12-12-2022

A Measurement Study of Live 360 Video Streaming Systems

Jun Yi

A Measurement Study of Live $360°$ Video Streaming Systems

by

Jun Yi

Under the Direction of Yubao Wu, Ph.D and Zhisheng Yan, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2022

ABSTRACT

360° live video streaming is becoming increasingly popular. While providing viewers with enriched experience, 360° live video streaming is challenging to achieve since it requires a significantly higher bandwidth and a powerful computation infrastructure. A deeper understanding of this emerging system would benefit both viewers and system designers. Although prior works have extensively studied regular video streaming and 360° video on demand streaming, we for the first time investigate the performance of 360° live video streaming. We conduct a systematic measurement of YouTube's 360° live video streaming using various metrics in multiple practical settings. Our research insight will help to build a clear understanding of today's 360° live video streaming and lay a foundation for future research on this emerging yet relatively unexplored area.

To further understand the delay measured in YouTube's 360° live video streaming, we conduct the second measurement study on a 360° live video streaming platform. While live 360° video streaming provides an enriched viewing experience, it is challenging to guarantee the user experience against the negative effects introduced by start-up delay, event-to-eye delay, and low frame rate. It is therefore imperative to understand how different computing tasks of a live 360° streaming system contribute to these three delay metrics. Our measurement provide insights for future research directions towards improving the user experience of live 360° video streaming.

Based on our measurement results, we propose a motion-based trajectory transmission method for 360° video streaming. First, we design a testbed for 360° video playback. The testbed can collect the users viewing data in real time. Then we analyze the trajectories of the moving targets in the 360° videos. Specifically, we utilize optical flow algorithms and gaussian mixture model to pinpoint the trajectories. Then we choose the trajectories to be delivered based on the size of the moving targets. The experiment results indicates that our method can obviously reduce the bandwidth consumption.

INDEX WORDS: 360° Video, Video Streaming, Motion, Measurement Study

A Measurement Study of Live 360° Video Streaming Systems

by

Jun Yi

ACKNOWLEDGMENTS

To be honest, it would be impossible for me to finish the dissertation without the guidance of my committee members, help from my group, and support from my family and friends. I would like to show my great gratitude to all of them. First, I would like to give my deepest thanks to my advisors, Dr.Zhisheng Yan and Dr. Yubao Wu, for their guidance and support. They patiently inspired me and financially supported my research. Dr. Yan helped me on my research with great patience. Dr. Wu not only provided valuable suggestions and guidance when I was trapped in my research, but also motivated me to pursue higher career growth. His hard-working, brilliant and keen has set a good example for me to follow. I really appreciate their help.

I am also very thankful to the professors and staffs at our department. One of the most important person is Tammie Dudley. During the past years, she really help me a lot. Especially during the pandemic, she always encouraged me and asked me never give up.

Last but not least, it is a pleasure to thank everybody who made the dissertation possible especially during this pandemic season, as well as express my apologies that I could not mention personally one by one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Live video streaming has played a pivotal role in shaping our lives in entertainment, surveillance and teleconference. Based on video streaming market report in 2016 (rep 2016), the global video streaming market will grow to 70.05 billion by 2021. In 2018, live video streaming is overtaking the growth of other types of online videos, with an impressive annual market increase of 113% (web 2018). With the emergence of 360° videos, 360° live video streaming is becoming a new way to broadcast our everyday life. Viewers are allowed to freely switch viewing directions of the panoramic content by dragging the mouse of a desktop, swiping the screen of a smartphone or moving the head around with a head-mounted display (HMD). Compared with regular live video streaming, 360° live video streaming can render a panoramic view and enrich viewer experience in various applications. Major social media websites such as Facebook and YouTube have all supported 360° live video streaming.

Despite the promising experience, achieving 360° live video streaming is challenging. Due to its panoramic nature, 360° live video streaming needs to transport high-quality data of multiple viewports and thus requires a significantly higher network bandwidth. Furthermore, the computation time for video encoding, decoding and rendering must be minimized on cameras and viewing devices to allow real-time 360° live video playback.

Delay is critical to live video streaming. Different delay metrics have various impacts on user experience. Complex initialization between a client and a server may lead to an excessive *start-up delay*, which decreases users' willingness to continue the viewing. The start-up delay may in turn result in a long *event-to-eye delay*, i.e., the time interval between the moment an event happens on the remote scene and the moment when the event is displayed on the client device. Long event-to-eye delay causes significant lags in streaming of live events such as sports, concerts, and business meetings. Moreover, the *frame rate* of a live video is determined by how fast frames can be pushed through the system pipeline. A low frame rate would make the video playback not smooth.

Facing these challenges, it is imperative to have an clear understanding of 360° live video streaming and thus to assist both viewers and system designers of this technology. For example, the viewers would like to know which video quality should be chosen when using 360° live video streaming service. For developers, an understanding of the system would help to identify the problems that exist in today's 360° live video streaming. This could be the first step to design effective algorithms to optimize the 360° live video streaming performance.

My research focuses on understanding the 360° live video streaming service, especially in digging into a task-level understanding of the 360° live video streaming pipeline. In summary, our contributions can be summarized as follows.

- Today's 4K 360° live video streaming suffers from frequent rebuffering for 33% of the playback time. In contrast, the smooth playback of a 360° video at 1080p and below can be supported. However, this might imply unsatisfactory user experience in some applications since only 15% $\sim$ 20% of the panorama would be viewed.

- One-way video delay is extremely high in 360° live video streaming. Viewers need to wait for 13 seconds (480p) to 42 seconds (4K) to see the most recent events of the remote site. Such delay performance could be acceptable for certain broadcasting applications, but would be unsuitable for real-time interactive applications.

- While DASH can reduce the rebuffering events and one-way video delay, it still does not solve the problems of 4K 360° live video streaming. Surprisingly, viewers might not necessarily have better experience in DASH-based 4K 360° live video streaming than in non-DASH-based 1080p streaming.

- Although YouTube 360° live video streaming player has adopted a strategy to skip frames in order to reduce the one-way video delay and keep the live video up to date, the buffer management algorithm is still conservative to achieve real-time requirement.

- We identify the diverse relationship between the time consumption breakdown across the system pipeline and the three delay metrics in live 360° video streaming.

- We build an open research prototype `Zeus`[1] using publicly available hardware and software to enable task-level delay measurement. The methodology for building `Zeus` can be utilized in future 360° video research.

- We leverage `Zeus` to perform a comprehensive measurement study to dissect the time consumption in live 360° video streaming and understand how each task affects different delay metrics.

- We perform a comparison of `Zeus` against a commercial live 360° video streaming system built on Ricoh Theta V and YouTube and validate that our measurement results are representative of real world systems.

The rest of the proposal is organized as follows: Chapter 2 presents the background of 360° video streaming. Chapter 3 investigates the problems of YouTube 360° live video

[1]https://github.com/junyiwo/Zeus

streaming, which provides insights on commercial live video streaming platforms. Chapter 4 discusses the delay analysis and proposed method for delay measurement in 360° live video streaming platforms, which help build a deeper understanding of delay in commercial live video streaming. Chapter 5 introduces a motion-based transmission method for 360° video streaming. Chapter 6 concludes our work.

CHAPTER 2
BACKGROUND

In this chapter, we provide the background information of our topic. We review the relative concepts in the fields of 360° video, live video streaming, measurement study in the subsections respectively.

2.1 360° video

360° videos are also known as immersive videos. They are video recording where each direction of scenes will be recorded at the same time. 360° videos are normally generated by two different types of cameras: a rig of normal cameras or multiple fish eye cameras embedded into one camera. The rig of cameras usually contains more than five cameras that cover all the directions(rig 2021). After being recorded by different cameras, the images will be stitched to generate one 360° image by the camera itself or other platforms installing software like Open Broadcaster Studio.

There are three major map projection methods to transform 360 videos into a 2D plane before encoding. Specifically, they are equiractangular projection, pyramid projection and cube projection. 360° videos need to be mapped since currently no encoder can encode special videos. Equirectangular projection will first transform spherical coordinates of the stitched sphere images into planar coordinates (Su & Grauman 2017; Wang et al. 2020, 2017). Once received, the client will run the reverse projection which transforms the rectangular from the plane back onto the sphere. However, this method introduces obvious distortion at the poles of sphere. Pyramid projection maps the viewport to the base of the pyramid and slides for the other part of a video frame (Nasrabadi et al. 2017; Duanmu et al. 2017; Zhou et al. 2019). However, it causes severe quality variance between the base and the slides, which decrease the user's experience. For the cube projection(Cheng et al. 2018; Sauer et al. 2018; Duanmu et al. 2018), video frames will be mapped to six faces of a cube. However, it also has geometry distortion.

2.2 Live Video Streaming

Live video streaming is a new method for digital communication. Compared with Video on Demand (VoD), live video streaming is when the streamed video being sent over the Internet in real time instead of being recorded or stored before streaming. Currently, live video streaming has an important impact on various business (web 2018). For example, more and more schools and colleges utilize live video streaming for online classes during the pandemic.

Compared with VoD, live video streaming provide better user experience. For example, users can have real time interactions even if the video streaming was delivered to multiple channels. Meanwhile, live video streaming is more authentic since there is no post production. Moreover, live video streaming can save budget. For example, businesses that use live video streaming don't require everyone to stay in the same location. Everyone can receive the same video streaming simultaneously. Thus, there is no need to pay the travel fee or any other fee that needed to be considered.

2.3 DASH

Dynamic Adaptive Streaming over HTTP (DASH) has been widely used in live video streaming and VoD since it was proposed in 2011(DAS 2021). Now it can supoort a wide range of devices such as mobile phones, laptops and tablets.

To delivery the video content, DASH divides them into small chunks which are several seconds long. The HTTP server can save all the video chunks with different encoding rate.Thus client players can download video chunks based on their network performance by HTTP (Gadaleta et al. 2017; Sodagar 2011; Lederer et al. 2012). Specifically, HTTP server will first sends client players a list of the available media segment URLs in a Media Presentation Description (MPD) file (Zhao et al. 2014). Then the client players fetch information like maximum bandwidth and minimum bandwidth for the video, video resolution, and various encoded alternatives of video chunks. Based on the MPD file and the network performance, the client players can pick up an appropriate video stream and send an request to the HTTP server.

Compared with other adaptive video streaming technologies such as HTTP Live Streaming (HLS) and Microsoft Smooth Streaming, DASH has better performance(Zhu & Song 2015; Pan et al. 2016; Durak et al. 2020). For example, DASH support continued adaptation to the bandwidth situation of the client. Meanwhile, it has smaller startup delay and video buffering during the play. Moreover, it can bypass the firewall due to the usage of HTTP.

CHAPTER 3

A Measurement Study of Youtube 360° Live Video Streaming

In this chapter, we study the issues about today's commercial 360° live video streaming systems by conducting a measurement study on YouTube 360° live video streaming.

3.1 Motivation

360° live video streaming is becoming increasingly popular due to the development of hardware infrastructure. 360° cameras is expected to grow by 1.68 billion between 2021 and 2025 (360 2021). 360° live streaming has been widely used in multiple industries such as education, commercial and sports. Compared with traditional live video streaming, 360° live video streaming provide uses an enriched experience due to the panoramic viewport. With the increasing adoption of virtual reality, it is not hard to imagine a future where a 360-degree video would become the new norm rather than being the novelty it is today.

While providing viewers with enriched experience, 360° live video streaming is challenging to achieve since it requires a significantly higher bandwidth and a powerful computation infrastructure. A deeper understanding of this emerging system would benefit both viewers and system designers. Although prior works have extensively studied regular video streaming and 360° video on demand streaming, we for the first time investigate the performance of 360° *live* video streaming. We conduct a systematic measurement of YouTube's 360° live video streaming using various metrics in multiple practical settings.

Our key findings suggest that viewers are advised not to live stream 4K 360° video, even when dynamic adaptive streaming over HTTP (DASH) is enabled. Instead, 1080p 360° live video can be played smoothly. However, the extremely large one-way video delay makes it only feasible for delay-tolerant broadcasting applications rather than real-time interactive applications. More importantly, we have concluded from our results that the primary design weakness of current systems lies in inefficient server processing, non-optimal rate adaptation, and conservative buffer management. Our research insight will help to build a clear understanding of today's 360° live video streaming and lay a foundation for future research on this emerging yet relatively unexplored area.

3.2 Related Work

In this section, we introduce the related work on video streaming measurement study. Based on the video content, we divide the related work into two categories: regular live video streaming and 360° video streaming.

Regular live video streaming. Regular live video streaming has been implemented on different platforms. Yu et al (Yu et al. 2014) conducted a measurement study of three mobile video call applications: FaceTime, Google Plus Hangout, and Skype. Through measurements over a wide range of wireless network conditions, they showed that mobile live video streaming quality is highly vulnerable to bursty packet loss and long packet delay. For the broadcast applications, Tang et al (Tang et al. 2016a) described the characteristics of live video streaming on Meerkat and Periscope and revealed the relationship among different roles in the broadcast system. For example, the interaction between uploaders and viewers shapes the video content. Ding et al (Ding et al. 2011) focused on YouTube uploaders' characteristics (gender, age and geography distribution) and behavior. By analyzing 10,000 uploaders' information, they demonstrated that most uploaders prefer to upload a contiguous snippet of a video that is originally distributed outside of YouTube. Hamilton et al

(Hamilton et al. 2014) investigated the behavior of Twitch's viewers and uploaders. It is interesting to observe that viewers tend to regard Twitch as a virtual place of social activity. Although these works have shed some light on regular live video streaming system design, the observation and principle cannot be directly used in 360° live video streaming systems.

360° video streaming. Recent efforts have been made towards the encoding/projection (Xiao et al. 2017; Xie et al. 2017), view-adaptive streaming (Nguyen et al. 2018a; Qian et al. 2018a) and energy optimization (Yan et al. 2018) of 360° video streaming. Xiao et al (Xiao et al. 2017) proposed a new encoding technique by converting the video encoding into a storage costs optimization problem. Anh et al(Nguyen et al. 2018a) proposed a methodology to collect saliency maps for 360° video based on previous study on human attention behaviors in head-mounted display. They built a Deep Convolutional Neural Network (DCNN) based on the generated saliency map. Then, a head movement prediction model is proposed to predict user head movement using saliency maps generated from DCNN.

Meanwhile, researchers have attempted to understand 360° VoD streaming through measurement studies (Zhou et al. 2017a; Afzal et al. 2017b; Jiang et al. 2017; Lo et al. 2017). Afzal et al (Afzal et al. 2017b) characterized 360° videos by examining thousands of YouTube videos across more than a dozen categories and reached the conclusion that 360° videos are less variable in terms of bit rate and have less motion than regular videos. Zhou et al (Zhou et al. 2017a) reverse-engineered the encoding solution of Oculus 360° VoD systems and identified it as offset cubic projection. Compared with the standard cubic encoding, the offset cubic projection encodes a distorted version of the spherical surface, devoting more information to the view in a chosen direction. Jiang et al (Jiang et al. 2017) analyzed the energy consumption of 360° streaming on HMD under 8 test cases with different configurations and provided a detailed energy consumption breakdown of the system. They showed that the HMD streaming overhead and network transmission account for approximately half of the energy consumption. Lo et al (Lo et al. 2017) demonstrated the strength and weakness of tile-based streaming for 360° videos over 4G networks. They observed that the tile-based video streaming can clearly save the bandwidth but it may decrease the coding efficiency of tiles.

Despite the insight obtained from these studies, our understanding about 360° live video streaming is still limited. In this paper, we conduct a systematic study to facilitate viewers' interaction with this new technology as well as to lay a foundation for future system design in user experience and system performance optimization.

3.3 Measurement Setup

3.3.1 System Architecture

There are two types of architecture for 360° live video streaming systems: direct broadcast and indirect broadcast. For the indirect broadcast, a 360° camera is used to capture panoramic images and stitch them into the equirectangular format. After being captured by the camera, the 360° video will be transmitted to a client via USB or Ethernet connection. The client can be a desktop, laptop or mobile device on which a streaming software is installed to encode the video. Then the video will be uploaded to a third-party server such as YouTube's video server. The server will transcode the video into multiple resolutions to enable DASH support and then broadcast the video to viewers. After receiving the 360° video, a player will decode the video and render the spherical images for displaying. On the other hand, for direct broadcast, the camera is connected with a router through wireless

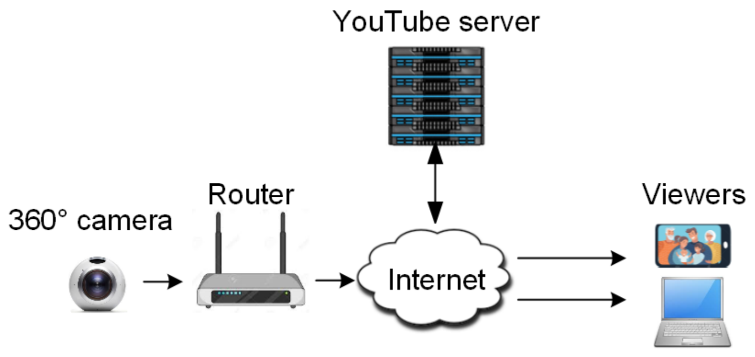


Figure 3.1 System architecture of the direct broadcast system.

networks so that the 360° video can be directly uploaded to a third-party server without a desktop/mobile relay. These two systems have been widely applied.

In this paper, we build a direct broadcast system to transport the 360° live video. This is a more practical case in everyday life as uploaders do not need extra devices to use as the relay client when broadcasting the video. Figure 3.1 shows the architecture of the direct broadcast system used in our measurement study.

3.3.2 Testbed

We build a testbed that consists of three components to study 360° live video streaming over YouTube. The 360° camera is a Ricoh Theta V(Ric 2021), which can provide 360° live video streaming with different resolutions and bitrates. For the router, we use ASUS RT-AC3100, a dual band gigabit router. It can support 2.4 GHz and 5 GHz wireless signals simultaneously. However, we have verified that the camera only connects to the 2.4 GHz wireless service in our measurement since the router complies to 5GHz standard W58 but the Ricoh THETA V only complies to standard W52 (Oda 2018). The viewing player lies in a desktop, residing in a normal university building. For the desktop, the CPU is an Intel Xeon W-2135 with 12 cores at 3.7 Ghz, and the graphic card is a GeForce GTX 1080Ti. The high-end configuration makes it feasible to process 4K 360° live video streaming. The live stream is sent to the desktop through a university LAN connection via a Gigabit Ethernet cable. We have confirmed that the upload bandwidth from the camera to the YouTube server is 60 Mbps and the download bandwidth is 16 Mbps.

To enable the end-to-end system, we implement several system components. First, we install a plug-in (ric 2018) on the camera to establish the connection. The plug-in uses Real-Time Messaging Protocol (RTMP), which is compatible to YouTube. We input the server URL and stream key into the plug-in so that uploaders can easily live stream 360° video to YouTube. The quality of camera capture can also be configured through the plug-in. On the viewer side, we develop an 360° video player for video playback using IFrame player API (**?**). The IFrame player API can embed a YouTube video player on a webpage and control the player using JavaScript. After receiving the video file chunks, our player offloads much of the processing work (decoding, audio synchronization) to dedicated video/graphic card of the computer to expedite the video processing.

3.3.3 Metrics and Methodology

To evaluate the performance of the YouTube 360° live video streaming service, we use several important metrics (Dobrian et al. 2011; Yu et al. 2014) for live video streaming.

- *Rebuffering ratio* is calculated as the rebuffering time divided by the duration of the entire live video session. It is used to evaluate how long would the video stay in the "freeze" status.

- *Rebuffering frequency* is calculated as the total number of rebuffering events divided by the duration of the entire live video session. Although the total stalling time could be short, it is likely that the viewers may suffer from frequent rebuffering events which also degrades viewing experience.

- *One-way video delay* is defined as the time difference between when a frame is captured and when it is displayed on the screen. It measures how long it would take for a viewer to see a remote event after the event occurs. The one-way video delay includes the time for camera stitching and processing, video upload, YouTube processing, downlink transmission, video decoding, and video rendering/display.
- *Initial delay* is the time difference between when the viewer sends a request to start the live streaming to when the first frame is displayed.

Measuring these performance metrics is non-trivial. First, we have no access to the rebuffering information on the YouTube website. During the live video session, we use `getPlayerState()` function to monitor the live video streaming session. It outputs a set of values, indicating the playback status– not started, ended, paused, playing, and buffering. We also output information such as video quality and play time using `getPlaybackQuality()` and `getDuration()` functions. Based on the system logs, we can compute the rebuffering metrics.

Furthermore, it is challenging to collect the delay information since there is no API support for inserting a timestamp into each frame on the camera side. Existing tools, such as gStreamer and FFmpeg, cannot be directly applied on the camera. Even though frame timestamp is available, it is unlikely to extract the application-layer information as the video packets are encrypted by HTTPS. Therefore, we follow the methodology used in (Yu et al.

2014) to measure the one-way video delay. The key idea is to let the camera capture a stopwatch such that the camera-side timestamps become available. A second stopwatch is run in the viewing device. By measuring the difference between the two clocks when the same frame is shown, we are able to derive the one-way video delay. The configuration details can be found in (Yu et al. 2014). For the initial delay, we can obtain it directly from the second stopwatch on the viewing client. To automatically collect the delay data, we write a script using Python 3.5 to extract the timestamps for both stopwatches.

We carry out the measurement inside a typical office building. The shooting scene generally contains professionals and office supplies such as desks and printers. As we focus on the system performance from frame capture to frame display rather than the user interaction and viewport switch, the viewer viewport does not change during the live session. The camera can capture raw video at 480p, 720p, 1080p and 4K and the viewer will pick one version upon live streaming request. The video frame rate is fixed at 30 frames per second. We perform measurement studies on various practical settings to simulate real-world application scenarios. For each video session, we measure the performance for 3 minutes and we repeat this for 20 times. The measurement scenarios are divided into four categories.

- *Default*: the camera is fixed on a table. The video with the viewer-requested resolution will be streamed.

- *Moving*: unlike Default, the uploader holds the camera and walks around the building during the measurement. Other settings remain the same.

- *DASH*: The difference between DASH and Default is that the viewer-requested video will be transcoded into more versions at the YouTube server in DASH scenario for downlink streaming (144p, 240p, 360p, 480p, 720p, 1080p, 1440p and 4K). Note that the downlink-streamed DASH videos would not have higher resolution than the viewer-requested version.

- *Moving DASH*: on top of Moving, the viewer also adopts DASH for downlink 360° live video playback.

3.4 Measurement Results

In this section, we present the results and analysis obtained from the measurement studies.

3.4.1 Rebuffering Events

In this set of measurements, we measure the rebuffering events of 360° live video streaming. Figure 3.2 illustrates the distribution of rebuffering time of all rebuffering events across all video sessions in 4 measurement scenarios. We can observe that if the viewer-requested resolution is 1080p and below, the rebuffering time is only around 0.5s to 0.8s. We also identify in our data that the average frequency of rebuffering events is 0.6 times/min for those sessions with a viewer-requested resolution at 1080p and below. This indicates a negligible rebuffering for low-resolution live video streaming. However, if the viewer-requested video is 4K, the viewer tends to suffer from an average of 6s stall with a longest stall up to 20s. The major reason of this phenomenon is that 4K live streaming is still challenging in today's downlink networks. The rendering and processing of 360° videos further add computation burden to the end-to-end pipeline, thereby causing delayed frame delivery and frequent rebuffering.

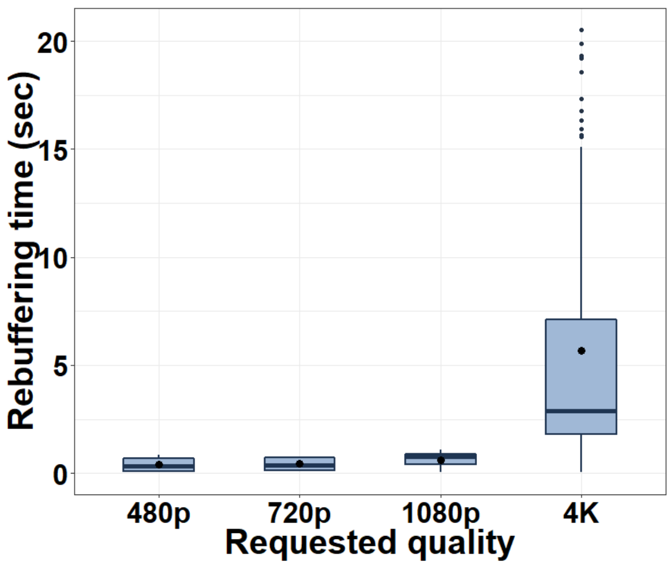


Figure 3.2 Distribution of rebuffering time across all sessions.

We also individually examine the rebuffering ratio and rebuffering frequency when a 4K video is requested by the viewer in Figure 3.3. We observe that the rebuffering ratio is 23% and the rebuffering frequency is more than 4 times/min even under Default. In Moving, the viewer will suffer worse experience since the live video session stalls for approximately 40% of the time and the rebuffering occurs more than 6 times per minute. In fact, we have verified that the moving camera introduces unstable upload bandwidth. Furthermore, the dynamic content captured by the camera will enlarge the volume of the video, making the encoding/decoding/rendering more challenging. In addition, it is important to observe that the rebuffering ratio decreases when adaptive dynamic downlink streaming is enabled in Moving DASH and DASH. However, since the camera-captured 4K video will be transcoded into lower resolutions in DASH-enabled cases, the actual viewed video quality is far from satisfactory. We will investigate the impact of DASH in Section 4.3.

Based on our results on rebuffering events, we can conclude that viewers would experience significant rebuffering when requesting 4K 360° live video on YouTube. To ensure

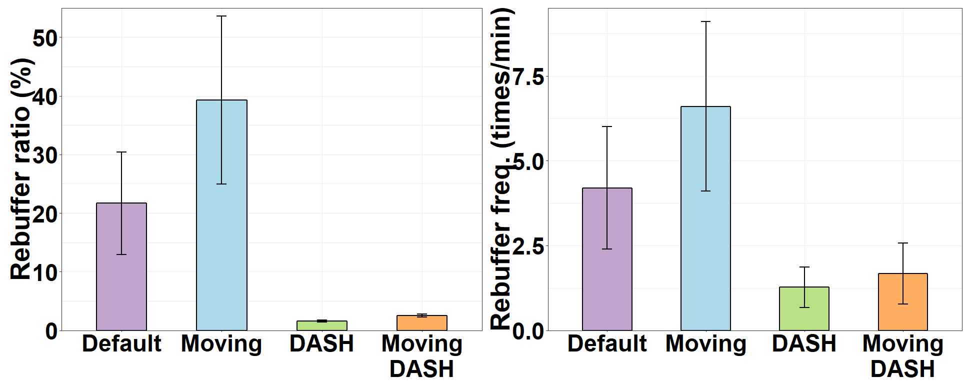


Figure 3.3 Rebuffering ratio and frequency of 360° live video streaming in 4K resolution.

the smooth non-stalling playback, viewers are suggested to request a resolution at 1080p or below. However, it is critical to note that viewer experience is likely to be unsatisfactory. This is because the resolution of 1080p or below will become even lower during viewing since only 15% ∼ 20% of the panorama would be actually viewed by the viewer.

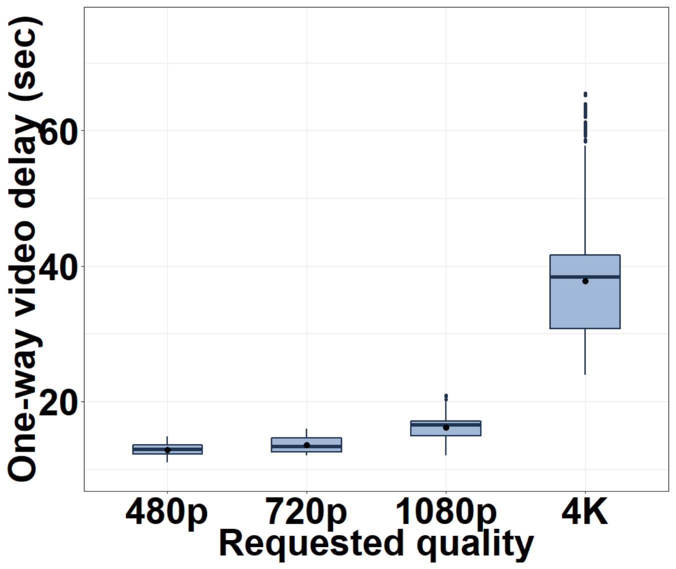
3.4.2 Delay Performance

In this section, we evaluate various delay of 360° live video streaming service.

3.4.2.1 One-way Video Delay

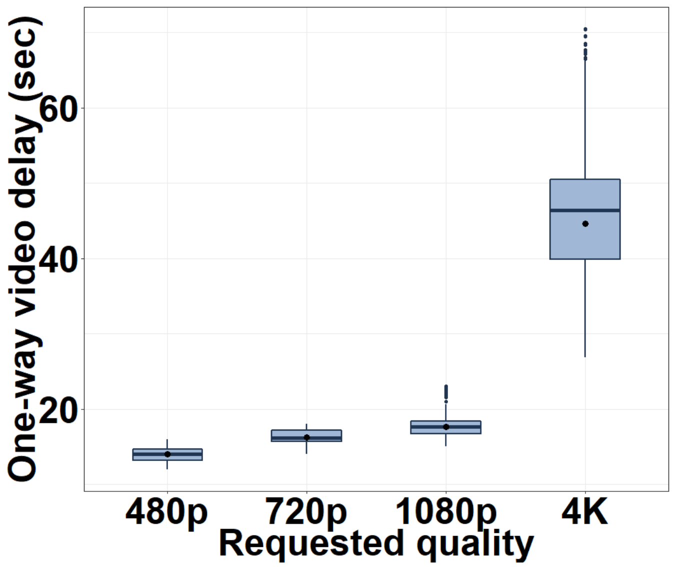
Figure 3.4 shows the boxplot of one-way video delay under the scenarios of Default and Moving. We observe that the one-way video delay for 4K 360° videos is significantly higher than lower resolution. The delay range of 4K 360° live video streaming (25s to 60s) is also clearly larger. Similar to what we explained in Section 4.1, this is because each frame in 4K video contains over three times as many pixels as in lower-resolution videos. The excessive size of the video consumes more time for video transport and processing.

Another interesting phenomenon is that movement has larger impact on 4K video than



Default

Figure 3.4 One-way video delay in different scenarios.



Moving

Figure 3.5 One-way video delay in different scenarios.

lower-resolution video. While the one-way video delay of 4K video increases more than 6s in Moving, lower-resolution videos show similar one-way video delay under Default and Moving. The reason is that the network condition is significantly better than the required bandwidth

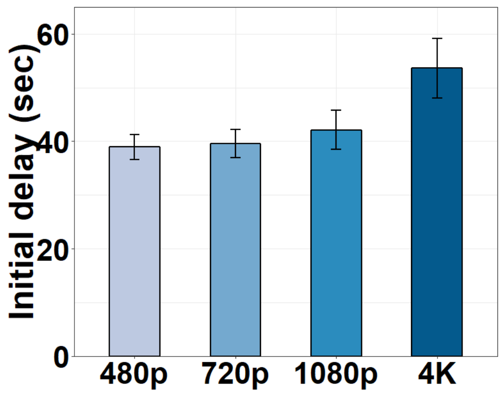


Figure 3.6 Initial delay in default scenarios.

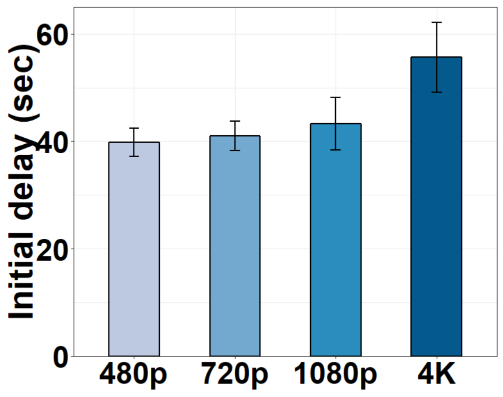


Figure 3.7 Initial delay in moving scenarios..

for lower-resolution video. Therefore, the network speed spikes introduced by the camera movement have little impact on the one-way video delay.

3.4.2.2 Initial Delay

Figure 3.6 and figure 3.7 shows the initial delay of 360° live video streaming in different scenarios. Even for the lowest resolution, viewers need to initially wait for approximately 39s before watching the 360° live video. The start-up waiting time for 4K videos can reach 55s. We observe from our results that the large initial delay is attributed to the heavy pre-processing completed in YouTube server. In order to allow the option of DASH streaming, the YouTube server has to transcode the viewer-requested 360° video into multiple resolutions. The initial transcoding takes far more time than the subsequent frame by frame transcoding since the YouTube server initially needs to set up the encoding configuration, generate the Media Presentation Description (MPD) file, and possibly convert the projection/format of the uploaded 360° video. Therefore, we see a larger initial delay than the one-way delay in Figure 3.4.

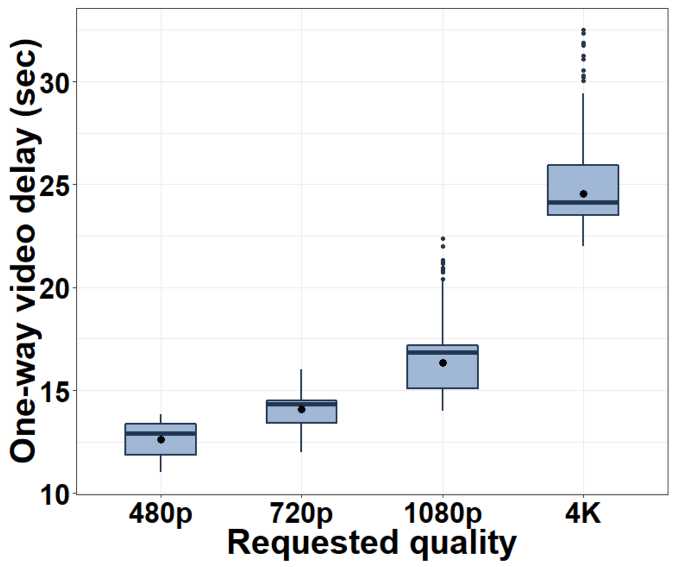


Figure 3.8 One-way video delay in DASH scenario.

From the above mentioned results, we can arrive at the conclusion that delay is probably the biggest issue for 360° live video streaming, even for low-resolution videos. Although the large initial delay and one-way video delay may be tolerant in some broadcasting applications, they are far from the requirement of real-time interactive playback. The large one-way video delay will make the viewing experience lag far behind the actual remote events. We note that server processing has played a primary role in the undesirable delay performance. New video/MPD preparation schemes at the server may be needed to expedite the 360° live video streaming.

3.4.3 Impact of DASH

We have shown in Figure 3.3 that using DASH in downlink streaming can reduce the re-buffering events. In this section, we conduct more experiments to investigate the impact of DASH on 360° live video streaming, especially for 4K resolution.

Figure 3.8 shows the one-way video delay of 360° live video streaming under different

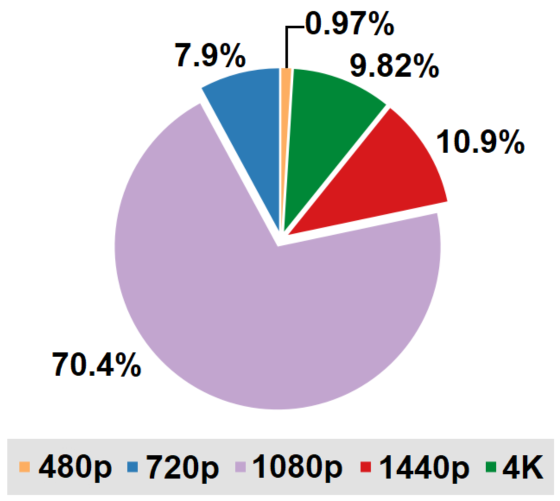


Figure 3.9 Resolution of DASH & Moving DASH sessions.

viewer-requested resolutions in DASH scenario. We observe that the one-way video delay is around 24.5s, which is smaller than the delay in Default and Moving scenarios. This is because DASH can dynamically change the streamed video quality to match the network condition. To further understand the viewing experience in DASH, we analyze the video quality actually received and viewed during the measurement sessions. Figure 3.9 plots the percentage of actually viewed video resolution under the viewer-requested resolution of 4K across both moving and non-moving scenarios. We observe that although the requested resolution is 4K, viewers will only watch a 4K video for 10% of the video session. Instead, viewers spend 70% of the time viewing a 1080p video. Furthermore, the viewers will to need switch among a total of five resolutions during the 3-min session. This could lead to mediocre user experience with an one-way video delay of 25s, video resolution of around 1080p, and quality variation among 5 resolutions. In fact, viewers are even suggested to directly request the non-DASH 1080p video which can achieve a one-way video delay of 18s and a constant video quality at 1080p.

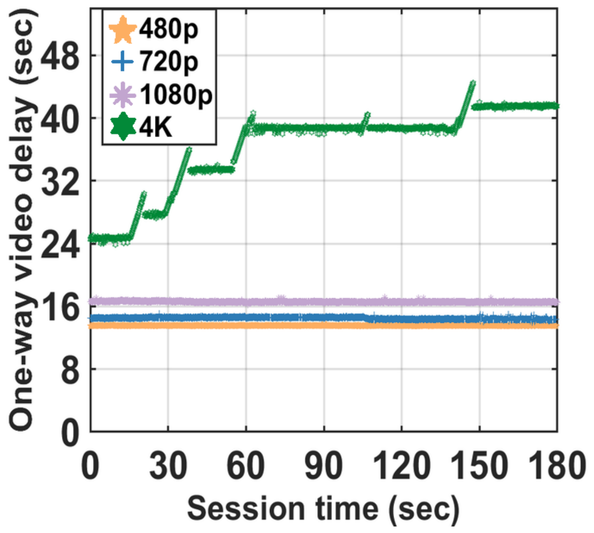


Figure 3.10 Per-session one way video delay in Default.

The reason of this nonpositive effects of DASH on 4K 360° live video is that a conservative quality increment algorithm is adopted in the player. To further optimize the 360° live video streaming system, it is imperative for system designers to develop new rate adaptation algorithms suitable for 4K 360° live videos.

3.4.4 Session Trace Analysis

We have so far focused on the statistical results of the system performance. In this section, we examine a 4K 360° live video session to identify more technical issues of existing streaming solutions.

Specifically, we show the trace of one-way video delay for different viewer-requested resolutions under Default scenario in Figure 3.10. We see that the one-way video delay of lower-resolution videos is stable. This is expected since lower-resolution videos can be easily processed and transported in a typical computing and network environment. The large delay stems from the initial setup as we discussed in Figure 3.6 and Figure 3.7.

For the 4K resolution, the one-way video delay continuously climbs. This is mostly due to the computation and networking constraints in handling 4K 360° live video. It is also interesting to notice that the one-way video delay of 4K video always suddenly drops after a spike and this pattern repeats across the session. After studying the trace of rebuffering status, we learn that the delay spike occurs when the player is rebuffering. Once the buffer is filled with enough video data, we observe that the player will drop a number of frames and then resume the playback, thereby leading to the sudden drop of one-way video delay. It is also worthwhile to point out that, no matter how long the rebuffering is, the one-way video delay always drops down approximately 2.5s when the live video recovers from the stall. This is because the player always skips roughly 2.5 seconds of frames when the buffer is ready for playback.

In sum, while the existing player for 360° live video streaming does implement a scheme to skip frames in order to keep the live video up to date, the one-way video delay is too large to be acceptable. The inconsistency between the remote site and viewer will disable all interactive applications. Therefore, new buffer management algorithms are needed to address the trade-off between interactivity and scene fidelity.

3.5 DISCUSSION AND FUTURE WORK

Other network environment. In this paper, we only focus on measuring the performance of 360° live video streaming in Wi-Fi upload and wired download environment. Indeed, the network condition could be more diverse in practice. For example, the video may be uploaded by LTE network through a smartphone or the viewer can use a mobile HMD to download the video via wireless networks. However, we point out that the system performance of other network conditions are unlikely to be better than our setup. Hence, we believe that our observation can generally be extended to other network environment.

Other 360° live video streaming platforms. We achieve a deeper understanding of 360° live video streaming on YouTube using a Ricoh 360° camera. However, there are other platforms (Facebook and Twitch) and cameras (Samsung and Nokia) supporting 360° live video streaming. The parallel computing and media processing capability of different online platforms can be different and may impact the 360° live video streaming performance. Similarly, the stitching quality and overhead of cameras may also determines the viewer experience. In this paper, we take the first step to study commercial 360° live video streaming. A comprehensive full-scale study of 360° live video streaming pipeline is needed as future work to provide further insight on optimizing the system performance. More specific metrics will be used to further understand the system. Especially for 4K 360° live video streaming, we will try to pinpoint the component for the relatively inferior performance.

3.6 CONCLUSION

This chapter proposes a new method to measure the 360° live video streaming in YouTube. Specifically, we have built an end-to-end measurement testbed to investigate the performance of 360° live video streaming on YouTube. We study the system performance and user experience under various requested resolutions (up to 4K). Our observation provides important insight for both viewers and developers. Due to frequent rebuffering and annoying delay performance, viewers are suggested not to live stream 4K 360° videos, even when DASH is enabled. Although delay-tolerant applications can be achieved by broadcasting 1080p video, the excessive one-way video delay will fail any interactive application. For system designers, it is important to devise new algorithms to improve the performance of video server processing/transcoding, player rate adaptation, and player buffer management. The enhanced designs will need to maximize the video quality while keeping the streamed content up to date. We believe the results of this research can enable a suite of future works on live 360° videos, an emerging yet relatively unexplored topic in multimedia systems community.

CHAPTER 4

An Analysis of Delay in Live 360° Video Streaming Systems

In this chapter, we extend our work to delay analysis over the whole pipeline of live 360° video streaming systems. Specifically, we will analyze relationships between different delay metrics and task-level measurement results.

4.1 Motivation

Live video streaming services have been prevalent in recent years (web 2019). With the emergence of 360° cameras, live 360° video streaming is emerging as a new way to shape our life in entertainment, online meetings, and surveillance. A recent study shows that about 70% of users are interested in streaming live sports in 360° fashion (360 2019). Compared with live regular video streaming, live 360° video streaming offers an immersive viewing experience by first recording a scene in all directions with an omnidirectional camera and then live streaming the 360° video to a user.

Delay is critical to live video streaming. Different delay metrics have various impacts on user experience. Complex initialization between a client and a server may lead to an excessive *start-up delay*, which decreases users' willingness to continue the viewing. The start-up delay may in turn result in a long *event-to-eye delay*, i.e., the time interval between the moment an event happens on the remote scene and the moment when the event is displayed on the client device. Long event-to-eye delay causes significant lags in streaming of live events such as sports, concerts, and business meetings. Moreover, the *frame rate* of a live video is determined by how fast frames can be pushed through the system pipeline. A low frame rate would make the video playback not smooth.

Guaranteeing user experience in *360°* live video streaming against the above negative effects of delay is especially challenging. First, compared to regular videos, live 360° videos generate far more data and require additional processing steps to stitch, project, and display the omnidirectional content. Second, the aforementioned delay metrics have an independent effect on user experience. For example, a short event-to-eye delay does not guarantee a high frame rate. To prevent undesirable user experience caused by delays, a key prerequisite is to understand how different components of a live 360° streaming system contribute to the three delay metrics. In particular, we must answer the following questions: (1) what tasks does a live 360° video streaming system have to complete, and (2) how does the time spent on each task affect user experience?

While a number of measurement studies have been conducted on regular 2D live video streaming (Wang et al. 2016; Siekkinen et al. 2018; Tang et al. 2016b), the delay of live 360° video streaming has not been well understood. Recent works in 360° video streaming focused on rate adaptation algorithms (Corbillon et al. 2017a, 2018; Nguyen et al. 2018b; Chen et al. 2019) and encoding/projection methods (Corbillon et al. 2017b; Nasrabadi et al. 2017; Zhou et al. 2017b). The only two existing measurement studies on live 360° videos (Yi et al. 2019; Liu et al. 2019) were performed on commercial platforms; both were only able to treat the system as a black box and performed system-level measurements. They were not able to dissect the streaming pipeline to analyze how each task of a live 360° video streaming system contributes to the start-up delay, event-to-eye delay, and frame rate.

In this paper, we aim to bridge this gap by conducting an in-depth measurement study of the time consumption across the end-to-end system pipeline in live 360° video streaming. Such an analysis can pinpoint the bottleneck of a live 360° video streaming system in terms of different delay metrics, thus prioritizing the system optimization efforts. To our best knowledge, the proposed measurement study is the first attempt to understand the task-level time consumption across the live 360° video streaming pipeline and their impacts on different delay metrics and user experience.

Performing such a measurement study is non-trivial because commercial live 360° video streaming platforms are usually implemented as a black box. The closed-source implementation makes it almost impossible to measure the latency of each computing task directly. To tackle this challenge, we build a live 360° video streaming research prototype, called ***Zeus***, using publicly available hardware devices, SDKs, and open-source software packages. Composed of five components – a 360° camera, camera-server transmission, a video server, server-client transmission, and a video client, `Zeus` can be easily replicated for future live 360° video streaming studies in areas such as measurement, modeling, and algorithm design.

Using `Zeus`, we evaluate micro-benchmarks to measure the time consumption of each task in all five system components. Our measurement study has three important findings. First, video frame copying between the CPU and GPU inside the camera consumes non-negligible time, making it a critical task towards achieving a desired frame rate on the camera (typically 30 frames per second, or fps). Second, stitching a 360° video frame surprisingly has only a minor effect on ensuring the frame rate. Third, server initialization before live streaming 360° videos is very time-consuming. The long start-up delay leads to a significant event-to-eye delay, indicating an annoying streaming lag between what happens and what is displayed. Overall, the camera is the bottleneck for frame rate whereas the server is the obstacle for low start-up and event-to-eye delay.

Because of the implementation differences between `Zeus` and commercial live 360° video streaming platforms, the absolute values of the results obtained with `Zeus` may potentially differ from those measured on commercial platforms. Therefore, we further perform measurements on a commercial system, built using Ricoh Theta V and YouTube, treating it as a black box and compare its component-level time consumption to the values obtained with `Zeus`. We observe that the time consumption of each component in `Zeus` has a strong correlation with that of the commercial system, suggesting that our findings can be generalized to real-world live 360° video streaming systems.

4.2 Related Work

Regular live video streaming. A corpus of work has gained understanding of regular live video streaming platforms. Wang et al. (Wang et al. 2016) analyzed the delay of Periscope, a popular live video streaming App and showed that the client-buffering, chunking, and polling are the major contributors to the high latency of HTTP Live Streaming (HLS) systems. Siekkinen et al. (Siekkinen et al. 2018) studied user experience on mobile live video streaming and observed that video transmission time is highly affected by live streaming protocols. Researchers (Nihei et al. 2018; Sun et al. 2019) studied encoding methods to reduce the transmission time introduced by bandwidth variance. Although these works are beneficial to regular live video streaming, the observations cannot be applied to 360° videos because multiple video views and extra processing steps of the live 360° video streaming.

360° video-on-demand streaming. Measurement studies on streaming pre-recorded 360° videos have been conducted (Afzal et al. 2017a; Zhou et al. 2017b; de la Fuente et al. 2019; Grzelka et al. 2019). Afzal et al. (Afzal et al. 2017a) characterized 360° videos by examining thousands of pre-recorded YouTube videos across more than a dozen categories and concluded that latency is a highly important criterion when viewing 360° videos in VR headsets. Zhou et al. (Zhou et al. 2017b) studied the encoding solution and streaming strategy of Oculus 360° video-on-demand (VoD) streaming. They reverse-engineered the offset cubic projection adopted by Oculus which encodes a distorted version of the spherical surface and devotes more information to the view in a chosen direction. Previous studies also showed that the delay of 360° VoD streaming affects viewport-adaptive streaming algorithms (de la Fuente et al. 2019; Grzelka et al. 2019) and the rendering quality. Despite all efforts on 360° VoD measurement studies, none of them considers the 360° camera and the management of a live streaming session, which are essential components in 360° *live* video streaming. Thus, these works provide limited insight to live 360° video streaming.

Live 360° video streaming. Two recent works (Yi et al. 2019; Liu et al. 2019) have studied commercial live 360° video streaming systems. Jun et al. (Yi et al. 2019) investigated the YouTube platform for up to 4K resolution and showed that viewers suffer from a high event-to-eye delay in live 360° video streaming. Liu et al. (Liu et al. 2019) conducted a crowd-sourced measurement on YouTube and Facebook. Their work verified the high event-to-eye delay and showed that viewers experience long session stalls. Chen et al. (Chen et al. 2019) proposed a stitching algorithm for tile-based live 360° video streaming under strict time budgets. Despite the improved understanding of commercial live 360° video streaming platforms, none of the existing studies dissected the delay of a live 360° streaming pipeline at the component or task level. They failed to show the impacts of components/tasks on delay metrics (start-up, event-to-eye, and frame rate). Our work delves into each component of a canonical live 360° video system and presents an in-depth delay analysis.

4.3 Canonical System Architecture

In live 360° video streaming, a 360° camera captures the surrounding scenes and stitches them into a 360° equirectangular video frame. The 360° camera is connected to the Internet so that it can upload the video stream to a server. The server extracts the video data and keeps them in a video buffer in memory. The server will not accept client requests until the buffered video data reach a certain threshold. At that time, a URL to access the live streaming session will become available. Clients (PCs, HMDs, and smartphones) can initiate the live streaming via the available URL. The server first builds a connection with the client and then streams data from the buffer. Upon receiving data packets from the server, the client will decode, project, and display 360° video frames on the screen.

As shown in the system architecture in Figure 4.1, the above workflow can be naturally divided into five *components* – a camera, camera-server transmission (CST), a server, server-client transmission (SCT), and a client. These components must complete several computing *tasks* in sequence.

First, the 360° camera completes the following tasks.

- *Video Capture* obtains multiple video frames from regular cameras and stores them in memory.
- *Copy-in* transfers these frames from the memory to the GPU.
- *Stitching* utilizes the GPU to stitch multiple regular video frames into an equirectangular 360° video frame.

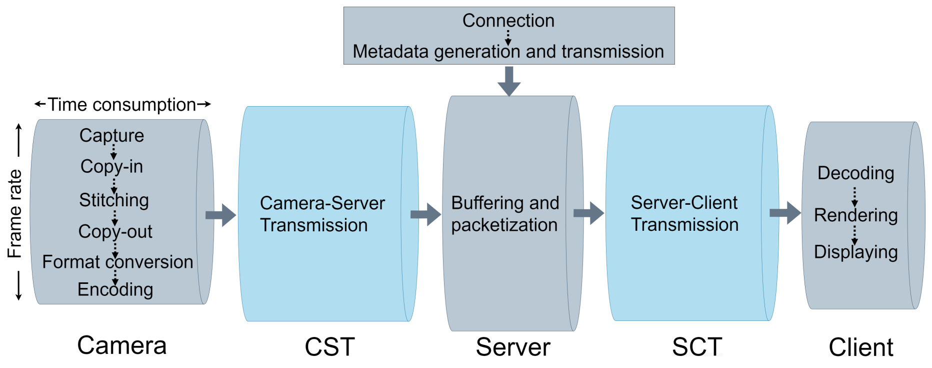


Figure 4.1 The architecture of live 360° video streaming and the tasks of the 5 system components. The top rectangle shows one-time tasks whereas the 5 bottom pipes show the pipeline tasks that must be passed through for every frame.

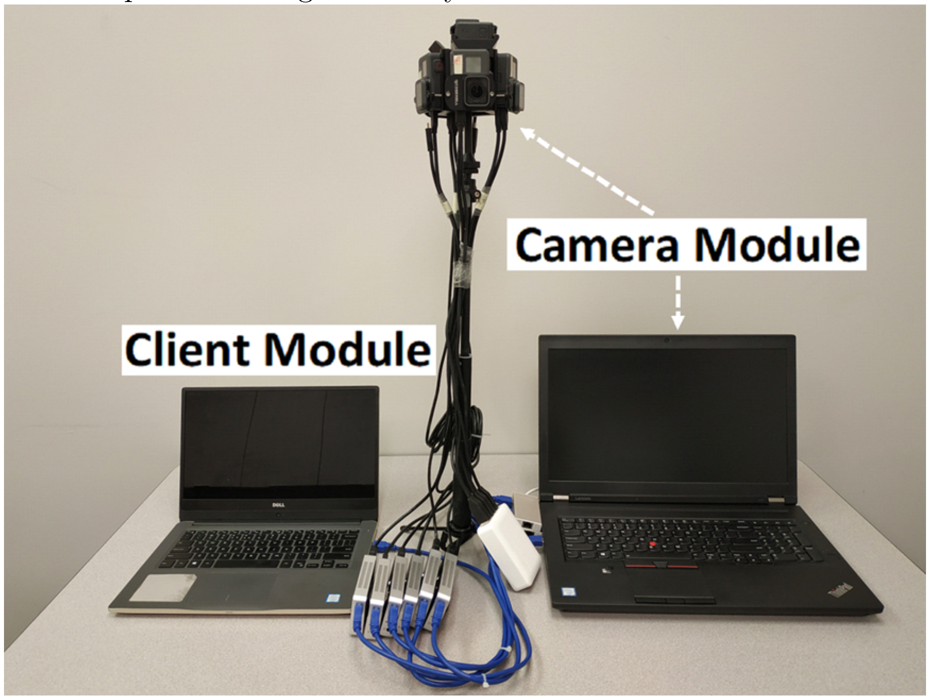


Figure 4.2 The `Zeus` prototype.

- *Copy-out* is the process of transferring the equirectangular 360° video frame from the GPU to the memory.

- *Format Conversion* leverages the CPU to convert the stitched RGB frame to the YUV format.

- *Encoding* is the task that compresses the YUV equirectangular 360° video frame using an H.264 encoder.

Then the CST component, e.g., WiFi plus the Internet, delivers data packets of the 360° video frame from the camera to the server.

Next, the following tasks are accomplished at the server.

- *Connection* is the task where the server builds a 360° video transfer connection with the client after a user clicks the live streaming URL.

- *Metadata Generation and Transmission* is the process of producing a metadata file for the live 360° video and sending it to the client.

- *Buffering and Packetization* is the process where the video data wait in the server buffer, and then, when they are moved to the buffer head, the server packetizes them for streaming.

The SCT component will then transmit data packets of the 360° video from the video server to the video client.

Finally, the client completes the tasks detailed below.

- *Decoding* converts the received packets into 360° video frames.

- *Rendering* is a special task for 360° videos that projects an equirectangular 360° video frame into a spherical frame and then renders the pixels of the selected viewport.

- *Display* is the process for the client to send the viewport data to the display buffer and for the screen to refresh and show the buffered data.

It should be emphasized that the connection and metadata generation and transmission are *one-time tasks* for a given streaming session between the server and a client, whereas all other tasks are *pipeline tasks* that must be passed through for every video frame.

4.4 Dissecting Delay Metrics

In this section, we identify three main delay metrics that affect user experience and explain how they are affected by the time consumption for different components, denoted by the length of each pipe as shown in Figure 4.1.

Start-up delay. This is the time difference between the moment when a client sends a streaming request and the moment when the first video frame is displayed on the client screen. An excessive start-up delay is one primary reason that decreases users' willingness to continue video viewing (sta 2020). Formally, given the time consumption for the one-time connection and metadata generation and transmission $T_{srv,once}$, the server-client transmission of a frame T_{sct}, and the time to process and display a frame on the client device T_{clnt}, the start-up delay D_{start} can be expressed as,

$$D_{start} = T_{srv,once} + T_{sct} + T_{clnt} \tag{4.1}$$

The time consumption in the camera and camera-server transmission does not affect the start-up delay. This is attributed to the system architecture where the live streaming will not be ready until the server buffers enough video data from the camera. Therefore, there should have been video frames already in the server before a streaming URL is ready, and a client request is accepted.

Event-to-eye delay. This is the time interval between the moment when an event occurs on the camera side and the moment when the event is displayed on the client device. A long event-to-eye delay will make users perceive a lag in live broadcasting of sports and concerts. It will also decrease the responsiveness of real-time communication in interactive applications such as teleconferences. It is evident that all tasks in live 360° streaming contribute to the event-to-eye delay $D_{event-to-eye}$. After camera capture, video frames must go through and spend time at all system components before being displayed on the screen, i.e.,

$$D_{event-to-eye} = T_{cam} + T_{cst} + T_{srv,once} + T_{srv,pipe} + T_{sct} + T_{clnt} \tag{4.2}$$

where T_{cam}, T_{cst}, $T_{srv,pipe}$ are the time consumption of a frame on the camera, camera-server transmission, and the pipeline tasks in the server (buffering and packetization). Note that although the one-time connection and metadata tasks are not experienced by all frames, their time consumption will be propagated to subsequent frames, thus contributing to the event-to-eye delay.

Frame rate. This indicates how many frames per unit time can be processed and pushed through the components in the system pipeline. The end-to-end frame rate of the system, FR, must be above a threshold to ensure the smoothness of video playback on the client screen. It is determined by the minimum frame rate among all system components and can be formally represented as follows,

$$FR = \min\{FR_{cam}, FR_{cst}, FR_{srv}, FR_{sct}, FR_{clnt}\} \tag{4.3}$$

where $FR_{cam}, FR_{cst}, FR_{srv}, FR_{sct}, FR_{clnt}$ are the frame rate of each system component. It is important to note that the frame rate of a component, i.e., how many frames can flow through the pipe per unit time, is not necessarily the inverse of the per-frame time consumption on that component if multiple tasks in a component are executed in parallel by different hardware units. As illustrated in Figure 4.1, the end-to-end frame rate is determined by the radius rather than the length of each pipe.

Dissection at the task level. Since the tasks within each component are serialized, the time consumption and frame rate for each component (e.g., T_{cam}) can be dissected in the same way as before. We omit the equations due to page limit.

4.5 The Zeus Research Prototype

Commercial live 360° video streaming systems are closed-source and there is no available tool to measure the latency breakdown of commercial cameras (e.g., Ricoh Theta V), servers (e.g., Facebook), and players (e.g., YouTube) at the task level. To enable measuring the impact of the time consumption at the task level on live 360° video experience, we build a live 360° video streaming system prototype, Zeus, shown in Figure 4.2, as a reference implementation to the canonical architecture. We build Zeus using only publicly available hardware and software packages so that the community can easily reproduce the reference implementation for future research.

Hardware design. The 360° camera in Zeus consists of six GoPro Hero cameras ($400 each) (gop 2020) held by a camera rig and a laptop serving as the processing unit. The camera output is processed by six HDMI capture cards and then merged and fed to the laptop via three USB 3.0 hubs. The laptop has an 8-core CPU at 3.1 GHz and an NVIDIA Quadro P4000 GPU, making it feasible to process, stitch, and encode live 360° videos. The video server runs Ubuntu 18.04.3 LTS. The client is a laptop running Windows 10 with an Intel Core i7-6600U CPU at 2.6 GHz and an integrated graphics card.

Software design. The six cameras are configured in the `SuperView` mode to capture wide-angle video frames. We utilize the VRWorks 360 Video SDK (sdk 2019) to capture regular video frames in a pinned memory. To reduce the effects of camera lens distortion during stitching, we first utilize the OpenCV function `cv.fisheye.ca- librate()` and the second-order distortion model (sec 2013) to calculate camera distortion parameters (Zhang 2000). Video frames are then calibrated during stitching to guarantee that the overlapping area of two adjacent frames will not be distorted. We copy the frames to the GPU via `cudaMemcpy2D()` and use `nvssVideoStitch()` for stitching. Finally, we use FFmpeg for encoding and streaming the 360° video. We use Real-Time Message Protocol (RTMP) in the camera to push the live video for low-delay transmission. This is similar to most commercial cameras, e.g., Ricoh Theta V and Samsung Gear 360.

For the video server, we run a Nginx-1.16.1 server. We use the HTTP-FLV protocol to stream the video from the server to the client because it can penetrate firewalls and is more acceptable by web servers, although other popular protocols, e.g., HLS, could have also been used. The HLS protocol consumes time for chopping a video stream into video chunks with different video quality, thus the start-up delay might be higher. To enable the server to receive RTMP live video streams from the 360° camera and deliver HTTP-FLV streams to the client, Nginx is configured as *nginx-http-flv-module* (mod 2018).

We design an HTML5 based video client using FLV.js, a flash-based module written in JavaScript. Three.js is used to fetch a video frame from Flv.js and project it onto the sphere format using `render()`. The sphere video frame is stored at the HTML5 element `<canvas>`, which will be displayed on webpages. The client is embedded in a Microsoft Edge browser with hardware acceleration enabled to support the projection and decoding.

Measuring latency. We can measure the time consumption of most tasks by inserting timestamps in `Zeus`. The exceptions are the camera-server transmission (CST) and server-client transmission (SCT), where the video stream is chunked into packets for delivery since both the RTMP and HTTP protocols are built atop TCP. As frame ID is not visible at the packet level, we cannot identify the actual transmission time of each frame individually. We instead approximate this time as the average time consumption for transmitting a video frame in CST and SCT. For example, for the per-frame time consumption of CST, we first measure the time interval between the moment when the camera starts sending the first frame using `stream_frame()` and the moment when the server stops receiving video data in `ngx_rtmp_live_av()`. We then divide this time interval by the number of frames transmitted.

4.6 Results

In this section, we report the time consumption of the tasks across system components and discuss their effects on the start-up delay, event-to-eye delay, and frame rate. We also evaluate the time consumption of the tasks under varying impact factors to expose potential mitigation of long delay that affects user experience.

4.6.1 Experimental setup

We carry out the measurements inside a typical lab environment located in a university building, which hosts the camera and the client. We focus on a single client in this paper and leave multiple-client scenarios as future work. To mimic the real-world conditions experienced by commercial 360° video systems, we place the server at another university campus over 800 miles away. Although the camera and the client are in the same building, this does not affect the results significantly as the video data always flows from the camera to the server and then to the client.

The camera is fixed on a table so that the video content generally contains computer desks, office supplies, and lab personnel. By default, each GoPro camera captures a 720p regular video, and the stitched 360° video is configured as 2 Mbps with the resolution ranging from 720p to 1440p (2K). We fix the resolution during a session and do not employ adaptive streaming because we want to focus on the most fundamental pipeline of live 360° video streaming without advanced options. The frame rate of videos is fixed at 30 fps. The Group of Pictures (GOP) value of the H.264 encoder is set as 30. A user views the live 360° video using a laptop client. A university WiFi is used for the 360° camera to upload the stitched video and for the video client to download the live video stream. The upload and download bandwidth of the university WiFi are 16 Mbps and 20 Mbps, respectively. For each video session, we live stream the 360° video for 2 minutes and repeat this 20 times. The average and standard deviation of the results are reported.

4.6.2 360° Camera

4.6.2.1 Video Capture Task

We vary the resolutions of the captured regular videos and show the video capture time in Figure 4.3. The video capture time is short in general. It takes 1.68 ms to capture six 480p video frames and 2.05 ms for six 720p frames. Both resolutions provide abundant details for stitching and are sufficient to generate 360° videos ranging from 720p to 1440p that are currently supported in today's live 360° video platforms (YT 2020; FB 2020). While capturing six 1080p and 1440p regular frames would consume more time, such high resolutions

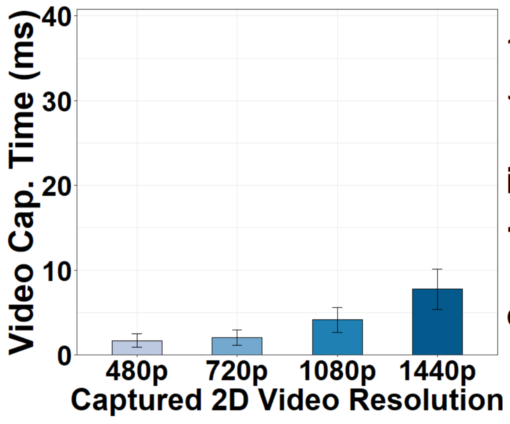


Figure 4.3 Video capture time versus capture resolutions.

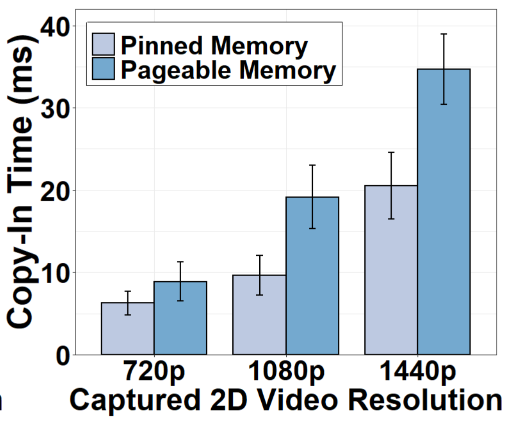
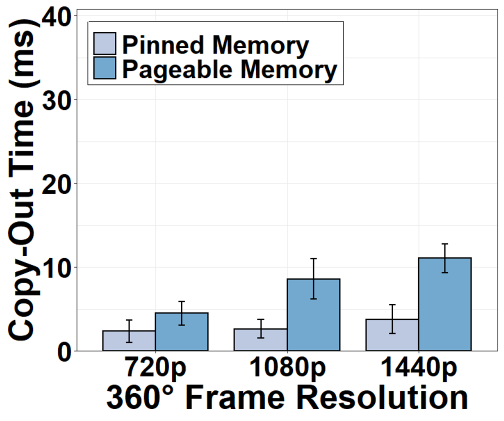


Figure 4.4 Copy-in time from different memory locations.

of input regular videos are typically not required in current live 360° video applications.

4.6.2.2 Copy-in and Copy-out Tasks

Figures 4.4-4.5 show that the CPU-GPU transfer time is non-negligible. It takes 6.28 ms to transfer six 720p video frames from pinned memory to GPU before stitching and as high as 20.51 ms for copying in six 1440p frames. The copy-out time is shorter than the copy-in time, taking 2.33 ms for a 720p 360° frame using the pinned memory and 4.47 ms using the pageable memory. This is because the six 2D regular frames have been stitched into one 360° frame, which reduces the amount of video data to be transferred. The results indicate that transferring video data for GPU stitching does introduce extra processing and such overhead can only be justified if the stitching speed in the GPU is superior. Moreover, it is evident that pinned memory is preferred in CPU-GPU transfer. Pinned memory can directly communicate with the GPU whereas pageable memory has to transfer data between the GPU and the CPU via the pinned memory.



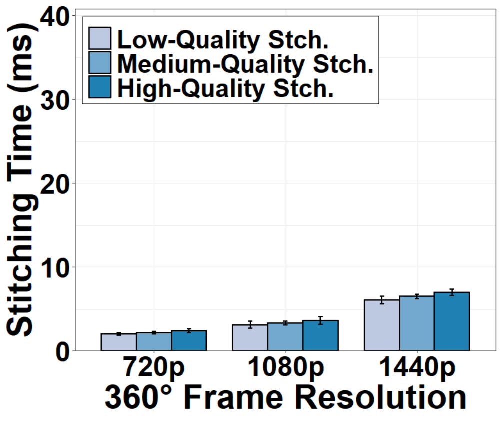


Figure 4.5 Copy-out time from different memory locations.

Figure 4.6 Frame stitching time vs. stitching options.

4.6.2.3 Stitching Task

We measure the stitching time using different stitching quality options in the VRWorks 360Video SDK, which execute different stitching algorithms. For example, "high stitching quality" applies an extra depth-based mono stitching to improve the stitching quality and stability. Surprisingly, the results in Figure 4.6 show that stitching time is not a critical obstacle compared to the CPU-GPU transfer. It takes as low as 1.98 ms for stitching a 720p equirectangular 360° video frame with high stitching quality and 6.98 ms for a 1440p frame. This is in sharp contrast to previous 360° video research (Silva et al. 2016) that stressed the time complexity of live 360° video stitching and proposed new stitching methods to improve the stitching speed. The short stitching time is attributed to the fact that, given the fixed positions of six regular cameras, modern GPUs and GPU SDKs can reuse the corresponding points between two adjacent 2D frames for stitching each 360° frame without having to recalculate the overlapping areas for every frame.

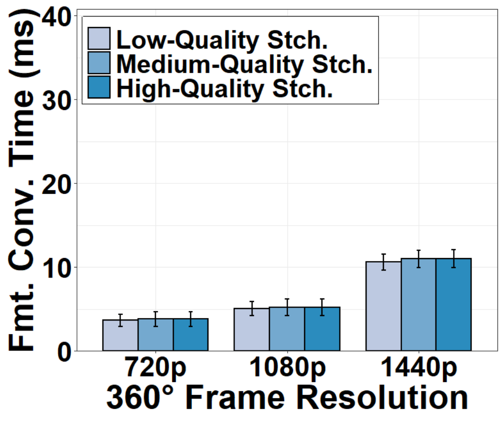


Figure 4.7 Format conversion time vs. stitching options.

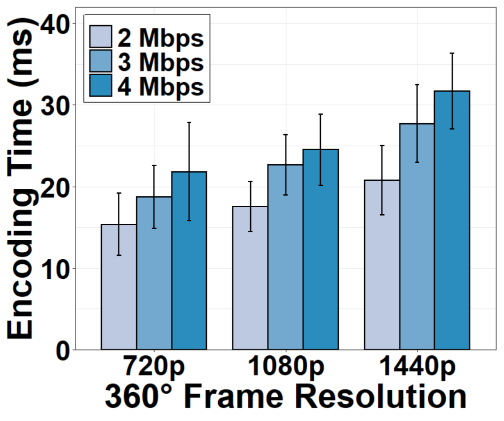


Figure 4.8 Encoding time under different bitrates.

4.6.2.4 Format Conversion Task

Figure 4.7 shows the time consumption for converting the stitched 360° frame to YUV format before encoding. This time is 3.75 ms for a 720p video frame and it is increased to 10.86 ms for a 1440p frame. We also observe that the stitching quality has a negligible effect. This is because format conversion time is primarily determined by the number of pixels to be converted rather than the choice of stitching algorithms.

4.6.2.5 Encoding Task

Figure 4.8 illustrates the encoding time under different encoding parameters. As expected, encoding time is one of the major tasks in the camera. Encoding a 1440p 360° frame at 2 Mbps consumes 20.74 ms on average; the encoding time is reduced to 15.35 ms when the resolution is 720p as fewer pixels need to be examined and encoded. We also observe that decreasing the bitrate by 1 Mbps can result in a 16.68% decrease in the encoding time. To achieve a lower bitrate in an encoder, a larger quantization parameter (QP) is typically used to produce fewer non-zero values after the quantization, which in turn reduces the time to encode these non-zero coefficients. Given the importance of encoding in the overall camera time consumption, a tradeoff between frame rate and encoding quality must be struck in the camera.

Furthermore, it is interesting to see that the encoding time increases as the GOP increases, and then it starts decreasing once the GOP reaches a certain threshold. Increasing the GOP length enforces the encoder to search more frames to calculate the inter-frame residual between the I-frame and other frames, leading to a larger encoding time. However, an I-frame is automatically inserted at scene changes if the GOP length is too long, which will decrease the encoding time. Our results indicate that the GOP threshold for the automatic I-frame insertion is somewhere from 40 to 50.

4.6.2.6 Impact on Delay Metrics

Our camera can achieve live streaming of 720p 360° videos at 30 fps, which is consistent with the performance of state-of-the-art middle-end 360° cameras such as Ricoh Theta S (RSc 2020). The camera conducts a sequence of tasks for a frame one by one and does not utilize parallel processing. Therefore, the frame rate of the camera output is simply an inverse of the total time consumption of all tasks in the camera. This is consistent to our results that the overall time consumption of camera tasks for a 720p frame is less than 33.3 ms. Our results suggest that certain tasks can be optimized to improve the output quality of the 360° camera. In addition to the well-known encoding task, the optimization of CPU-GPU transfer inside the camera is important, since this task consumes a noticeable amount of time. On the other hand, there is little scope to further improve the stitching task since the current stitching time is already low. Moreover, the parameter space of major tasks, such

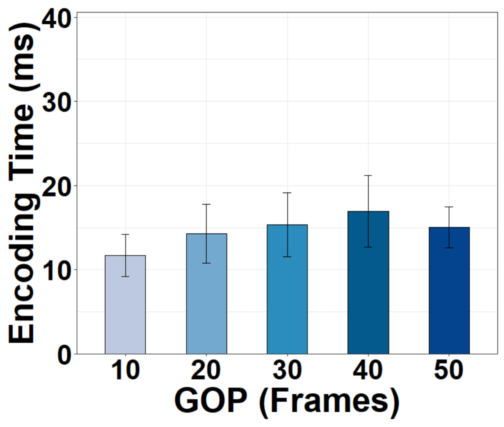


Figure 4.9 Encoding time of a 720p frame versus GOP.

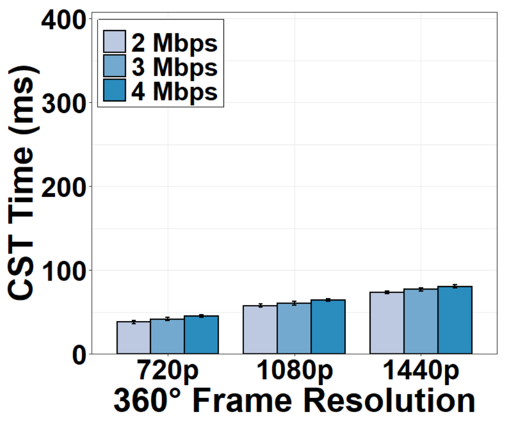


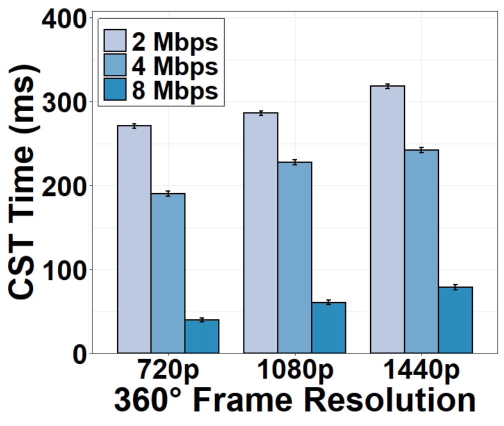
Figure 4.10 CST time under different bitrates.

as encoding and CPU-GPU transfer, should be explored to balance the frame rate and the video quality. These efforts can potentially improve the frame rate to support live streaming of higher-quality videos that are only offered in high-end cameras or even unavailable in today's markets.

Note that tens of milliseconds spent on the camera will not affect the event-to-eye delay in equation (4.2) significantly. The typical event-to-eye delay requirement for interactive applications is no more than 200 ms (Szigeti & Hattingh 2004), and it can be further relaxed to 4-5 seconds for live broadcasting of events (Xiao 2008). We also reiterate that the camera has no effect on the start-up delay as defined in equation (4.1).

4.6.3 Camera-Server Transmission

We vary the bitrate and resolution of 360° videos sent by the camera and show the CST time in Figure 4.10. The transmission time over the Internet is generally long compared to the time consumption in the camera. It is clear that the CST time increases when the encoding quality is higher. For example, it takes 37.83 ms to transmit a 720p 360° frame at 2 Mbps



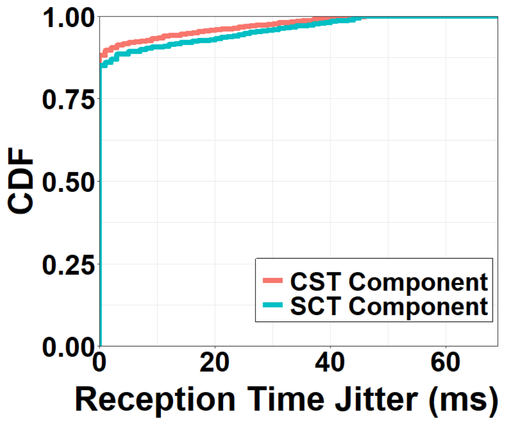


Figure 4.11 CST time versus upload bandwidth.

Figure 4.12 Jitter of packet reception time.

and as long as 73.23 ms for a 1440p frame.

In addition, we throttle the upload bandwidth to 2, 4, and 8 Mbps using *NetLimiter* and evaluate the impact of network conditions on the CST time given the same video bitrate of 2 Mbps. Figure 4.11 shows that, when the upload bandwidth is reduced to 2 Mbps, the CST time dramatically increases to 270.79 ms for a 720p 360° frame, 286.13 ms for a 1080p frame, and 318.17 ms for a 1440p frame. We also observe that when the upload bandwidth is 8 Mbps, the CST time is similar to the case when there is no bandwidth throttling as in Figure 4.10. This confirms that 8 Mbps is sufficient to support the 360° video transmission.

4.6.3.1 Impacts on Delay Metrics

The time consumption in the CST component generally has no effect on the frame rate, since the CST component handles video data *packet by packet* continuously. As long as consecutive packets are pushed back to back to the CST component, the output frame rate of the CST will not change regardless of the processing time of a packet. One exception might be when the variance of the packet transmission time in the CST component (jitter) is very large. Fortunately, Figure 4.12 shows that 90% of the packets are received 2 ms after the reception of their previous packets. Thus, packets flow through the CST component continuously and the negative effects on frame rate are not observed.

Similar to the camera, the CST component does not affect the start-up delay. However, the large CST time plays an essential role in satisfying the requirement of event-to-eye delay, especially when streaming high-quality videos in live interactive applications.

Since modern WiFi (802.11ac) has sufficient bandwidth to support a reasonable CST time and stable delay jitter, future efforts should focus on improving the transmission design in terms of the robustness against challenged networks.

4.6.4 Video Server

4.6.4.1 Connection Task

Once enough 360° video frames are received from the camera, the server is ready to accept a client request by proceeding to the connection task. Figure 4.13 shows that the time consumption on the connection task is long, taking around 900 ms. The connection task starts with a TCP three-way handshake between the client and the server which consumes tens of milliseconds. Then the server spends the majority of time (hundreds of milliseconds) preparing the initial response to the client, which includes information about the streaming session. It creates new threads, initializes data structures for the live video stream management, and registers different handler functions, e.g., `ngx_http_request_handler`, for accepting the client request. Finally, the server transmits the initial HTTP response (excluding video data) to the client. Since the amount of data transmitted during the connection task is small, increasing download bandwidth does not reduce the connection time in a noticeable

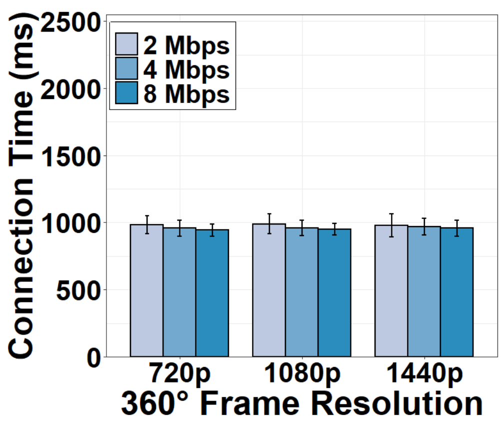


Figure 4.13 Connection time under different download bandwidths.

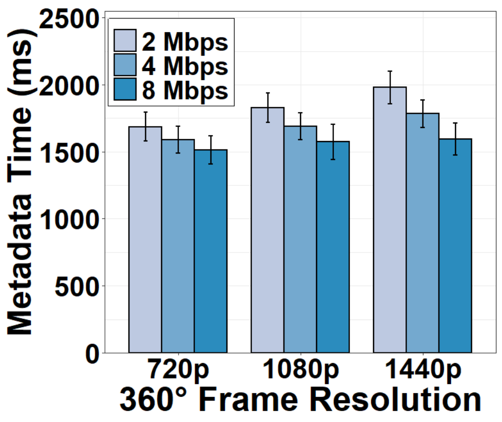


Figure 4.14 Metadata gen. and tx time under different download bandwidths.

way.

4.6.4.2 Metadata Generation and Transmission Task

Figure 4.14 shows the metadata generation and transmission time for download bandwidth of 2, 4, and 8 Mbps. The time consumption is long because the server must create and transmit a metadata file detailing the format, encoding, and projection parameters of the 360° video. This procedure includes retrieving video information from the camera, registering functions and creating data structures, generating live video streaming threads to build the metadata file, and sending it to the client. Since this is not a parallel process, it takes a long time to execute these steps. The shortest time is 1512.90 ms for a 720p video stream under the download bandwidth of 8 Mbps. Since reducing the bandwidth from 8 Mbps to 2 Mbps only reduces the task time slightly, we can infer metadata generation dominates this task.

4.6.4.3 Buffering and Packetization Task

We found that the time consumption for the server to buffer video data and packetize it for downlink streaming is negligible. In other words, the server buffer is very small in order to send out the received camera captured frames as soon as possible. Moreover, the Nginx server utilizes pointers to record the locations of the received video data in the server buffer and then directly fetches the video data using the pointers when adding FLV headers to generate HTTP-FlV packets. No memory copy or transfer is needed for the received video data, expediting the packetization task.

4.6.4.4 Impacts on Delay Metrics

Since the connection and metadata generation and transmission in the server occurs before any video frames are pushed into the pipeline for streaming, they do not affect frame rate. Given the negligible buffering and packetization time, the end-to-end frame rate would not be impacted by the server.

However, the large time consumption of the connection and metadata generation and transmission tasks introduces an excessive start-up delay that may degrade the users' retention of viewing after initializing the video session. The start-up delay in turn yields a long event-to-eye delay. Even though the connection and metadata tasks occur only once, video data are accumulated in the server buffer during the session start-up. The subsequent video frames have to wait until previous frames are sent out, and thus, they also experience a long event-to-eye delay. The long event-to-eye delay can undermine the responsiveness requirement ($\sim$ 200 ms (Szigeti & Hattingh 2004)) of interactive video applications. To relieve the negative effects of long start-up and event-to-eye delay, researchers should focus

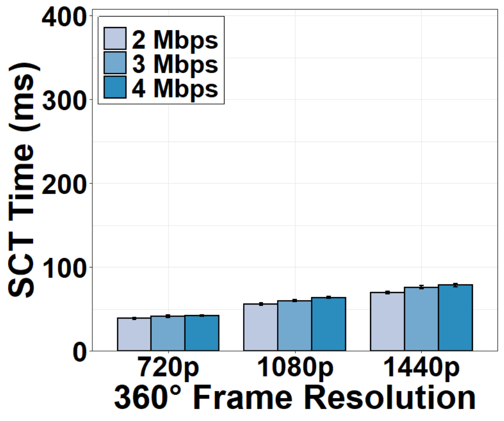


Figure 4.15 SCT time under different bitrates.

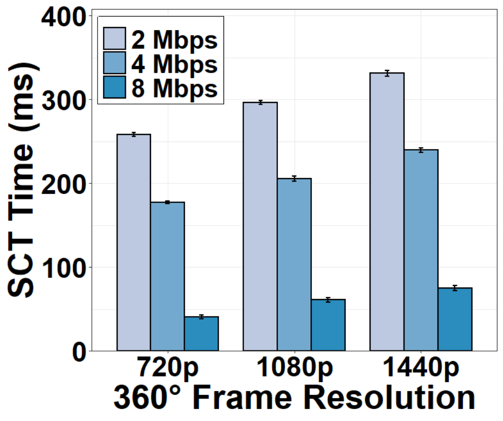


Figure 4.16 SCT time versus download bandwidth.

on optimizing the workflow and data management in the server to minimize the preparation step during the connection task.

4.6.5 Server-Client Transmission

Figure 4.15-4.16 show the SCT time for streaming a 360° frame from the server to the client. The time consumption is similar to the CST task, taking 41.07 ms for a 720p 360° frame and 74.98 ms for a 1440p frame. This is because the camera and the client are equally far away from the server in our setup, and both the upload bandwidth and download bandwidth are high enough to support the video to be streamed. Similar to the CST time, the SCT time decreases as the video quality degrades and the download bandwidth increases.

4.6.5.1 Impacts on Delay Metrics

Unlike the CST component, the SCT component is an important contributor to the start-up delay because the first frame has to be streamed to the client before the display. On the other hand, their impact on the event-to-eye delay and frame rate are similar. Users will experience lag of events if the SCT time is high. If the network conditions are not stable, the

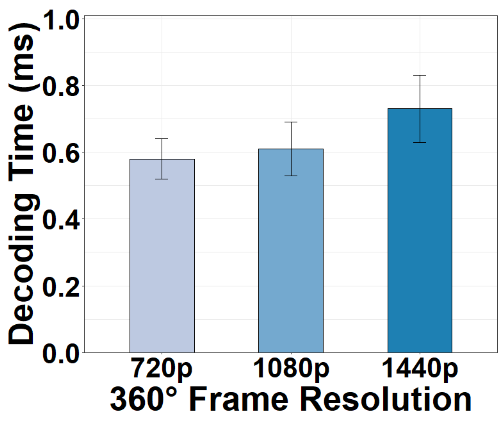


Figure 4.17 Decoding time versus 360° frame resolutions.

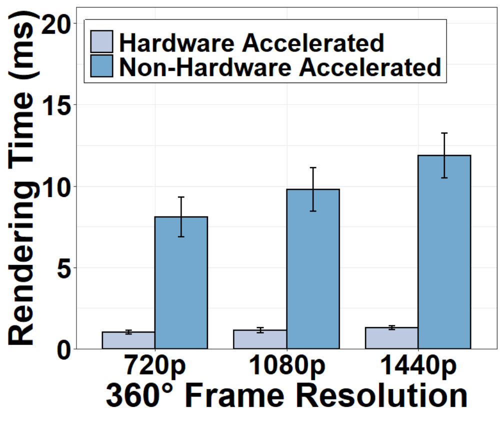


Figure 4.18 Rendering time of different hardware options.

continuous packet reception in Figure 4.12 may not hold for the SCT component, resulting in a reduced frame rate.

4.6.6 Client

4.6.6.1 Decoding Task

Figure 4.17 shows the decoding time of a 360° frame in the client. The decoding time is negligible; its average value over different resolutions is 0.62 ms. Modern computers use dedicated hardware decoder for video decoding, significantly expediting the complex decoding procedure that could have taken much longer in the CPU.

4.6.6.2 Rendering Task

We show the rendering time under different hardware configurations in Figure 4.18. We see that the rendering time is also negligible, and the hardware acceleration expedites the task. The time consumption for projecting the equirectangular frame and rendering the viewport using GPU-based hardware acceleration is 1.29 ms for a 1440p video frame, an 89.13% decrease from the non-acceleration mode. The performance improvement is achieved by the massive parallelism of the GPU processing. Note that, although video frames are transferred to the client GPU for rendering, this process is much less time consuming than the CPU-GPU frame transfer in the camera, because a video frame is fetched from the WiFi module to the GPU through Direct Memory Access.

4.6.6.3 Display Task

The display task involves two steps. First, the viewport data are sent to the display buffer. Second, the screen refreshes at a certain frequency to display the most recently buffered data. We found that the time consumption for sending data to the display buffer is negligible, and thus the display time is determined by the refresh frequency. In our case, the screen refreshes at 60 Hz, resulting in a 16.67 ms display time.

4.6.6.4 Impact on Delay Metrics

The frame rate of the client output is the inverse of its time consumption because of the non-parallel frame-processing similar to the camera. Although extra projection and rendering are needed for 360° videos, the tasks of the client can be completed with a fairly short time consumption to achieve the 30-fps frame rate. Similarly, the client contribution to the start-up delay and the event-to-eye delay is much less than the contribution of the server or SCT component. We conclude that the client has minor impacts on the user experience due to its negligible contribution to the three delay metrics, and thus, modern 360° video clients are ready for high-quality high-requirement applications.

4.7 Cross Validation

To confirm that the results collected by `Zeus` can be generalized to commercial live 360° video streaming systems and provide insight for system optimization, we conduct a cross validation by comparing `Zeus` to a system using commercial products. As it is infeasible to break down the task-level time consumption of commercial products, we treat them as black boxes and compare the component-level time consumption.

Experiment setup. The commercial system uses a Ricoh Theta V as the camera which has an Adreno 506 GPU for 360° video processing. An open-source plug-in (Ric 2019) is installed in the camera so that it can communicate through WiFi with the server which is the YouTube live streaming service. For the commercial client, we use an embedded YouTube client via `IFrame APIs` and install it on the same laptop used in `Zeus`.

Although dissecting the time consumption of each task is infeasible on commercial products, we can utilize high-level APIs provided by these products to measure the time consumption on each component. We calculate the time consumption of a frame in the Ricoh Theta V by recording the timestamps when it begins to generate a 360° video frame (via `scheduleStreaming.setSchedule()`) and when the frame transmission starts (via `rtmpExtend.startSteam()`). For the YouTube server, we monitor its status change via the webpage interface used to configure the camera URL. We measure the time spent on the server through the timestamps when a packet is received in the YouTube server and when the server starts streaming. The time consumption on the YouTube client can be measured by monitoring the buffering status `YT.PlayerState`. To calculate the frame transmission time on the CST and SCT components, we record the timestamps when the first packet is

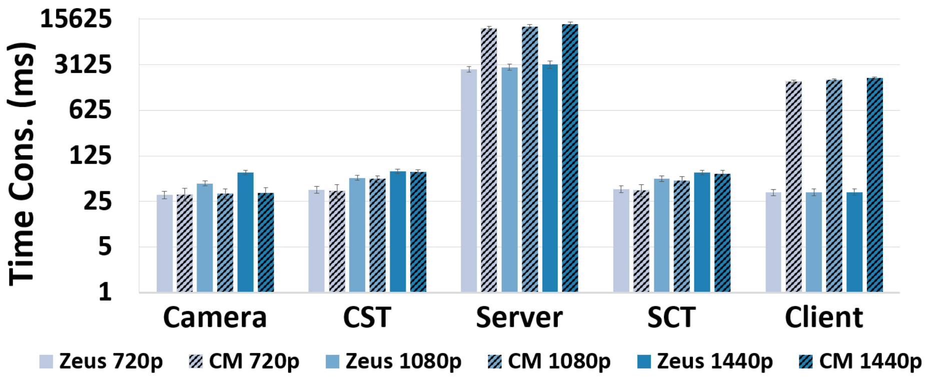


Figure 4.19 Comparison of component time between Zeus and a commercial system (denoted by CM).

sent and when the receiver stops receiving data and then divide this time interval by the total number of frames transmitted.

Cross-validation results. Figure 4.19 shows the comparison of the time consumption across the five system components of the two systems. We observe that the distribution of the time consumption across system components on Zeus is similar to that on the commercial system. Specifically, the time consumption in the camera, camera-server transmission, and server-client transmission is almost the same, and the server in both systems consumes significant time. We quantify the similarity between the two systems by calculating the Pearson Correlation Coefficient (PCC) (pcc 2020), the Distance Correlation (DC) (dc 2020), and the Cosine Similarity (CS) (cs 2020) of the time distribution across the five components. In addition to the default static camera scenario, we further compare the moving camera scenario, where a person holds the camera rig and walks around while live streaming 360° videos.

The results in Table 4.1 show the correlation between the two systems under static and moving scenarios. The PCC and DC values are larger than 0.98 in both scenarios, indicating the distribution of time across the five components in the two systems has a strong positive

Table 4.1 Correlation of time consumption across five components between `Zeus` and the commercial system.

Motion	Resolution	PCC	DC	CS
Static	720p	0.989045	0.993842	0.990239
	1080p	0.987980	0.994173	0.990135
	1440p	0.987269	0.994539	0.990206
Moving	720p	0.990334	0.994896	0.992691
	1080p	0.990994	0.995165	0.992799
	1440p	0.992019	0.995811	0.993636

correlation. The high CS value further implies that the 5-element vectors of component time for both systems point to roughly the same direction, indicating that the most time-consuming component of the two systems is the same (the server).

The strong correlation and similarity of the component-level measurement results with the commercial live 360° video streaming system indicate that our results with `Zeus` are representative of commercial live 360° video streaming systems. Our insights can thus be generalized to minimize the negative effects on user experience caused by different delay metrics in such systems.

We also observe that the YouTube server consumes more time because it handles a larger number of clients than the `Zeus` server. In addition, it uses DASH that chunks and transcodes a video into multiple versions and creates an MPD file, which also contributes to the latency. The longer time at the YouTube client is attributed to its larger player buffer ($\sim$ 1500 ms) compared to `Zeus` ($\sim$ 40 ms).

4.8 More Usage Scenarios

We have so far focused on the measurement result obtained from the default scenario defined in section 4.6.1. In this section, we present the macro-benchmark results under multiple realistic usage scenarios of the 360° video camera sensing system. We aim to compare the measurement results in order to uncover the impacts of the time consumption on delay metrics under different scenarios. The requested video quality for camera sensing is set to be 720p with a bitrate of 2 Mbps. In addition to the Default scenario, we evaluate the following additional usage scenarios of 360° video camera sensing.

- Simple: The experiment configuration of this scenario is based on the Default scenario. The only difference is that all the lights in the lab are turned off. The captured content is almost dark except for a laptop screen showing a timer that is used to remind us of the experiment time.

- Lab Moving: Building on top of the Default scenario, the Lab Moving scenario involves a moving camera. A person holds the camera rig and walks around the lab while the camera captures content.

- Street Still: Instead of fixating the camera in the lab in the Default scenario, Street Still places the camera in a fixed position on the sidewalk of a street in a city center. Other settings remain the same as in the Default scenario.

- Street Moving: Based on the Street Still, we configure the Street Moving scenario by having a person holding the camera rig while walking on the street sidewalk.

Figure 4.20 to Figure4.23 show the time breakdown of the camera component under different scenarios. We observe that motion has an impact on the time consumption of the camera. Both the Lab Moving and Street Moving scenarios consume more time on the camera than their counterparts in the still scenarios (Default and Street Still) respectively. We notice that the increased time stems mainly from the encoding. For example, it takes 8.12

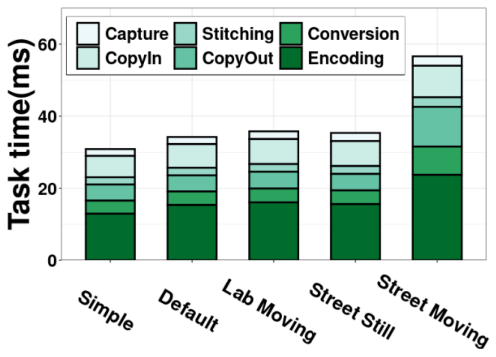


Figure 4.20 Camera Tasks.

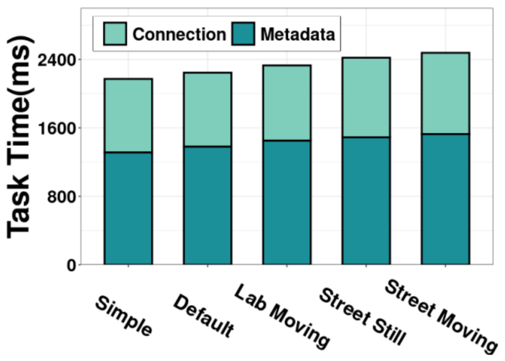


Figure 4.21 Server Tasks.

ms more time for encoding in Street Moving than in Street Still. The reason is that camera motion increases the complexity and diversity of the captured video content, making it more difficult for the encoder to find the redundancy between frames for encoding. It can also be seen that the time increase from Default to Lab Moving is smaller than that from Street Still to Street Moving. This is because the relatively simple background of the lab environment does not introduce much variance to the video content. The time breakdown of other camera tasks in different scenarios presents a similar pattern to the Default scenario. Regarding the impacts on the delay metrics, we find that our insight obtained from the Default scenario still holds in all scenarios. The only difference comes from the camera frame rate. Specifically, the recommended 24–30 fps frame rate can be achieved in all scenarios except for Street Moving. This identifies a possible bottleneck of the moving camera in the wild, which opens a new door for future research in optimizing the computing workflow of 360° camera, especially the encoding in moving scenarios.

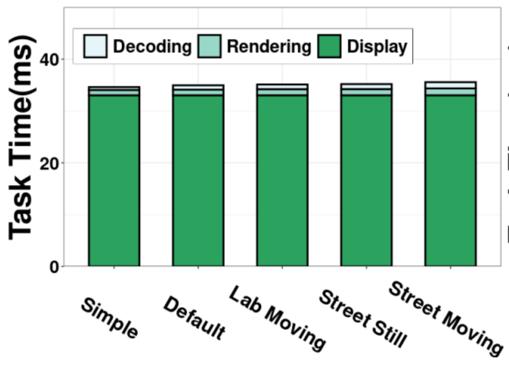


Figure 4.22 Client Tasks.

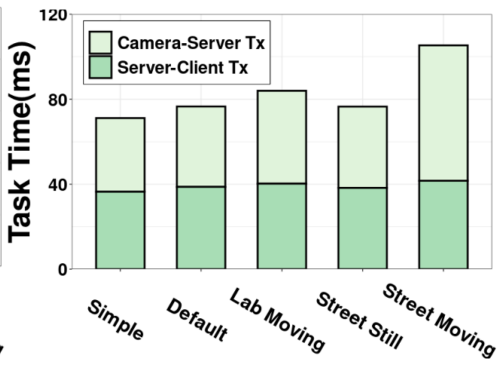


Figure 4.23 Transmission Tasks.

4.9 Conclusion

In this chapter, we conduct the first in-depth analysis of delay across the system pipeline in live 360° video streaming. We have identified the subtle relationship between three important delay metrics and the time consumption breakdown across the system pipeline. We have built the `Zeus` prototype to measure this relationship and study the impacts of different factors on the task-level time consumption. We further validate that our measurement results are representative of commercial live 360° video streaming systems.

Our observations provide vital insights in today's live 360° video streaming systems. First, the bottleneck of achieving a higher frame rate is the 360° camera. While there is little space for improving the stitching, optimizing the encoding and CPU-GPU transfer may elevate the achievable frame rate to the next level. Second, the most critical component to satisfy the requirement of start-up delay and event-to-eye delay is the server. Workflow optimization and server management can be utilized to mitigate the negative effects. In light of these insights, future work can be focused on algorithm design in the camera to improve frame rate and in the video server to shorten the delays as well as to support multiple clients.

CHAPTER 5

A Motion-Based Trajectory Transmission Method for 360° Video Streaming

5.1 Motivation

360° video streaming has become more and more popular in recent years. It has been applied to a diverse set of applications such as entertainment, smart home and art. 360° video market and virtual reality (VR) are expected to reach multi-million by 2026 with an impressive increase compound annual increase rate (web 2018).Due to the development of the 360° cameras (cam 2022), it is convenient for users to create 360° content and upload to commercial meida platforms. Meanwhile, 360° videos provide enriched user experience due to the ultra-high video resolution and the arbitrary viewing directions. Users can switch to different viewing directions by dragging the mouse of a desktop, swiping the screen of a smartphone or moving the head around with a head-mounted display (HMD).

Despite the better user experience, 360° video streaming is still challenging to achieve. First, 360° video streaming requires a very large bandwidth consumption since the videos records all angles of viewing direction which generate more video data to be delivered. Second, the huge amount of generated video data increases the working load of different processes like encoding, decoding and rendering which can lead to a large initial delay.

Considering these challenges, a lot of studies have been focused on 360° video streaming algorithms. For example, existing work on viewport prediction algorithms (Qiao et al. 2020; Li et al. 2022; Qian et al. 2018b, 2016) focused on predict a short-term or long-term viewport for users. Other research efforts towards saliency map prediction (Chao et al. 2018; Nguyen et al. 2018b; Zhu et al. 2019; Zhang et al. 2018). These works aim to improve the viewport prediction accuracy meanwhile focus on the region of interest. However, all these works requires the hardware has a high computation ability.

In this chapter, we present a novel trajectory-based approach for the 360° video streaming. To achieve this goal, we first analyze the trajectories of the moving targets in the 360° videos. For a given video, we utilize optical flow algorithms and Gaussian mixture model to pinpoint the trajectories. Then we choose the trajectories to be delivered based on the size of the moving targets. To help users find the interested content of the 360 videos, we also visualize the current viewing angle and a default viewing trajectory as a small playback window on top the default playback screen. Our contributions can be summarized in two perspectives. First, we proposed an light-weight trajectory-based viewport prediction methods. Our method does not require heavy computation. Thus, it can be applied to more hardware devices. Second, our algorithms reduced the bandwidth consumption of 360° video streaming.

5.2 Related Work

Viewport prediction algorithms based on users' behavior. There are many studies proposed viewport algorithms(Bao et al. 2016; Qian et al. 2018b; Corbillon et al. 2017b; Akcay et al. 2021; Yaqoob & Muntean 2021) based on users head movement. Bao et al(Bao et al. 2016) collected motion data for 153 subjects and designed a regression models to predict the viewports. Their method can achieve a low failure ration when the prediction window is short. Qian et al(Qian et al. 2018b) collected head movement traces from 130 diverse users and proposed a tile-based viewport adaptive algorithm which can allocate higher bitrate to the users' interested area. The results show that their method can guarantee the users experience while save up to 35% bandwidth. Corbillon et al (Corbillon et al. 2017b) utilized the head movement data to predict the viewport. Their method downloads the video version with the high quality region closer to where the user will watch. Their method allows an increase of 102% of the displayed quality for the same bandwidth budget

Viewport prediction algorithms based on saliency map. Many research works focus on utilizing the saliency map to predict the viewport(Qiao et al. 2020; Zhu et al. 2019; Li et al. 2022; Nguyen et al. 2018b). Qiao et al (Qiao et al. 2020) studied more than 200 videos and designed a Multi-Task Deep Neural Network(MT-DNN) for view port saliency prediction. Their method improves the performance compared with state-of-the-art methods such as SalGAN and DeepVS. Li et al(Li et al. 2022) utilized convolutional neural network to design a field of view (FoV) prediction method. The method extracts the saliency features for FoV prediction. The results show that their method is better than the other prediction methods. Anh et al (Nguyen et al. 2018b) also proposed an saliency map prediction algorithm for viewport prediction. They use deep neural network to train a self-built 360° video dataset and the result show that they can accurate predict at 0.5-1.0s. However, all these methods require a high computation.

5.3 METHOD

5.3.1 Testbed

We build a testbed to run our method. For the video server, we run a Nginx-1.16.1 server to support 360° video streaming. The viewing player lies in a desktop. We design an HTML5 based video client using FLV.js, a flash based module written in JavaScript. Three.js is used to fetch a video frame from Flv.js and project it onto the sphere format using render(). The sphere video frame is stored at the HTML5 element ¡canvas¿, which will be displayed on

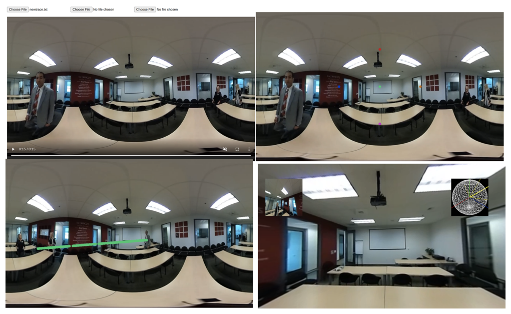


Figure 5.1 Testbed 360° player.

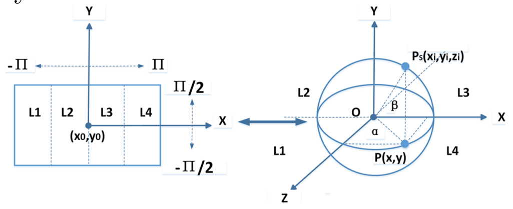


Figure 5.2 360° video frame format conversion between sphere format and equirectangular format.

webpages. desktop, the CPU is an Intel Xeon W-2135 with 12 cores at 3.7 Ghz, and the graphic card is a GeForce GTX 1080Ti.

5.3.2 Conversion between 3D and 2D

Pinpoint the position of the viewport on the original 360° video is not easy. 360° videos are saved as the equirectangular format. However, users can only watch part of the video in the sphere format from the video player built by Three.js. Since Three.js has a different coordinate system compared with the original 360 videos, we cannot directly pinpoint the position of the viewport on the original 360° video frame from the video player. Figure 5.2 shows the conversion between the sphere format and and equirectangular format. As we can see from the figure, the 360° video frame can be divided into four parts considering the (x_i, z_i) position.

$$L1 : x_i \leq 0, z_i > 0; \tag{5.1}$$

$$L2 : x_i \leq 0, z_i \leq 0; \tag{5.2}$$

$$L3 : x_i > 0, z_i \leq 0; \tag{5.3}$$

$$L4 : x_i > 0, z_i > 0; \tag{5.4}$$

Assume the position of a random pixel P_s is (x_i, y_i, z_i) in the sphere format and the corresponding position in the equirectangular format would be (x,y). Thus, we have

$$x = \alpha/\pi * W_{canvas}/2 + W_{canvas}/2 \tag{5.5}$$

$$y = 2 * \beta/\pi * H_{canvas}/2 + H_{canvas}/2 \tag{5.6}$$

where α is the angle between the z axis and the line OP. β is the angle between the line OP and the line OP_s.W_{canvas} and H_{canvas} are the width and the height of the video player, respectively. For the α angle, it can be calculated by the formulas below:

$$\alpha = -\pi - \arctan(x_i/z_i)(P_s \in L1) \tag{5.7}$$

$$\alpha = -\arctan(x_i/z_i)(P_s \in L2) \tag{5.8}$$

$$\alpha = -\arctan(x_i/z_i)(P_s \in L3) \tag{5.9}$$

$$\alpha = \pi - \arctan(x_i/z_i)(P_s \in L4) \tag{5.10}$$

The β angle can be calculted by the formula below:

$$\beta = \arctan(-y_i/\sqrt{{x_i}^2 + {z_i}^2}) \tag{5.11}$$

After we get the trajectory from the 360° equirectangular videos, we also need to convert the coordinate into Three.js sphere coordinate system. Below we show the formulas to do the conversion.

$$\alpha = (x - W_{canvas}/2)/(W_{canvas}/2) * \pi \tag{5.12}$$

$$\beta = (y - H_{canvas}/2)/(H_{canvas}/2) * \pi/2 \tag{5.13}$$

Thus, the corresponding camera position $(x_c, y_c z_c)$ can be calculated based on the formula

For $0 < \alpha \leq \pi/2$

$$x_c = l_s * \sin\alpha/(-r) \tag{5.14}$$

$$z_c = -l_s * \cos(\alpha)/(-r) \tag{5.15}$$

For $\pi/2 < \alpha \leq \pi$

$$x_c = l_s * \sin(\pi - \alpha)/(-r) \tag{5.16}$$

$$z_c = -l_s * \cos(\pi - \alpha)/(-r) \tag{5.17}$$

For $-\pi/2 < \alpha \leq 0$

$$x_c = -l_s * \sin(-\alpha)/(-r) \tag{5.18}$$

$$z_c = -l_s * \cos(-\alpha)/(-r) \tag{5.19}$$

For $-\pi < \alpha \leq -\pi/2$

$$x_c = -l_s * \sin(\pi + \alpha)/(-r) \tag{5.20}$$

$$z_c = l_s * \cos(\pi + \alpha)/(-r) \tag{5.21}$$

For the y_c

$$y_c = l_t * /(-r) \tag{5.22}$$

where l_s and l_t can be calculated using formula 23 and 24. r is the radius of the rendered sphere which can be configured in Three.js.

$$l_s = r * \cos(\beta) \tag{5.23}$$

$$l_s = r * \sin(\beta) \tag{5.24}$$

5.3.3 Pinpoint the Trajectory

To track the trajectory from the 360° videos, we use gaussian mixture model (GMM) and optical flow algorithms. Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters (gau 2022). Compared with those complex AI algorithms, GMM can be applied for systems that do not have powerful computation ability. To achieve the goal, we first need to detect the motion target by extracting the moving foreground from the background of the video. Second, we need to filter some motion targets since they are noise detected by GMM. After that we need to extract the features of the trajectory. We calculate the average size of the moving target on the trajectory as the feature. It can be calculated as:

$$F_t = \frac{1}{k} \sum_{n=1}^{k} f_s \tag{5.25}$$

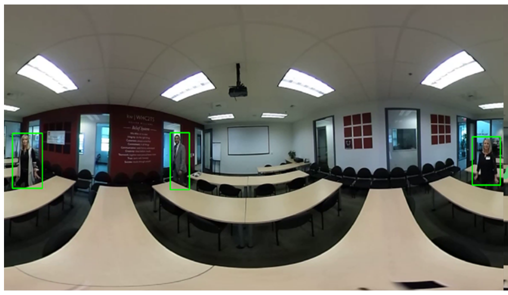


Figure 5.3 Moving target detection.

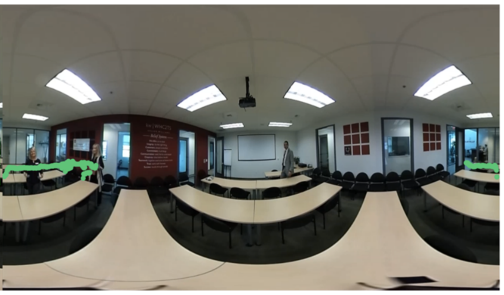


Figure 5.4 Woman in the mid.

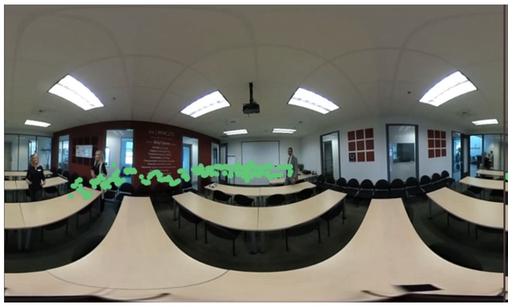


Figure 5.5 Man in the mid.

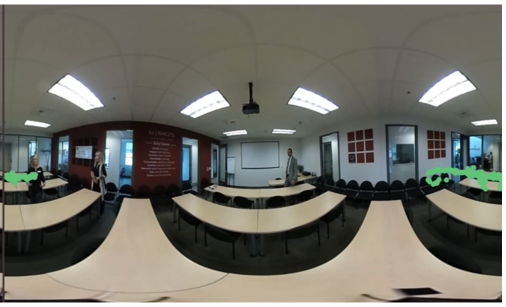


Figure 5.6 Woman in the left.

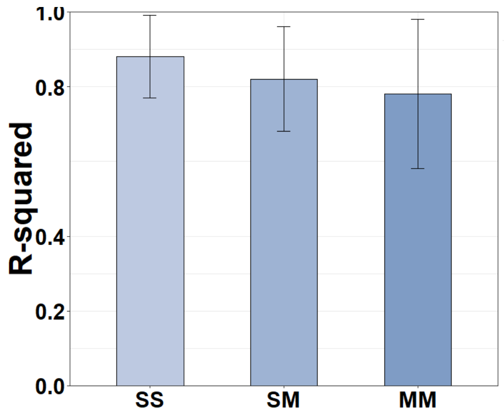
where F_t is the average size of each trajectory. k is the total number of the contours for each moving target. f_s is the size of each contour on the trajectory. Figure 5.3 shows the contours of the moving targets. Figure 5.4 to Figure 5.6 shows the trajectories of the moving targets in the video.

5.4 Experiment Results

5.4.1 Experiment Setup

To verify our method, we use a public dataset(Dharmasiri et al. 2021). It is an aggression of six different published datasets which include 88 different 360° videos. Considering the moving direction of the targets and the total amount in the 360° videos, we define three different types of video.

Single-Single video: there is one moving target in the 360° video moving toward one



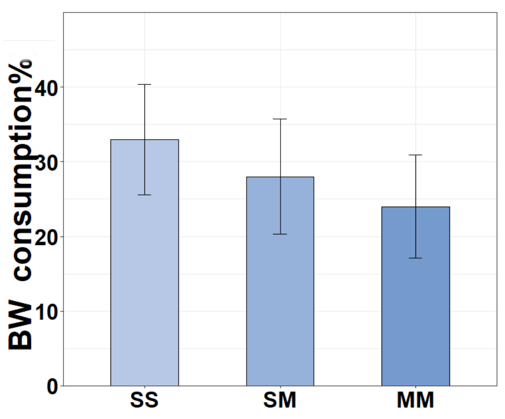


Figure 5.7 R-squared values for three types of videos.

Figure 5.8 Bandwidth Consumption for three types of videos.

single direction.

Single-Multiple video: there is one moving target in the 360° video moving toward random directions.

Multiple-Multiple video: there are multiple moving targets in the 360° video moving toward random directions.

5.4.2 R-squared values

After pinpoint the trajectory, we use r-squared values to verify if our trajectory can present the real one after filtering the noise. Figure 5.7 shows the r-squared values for three different types of videos. We can see that the average R-squared values is around 0.8 which indicates a good fitness. The reason is because that the targets are moving slowly and in a relatively simple trajectory. We also observe that it has a lower R-squared values when there are multiple moving trajectories. The reason is that some trajectories are in random directions. Thus, it is difficult to find a line or curve to fit the trajectory.

5.4.3 Bandwidth Consumption

To check the effectiveness of our method, we further verify the bandwidth consumption for different types of videos. Figure 5.8 presents the results. We observe that our method can obviously reduce the bandwidth consumption by approximately 75%. We also observe that it would save more bandwidth for Multiple-Multiple videos. the reason is because that it has more complex video content in that type of video. Thus, the viewport account for a relatively small percent of the whole video.

CHAPTER 6
Conclusion

In this dissertation, we first present the measurement study on 360° live video streaming systems.Then we proposed a motion-based trajectory transmission method for 360° live video streaming. We perform the first in-depth study of delay across the computing tasks in 360° video camera sensing in this paper. We have characterized the three important delay metrics by the time consumption breakdown across the tasks. A prototype Zeus has been developed to measure this relationship and examine the task-level time consumption in various usage scenarios. We finally compare Zeus with commercial systems and validate that our measurement results are representative. Our findings provide critical insight for improving 360° video camera sensing. First, the bottleneck of a higher frame rate is the 360° camera. Although the space for optimizing the stitching is limited, enhancing the encoding and CPU-GPU transfer may elevate the frame rate to the next level. Second, the server plays a decisive role in meeting the requirement of start-up delay and event-to-eye delay. The existing sever workflow can be optimized to reduce the server time. In light of these observations, future work should focus on algorithm design in the camera to improve frame rate as well as in the server to shorten the delays and to support multiple clients. After optimizing Zeus in terms of camera frame rate and server delay, it is important to compare Zeus with emerging commercial systems that support 360° video live streaming through advanced cameras, such as Ricoh Theta Z1. A full scale research including extensive experiments are needed to confirm the applicability of the findings in this paper in the context of challenging 4 K 360° video systems. Moreover, since user experience is also determined by the quality of the received 360° videos, it is critical to understand how the delayed and missed packet delivery would affect the distortion and quality of the video frames viewed by the users.

REFERENCES

2013, Second-order intercept point, https://en.wikipedia.org/wiki/Second-order_intercept_point

2016, Businesswire, https://www.businesswire.com/news/home/20160512005924/en/Video-Streaming-Market-Worth-USD-70.05-Billion-by-2021—Key-Players-are-Netflix-Hulu-Amazon—Research-and-Markets

2018, Nginx-http-flv-module, https://github.com/winshining/nginx-http-flv-module

2018, PLUG-IN STORE, https://pluginstore.theta360.com/

2018, Stats You Need To Know About Live-Streaming Video In 2018, https://www.talkpoint.com/stats-you-need-to-know-about-live-streaming-video-in-2018/

2019, 47 Must-Know Live Video Streaming Statistics, https://livestream.com/blog/62-must-know-stats-live-video-streaming

2019, Virtual reality and 360-Degree are the future of live sports video streaming, https://www.bandt.com.au/virtual-reality-360-degree-future-live-sports-video-streaming/

2019, VRWorks - 360 Video, https://developer.nvidia.com/vrworks/vrworks-360video

2019, Wireless Live Streaming, https://pluginstore.theta360.com/

2020, Cosine Similarity, https://en.wikipedia.org/wiki/Cosine_similarity

2020, Distance correlation, https://en.wikipedia.org/wiki/Distance_correlation

2020, Facebook 360 Video, https://facebook360.fb.com/live360/

2020, GoPro Hero6, https://www.aircraftspruce.com/catalog/avpages/goprohero6.php?utm_source=google&utm_medium=organic&utm_campaign=shopping&utm_term=

11-15473

2020, Pearson correlation coefficient, https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

2020, Ricoh Theta S, https://theta360.com/en/about/theta/s.html

2020, The Video Problem: 3 Reasons Why Users Leave a Website with Badly Implemented Video, https://bitmovin.com/video-problem-3-reasons-users-leave-website-badly-implemented-video/

2020, YouTube, https://www.youtube.com/

2021, 360-degree video, https://en.wikipedia.org/wiki/360-degree_video

2021, CISION, https://www.prnewswire.com/news-releases

2021, DASH, https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP

2021, Theta, https://theta360.com/en/about/theta/v.html

2022, 360 Degree Camera Market Size Worth Around 4.64 BN by 2030, https://www.globenewswire.com/news-release/2022/03/10/2401118/0/en/360-Degree-Camera-Market-Size-Worth-Around-US-4-64-BN-by-2030.html

2022, Gaussian mixture models, https://scikit-learn.org/stable/modules/mixture.html

Afzal, S., Chen, J., & Ramakrishnan, K. 2017a, in Proceedings of the Workshop on Virtual Reality and Augmented Reality Network, 1–6

Afzal, S., Chen, J., & Ramakrishnan, K. K. 2017b, in Proceedings of the Workshop on Virtual Reality and Augmented Reality Network (VR/AR Network ’17), 1–6

Akcay, M. N., Kara, B., Ahsan, S., Begen, A. C., Curcio, I., & Aksu, E. 2021, in ACM Multimedia Asia, 1–5

Bao, Y., Wu, H., Zhang, T., Ramli, A. A., & Liu, X. 2016, in 2016 IEEE International Conference on Big Data (Big Data), IEEE, 1161–1170

Chao, F.-Y., Zhang, L., Hamidouche, W., & Deforges, O. 2018, in 2018 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), IEEE, 01–04

Chen, B., Yan, Z., Jin, H., & Nahrstedt, K. 2019, in Proceedings of the 10th ACM Multimedia Systems Conference, ACM, 1–12

Cheng, H.-T., Chao, C.-H., Dong, J.-D., Wen, H.-K., Liu, T.-L., & Sun, M. 2018, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1420–1429

Corbillon, X., De Simone, F., Simon, G., & Frossard, P. 2018, in Proceedings of the 9th ACM Multimedia Systems Conference, ACM, 237–249

Corbillon, X., Devlic, A., Simon, G., & Chakareski, J. 2017a, in Proceedings of the 25th ACM international conference on Multimedia, ACM, 943–951

Corbillon, X., Simon, G., Devlic, A., & Chakareski, J. 2017b, in 2017 IEEE international conference on communications (ICC), IEEE, 1–7

de la Fuente, Y. S., Bhullar, G. S., Skupin, R., Hellge, C., & Schierl, T. 2019, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 9, 18

Dharmasiri, A., Kattadige, C., Zhang, V., & Thilakarathna, K. 2021, in Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, 114–121

Ding, Y., Du, Y., Hu, Y., Liu, Z., Wang, L., Ross, K., & Ghose, A. 2011, in Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, 361–370

Dobrian, F., Sekar, V., Stoica, I., & h. Zhang. 2011, in Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11), 362–373

Duanmu, F., He, Y., Xiu, X., Hanhart, P., Ye, Y., & Wang, Y. 2018, in 2018 Data Compression Conference, IEEE, 404–404

Duanmu, F., Kurdoglu, E., Hosseini, S. A., Liu, Y., & Wang, Y. 2017, in Proceedings of the Workshop on Virtual Reality and Augmented Reality Network, 13–18

Durak, K., Akcay, M. N., Erinc, Y. K., Pekel, B., & Begen, A. C. 2020, in 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP), IEEE, 1–6

Gadaleta, M., Chiariotti, F., Rossi, M., & Zanella, A. 2017, IEEE Transactions on Cognitive Communications and Networking, 3, 703

Grzelka, A., Dziembowski, A., Mieloch, D., Stankiewicz, O., Stankowski, J., & Domański, M. 2019, in 2019 Picture Coding Symposium (PCS), IEEE, 1–5

Hamilton, W. A., Garretson, O., & Kerne, A. 2014, in Proceedings of the SIGCHI conference on human factors in computing systems, 1315–1324

Jiang, N., Swaminathan, V., & Wei, S. 2017, in Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '17), 55–60

Lederer, S., Müller, C., & Timmerer, C. 2012, in Proceedings of the 3rd multimedia systems conference, 89–94

Li, J., Han, L., Zhang, C., Li, Q., & Liu, Z. 2022, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)

Liu, X., Han, B., Qian, F., & Varvello, M. 2019, in Proceedings of the 10th ACM Multimedia Systems Conference, ACM, 154–164

Lo, W.-C., Fan, C.-L., Yen, S.-C., & Hsu, C.-H. 2017, in 2017 Asia-Pacific Network Operations and Management Symposium (APNOMS), 205–210

Nasrabadi, A. T., Mahzari, A., Beshay, J. D., & Prakash, R. 2017, in Proceedings of the

25th ACM international conference on Multimedia, ACM, 1689–1697

Nguyen, A., Yan, Z., & Nahrstedt, K. 2018a, in 2018 ACM Multimedia Conference on Multimedia Conference (MM '18), 1190–1198

Nguyen, A., Yan, Z., & Nahrstedt, K. 2018b, in 2018 ACM Multimedia Conference on Multimedia Conference, ACM, 1190–1198

Nihei, K., Yoshida, H., Kai, N., Satoda, K., & Chono, K. 2018, in 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, 1–7

Oda, C. 2018, THETA V Client Mode Configuration Guide, https://community.theta360.guide/t/theta-v-client-mode-configuration-guide/2565

Pan, W., Cheng, G., Wu, H., & Tang, Y. 2016, in 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), IEEE, 1–6

Qian, F., Han, B., Xiao, Q., & Gopalakrishnan, V. 2018a, in Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom '18), 99–114

Qian, F., Han, B., Xiao, Q., & Gopalakrishnan, V. 2018b, in Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, 99–114

Qian, F., Ji, L., Han, B., & Gopalakrishnan, V. 2016, in Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges, 1–6

Qiao, M., Xu, M., Wang, Z., & Borji, A. 2020, IEEE Transactions on Multimedia, 23, 748

Sauer, J., Wien, M., Schneider, J., & Bläser, M. 2018, in 2018 Picture Coding Symposium (PCS), IEEE, 66–70

Siekkinen, M., Kämäräinen, T., Favario, L., & Masala, E. 2018, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 14, 1

Silva, R. M., Feijó, B., Gomes, P. B., Frensh, T., & Monteiro, D. 2016, in ACM SIGGRAPH

2016 Posters, 1–2

Sodagar, I. 2011, IEEE multimedia, 18, 62

Su, Y.-C., & Grauman, K. 2017, Advances in Neural Information Processing Systems, 30, 529

Sun, K., Zhang, H., Gao, Y., & Wu, D. 2019, Journal of Communications and Networks, 21, 339

Szigeti, T., & Hattingh, C. 2004, Cisco, San Jose, CA, Dec, 1

Tang, J. C., Venolia, G., & Inkpen, K. M. 2016a, in Proceedings of the 2016 CHI conference on human factors in computing systems, 4770–4780

Tang, J. C., Venolia, G., & Inkpen, K. M. 2016b, in Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, ACM, 4770–4780

Wang, B., Zhang, X., Wang, G., Zheng, H., & Zhao, B. Y. 2016, in Proceedings of the 2016 Internet Measurement Conference, 485–498

Wang, F.-E., Yeh, Y.-H., Sun, M., Chiu, W.-C., & Tsai, Y.-H. 2020, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 462–471

Wang, Y., Li, Y., Yang, D., & Chen, Z. 2017, in 2017 IEEE Visual Communications and Image Processing (VCIP), IEEE, 1–4

Xiao, M., Zhou, C., Liu, Y., & Chen, S. 2017, in Proceedings of the ACM International Conference on Multimedia (MM ’18), 708–716

Xiao, X. 2008, Technical, commercial and regulatory challenges of QoS: An internet service model perspective (Morgan Kaufmann)

Xie, L., Xu, Z., Ban, Y., Zhang, X., & Guo, Z. 2017, in Proceedings of the ACM on Multimedia Conference (MM ’17), 315–323

Yan, Z., Song, C., Lin, F., & Xu, W. 2018, in Proceedings of the International Workshop on Mobile Computing Systems & Applications (HotMobile '18), 13–18

Yaqoob, A., & Muntean, G.-M. 2021, IEEE Transactions on Broadcasting, 67, 746

Yi, J., Luo, S., & Yan, Z. 2019, in Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, ACM, 49–54

Yu, C., Xu, Y., Liu, B., & Liu, Y. 2014, in IEEE INFOCOM 2014-IEEE Conference on Computer Communications, IEEE, 1456–1464

Zhang, Z. 2000, IEEE Transactions on pattern analysis and machine intelligence, 22

Zhang, Z., Xu, Y., Yu, J., & Gao, S. 2018, in Proceedings of the European conference on computer vision (ECCV), 488–503

Zhao, M., Gong, X., Liang, J., Wang, W., Que, X., & Cheng, S. 2014, IEEE Transactions on Circuits and Systems for Video Technology, 25, 451

Zhou, C., Li, Z., & Liu, Y. 2017a, in Proceedings of the ACM on Multimedia Systems Conference (MMSys '17) (ACM)

Zhou, C., Li, Z., & Liu, Y. 2017b, in Proceedings of the 8th ACM on Multimedia Systems Conference, ACM, 27–37

Zhou, Y., Tian, L., Zhu, C., Jin, X., & Sun, Y. 2019, IEEE Journal of Selected Topics in Signal Processing, 14, 118

Zhu, G., & Song, H. 2015, Telecommunications Science, 31, 22

Zhu, Y., Zhai, G., Min, X., & Zhou, J. 2019, IEEE Transactions on Multimedia, 22, 2331