An Analysis of Delay in Live 360° Video Streaming Systems

Jun Yi[1], Md Reazul Islam[1], Shivang Aggarwal[2]
Dimitrios Koutsonikolas[2], Y. Charlie Hu[3], Zhisheng Yan[1]
[1] Georgia State University,[2] University at Buffalo, SUNY, [3] Purdue University

ABSTRACT

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. Although prior works have studied commercial live 360° video streaming systems, none of them has dug into the end-to-end pipeline and explored how the task-level time consumption affects the user experience. In this paper, we conduct the first in-depth measurement study of task-level time consumption for five system components in live 360° video streaming. We first identify the subtle relationship between the time consumption breakdown across the system pipeline and the three delay metrics. We then build a prototype Zeus to measure this relationship. Our findings indicate the importance of CPU-GPU transfer at the camera and the server initialization as well as the negligible effect of 360° video stitching on the delay metrics. We finally validate that our results are representative of real world systems by comparing them with those obtained with a commercial system.

CCS CONCEPTS

• **Information systems → Multimedia information systems**.

KEYWORDS

Live 360° video streaming; prototype design; measurement study

ACM Reference Format:
Jun Yi, Md Reazul Islam, Shivang Aggarwal, Dimitrios Koutsonikolas, Y. Charlie Hu, Zhisheng Yan. 2020. An Analysis of Delay in Live 360° Video Streaming Systems. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394171.3413539

1 INTRODUCTION

Live video streaming services have been prevalent in recent years [3]. 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 [4].

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 [26, 30, 31], the delay of live 360° video streaming has not been well understood. Recent works in 360° video streaming focused on rate adaptation algorithms [15–17, 24] and encoding/projection methods [18, 23, 35]. The only two existing measurement studies on live 360° videos [22, 33] 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

MM '20, October 12–16, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3413539

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.

In summary, our contributions can be summarized as follows.

- We identify the diverse relationship between the time consumption breakdown across the system pipeline and the three delay metrics in live 360° video streaming (Section 4).
- 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 (Section 5).
- 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 (Section 6).
- 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 (Section 7).

2 RELATED WORK

Regular live video streaming. Siekkinen et al. [26] studied user experience on mobile live video streaming and observed that video transmission time is highly affected by live streaming protocols. Researchers [25, 28] 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. Zhou et al. [35] 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 [19, 20] 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. Jun et al. [33] 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. [22] 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. [15] 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.

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

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

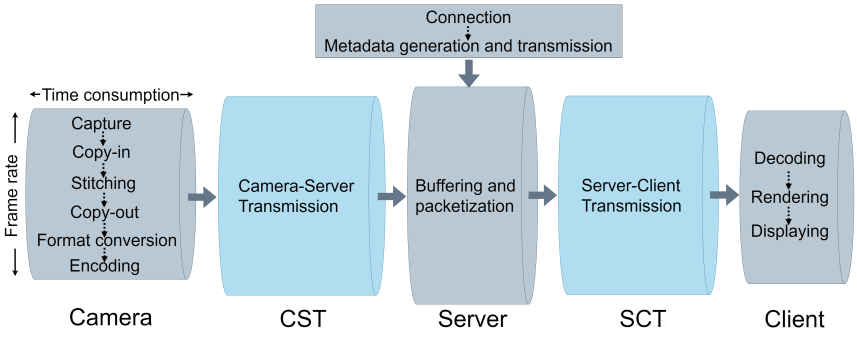

Figure 1: The architecture of live 360° video streaming and the tasks of the 5 system components. The top rectangle shows one-time tasks whereas the 5 bottom pipes show the pipeline tasks that must be passed through for every frame.

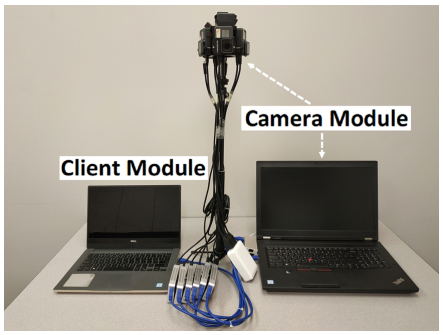

Figure 2: The Zeus prototype.

- *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 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 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 [13]. 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,

$$D_{start} = T_{srv,once} + T_{sct} + T_{clnt} \quad (1)$$

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.,

$$D_{event-to-eye} = T_{cam} + T_{cst} + T_{srv,once} + T_{srv,pipe} + T_{sct} + T_{clnt} \quad (2)$$

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,

$$FR = \min\{FR_{cam}, FR_{cst}, FR_{srv}, FR_{sct}, FR_{clnt}\} \quad (3)$$

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 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.

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 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) [10] 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 [5] 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.calibrate()` and the second-order distortion model [1] to calculate camera distortion parameters [34]. 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* [2].

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.

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.

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.

6.2 360° Camera

6.2.1 Video Capture Task. We vary the resolutions of the captured regular videos and show the video capture time in Figure 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 [9, 14]. While capturing six

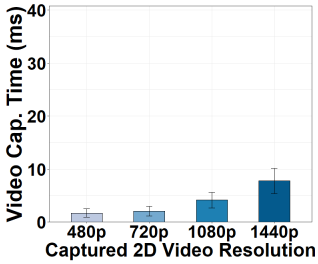

Figure 3: Video capture time versus capture resolutions.

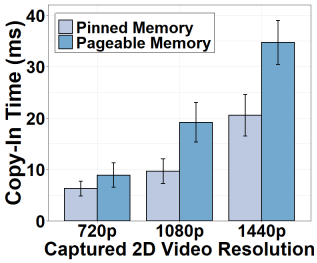

Figure 4: Copy-in time from different memory locations.

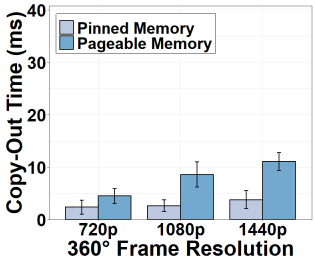

Figure 5: Copy-out time from different memory locations.

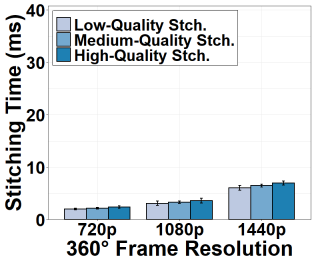

Figure 6: Frame stitching time vs. stitching options.

1080p and 1440p regular frames would consume more time, such high resolutions of input regular videos are typically not required in current live 360° video applications.

6.2.2 Copy-in and Copy-out Tasks. Figures 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.

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 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 [21, 27] 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.

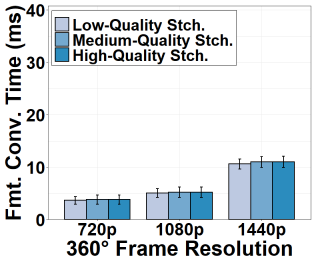

Figure 7: Format conversion time vs. stitching options.

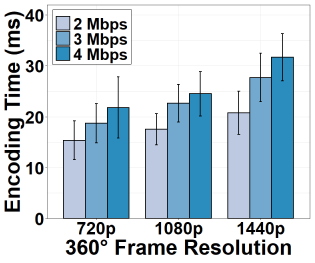

Figure 8: Encoding time under different bitrates.

6.2.4 Format Conversion Task. Figure 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.

6.2.5 Encoding Task. Figure 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.

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 [12]. 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

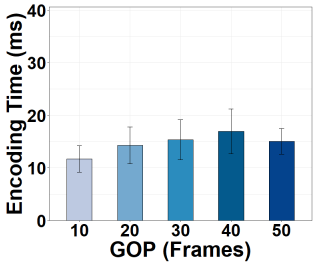

Figure 9: Encoding time of a 720p frame versus GOP.

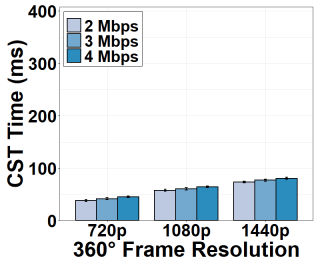

Figure 10: CST time under different bitrates.

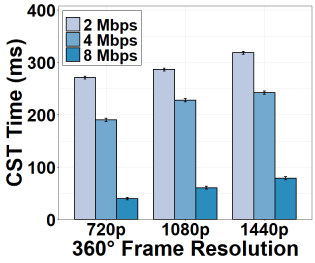

Figure 11: CST time versus upload bandwidth.

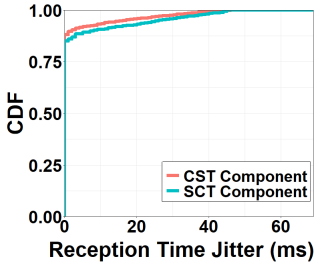

Figure 12: Jitter of packet reception time.

little scope to further improve the stitching task since the current stitching time is already low. Moreover, the parameter space of major tasks, such 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 (2) significantly. The typical event-to-eye delay requirement for interactive applications is no more than 200 ms [29], and it can be further relaxed to 4-5 seconds for live broadcasting of events [32]. We also reiterate that the camera has no effect on the start-up delay as defined in equation (1).

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 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 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 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 10. This confirms that 8 Mbps is sufficient to support the 360° video transmission.

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

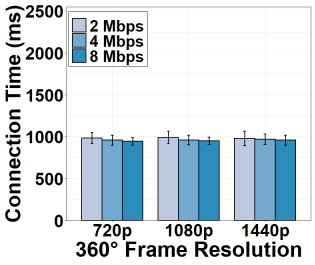

Figure 13: Connection time under different download bandwidths.

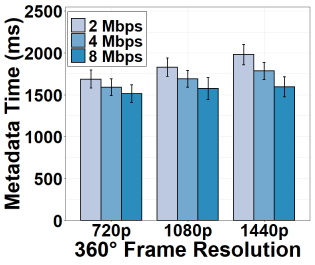

Figure 14: Metadata gen. and tx time under different download bandwidths.

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 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.

6.4 Video Server

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 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 way.

6.4.2 Metadata Generation and Transmission Task. Figure 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

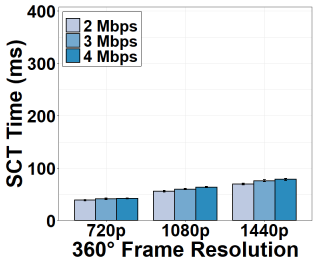

Figure 15: SCT time under different bitrates.

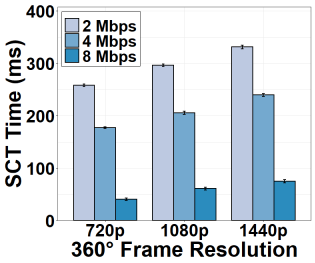

Figure 16: SCT time versus download bandwidth.

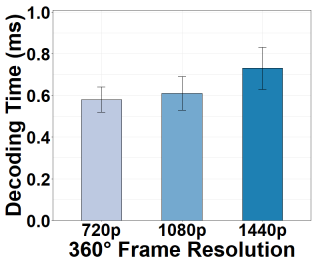

Figure 17: Decoding time versus 360° frame resolutions.

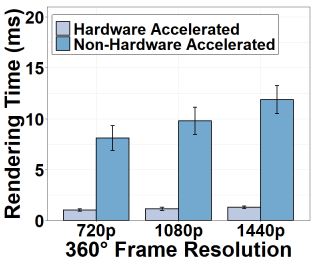

Figure 18: Rendering time of different hardware options.

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.

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.

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 (~ 200 ms [29]) of interactive video applications. To relieve the negative effects of long start-up and event-to-eye delay, researchers should focus on optimizing the workflow and data management in the server to minimize the preparation step during the connection task.

6.5 Server-Client Transmission

Figure 15-16 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.

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 continuous packet reception in Figure 12 may not hold for the SCT component, resulting in a reduced frame rate.

6.6 Client

6.6.1 Decoding Task. Figure 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.

6.6.2 Rendering Task. We show the rendering time under different hardware configurations in Figure 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.

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.

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.

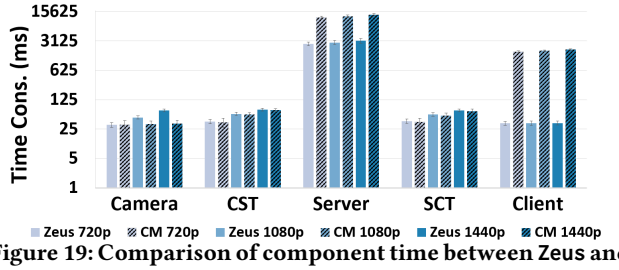

Figure 19: Comparison of component time between Zeus and a commercial system (denoted by CM).

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 [6] 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 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 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) [11], the Distance Correlation (DC) [8], and the Cosine Similarity (CS) [7] 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.

Table 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

The results in Table 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 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).

8 CONCLUSION AND FUTURE WORK

In this paper, 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.

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation Grant OAC–1948467.

REFERENCES

[1] 2013. Second-order intercept point. https://en.wikipedia.org/wiki/Second-order_intercept_point.

[2] 2018. Nginx-http-flv-module. https://github.com/winshining/nginx-http-flv-module.

[3] 2019. 47 Must-Know Live Video Streaming Statistics. https://livestream.com/blog/62-must-know-stats-live-video-streaming.

[4] 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/.

[5] 2019. VRWorks - 360 Video. https://developer.nvidia.com/vrworks/vrworks-360video.

[6] 2019. Wireless Live Streaming. https://pluginstore.theta360.com/.

[7] 2020. Cosine Similarity. https://en.wikipedia.org/wiki/Cosine_similarity.

[8] 2020. Distance correlation. https://en.wikipedia.org/wiki/Distance_correlation.

[9] 2020. Facebook 360 Video. https://facebook360.fb.com/live360/.

[10] 2020. GoPro Hero6. https://www.aircraftspruce.com/catalog/avpages/goprohero6.php?utm_source=google&utm_medium=organic&utm_campaign=shopping&utm_term=11-15473.

[11] 2020. Pearson correlation coefficient. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

[12] 2020. Ricoh Theta S. https://theta360.com/en/about/theta/s.html.

[13] 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/.

[14] 2020. YouTube. https://www.youtube.com/.

[15] Bo Chen, Zhisheng Yan, Haiming Jin, and Klara Nahrstedt. 2019. Event-driven stitching for tile-based live 360 video streaming. In *Proceedings of the 10th ACM Multimedia Systems Conference*. ACM, 1–12.

[16] Xavier Corbillon, Francesca De Simone, Gwendal Simon, and Pascal Frossard. 2018. Dynamic adaptive streaming for multi-viewpoint omnidirectional videos. In *Proceedings of the 9th ACM Multimedia Systems Conference*. ACM, 237–249.

[17] Xavier Corbillon, Alisa Devlic, Gwendal Simon, and Jacob Chakareski. 2017. Optimal set of 360-degree videos for viewport-adaptive streaming. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 943–951.

[18] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE international conference on communications (ICC)*. IEEE, 1–7.

[19] Yago Sanchez de la Fuente, Gurdeep Singh Bhullar, Robert Skupin, Cornelius Hellge, and Thomas Schierl. 2019. Delay impact on MPEG OMAF's tile-based viewport-dependent 360° video streaming. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2019), 18–28.

[20] Adam Grzelka, Adrian Dziembowski, Dawid Mieloch, Olgierd Stankiewicz, Jakub Stankowski, and Marek Domański. 2019. Impact of Video Streaming Delay on User Experience with Head-Mounted Displays. In *2019 Picture Coding Symposium (PCS)*. IEEE, 1–5.

[21] Wei-Tse Lee, Hsin-I Chen, Ming-Shiuan Chen, I-Chao Shen, and Bing-Yu Chen. 2017. High-resolution 360 Video Foveated Stitching for Real-time VR. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 115–123.

[22] Xing Liu, Bo Han, Feng Qian, and Matteo Varvello. 2019. LIME: understanding commercial 360° live video streaming services. In *Proceedings of the 10th ACM Multimedia Systems Conference*. ACM, 154–164.

[23] Afshin Taghavi Nasrabadi, Anahita Mahzari, Joseph D Beshay, and Ravi Prakash. 2017. Adaptive 360-degree video streaming using scalable video coding. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 1689–1697.

[24] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. 2018. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 1190–1198.

[25] Koichi Nihei, Hiroshi Yoshida, Natsuki Kai, Kozo Satoda, and Keiichi Chono. 2018. Adaptive bitrate control of scalable video for live video streaming on best-effort network. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.

[26] Matti Siekkinen, Teemu Kämäräinen, Leonardo Favario, and Enrico Masala. 2018. Can you see what I see? Quality-of-experience measurements of mobile live video broadcasting. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 14, 2s (2018), 1–23.

[27] Rodrigo MA Silva, Bruno Feijó, Pablo B Gomes, Thiago Frensh, and Daniel Monteiro. 2016. Real time 360 video stitching and streaming. In *ACM SIGGRAPH 2016 Posters*. 1–2.

[28] Kairan Sun, Huazi Zhang, Ying Gao, and Dapeng Wu. 2019. Delay-aware fountain codes for video streaming with optimal sampling strategy. *Journal of Communications and Networks* 21, 4 (2019), 339–352.

[29] Tim Szigeti and Christina Hattingh. 2004. Quality of service design overview. *Cisco, San Jose, CA, Dec* (2004), 1–34.

[30] John C Tang, Gina Venolia, and Kori M Inkpen. 2016. Meerkat and periscope: I stream, you stream, apps stream for live streams. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 4770–4780.

[31] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y Zhao. 2016. Anatomy of a personalized livestreaming system. In *Proceedings of the 2016 Internet Measurement Conference*. 485–498.

[32] XiPeng Xiao. 2008. *Technical, commercial and regulatory challenges of QoS: An internet service model perspective*. Morgan Kaufmann.

[33] Jun Yi, Shiqing Luo, and Zhisheng Yan. 2019. A measurement study of YouTube 360° live video streaming. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 49–54.

[34] Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence* 22 (2000).

[35] Chao Zhou, Zhenhua Li, and Yao Liu. 2017. A measurement study of oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 27–37.