to a DASH segment consisting of a single tile which is the basic unit for scheduling. The tiled segments are encoded by the tiled-segment encoder, and each of them is sent through one of the multiplexed streams.

3.2 DASH Client

- **Preemptive multiplexed DASH client** generates the regular/urgent tiled segment requests. The tiled-segment requests are transmitted over the regular and urgent streams within a DASH session, where each stream carries only one tile request.
- **Video player** decodes and shows the 360° videos to the viewer and provides the buffer status to the ABR algorithm.
- **Head tracker** computes the viewer orientations and sends the results to the fixation prediction algorithm.
- **Fixation prediction algorithm** predicts the future viewport center based on the HMD orientation and video content. Different state-of-the-art fixation prediction algorithms [19, 8, 31] may be adopted here.
- **Adaptive Bit Rate (ABR) algorithm** is the core component of our system. It decides: (i) the quality level and priority level of each tiled segment and (ii) the time to request each of them. We present the details of our ABR algorithm design in the rest of the thesis.
- **Tile selector** determines the tiled segments to request based on the fixation prediction results to avoid missing tiles.
- **Tiled-segment decoder** decodes the tiled segments received from the server. It also synchronizes the tiled segments of the same segment duration.

The components operate as follows. Each 360° raw video is encoded and segmented into tiled segments at multiple quality levels. In each streaming session, the ABR algorithm takes inputs from several components (client, video player, fixation prediction algorithm, and tile selector) and determines the priority/quality levels of the tiled segments. The selected tiles are then passed to the DASH client, which are requested as *regular tiled*

segments. Due to sudden head movements (mismatch viewports), some tiles that were not requested earlier may become urgently needed because of the approaching playout time. Therefore these tiles are requested with higher priority as *urgent tiled segments*. To be more precise, by invoking the fixation prediction algorithm multiple times (see Fig. 1.2), our ABR algorithm gets updates on predicted viewports (which are more accurate), and generates the appropriate urgent tiled segment requests.

3.3 Component Designs

3.3.1 Fixation Predictor

Several algorithms can be used for fixation prediction, e.g., Dead-Reckoning (DR) [29] and neural-network [8, 31] based algorithms. We implement a DR algorithm based on speed and acceleration [29]. Let θ^c and ϕ^c be the yaw and pitch (in degrees) of the viewport center. We write their angular speeds at time t as $v_t^{\theta^c}$ and $v_t^{\phi^c}$; and their angular accelerations as $a_t^{\theta^c}$ and $a_t^{\phi^c}$. These values are computed as weighted moving averages, where the sampling rate is 500 ms, and the weight of the latest measurement is 0.9, if not otherwise specified. The DR algorithm predicts the viewport center τ s later than the current time t as:

$$\hat{\theta}^c_{t+\tau} = \theta^c_t + v_t^{\theta^c}\tau + (1/2)a_t^{\theta^c}\tau^2; \tag{3.1a}$$

$$\hat{\phi}^c_{t+\tau} = \phi^c_t + v_t^{\phi^c}\tau + (1/2)a_t^{\phi^c}\tau^2. \tag{3.1b}$$

We note that some DR algorithms omit the acceleration terms. We empirically test both versions with the dataset [26] and find that skipping the acceleration terms results in fewer missing tiles[1]. Thus we report the results from the DR algorithm without considering acceleration.

3.3.2 Tile Selector

The HMD viewport can be described by the center (θ^c, ϕ^c) and the radius r_v. The tile selector finds the required tiles by: (i) creating a viewport plane that is tangent to the sphere at the viewport center, (ii) projecting the points from the viewport to the sphere, (iii) mapping the points from the sphere to the 360° video content, and (iv) identifying the overlapped tiles. The process is however time-consuming, and we propose a real-time algorithm to *approximate* it. We consider the popular equirectangular projection in our discussion, while the same concepts can be applied to other projections.

[1]Probably because there are quite a few sudden head movements.

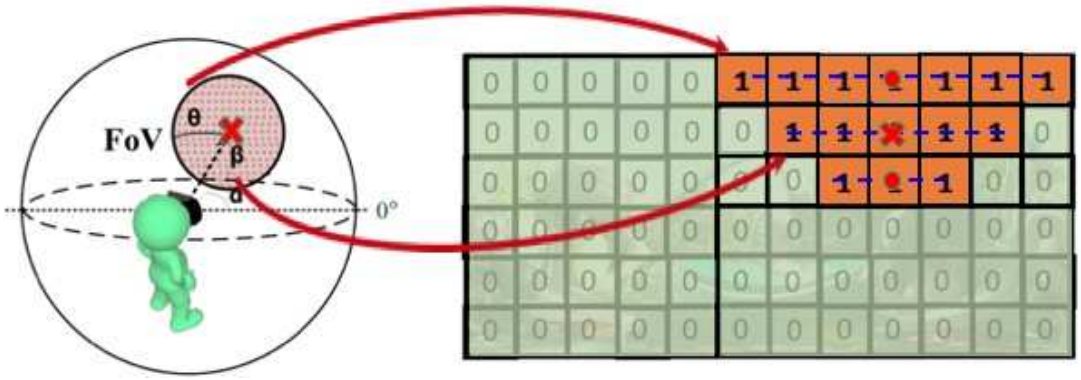


Figure 3.2: Illustration of tile selector process.

The key idea is to approximate the viewport on the video content with the center as (x^c, y^c) and the width as a function of the pitch value ϕ^ϵ as shown in Fig. 3.2. The derivation of the center is straightforward: $x^c = W\frac{\theta^c+180^\circ}{360^\circ}$ and $y^c = H\frac{90^\circ-\phi^c}{180^\circ}$, where $W \times H$ is the resolution in pixel. The pitch values in the viewport are between $\phi^c \pm r_v$, and we approximate the width at $\phi^c - r_v \leq \phi^\epsilon \leq \phi^c + r_v$ as:

$$w(\phi^\epsilon) = \frac{2r_v\sqrt{1-\cot^2 r_v \cdot \tan^2(\phi^\epsilon - \phi^c)}}{\cos\phi^\epsilon}\frac{W}{360^\circ}. \tag{3.2}$$

Having (x^c, y^c) and $w(\phi^\epsilon)$, we can compute the overlapped tile set $\mathbf{S}(r_v)$. More specifically, we start from the center and move up/down at integer number of tiles without exceeding the viewport. At each tile row, we compute $w(\phi^\epsilon)$ to determine the width of the viewport on the video content. In rare cases where $\phi^\epsilon < -90^\circ$ or $\phi^\epsilon > 90^\circ$, we let $w(\phi^\epsilon) = W$. We note that while the viewport radius r_v should be no smaller than that of the HMD; it can be set slightly *larger* to accommodate the imperfect fixation prediction.

Chapter 4

Preemptive Multiplexed Adaptive Bit Rate Algorithms

In this section, we discuss the existing solutions and present our proposed ABR algorithms.

4.1 Existing Solutions

There have been quite a few ABR algorithms [42, 22, 41] that adapt to network dynamics by selecting suitable video quality levels for better user experience. Unfortunately, most existing ABR algorithms are designed for 2D planar video streaming over HTTP/1.1. They are not designed for tiled streaming, and thus cannot allocate diverse quality levels among tiles in the same video frame. Sending *all* the tiles at the same quality level prevents the ABR algorithms from allocating more resources to the tiles in the viewport. That's why a preemptive multiplexed ABR algorithm is required for tiled 360° videos.

We first augment an existing ABR algorithm to adapt to the tiled video streaming and verify the effectiveness of urgent tile streams. After that, we design a new preemptive multiplexed ABR algorithm to further optimize the results.

4.2 Design Criteria

We design our ABR algorithms based on the following criteria:

- Achieve high average video quality. This is crucial to the visual quality experience.
- Avoid large quality jumps. In addition to maximizing the average video quality of the tiles in the viewport, avoiding large spatial/temporal jumps is also important. This is because large quality jumps negatively affect the viewing experience [7].

Fig. 4.1 shows the illustration of the high spatial quality jump. We see that the video quality at the upper side is relatively lower than that of at the other area, which severely degrades the user viewing experience.

- Avoid buffer under-run. Once the video starts playing, buffer under-run leads to playout stalls or black holes, which severely affect viewer experiences [7]. Some minimal buffer occupancy should be maintained and the ABR algorithm should be responsive to network and viewer dynamics.

Figure 4.1: Illustration of the large spatial quality jump.

While achieving each of the three optimization criteria is already difficult, the real challenge comes from the complex interplay among them. In this thesis, our proposed ABR algorithm focuses on striking a good balance among the three criteria.

4.3 AE ABR algorithm

We augment an existing ABR algorithm to design our first solution denoted as *AE*. Our AE ABR algorithm consists of two flows: (i) *regular tile* flow, which is event triggered and (ii) *urgent tile* flow, which is time triggered. More precisely, the regular tile flow is activated only when: (i) all previously requested regular tiled segments are received and (ii) the current buffer occupancy doesn't exceed the buffer high watermark B_h, which is a system parameter. Otherwise, the regular tile flow sleeps until both conditions are met.

The urgent tile flow is triggered once every U s if the current buffer occupancy is not lower than the buffer low watermark B_l, where the *urgent window* U and B_l are system parameters.

Upon being invoked at time t, the urgent tile flow requests for the tiled segments from $\mathbf{S}(r_v)$ in time $[t, t+U]$ that are not yet received. These urgent requests are made at a high priority. The quality level of the urgent tiled segments is determined by the throughput estimation T. T is calculated as the weighted moving average of the download size over the time period U, where the weight of the latest measurement is 0.9, if not otherwise specified. Given T, we set the urgent tiled segments at the highest possible quality level, while ensuring these tiled segments can be downloaded within the urgent window U. If the total segment size of the lowest quality level still exceeds $T \cdot P$, we skip tiled segments from the viewport edge until the total segment size fits the throughput.

The regular tile flow is built upon existing ABR algorithms [22], and we employ a popular buffer-based algorithm [17] denoted as *NETFLIX* in our discussions. NETFLIX only considers buffer occupancy to decide the suitable video quality level. Because we adopt a *strict priority scheduler* in QUIC, the urgent tiled segments can always be sent before the regular ones. Let B_o be the buffer occupancy (in time) and D be the total download time of urgent tiled segments between now and the preceding triggered time of the regular tile flow. Using D to approximate the time occupied by urgent requests in the upcoming round of regular requests, we consider $B_o - D$ as the *effective* buffer occupancy and pass it to the adopted ABR algorithm. The regular tiled segments are requested at low priority. Last, we note that the above principles can be augmented and applied to throughput-based and hybrid ABR algorithms [22].

4.4 PM ABR algorithm

4.4.1 Design Decision

Diverse tile priority levels based on proximity to the fixation center. Although all tiles in the viewport are visible by users, human visual systems have diverse sensitivity levels to tiles with different distances to the fixation center [16, 44]. We assume the fixation point is at the center of the viewport. This assumption can be lifted if an HMD with the eye-tracking capability [9] is adopted. Based on the survey of peripheral vision, Strasburger et al. [44] show that human can clearly see a colorful image only in a small 10° radius circle around the center and the visual acuity decreases as the radius increases. Based on this, we divide the viewport into three areas: foveal, intermediate, and outer areas as shown in Fig. 4.2. Tiles in the foveal area are assigned the highest quality and the highest streaming

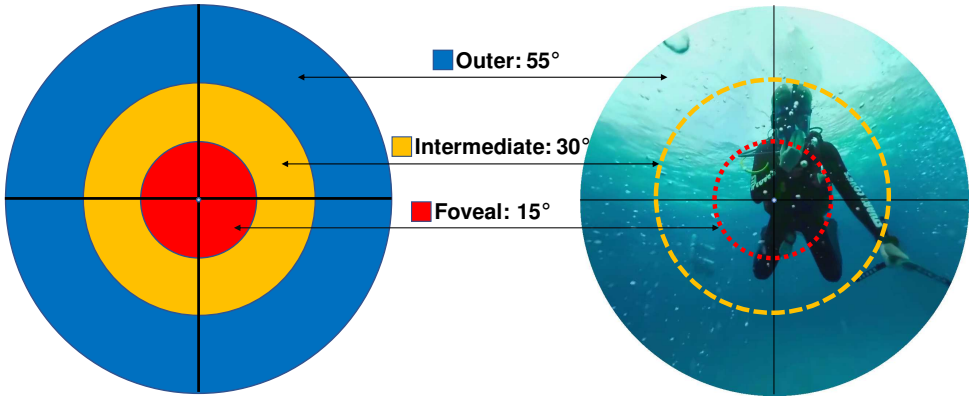


Figure 4.2: Illustration of three areas in the viewport with radii of 15°, 30°, and 55°. The radii are sample values from Oculus Rift [37], which can be adjusted by the system administrators.

priority; the tile in the intermediate area are assigned lower quality and priority; and the tiles in the outer area have the lowest quality and priority.

Parallel and independent flows for regular and urgent requests. Unlike traditional ABR algorithms that have a sequential flow. In this work, we employ two independent flows for: (i) regular, and (ii) urgent tiled segments. To be more precise, the regular flow aims to maintain buffer occupancy to prevent buffer under-run. The regular flow is event triggered. It is activated only when the following two conditions are both met: (i) all previous requested regular tiled segments are received and (ii) the current buffer occupancy does not exceed the buffer high watermark B_h. B_h is a system parameter to prevent buffer over-run. The urgent flow kicks in when some tiles in the future viewports are bound to be missing, which arise from imperfect fixation predictions. The urgent flow is time triggered and invoked every *urgent window* that is U s long, if the current buffer occupancy is not lower than the buffer low watermark B_l. Both U and B_l are system parameters. Capitalizing the advantage of multi-round fixation predictions, upon being invoked at time t, the urgent flow requests for the urgent tiled segments in the playout time $[t, t + U]$, which are not yet requested in the regular flow. Notice that, tiles of urgent requests will always have higher priority than tiles of regular requests during the video streaming.

Preemptive multiplexed streams to ensure timely delivery of urgent segment tiles. We opt for the preemptive multiplexed streaming over the weighted (non-preemptive, concurrent) one for two reasons. First, the urgent tiled segments typically have much closer playout time than most of the outstanding regular tiled segments. Second, each tiled segment is decodable only if it is *completely* received. For example, two half-completed tiled segments are totally useless; while a completed tiled segment can be decoded and shown

to the viewer. Therefore, preemptive multiplexed streaming is employed to maximize the number of *decodable* tiled segments with approaching playout time.

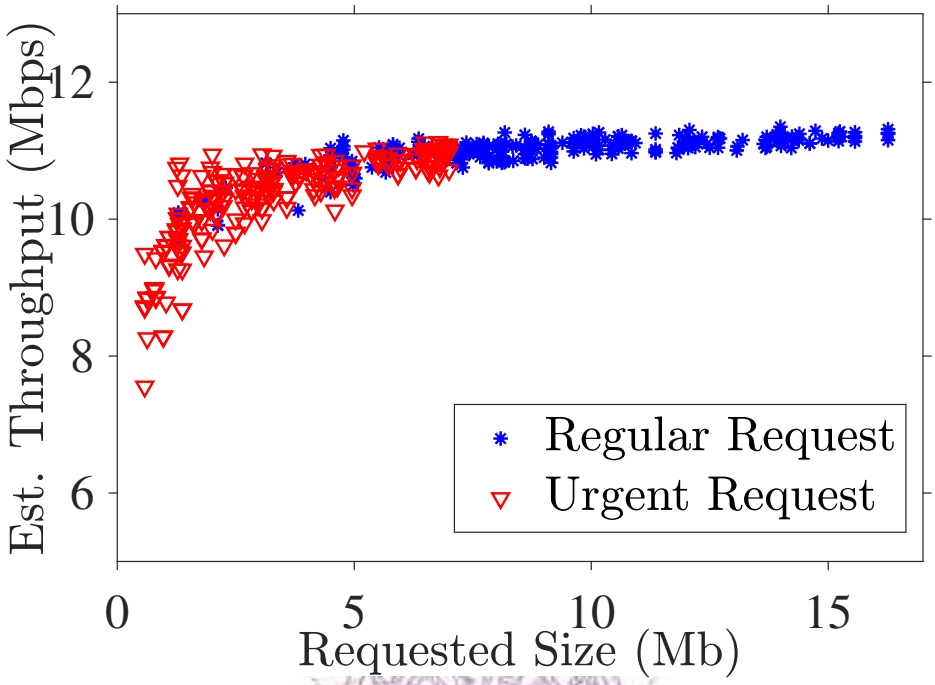


Figure 4.3: Throughput estimation with different requested size under 12 Mbps network bandwidth.

Estimating the network throughput with regular requests only. Throughout estimation is done by the DASH client when receiving the tiled segments. In our pilot experiments, we observe that the throughput estimation from the urgent requests often underestimates the available throughput. We record the estimated throughput over a 12 Mbps link (enforced by the Linux `tc` command) in Fig. 4.3. The figure illustrates that the estimated throughput deviates more from the ground truth (12 Mbps) when the requested size is smaller. Since urgent flow contains the missing tiles in the viewport, urgent requests usually contain fewer tiled segments and are smaller. Insufficient workloads could lead to the underestimation of available throughput. Hence, we only estimate the throughput using regular (larger) requests. We note that similar observations were reported in the literature [47].

4.4.2 Pseudo Code of PM ABR Algorithm

Overview. Table 4.1 summarizes the notations used in the thesis. Fig. 4.4 presents the block diagram of our proposed preemptive multiplexed (PM) ABR algorithm. The PM algorithm gets the latest throughput estimation T and prior requests information from the

Table 4.1: Notations Used in The Thesis

Symbol	Description
T	Estimated throughput
r_v	Viewport radius
B_h	High watermark
B_l	Low watermark
U	Urgent window size
J	Maximum quality jumps
V_c	Viewport center
B_o	Buffer occupancy
B_t	Target buffer occupancy
Q_i	Quality level of segment i
$S(V_c, r_v)$	Output set from tile selector given V_c and r_v
R_i^r	Regular request set of segment i with $<$ tile, quality, priority $>$ tuples
S_i^r	Regular tile set of segment i
R_i^u	Urgent request set of segment i with $<$ tile, quality, priority $>$ tuples
S_i^u	Urgent tile set of segment i
b_j^q	Bitrate of tile j at quality q

DASH client. The PM algorithm also gets the current buffer occupancy B_o from the video player and the predicted viewport center V_c from the fixation prediction algorithm. With these inputs and the system parameters from the system administrator, The PM algorithm generates the results and passes the segment number, tile numbers, quality levels, and priority to the DASH client. There are four levels of tile priority (from high to low):

1. Tiles in the foveal area for the urgent requests.
2. Tiles in the intermediate area for the urgent requests.
3. Tiles in the outer area for the urgent requests.
4. Tiles for the regular requests.

Moreover, we need to determine the tile set within a given viewport. Making *tile selector* as a function, for viewport center V_c and viewport radius r_v, the overlapped tile set is written as $S(V_c, r_v)$.

Tiled segments in the regular flow. We give the pseudo code of the regular flow in Algorithm 1. For segment i, given viewport center V_c and radius r_v, we obtain the

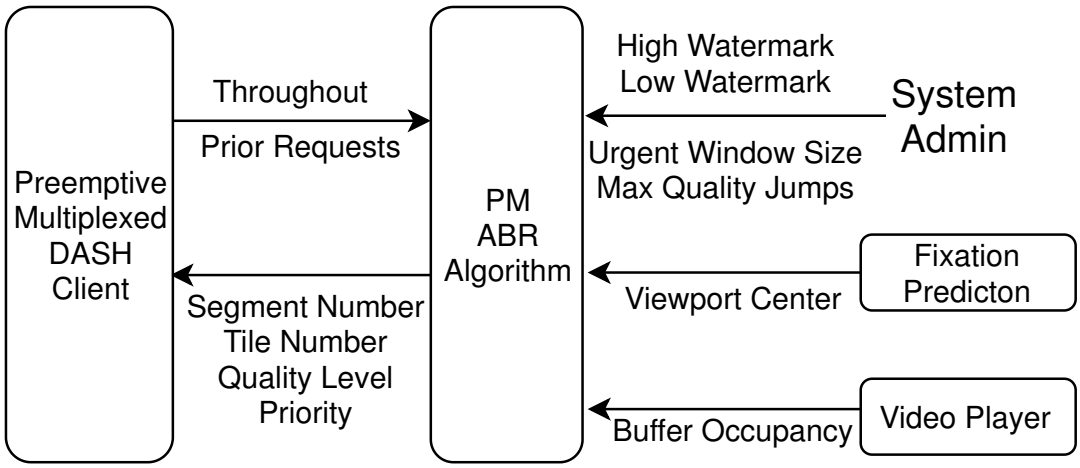


Figure 4.4: Block diagram of our proposed PM algorithm.

overlapped regular tile set S_i^r from $S(V_c, r_v)$ (line 3). Then total download budget T_b is computed, which considers T, B_o, B_t, and D (line 4). Here, B_t is the target buffer occupancy and D is the total download time of urgent requests between now and the preceding invocation time of the regular flow. This step is to the maintain buffer occupancy at B_t, in order to prevent buffer under-run. Moreover, B_h prevents buffer over-run. The quality Q_i is the maximum quality determined by summing bitrate of tiles in S_i^r at quality $q_{q\in[0:Q_{i-1}+J]}$ with the total the bitrate less than T_b, where J is the maximum quality jumps (line 5). The constraint prevents viewers from suffering large temporal quality changes. The information of each tile is added to R_i^r, which is a regular request set of segment i with <tile, quality, priority> tuples (line 6). Then, T_b is computed as the delta between the total throughput and the sum of the tile bitrates (line 7). Let T_b be the residual download budget, we increase r_v by 5° each time and decide the quality of tiles that have not yet assigned (lines 8-18). The steps avoid black holes in the viewport. Note that, the quality of these tiles are limited $[Q_i - J : Q_i]$, which is to prevent large spatial quality jumps due to imperfect fixation predictions. Last, we pass S_i^r and R_i^r to the DASH client, which then submits the requests.

Tiled segments in the urgent flow. We give the pseudo code of the urgent flow in Algorithm 2. For segment i, tile sets of the three regions, *foveal*, *intermediate*, and *outer*, are generated based on V_c, the corresponding region sizes, and S_i^r (lines 4-7). The tiles in three sets are first assigned quality at $Q_i - J$. They are then added to R_i^u at the predefined priority levels, where R_i^u is the urgent request set of segment i with $<$ tile, quality, priority$>$ tuples (line 8). This initial assignment guarantees that all tiles can be streamed to the client earlier and the different priority levels ensure that more important tiles are streamed to viewers sooner. The remaining T_b is then computed considering U

Algorithm 1 Regular Flow

1: Input: T, J, V_c, B_o, i
2: Output: S_i^r, R_i^r

3: $S_i^r \leftarrow S(V_c, r_v)$
4: $T_b \leftarrow T \times (B_o - D - B_t)$
5: $Q_i \leftarrow \mathbf{m}ax_{q \in [0:Q_{N-1}+J]}\, q$ s.t. $\sum_{j \in S_i^r} b_j^q \leq T_b$
6: $R_i^r = \{(j, Q_i, 4)\, \forall j \in S_i^r\}$
7: $T_b \leftarrow T_b - \sum_{j \in S_i^r} b_j^{Q_i}$
8: **while** $T_b > 0$ **do**
9: $\quad r_v \leftarrow r_v + 5^\circ$
10: $\quad S_t \leftarrow S(V_c, r_v) \setminus S_i^r$
11: $\quad Q^t \leftarrow \mathbf{m}ax_{q \in [Q_i - J:Q_i]}\, q$ s.t. $\sum_{j \in S_i^r} b_j^q \leq T_b$
12: $\quad$ **if** Q^t exits **then**
13: $\quad\quad S_i^r \leftarrow S_i^r \cup S_t$
14: $\quad\quad R_i^r = R_i^r \cup \{(j, Q^t, 4)\, \forall j \in S_t\}$
15: $\quad\quad T_b \leftarrow T_b - \sum_{j \in S_i^r} b_j^{Q^t}$
16: $\quad$ **else**
17: $\quad\quad$ **break**

and the total bitrate allocated to the tiles (line 8). If there is still T_b left, we upgrade the quality from the *foveal* to the *outer* area; and the highest possible quality level is limited at Q_i to avoid large spatial quality jumps (lines 10-17). Last, we pass S_i^u and R_i^u to the DASH client, which then submits the requests.

Algorithm 2 Urgent Flow

Input: T, i, J, V_c, Q_i, U, S_i^r

Output: S_i^u, R_i^u

$F(i)$, tile sets of foveal (0), intermediate (1), and outer (2)

$foveal = 15°$, $intermediate = 30°$, and $outer = 55°$

$F(0) \leftarrow S(V_c, foveal) \setminus S_i^r$

$F(1) \leftarrow S(V_c, intermediate) \setminus (S_i^r \cup F(0))$

$F(2) \leftarrow S(V_c, outer) \setminus (S_i^r \cup F(0) \cup F(1))$

$S_i^u \leftarrow F(0) \cup F(1) \cup F(2)$

$R_i^u \leftarrow \{(j, Q_i - J, n+1)\, \forall j \in F(n) \text{ and } n \in \{0,1,2\}\}$

$T_b \leftarrow T \times U - \sum_{j \in S_i^u} b_j^{Q_i - J}$

for n in $\{0,1,2\}$ **do**

    $Q_t \leftarrow \mathbf{m}ax_{q \in [Q_i - J : Q_i]}\, q$ s.t. $\sum_{j \in F(n)} b_j^q \leq T_b$

    if Q^t exits **then**

        $T_b \leftarrow T_b - \sum_{j \in F(n)} b_j^{Q^t}$

        Update $R_i^u \leftarrow \{(j, Q_t, n+1)\, \forall j \in F(n) \text{ and } n \in \{0,1,2\}\}$

    else

        break

Chapter 5

Implementations and Evaluations

In this section, we evaluate our solution through a real testbed.

5.1 Implementations

We build a preemptive multiplexed DASH streaming testbed of 360° tiled videos using the following open-source projects: (i) *QUIC client/server* in Chromium [11], which includes a QUIC stack written in C++, (ii) *AStream*, which is a python-based DASH player including several ABR algorithms. Our PM, AE, and other baseline ABR algorithms are implemented on this testbed for evaluations. Since the player and QUIC client are run on multiple processes, `ipc` messages and *shared memory* [25] are employed to pass data back and forth. Moreover, we extend the MPD files to transmit individual tiled segment sizes at various quality levels, which are needed by the PM algorithm.

5.2 Setup

We execute the DASH client and server on two workstations with Intel i7 CPU and 16 GB RAM running Ubuntu 18.04. The two workstations are connected by our wired campus Intranet. We leverage `tc` [3] in Linux to emulate the diverse network conditions. Moreover, `tcpdump` [28] is used to collect network traces during video streaming for analysis. In order to compare different ABR algorithms, we adopt a public 360° dataset [26]. In particular, we randomly select 10 users among 50 users watching 10 videos from the dataset. Each video lasts for 60 s. Figures plot the average results with 95% confidence intervals over 10 runs if possible. In addition to our two ABR algorithms, we also implement the other baseline ABR algorithm [40] (denote as PSHD17). We focus on the ABR algorithm without considering the server push. Following their spirit, each frame is divided into three regions: viewport $(55°)$, adjacent $(65°)$, and background (the rest). All the tiles are

first assigned the lowest quality level; they are upgraded to the highest possible quality from the viewport to the background until the available throughput is used up.

Several parameters are varied in our experiments. From the network aspects, we set the end-to-end network bandwidth $C \in \{12, \mathbf{10}, 8\}$ Mbps, network delay $d \in \{\mathbf{5}, 10, 30\}$ ms, and packet loss rate $L \in \{0.5, \mathbf{1}, 2\}\%$, respectively. Bold font is used to indicate default value. In terms of system parameters, the initial buffer time is set to be 2 s, urgent window size $U = 0.5$ s, buffer high/low watermarks $B_h = 3$ s, $B_l = 1$ s and viewport radius $r_v = 55°$. The videos are encoded at 9 quality levels $\{15, 14, 12, 11, 10, 9, 8, 6, 5\}$ Mbps with 10×10 tiles. We evaluate the results by considering the following metrics:

- *Throughput utilization.* The ratio between the throughput and the available bandwidth.
- *Missing ratio.* The fraction of tiles within the viewport (based on the ground truth) that are not received in time.
- *Video quality.* The Viewport-PSNR (V-PSNR) [51] values based on the ground truth and we set the viewport radius as 50° as measured in Fan et al. [8].
- *Rebuffering time and counts.* The playout freeze duration and instances due to buffer under-run.

5.3 Results

5.3.1 Benefit from QUIC Protocol

We first compare the DASH streaming quality over different protocol stacks: HTTP/1.1, HTTP/2, and QUIC. We focus on the performance results of rebuffering counts/time from a random user; results from other users are similar. For fair comparisons, HTTP/2 and QUIC do not enable their prioritized schedulers. We instruct our DASH client to request the tiles overlapping with the ground-truth viewports at the highest quality level. We consider different L and C values and plot the results in Fig. 5.1. This figure reveals that HTTP/1.1 gives too many rebuffering counts: as high as 39 times in each 60 s video. This can probably be attributed to the fact that HTTP/1.1 doesn't support multiplexing. Figs. 5.1(c) and 5.1(d) also confirm that HTTP/1.1 leads to high rebuffering counts/time, even when the network bandwidth is less limited. Moreover, these two figures also reveal that HTTP/2 is slightly more sensitive to packet loss rates; when the packet loss rate is 2%, HTTP/2 suffers from buffer under-run. This can be explained by the head-of-line blocking, extra retransmission packets, and longer latency caused by the underlying TCP

protocol. In summary, compared to QUIC, streaming over HTTP/1.1 and HTTP/2 results in higher rebuffering counts and longer rebuffering time. Hence, we no longer consider HTTP/1.1 and HTTP/2 stacks in the rest of the experiments.

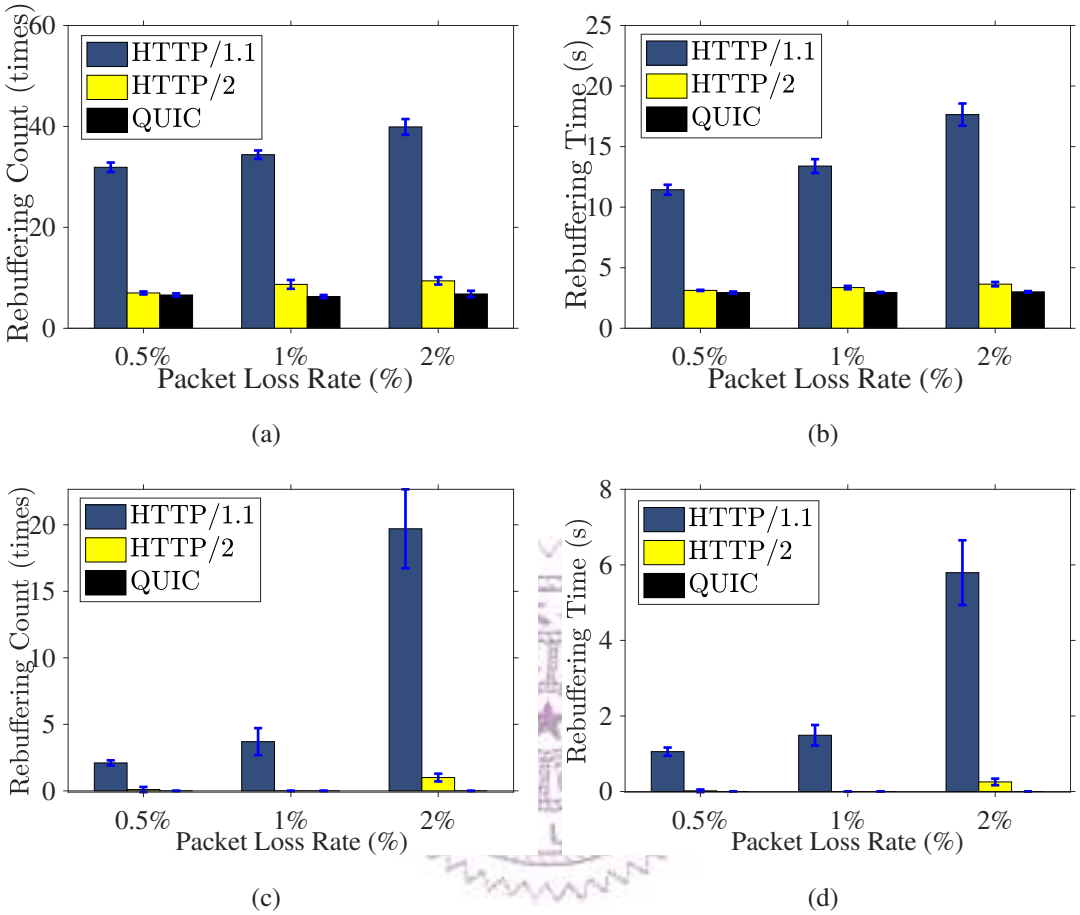


Figure 5.1: Comparison among protocol stacks under 5 Mbps network bandwidth: (a) rebuffering counts and (b) rebuffering time; under 8 Mbps network bandwidth: (c) rebuffering counts and (d) rebuffering time.

5.3.2 Effectiveness of Urgent Requests

We next compare the streaming performance with and without urgent tile streams, denoted as *AE* and *NETFLIX* respectively, and plot the results in Fig. 5.2. Fig. 5.2(a) presents the V-PSNR over time of a sample user on a sample video under 10 Mbps network bandwidth. This figure clearly shows the superior performance with urgent tiles in video quality. We next consider more metrics under diverse network bandwidth in Figs. 5.2(b). These figures show that our urgent tile streams are useful under diverse network bandwidth: as high as 21.5% missing ratio drop and 15.29 dB V-PSNR improvement on average are observed without excessive bandwidth utilization. To understand if the above observations

hold across users with diverse head movement patterns, we plot the aggregate results of individual users in Figs. 5.3. These two figures confirm that urgent tile streams are effective for all users. Furthermore, we also plot the aggregate results of individual videos in Fig. 5.4, and confirm that urgent tile streams work for all videos. *In summary, our urgent tiles effectively and constantly reduce missing ratios and increase the video quality without incurring excessive bandwidth utilization under different network bandwidth, user behavior, and video characteristics.* Last, we note that no rebuffering events are observed in these experiments.

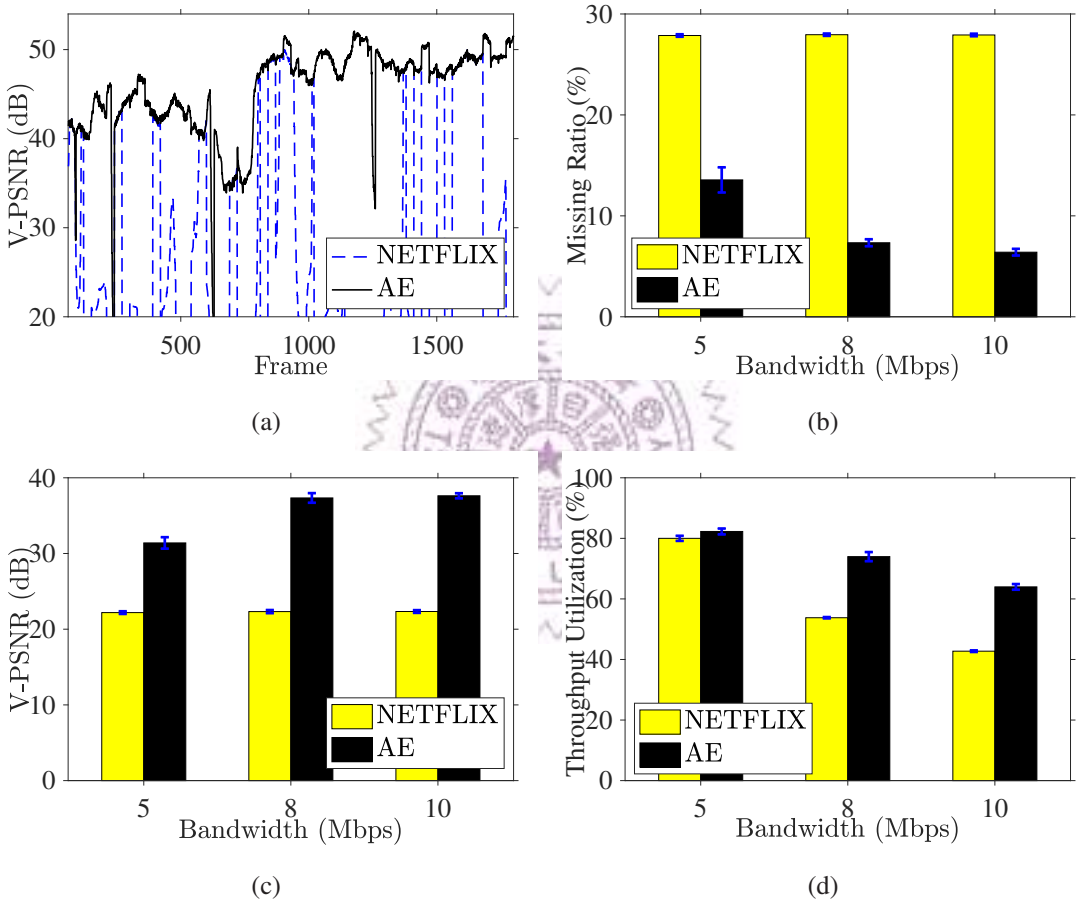


Figure 5.2: Effectiveness of our urgent tile streams, results from a sample user and a sample video: (a) V-PSNR over time at 10 Mbps, (b) average missing ratio, (c) average V-PSNR, and (d) average bandwidth utilization.

5.3.3 Enhancing Viewing Experience by the PM ABR Algorithm

Our proposed PM algorithm strikes a balance between video quality and buffer under-run. After verifying the effectiveness of urgent tiles, we next consider *PM* ABR algorithm and *PSHD17* to compare the results. We first evaluate the performance of the

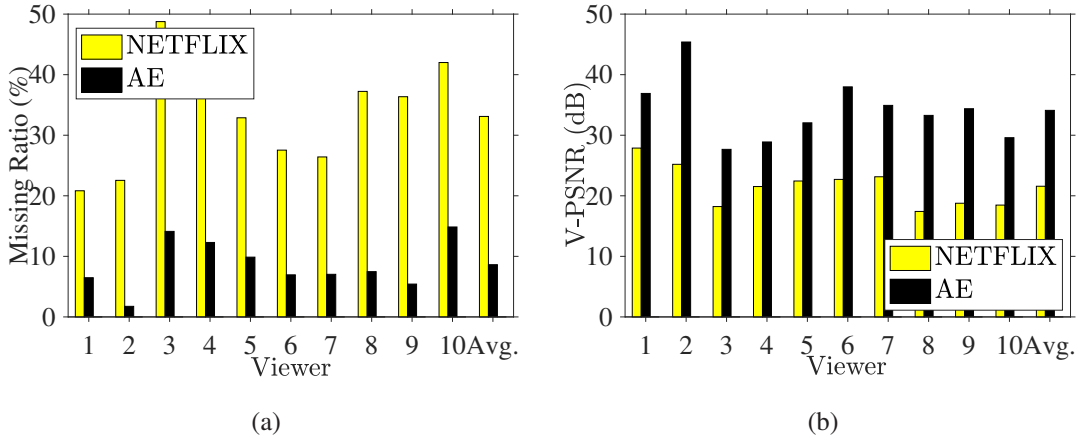


Figure 5.3: Effectiveness of our urgent tile streams, results from different users (with all videos): (a) average missing ratio and (b) average V-PSNR.

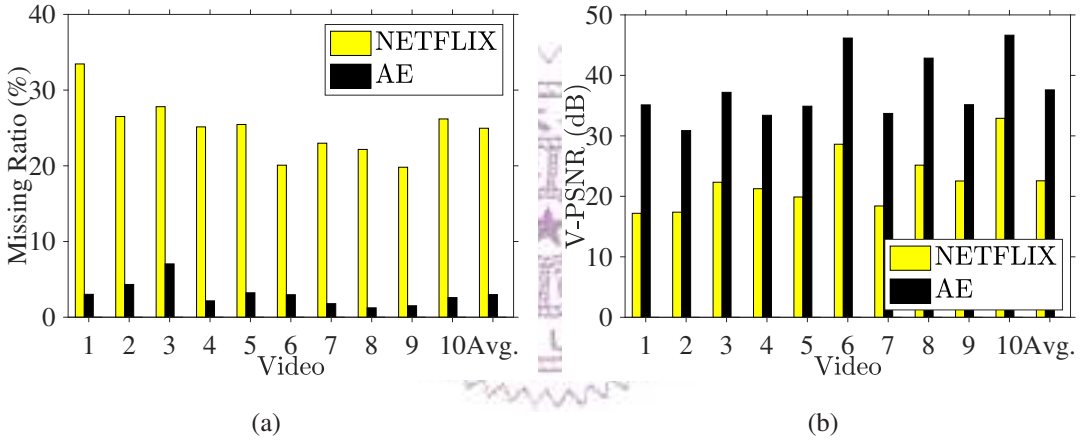
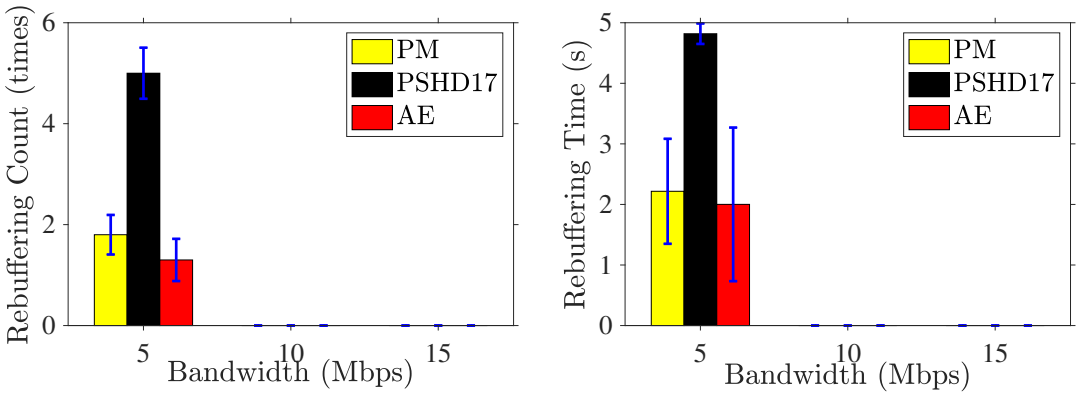


Figure 5.4: Effectiveness of our urgent tile streams, results from different videos (with all users): (a) average missing ratio and (b) average V-PSNR.

ABR algorithms under different network bandwidth settings. In this experiment, we randomly choose one sample viewer watching video *Shark Shipwreck*. We repeat the DASH streaming session ten times and report the results in Fig. 5.5. From Fig. 5.5(a), we see that under 5 Mbps, there are some playout stalls: the rebuffering counts are 1.8 times for PM, 5 times for PSHD17, and 1.3 times for AE on average. A similar trend is also observed in the rebuffering time results in Fig. 5.5(b). These two figures show that our PM algorithm does not suffer from high rebuffering counts/time compared to PSHD17 that always transmits *all* tiles, which leads to too much network traffic.

In Fig. 5.6(a), we observe some missing ratios in the viewports. More precisely, the missing ratios of 3.95% and 10.09% are recorded with the PM and AE algorithms under 5 Mbps network bandwidth. This reveals that AE aggressively skips some of the tiled

segments that are less likely to be watched. This may back-fire at AE for higher missing ratios. In summary, the proposed PM algorithm achieves a good balance between video quality and buffer under-run, compared to the baselines. Last, we note that the PM algorithm makes good use of the available throughput as shown in Fig. 5.6(b).

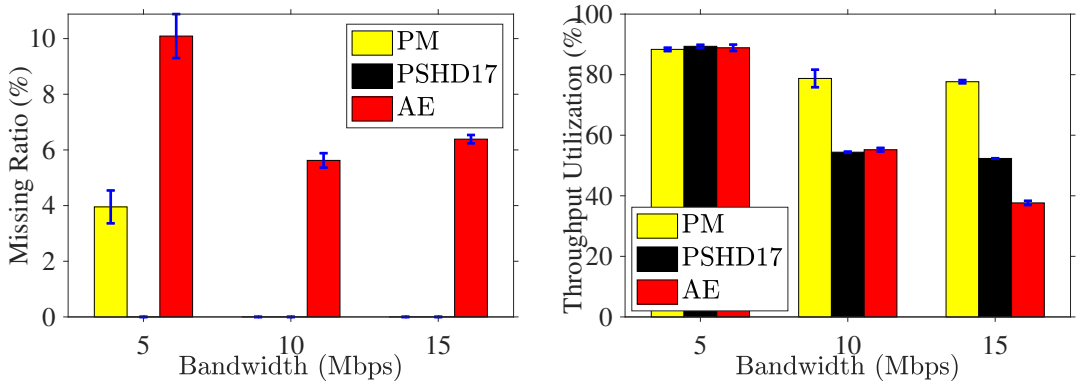


(a) (b)

Figure 5.5: Comparisons of rebuffering counts (time) among ABR algorithms under 5, 10, and 15 Mbps network bandwidth.

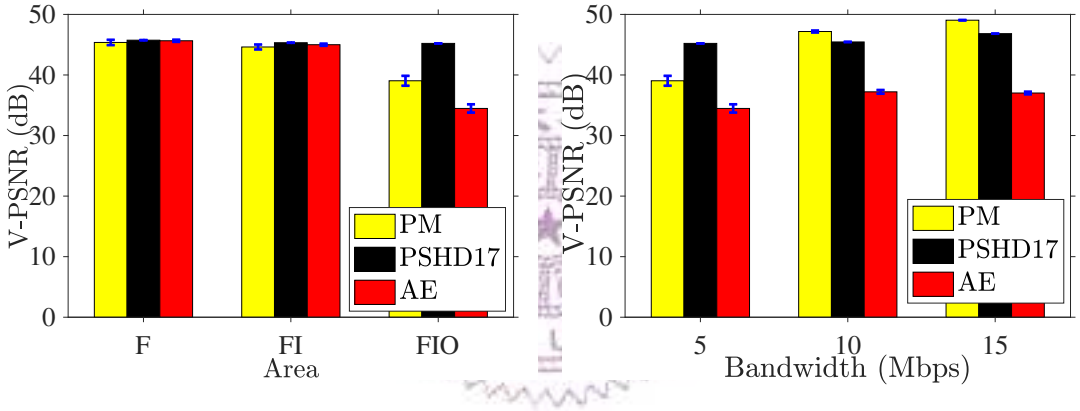
Our proposed PM algorithm leads to good video quality unless the network bandwidth is highly constrained. We report the V-PSNR in Fig. 5.7. Fig. 5.7(a) gives the V-PSNR in different areas (Foveal, Intermediate, and Other). We find that PSHD17 trades frequent buffer under-run (Figs. 5.5(a) and 5.5(b)) for slightly higher video quality, when all three regions are considered. This is not a huge concern for our PM algorithm, in fact, its video quality levels of (the more noticeable) foveal and intermediate regions are the same as those of PSHD17. Fig. 5.7(b) plots the video quality of the whole viewport under 5, 10, and 15 Mbps bandwidth. This figure reveals that the proposed PM algorithm outperforms the baselines under 15 Mbps (or more) bandwidth. At 15 Mbps, the achieved V-PSNR values are 49.04 dB for PM, 46.82 dB for PSHD17, and 37.02 for AE on average. The gap of at least 2.22 dB is a nontrivial boost. We note that the edge of our PM algorithm diminishes when the bandwidth is limited. This may be attributed to the encoded video bitrate of 7.73 Mbps, which may be too high to 5 Mbps bandwidth. PSHD17 achieves higher video quality *after* spending 5 s (5/60 = 8.33% of the session) on rebuffering, which results in degraded viewer experience.

The PM algorithm adapts to different viewers and videos. We first randomly select ten viewers watching *Shark Shipwreck*. Fig. 5.8 shows the results under 10 Mbps network bandwidth. In Fig. 5.8(a) we observe that the PM algorithm consistently utilizes available throughput with an average of 85.46%. In addition to zero missing ratios shown in Fig. 5.8(b), the PM algorithm also provides the best quality to all viewers among three



(a) (b)

Figure 5.6: Comparisons among ABR algorithms under 5, 10, and 15 Mbps network bandwidth: (a) missing ratio, and (b) throughput utilization.



(a) (b)

Figure 5.7: Comparisons of V-PSNR among ABR algorithms: (a) in different areas at 5 Mbps and (b) the whole viewport under 5, 10, and 15 Mbps.

ABR algorithms as revealed in Fig. 5.8(c): 47.20 dB for PM, 46.00 dB for PHSD17, and 34.28 dB for AE on average. We next select a viewer watching 10 different videos, and report the results in Fig. 5.9. The results show similar trends. We conclude that the PM algorithm adapts to various videos and viewers well. Compared to AE, our PM algorithm dynamically increases the viewport radius to achieve better viewing experiences.

The PM algorithm maintains a good video quality level. Fig. 5.10 reports the video quality levels over time under different network bandwidth. Fig. 5.10(a) plots the results at 5 Mbps network bandwidth. We see that although the PM algorithm sometimes suffers from slightly lower video quality, compared to AE, the PM algorithm recovers from those unfortunate events pretty fast. That is, the PM algorithm achieves video quality levels comparable to those of PSHD17 most of the time. When the network bandwidth is

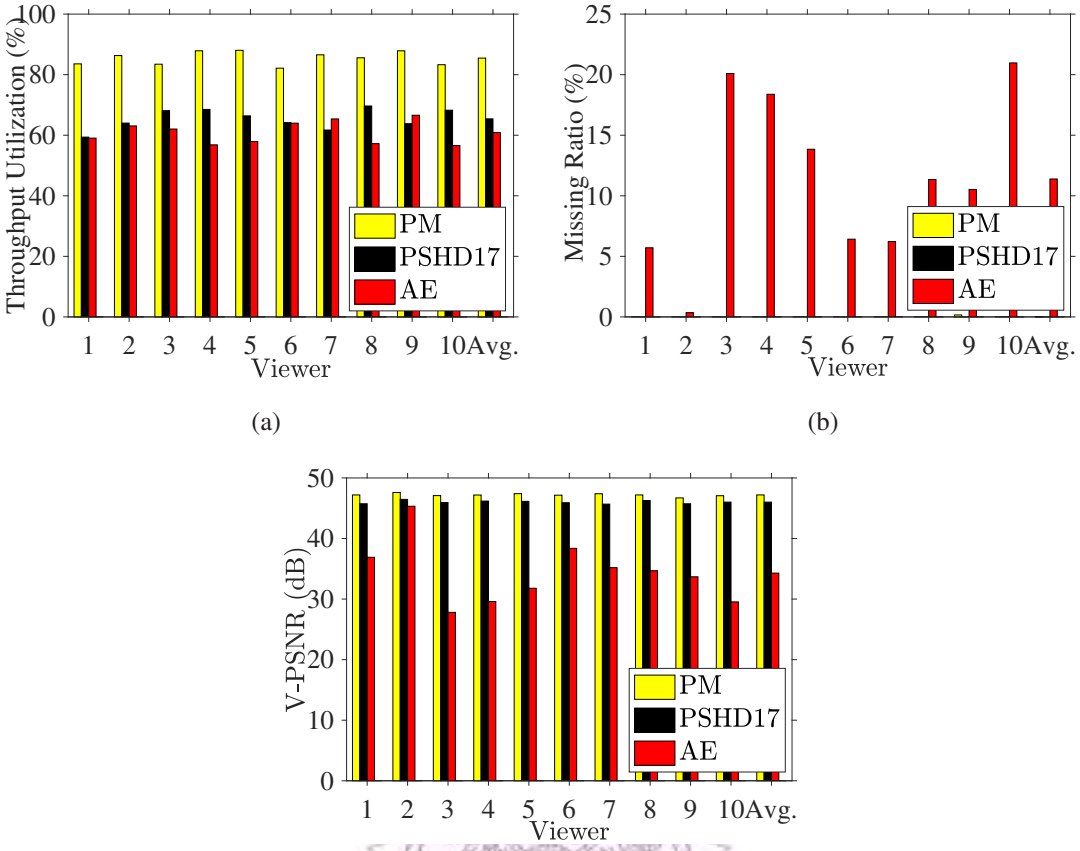


Figure 5.8: Comparisons among ABR algorithms from 10 different viewers under 10 Mbps network bandwidth: (a) average throughput utilization, (b) average missing ratio, and (c) average V-PSNR.

sufficient, PM constantly delivers good quality without large quality jumps as evidenced in Figs. 5.10(b) and 5.10(c).

5.3.4 Comparation of Imperfect Fixation Prediction

Among the causes of missing tiles, the imperfect fixation prediction (due to complex video content and user behavior) is the main one that may be further improved. More specifically, we next consider different viewport radius r_v values, because larger r_v results in more requested tiles, which essentially build some *cushion/buffer* zones to accommodate imperfect fixation predictions. Fig. 5.11 gives the missing ratios under different r_v at 10 Mbps. Fig. 5.11(a) gives the results from 1 sample user with a video, and Fig 5.11(b) gives the sample results from 10 users with a video; while Fig. 5.11(c) gives the results from 10 videos with a user. We observe that Different viewport radii have a very slight influence on PM. However, in terms of AE, there is a large gap between $52.5°$ and $55°$,

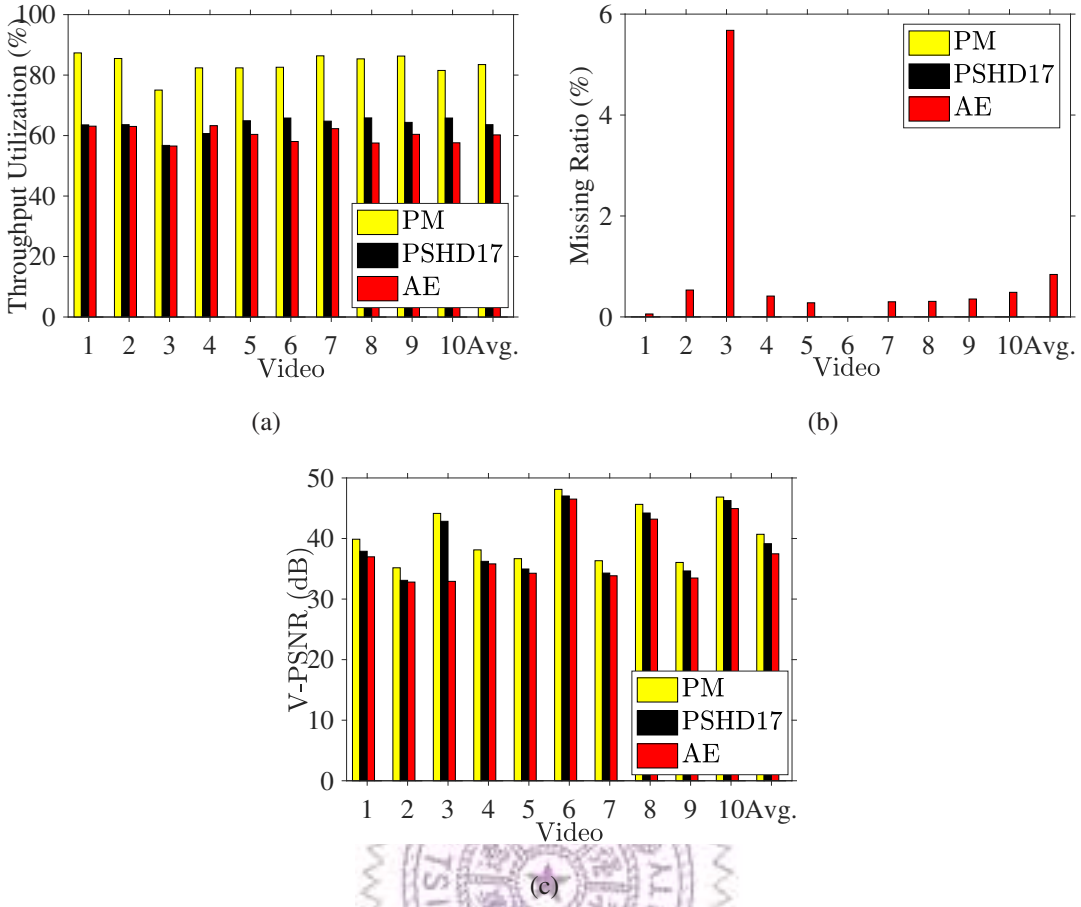


Figure 5.9: Comparisons among ABR algorithms from 10 different videos under 10 Mbps network bandwidth: (a) average throughput utilization, (b) average missing ratio, and (c) average V-PSNR.

while the slopes beyond 55° are flatter. The average V-PSNR values from $52.5°$ is at least 2.74 dB lower than that of $55°$.

5.3.5 Implication of Urgent Window Size

Next, we study how urgent window size affects the performance. We plot the results under different urgent window size U in Fig. 5.12. This figure reveals that frequently activating the urgent flow does not necessarily lead to low missing ratios. More precisely, the activation rate is empirically observed at a U value of about 0.5 s. The higher missing ratios when U is reduced could be attributed to: (i) shorter/more fragmented urgent window size may limit the (integer) number of tiled segments requested and (ii) shorter window size also turns urgent downloads more sensitive to network bandwidth fluctuation. Last, we observe that different user behavior, network condition, and fixation prediction algorithms affect the optimal U values. Adaptation of the urgent window size is among our future

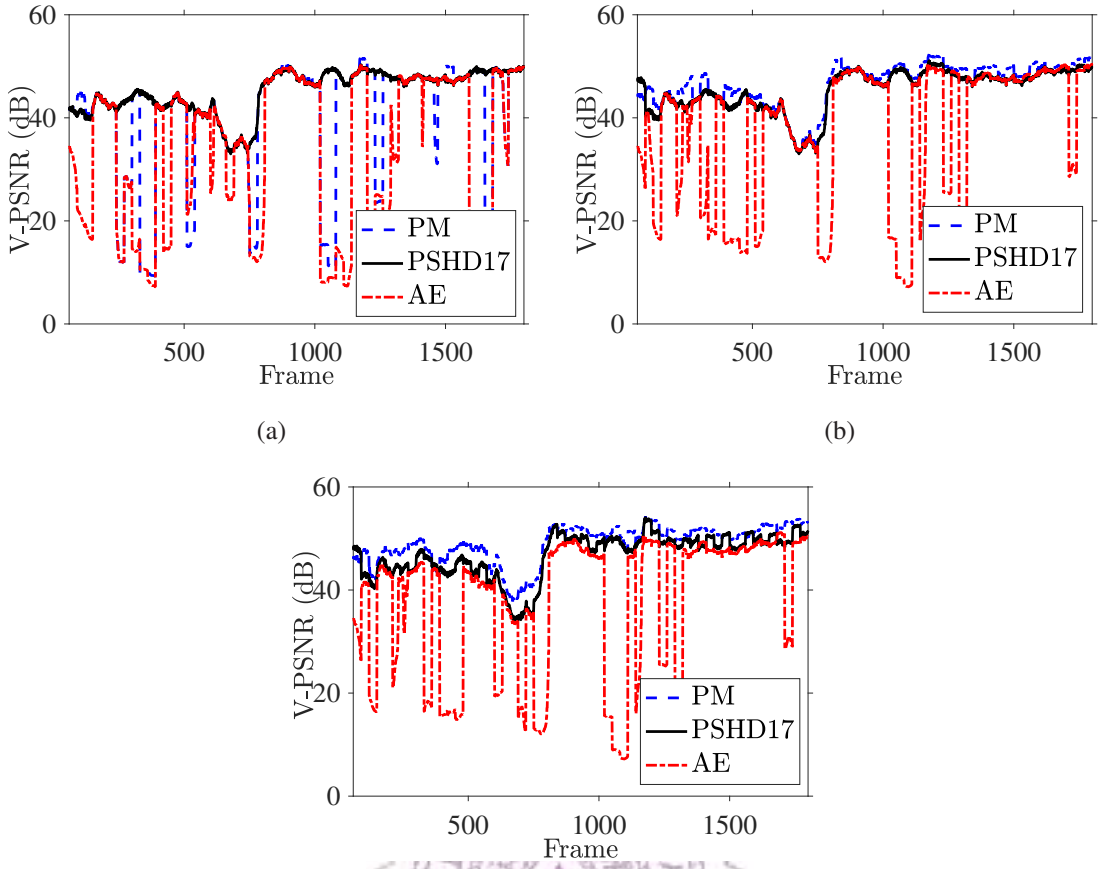


Figure 5.10: Comparisons of V-PSNR over time in the whole viewport among ABR algorithm under network bandwidth of: (a) 5, (b) 10, and (c) 15 Mbps.

tasks.

5.3.6 Approximation Approach of Tile Selector

We compare the approximated tile selector presented in Sec. 3.3.2 against the ordinary one. We compute the average prediction time and the number of different tiles between them using 100 random viewports. The approximation approach terminates in 0.06 ms, while the ordinary approach takes several seconds. The average number of different tiles is 1.73, which is not serious as also evidenced in the video quality (V-PSNR) results reported above.

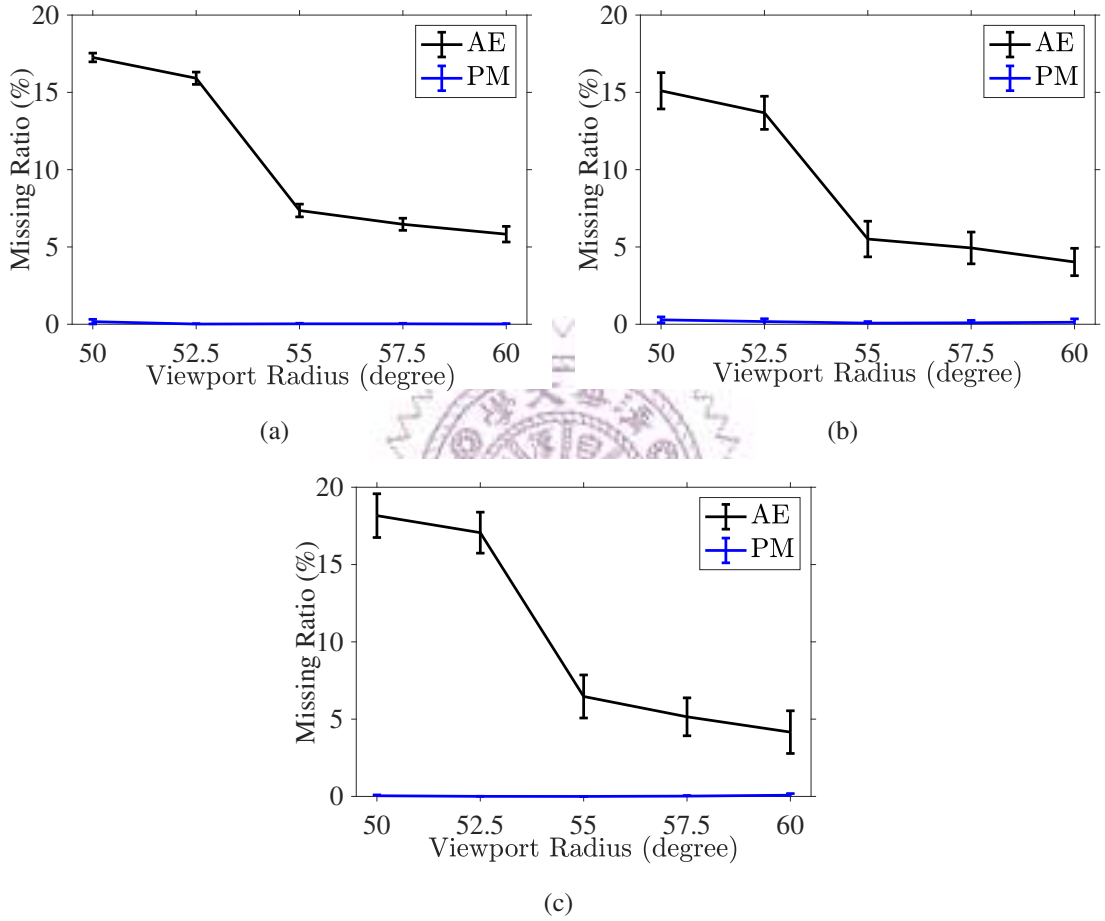


Figure 5.11: Missing ratios under diverse viewport radius: (a) a sample user with a sample video, (b) 10 users with a sample video and (c) 10 videos with a sample user.

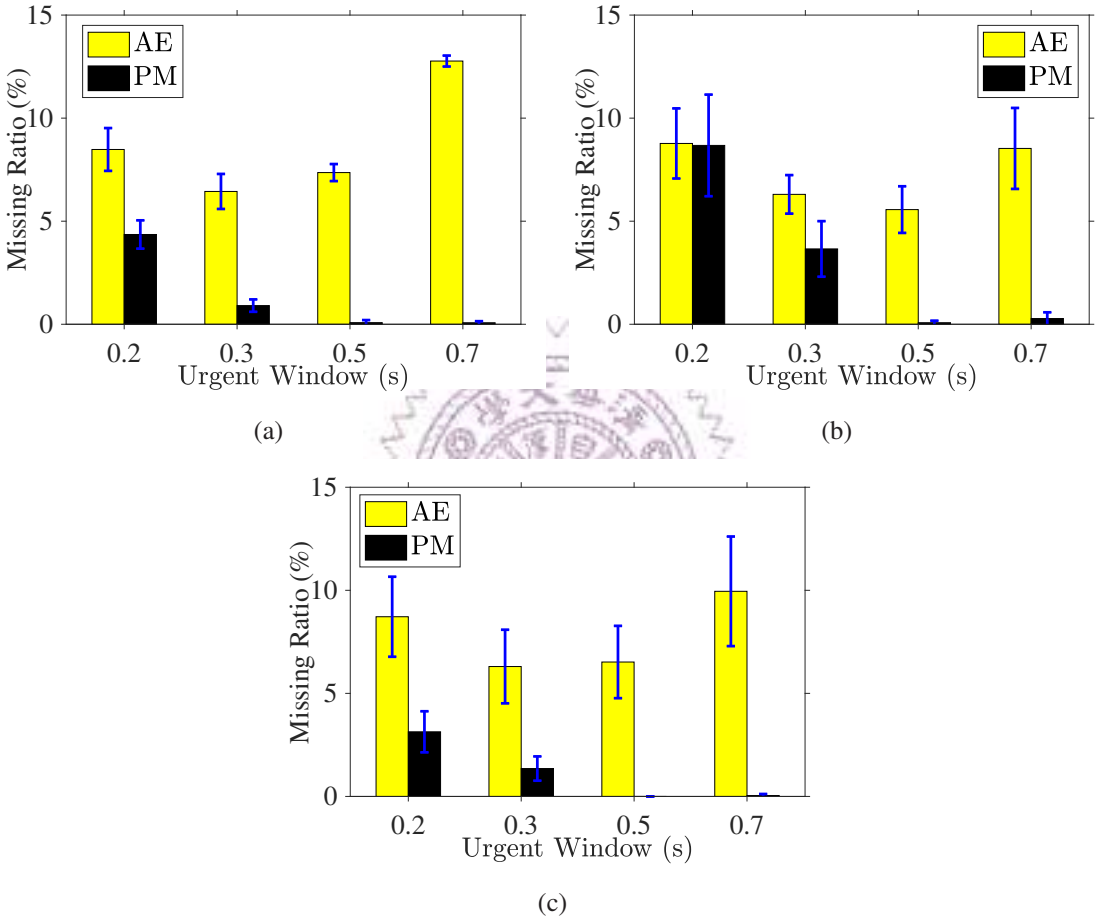


Figure 5.12: Missing ratios under different urgent window size: (a) a sample user with a sample video, (b) 10 users with a sample video and (c) 10 videos with a sample user.

Chapter 6

Related Work

6.1 Video Streaming over QUIC

Several studies in the literature investigate the potentials of QUIC on video streaming. For instance, Arisu and Begen [1] evaluate the performance of QUIC video streaming over wireless and cellular networks. They find that QUIC over UDP outperforms HTTP/2 over TCP, especially when the network is congested. Timmerer and Bertoni [45] also quantitatively compare the performance of video streaming over QUIC with other transport protocols. Bhat et al. [5] consider several state-of-the-art ABR algorithms to compare the video quality delivered by QUIC and HTTP/2. They find that existing ABR algorithms tend to switch to lower quality levels when QUIC is used. To address the problem, they propose retransmission over QUIC [4] to reduce the number of quality switches and increase the average bitrate. Palmer et al. [38] extend QUIC to support unreliable streams, and their proposed solution outperforms the ordinary HTTP/2 over TCP and QUIC over UDP. The aforementioned studies do no target 360° video DASH streaming, where extremely high bandwidth is required. Hayes et al. [12, 13, 14] partially solve the bandwidth problem with MultiPath TCP (MPTCP) and employ QUIC to mitigate the network congestion and fluctuation problems due to the large reordering buffers of MPTCP. Their studies, unfortunately, do not utilize the advantages of tiled streaming and always transmit the whole 360° video frames.

6.2 ABR Algorithms for 2D Planar Videos

There are many ABR algorithms for optimizing 2D planar video streaming [22, 41] and most of them are devised for HTTP/1.1. These ABR algorithms can be roughly classified into three types: (i) throughput-based, (ii) buffer-based, and (iii) hybrid algorithms. The throughput-based algorithms utilize the throughput estimations for making decisions. For

example, Jiang et al. [18] propose to select suitable video quality levels while striving for a balance among stability, fairness, and efficiency. Buffer-based algorithms only consider video buffer occupancy when determining proper video quality levels. For instance, Huang et al. [17] propose to build a rate map based on the available video bitrates and pick the most suitable rate according to the current buffer occupancy. Hybrid algorithms utilize both throughput and buffer level to make decisions. For example, Li et al. [24] propose to detect the congestion by observing throughput variations and consider the average buffer occupancy and target video rate when selecting the quality levels.

6.3 360° Tiled Video DASH Streaming

Streaming systems over multiple prioritized streams. Several 360° tiled video streaming studies [36, 48, 49, 33, 50] in the literature adopt HTTP/2 over TCP and leverage prioritized streams to better handle network and viewer dynamics. Tiles are assigned weights based on their algorithms. HTTP/2 allocates bandwidth resources to each prioritized stream based on its weight. As an example, Nguyen et al. [36] proposes to divide a video frame into multiple regions and assign tiles in the same region the same weight. After that, they monitor the download progress and estimate instantaneous throughput to determine whether some streams should be aborted to prevent buffer under-run. Xiao et al. [48] also utilize prioritized streams to determine the quality levels and weights of individual tiles in both spatial and temporal domains. To further capitalize on these prioritized streams, Ben Yahia et al. [49] and Nguyen et al. [33] propose to update stream weights during streaming sessions. In particular, in Nguyen et al. [33], each tile is assigned a weight when being requested. During the streaming sessions, the ongoing streams are monitored, and their weights may be dynamically updated. Ben Yahia et al. [49] propose to run the fixation prediction algorithm *twice* for better video quality due to the more accurate predictions. The aforementioned papers [36, 48, 49, 33, 50] present streaming systems, while the current thesis focuses on the ABR algorithm design.

ABR algorithms for 360° tiled streaming. While the above ABR algorithms [18, 17, 24] are not designed for 360° tiled streaming, some recent works specifically target 360° tiled streaming. Nguyen et al. [32] propose a client-based framework that trade off video bitrate and video quality when making decisions on tile's quality levels. Nguyen et al. [36] propose a throughput-based ABR algorithm, which partitions each 360 video into multiple regions. The tiles in each region are assigned a predetermined weight. Nguyen et al. [34, 35] take the prediction error of past frames into consideration when determining the quality levels of individual regions. To capitalize on HTTP/2, Petrangeli et al. [40] divide the video into tiles and classify them into three regions: viewport, adjacent, and

background. Based on their quality decision method, all tiles are first assigned the lowest quality level. The viewport tiles are first upgraded to the highest possible quality level, and the adjacent tiles are then upgraded if there exists residual bandwidth. The same procedure is then repeated between the adjacent and background tiles. They also leverage the server push to reduce the network overhead. We note that Petrangeli et al. [40] is the closest to our proposal. *Hence, we employ it as the baseline algorithm in our evaluations. In contrast to prior studies, the thesis is the first work developing an ABR algorithm for preemptive multiplexed streams in each DASH session*.

Chapter 7

Conclusion and Future Work

In this thesis, we designed an ABR algorithm for 360° tiled videos leveraging preemptive multiplexed streams. We built a real 360° tiled streaming testbed to evaluate our ABR algorithm with two start-of-the-art baseline ABR algorithms. The evaluation results show the merits of our proposed PM algorithm compared to the baseline algorithms: (i) Our algorithm averagely reduces the rebuffering counts by up to 3.2 and rebuffering time by up to 2.54 s, under constrained network bandwidth; (ii) Our algorithm achieves higher bandwidth utilization on average: at most 40.02% higher than the baseline algorithms; (iii) Our algorithm delivers good average V-PSNR at 39–49 dB under 5–10 Mbps bandwidth.

The evaluation results also shed some lights on future research directions.

- **More detailed performance comparison.** In the thesis, the experiments are run under stable network bandwidth, and thus video streaming performance under dynamic network has not been evaluated. Besides, due to the rise of machine learning, several state-of-the-art ABR algorithms employ Reinforcement Learning (RL) [20, 53] to stream 360° tiled videos and decide video quality level for the individual tile. We plan to implement their solutions and conduct more experiments to make the evaluation results more comprehensive.

- **Comparison among various prioritized schedulers.** Real-world Internet has dynamic network bandwidth and latency that complicates video streaming, especially 360° tiled video streaming. In this thesis, we utilize strict prioritized scheduler to optimize 360° tiled video streaming. However, different prioritized schedulers have inconsistent streaming performance under diverse network conditions. Therefore, we will compare streaming performance with different prioritized schedulers to identify the pros and cons.

- **Adaptation mechanisms for system parameters.** Most of the parameters in our system are empirically determined and remain the same value during the video

streaming. However, the adaptation algorithms for the system parameters are crucial for deploying our solutions in live networks. We plan to leverage machine learning to optimize the value of system parameters and dynamically change the value during video streaming.

Bibliography

[1] S. Arisu and A. C. Begen. Quickly starting media streams using QUIC. In *Proc. of Packet Video Workshop (PV'18)*, pages 1–6, Amsterdam, Netherlands, June 2018.

[2] I. Bauermann, M. Mielke, and E. Steinbach. H.264 based coding. In *In Proc. of International Conference Computer Vision and Graphics (ICCVG'04)*, pages 209–215, 2004.

[3] Bert Hubert. tc. https://linux.die.net/man/8/tc, 2018.

[4] D. Bhat, R. Deshmukh, and M. Zink. Improving QoE of ABR streaming sessions through QUIC retransmissions. In *Proc. of ACM International Conference on Multimedia (MM'18)*, pages 1616–1624, Seoul, South Korea, October 2018.

[5] D. Bhat, A. Rizk, and M. Zink. Not so QUIC: A performance study of DASH over QUIC. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'17)*, pages 13–18, Taipei, Taiwan, June 2017.

[6] Daniel Stenberg. libcurl. https://curl.haxx.se/libcurl/, 2019.

[7] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, and J. Zhan. Understanding the impact of video quality on user engagement. In *Proc. of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'11)*, pages 362–373, Toronto, Ontario, August 2011.

[8] C.-L. Fan, J. Lee, W.-C. Lo, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. Fixation prediction for 360° video streaming in head-mounted virtual reality. In *Proc. of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'17)*, pages 67–72, Taipei, Taiwan, June 2017.

[9] FOVE Inc. Eye tracking VR DEVKIT. http://tinyurl.com/y6rgj5mz, 2019.

[10] C. Fu, L. Wan, T. Wong, and C. Leung. The rhombic dodecahedron map: An efficient scheme for encoding panoramic video. *IEEE Transactions on Multimedia*, 11(4):634–644, 2009.

[11] Google Inc. The chromium projects. https://www.chromium.org/quic/playing-with-quic, 2019.

[12] B. Hayes, Y. Chang, and G. Riley. Omnidirectional adaptive bitrate media delivery using MPTCP/QUIC over an SDN architecture. In *Proc. of IEEE Global Communications Conference (GLOBECOM'17)*, pages 1–6, Singapore, Singapore, December 2017.

[13] B. Hayes, Y. Chang, and G. Riley. Controlled unfair adaptive 360 VR video delivery over an MPTCP/QUIC architecture. In *Proc. of IEEE International Conference on Communications (ICC'18)*, pages 1–6, Kansas City, MO, May 2018.

[14] B. Hayes, Y. Chang, and G. Riley. Scaling 360-degree adaptive bitrate video delivery over an SDN architecture. In *Proc. of International Conference on Computing, Networking and Communications (ICNC'18)*, pages 604–608, Maui, HI, March 2018.

[15] M. Hosseini and V. Swaminathan. Adaptive 360 VR video streaming: Divide and conquer. In *Proc. of IEEE International Symposium on Multimedia (ISM'16)*, pages 107–110, San Jose, CA, December 2016.

[16] C.-F. Hsu, A. Chen, C.-H. Hsu, C.-Y. Huang, C.-L. Lei, and K.-T. Chen. Is foveated rendering perceivable in virtual reality?: Exploring the efficiency and consistency of quality assessment methods. In *Proc. of ACM International Conference on Multimedia (MM'17)*, pages 55–63, Mountain View, California, October 2017.

[17] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'14)*, pages 187–198, Chicago, Illinois, August 2014.

[18] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)*, 22(1):326–340, February 2014.

[19] A. R. Jiménez, F. Seco, C. Prieto, and J. I. Guevara. A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU. In *Proc. of IEEE International Symposium on Intelligent Signal Processing (WISP'09)*, pages 37–42, Budapest, Hungary, August 2009.

[20] N. Kan, J. Zou, K. Tang, C. Li, N. Liu, and H. Xiong. Deep reinforcement learning-based rate adaptation for adaptive 360-degree video streaming. In *Proc. of IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP'19), pages 4030–4034, Brighton, United Kingdom, May 2019.

[21] H. Kimata, S. Shimizu, Y. Kunita, M. Isogai, and Y. Ohtani. Panorama video coding for user-driven interactive video application. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'09)*, pages 112–114, 2009.

[22] J. Kua, G. Armitage, and P. Branch. A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP. *IEEE Communications Surveys Tutorials*, 19(3):1842–1866, March 2017.

[23] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Y. F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi. The QUIC transport protocol: Design and internet-scale deployment. In *Proc. of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*, pages 183–196, Los Angeles, CA, August 2017.

[24] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, April 2014.

[25] Linux. Shared memory. shorturl.at/kNSW8, 2019.

[26] W.-C. Lo, C.-L. Fan, J. Lee, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu. 360° video viewing dataset in head-mounted virtual reality. In *Proc. of ACM International Conference on Multimedia Systems (MMSys'17)*, pages 211–216, Taipei, Taiwan, June 2017.

[27] W.-C. Lo, C.-L. Fan, S.-C. Yen, and C.-H. Hsu. Performance measurements of 360° video streaming to head-mounted displays over live 4g cellular networks. In *Proc. of Asia-Pacific Network Operations and Management Symposium(APNOMS'17)*, pages 205–210, Seoul, South Korea, September 2017.

[28] Luis MartinGarcia. Tcpdump. https://www.tcpdump.org/, 2018.

[29] A. Mavlankar and B. Girod. Video streaming with interactive pan/tilt/zoom. In *High-Quality Visual Experience*, pages 431–455. Springer, 2010.

[30] K. Ng, S. Chan, and H. Shum. Data compression and transmission aspects of panoramic videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(1):82–95, 2005.

[31] A. Nguyen, Z. Yan, and K. Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proc. of ACM International Conference on Multimedia (MM'18)*, pages 1190–1198, Seoul, Republic of Korea, October 2018.

[32] D. Nguyen, H. T. Tran, and T. C. Thang. A client-based adaptation framework for 360-degree video streaming. *Journal of Visual Communication and Image Representation*, 59:231–243, February 2019.

[33] 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 *Proc. of International Conference on Communications and Electronics (ICCE'18)*, pages 261–266, Hue, Vietnam, July 2018.

[34] D. V. Nguyen, H. T. T. Tran, A. T. Pham, and T. C. Thang. A new adaptation approach for viewport-adaptive 360-degree video streaming. In *Proc. of IEEE International Symposium on Multimedia (ISM'17)*, pages 38–44, Taichung, Taiwan, December 2017.

[35] 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*, 9(1):29–42, March 2019.

[36] M. Nguyen, D. H. Nguyen, C. T. Pham, N. P. Ngoc, D. V. Nguyen, and T. C. Thang. An adaptive streaming method of 360 videos over HTTP/2 protocol. In *Proc. of NAFOSTED Conference on Information and Computer Science (NICS'18)*, pages 302–307, Hanoi, Vietnam, November 2017.

[37] L. Oculus VR. Facebook Oculus Rift. https://www.oculus.com/, 2019.

[38] M. Palmer, T. Kruger, B. Chandrasekaran, and A. Feldmann. The QUIC fix for optimal video streaming. In *Proc. of the Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18)*, pages 43–49, Heraklion, Greece, December 2018.

[39] Parikshit Juluri. AStream. https://github.com/pari685/AStream, 2018.

[40] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. D. Turck. An HTTP/2-based adaptive streaming framework for 360° virtual reality videos. In *Proc. of ACM International Conference on Multimedia (MM'17)*, pages 306–314, Mountain View, California, October 2017.

[41] Y. Sani, A. Mauthe, and C. Edwards. Adaptive bitrate selection: A survey. *IEEE Communications Surveys Tutorials*, 19(4):2985–3014, July 2017.

[42] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys Tutorials*, 17(1):469–492, September 2015.

[43] T. Stockhammer. Dynamic adaptive streaming over HTTP –: standards and design principles. In *Proc. of the ACM Conference on Multimedia Systems (MMSys'11)*, pages 133–144, San Jose, CA, February 2011.

[44] H. Strasburger, I. Rentschler, and M. Jüttner. Peripheral vision and pattern recognition: A review. *Journal of Vision*, 11(5):13–13, 12 2011.

[45] C. Timmerer and A. Bertoni. Advanced transport options for the dynamic adaptive streaming over HTTP. Technical report, Alpen-Adria-Universitat Klagenfurt, Institute of Information Technology (ITEC), Austria, 2016.

[46] Transparency Market Research Inc. Head mounted display (helmet mounted display, wearable computing glasses) market. https://www.transparencymarketresearch.com/head-mounted-displays.html, 2015.

[47] C. Wang, A. Rizk, and M. Zink. Squad: A spectrum-based quality adaptation for dynamic adaptive streaming over HTTP. In *Proc. of International Conference on Multimedia Systems (MMsys'16)*, pages 1–112, Klagenfurt, Austria, May 2016.

[48] M. Xiao, C. Zhou, V. Swaminathan, Y. Liu, and S. Chen. BAS-360°: Exploring spatial and temporal adaptability in 360-degree videos over HTTP/2. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'18)*, pages 953–961, Honolulu, HI, April 2018.

[49] M. B. Yahia, Y. L. Louedec, G. Simon, and L. Nuaymi. HTTP/2-based streaming solutions for tiled omnidirectional videos. In *Proc. of IEEE International Symposium on Multimedia (ISM'18)*, pages 89–96, Taichung, Taiwan, Taiwan, December 2018.

[50] S.-C. Yen, C.-L. Fan, and C.-H. Hsu. Streaming 360° videos to head-mounted virtual reality using DASH over QUIC transport protocol. In *Proc. of ACM Workshop Packet Video Workshop (PV'19)*, pages 7–12, Amherst, MA, June 2019.

[51] M. Yu, H. Lakshman, and B. Girod. A framework to evaluate omnidirectional video coding schemes. In *Prof. of IEEE International Symposium on Mixed and Augmented Reality (ISMAR'15)*, pages 31–36, Fukuoka, Japan, September 2015.

[52] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj. HEVC-compliant tile-based streaming of panoramic video for virtual reality applications. In *Proc. of ACM International Conference on Multimedia (MM'16)*, pages 601–605, Amsterdam, The Netherlands, October 2016.

[53] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li. DRL360: 360-degree video streaming with deep reinforcement learning. In *Proc. of IEEE Conference on Computer Communications (INFOCOM'19)*, pages 1252–1260, Paris, France, May 2019.