Power-Efficient Live Virtual Reality Streaming Using Edge Offloading

Zichen Zhu
Rutgers University
Piscataway, NJ, USA
zichen.zhu@rutgers.edu

Xianglong Feng
Miami University
Oxford, OH, USA
fengx17@miamioh.edu

Zhongze Tang
Rutgers University
Piscataway, NJ, USA
zhongze.tang@rutgers.edu

Nan Jiang
Zhejiang Sci-Tech University
Hangzhou, Zhejiang, China
jiangn@zstu.edu.cn

Tian Guo
Worcester Polytechnic Institute
Worcester, MA, USA
tian@wpi.edu

Lisong Xu
University of Nebraska-Lincoln
Lincoln, NE, USA
xu@unl.edu

Sheng Wei
Rutgers University
Piscataway, NJ, USA
sheng.wei@rutgers.edu

ABSTRACT

This paper aims to address the significant power challenges in live virtual reality (VR) streaming (a.k.a., 360-degree video streaming), where the VR view rendering and the advanced deep learning operations (e.g., super-resolution) consume a considerable amount of power draining the battery-constrained VR headset. We develop *EdgeVR*, a power optimization technique for live VR streaming, which offloads the on-device VR rendering and deep learning operations to an edge server for power savings. To address the significantly increased motion-to-photon (MtoP) latency due to the edge offloading, we develop a live VR viewport prediction method to pre-render the VR views on the edge server and compensate for the round-trip delays. We evaluate the effectiveness of *EdgeVR* using an end-to-end live VR streaming system with an empirical VR head movement dataset involving 48 users watching 9 VR videos. The results reveal that *EdgeVR* achieves power-efficient live VR streaming with low MtoP latency.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**.

KEYWORDS

Live streaming, virtual reality, power efficiency

ACM Reference Format:
Zichen Zhu, Xianglong Feng, Zhongze Tang, Nan Jiang, Tian Guo, Lisong Xu, and Sheng Wei. 2022. Power-Efficient Live Virtual Reality Streaming Using Edge Offloading. In *32nd edition of the Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '22), June 17, 2022, Athlone, Ireland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3534088.3534351

NOSSDAV '22, June 17, 2022, Athlone, Ireland

ACM ISBN 978-1-4503-9383-6/22/06…$15.00
https://doi.org/10.1145/3534088.3534351

1 INTRODUCTION

The recent rapid growth in consumer virtual reality (VR) headsets [7, 11, 14, 17] has enabled a new interface for presenting multimedia content such as videos and games [30, 34]. VR video streaming (a.k.a., 360-degree video streaming), which provides users with a fully immersive video viewing experience, has emerged as a widely deployed VR application [1, 2]. In particular, live VR streaming, where the VR video content is generated on the fly during streaming, has become the most attractive VR video use case (e.g., live sports broadcast [4, 6]) and garnered increased interests from the community given its inherent performance challenges [20, 27, 41].

However, the quality of experience (QoE) of VR video streaming, including live streaming, is fundamentally constrained by the high power consumption and limited battery capacity on the VR headset. For example, preliminary power measurement studies such as [23] reveal that VR video streaming consumes significantly more power than traditional 2D video, due to the increased data volume and computation required by VR-specific features. Moreover, the recent adoption of deep learning techniques (e.g., super-resolution) in state-of-the-art VR streaming systems [18, 19], although significantly improving QoE and bandwidth efficiency, poses additional workload on the VR headset to worsen the power efficiency. Since the VR headsets are driven by power-constrained batteries, the video viewing session may have to be terminated for a battery recharge, which severely degrades the user's QoE. Also, since the headset is a wearable device, the on-device heat accumulation and dissipation due to the intensive power consumption can significantly impact the viewing experience as well.

Prior power optimization techniques for traditional 2D videos [21, 28, 36, 37] cannot be directly applied to VR streaming due to the unique user-centric viewport control through head movements in the VR experience, as well as the newly introduced on-device deep

learning operations for VR. Such VR-specific features and operations trigger power consumptions caused by additional sensing, computation, and view generation [23]. Several recent works have started investigating power optimization for VR videos, such as dynamically scaling VR display brightness based on user's eye movements [39], dynamically adapting the frame rate [22], and leveraging specialized hardware or edge-based video pre-processing [25]. These studies provide valuable insights on VR power optimization; however, they rely on pre-processing of past video or user data, which are available only in the offline video-on-demand (VOD) scenario where the content is pre-recorded, instead of *live VR streaming* where the content is generated on the fly. Other recent works proposed to pre-render the user's viewport at an edge server and pre-deliver it to the client device to reduce latency or power consumption [25, 31]; however, the existing solutions either target the VOD streaming case [25] or require ideal network condition and video streaming pipeline to meet the latency requirement [31].

We develop *EdgeVR*, an edge offloading-based mechanism to reduce the power consumption in live VR streaming. Built on top of our preliminary, work-in-progress prototype [43], *EdgeVR* offloads the VR-specific, computation-intensive view generation and deep learning tasks to an edge server and thus reduces the on-device power consumption caused by the intensive computations. Despite the power benefits from offloading, *EdgeVR* is challenged by the round-trip communication delay added to the user's interactive VR experience, which consists of the time for the headset to send the user's head orientation information to the edge server and that for the edge server to deliver the rendered view back to the headset. Such delay becomes an integral part of the motion-to-photon (MtoP) latency [3, 5, 24, 42], i.e., the latency between the user's head movement and the display of the corresponding VR view, which may cause motion sickness if beyond 20 ms [3, 5].

To address the critical MtoP latency challenge, while taking into account the unique data availability and timing challenges posed by live streaming, we develop a pre-rendering mechanism at the edge server, which predicts the viewports that the user would most likely watch based on an online modeling of the user's head movements. The edge server pre-renders these predicted viewports and pre-delivers them to the client before being requested. The lead time achieved by viewport prediction is able to compensate for the increased MtoP latency and achieve a premium QoE. We evaluate *EdgeVR* using an empirical VR head movement dataset involving 48 users watching 9 VR videos. Our experimental results indicate around 60% power savings compared to the original VR streaming system and an average 11 ms MtoP latency under the live VR streaming scenario, meeting the interactive VR requirement.

2 BACKGROUND

2.1 Live VR Streaming

Figure 1(a) shows the overall workflow of a typical live VR streaming system. The live VR video is first captured by a 360-degree camera and packaged to video segments of a few seconds each. The generated video segments are then deployed immediately to a content server. Next, the headset requests the 360-degree video segments via HTTP as they become available, conducts VR-specific computations (e.g., super-resolution and view rendering) based on the user's head orientation, and presents the high-quality 2D viewport to the user.

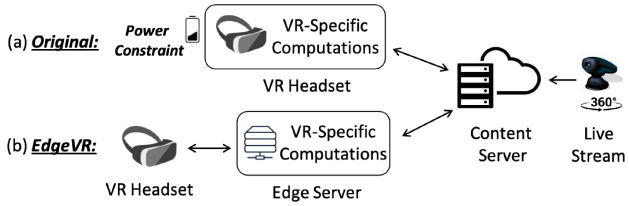

Figure 1: Overall workflow of (a) the original live VR streaming system; and (b) the proposed *EdgeVR* for power savings.

2.2 Power Consumption in VR Streaming

The live VR streaming system depicted in Figure 1(a) could pose significant challenges to the power consumption of the VR headset for two reasons. *(i)* **VR rendering**: Prior studies [23, 25] reported that VR rendering is the top power consumption sources in VR streaming. *(ii)* **Deep learning computations**: The newer development in VR streaming, which was not covered by the prior VR power measurement studies, is the adoption of deep learning techniques to improve user QoE and bandwidth efficiency. For example, recent VR streaming research [18, 19] adopted the super-resolution technique [26, 40] to reduce the bandwidth consumption, which streams a low-resolution video over the network and boosts the resolution at playback by executing a generative neural network model on the VR headset. In the neural network, the convolutional layers play the role of interpolation that takes in a low-resolution 2D image and outputs extra information in the feature dimension. Such computation-intensive deep learning operations would add additional power consumption to the already high power profile of VR rendering.

3 PROPOSED APPROACH: *EDGEVR*

We develop an edge offloading-based approach, namely *EdgeVR*, to minimize the on-device computation and thus power consumption in live VR streaming. Figure 1(b) depicts the overall system workflow of *EdgeVR*, as compared to the original workflow in Figure 1(a). In *EdgeVR*, the live VR video is first delivered to the edge server for processing and rendering. Then, the VR headset interacts with the edge server with the user head movement data to request the specific viewports. With the edge-based workflow, the most power-consuming component, i.e., the intensive computations caused by VR rendering and deep learning, can be offloaded to the edge server that is not power constrained. The VR headset now only needs to display the already rendered and resolution-boosted viewport, which would save significant amount of power.

Challenge: Motion-to-Photon Latency. The edge-based design presented in Figure 1(b) would pose a significant challenge to the QoE in VR streaming due to the increased motion-to-photon (MtoP) latency [3, 5, 24, 42]. The MtoP latency is defined as the time difference between when the user conducts a head movement and when the headset displays the corresponding view. This latency is critical in VR streaming, as a high MtoP latency may cause a

poor VR experience with disorientation and motion sickness [42]. We note that the MtoP latency would increase in the scenario of edge offloading, as it is now the sum of the network round-trip time and the computation/rendering time at the edge, which can be hundreds of milliseconds [31] and thus far beyond the requirement for interactive VR applications (i.e., 20 ms) [3, 5]. Moreover, such MtoP latency cannot be reduced via straightforward system or network engineering, simply due to the inevitable round-trip network delay and the video buffering. Therefore, a new thread of latency compensation approach tailored to the real-time requirement of VR is required. A key goal in *EdgeVR* is to develop a latency compensation mechanism to significantly reduce the round-trip delay caused by edge offloading (i.e., a few hundred milliseconds) to the acceptable level of VR MtoP latency (i.e., below 20 ms).

Solution: Live Viewport Prediction. Our key idea is to leverage viewport prediction to compensate for the impact of network delay on MtoP latency. Specifically, we leverage the user head movement to predict the user's future viewports and pre-render them at the edge server prior to the client's requests. The edge server then delivers the pre-rendered future viewports to the VR headset for display upon the actual user head movements. In order for *EdgeVR* to compensate for the aforementioned network delay, the live viewport prediction must satisfy two requirements: *(i)* sufficient prediction accuracy to cover the user's actual viewport; and *(ii)* sufficient lead time of the prediction to account for the network delay.

We develop a live viewport prediction scheme for *EdgeVR* by leveraging the real-time velocity and trajectory of the user head movement. The edge server collects the user's viewports in the previous frames, which form the user's past trajectory. Then, based on the past trajectory, we model the velocity of the head movement and estimate the trend of the viewing trajectory in the new frames. In particular, we set a sliding time window of K frames when forming the user viewing trajectory and update the window once the predictions on new frames are finished. Given the latest user feedback (i.e., the actual viewport) at frame c, we collect the viewports from frame i to frame c, where $i = c + 1 - K$. Based on the viewing trajectory with the K viewports, we calculate the velocity and estimate the new viewports from frame $c + 1$ to frame $c + \frac{T_{MtoP}}{2f}$, where f is the frame rate of the video, and T_{MtoP} is the MtoP latency. Upon receiving the user feedback for a new frame (i.e., frame $c + 1$), we shift the time window by 1, i.e., the new time window ranges from frame $i + 1$ to frame $c + 1$.

The user orientation can be described by yaw, pitch and roll, which can be further interpreted by three angles, namely β, θ and α. We consider a right-hand reference system where the roll, pitch and yaw axes are positive towards the user's nose, left ear and head, respectively. The user head movement can be described by angular velocity V_β, V_θ and V_α. In *EdgeVR*, we calculate the angular velocity of the three angles separately since they are orthogonal to each other. Given the angles in the previous K frames (i.e., vectors β, θ and α with length K), the angular velocity V_β, V_θ, and V_α can be calculated following Equation (1). The predicted user orientation, $\beta^{'}$, $\theta^{'}$ and $\alpha^{'}$ can be obtained using Equations (1) to (3).

$$V_\beta = \frac{\beta_K - \beta_0}{K}, V_\theta = \frac{\theta_K - \theta_0}{K}, V_\alpha = \frac{\alpha_K - \alpha_0}{K} \quad (1)$$

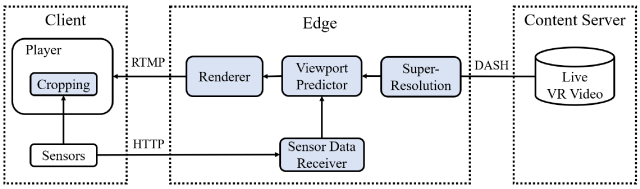

Figure 2: Architecture of the proposed *EdgeVR* system. The shaded blocks are *EdgeVR*-specific components.

$$T_p = \frac{T_{MtoP}}{2} \quad (2)$$

$$\beta^{'} = \beta_K + T_p \cdot V_\beta, \theta^{'} = \theta_K + T_p \cdot V_\theta, \alpha^{'} = \alpha_K + T_p \cdot V_\alpha \quad (3)$$

To mitigate the potential offsets due to slight head movements (e.g., caused by breathing), we extend the predicted viewport by a padding angle p (also referred to as *tolerance* hereafter). Assuming the original viewport angle is O, the angle of the predicted viewport will be $O + p$. Taking angle β (i.e., yaw) as an example, the predicted viewport has an angular range of $\beta \pm 0.5(O + p)$. In this work, we set $O = 90^\circ$ and $p = 30^\circ$ which increases the viewport by 33%.

4 EDGEVR SYSTEM IMPLEMENTATION

Our proposed *EdgeVR* system consists of three components: client, edge, and content server. Figure 2 illustrates the system architecture, in which the shaded blocks are *EdgeVR*-specific components. The *content server* provides HTTP-based live VR video stream following the DASH standard [33]. The *edge server* takes in the live VR video stream, renders the 2D viewport of each frame based on live viewport prediction, and streams the 2D frames to the client. The *client* involves a lightweight video player that crops and displays the 2D viewports based on the sensor samples of user head orientations.

4.1 Edge Server

4.1.1 Renderer. The *renderer* on the *edge server* plays the role of rendering the VR content. We adopt FFmpeg [10] as the overall multimedia framework on the edge server to process and render the live VR video. For rendering VR viewports in particular, we adopt FFmpeg360 [29], an open-source FFmpeg filter with the ability of rendering a specific viewport from a 360-degree video frame given the instant user head orientation. We leverage FFmpeg360 to process the input video with the predicted head orientation data. The rendered frames are passed to FFmpeg to be encoded into H.264 and delivered to the client via Real-Time Messaging Protocol (RTMP) [16].

4.1.2 Viewport Predictor. The role of the *viewport predictor* is to predict the user's future viewport for the time period of $T_{MtoP}/2$ following the velocity-based method presented in Section 3, leveraging the real-time sensor data received from the client. A key challenge in implementing the *viewport predictor* lies in the synchronization between the stream of the sensor data (i.e., head orientation) and the VR video stream. We addressed the challenge by implementing the *viewport predictor* as a filter in the FFmpeg framework used to implement the *renderer*, which eliminates the complex inter-process communication mechanisms (e.g., message queue, pipe, or UNIX domain socket) required by a standalone predictor module. The

predictor filter takes the frame right before the FFmpeg360 filter and makes the prediction. By using the value designed for private user data in the FFmpeg frame data structure, the *viewport predictor* can write the predicted viewports into the frame for the *renderer*. In our implementation, the *viewport predictor* and the *renderer* work in the same thread to avoid synchronization issues.

4.1.3 Viewport Scaling. As discussed in Section 3, we enlarge the predicted viewport to raise the tolerance of the prediction error. When rendering the enlarged viewport, we still require it to keep the same projection as the target viewport of the client. In practice, FFmpeg360 [29] uses perspective projection to generate the viewport from the entire frame. Considering there is only one object in the 360-degree scene, and the position of the user (i.e., the camera that generates the viewport) is fixed to the central point of the video sphere, we can further simplify the projection. Assuming the final viewport length is len_{vp} on the client, the following equation holds, where r is an intermediate parameter that reflects the depth of the 2-D panel viewport.

$$r = \frac{len_{vp}}{2} \cot \frac{O}{2} \tag{4}$$

Then, the enlarged viewport length len_{pred} can be calculated as

$$\frac{len_{pred}}{2} = r \tan \frac{p}{2} + \frac{len_{vp}}{2} \tag{5}$$

$$len_{pred} = len_{vp} \cot \frac{O}{2} \tan \frac{p}{2} + len_{vp} \tag{6}$$

In this way, when cropping the viewport O from the center of the enlarged viewport, we will obtain the viewport with len_{vp} as the length. With the enlarged size and the parameter r, we can calculate the enlarged viewport O' used in the rendering:

$$O' = 2 \tan^{-1} \frac{len_{pred}}{2r} \tag{7}$$

$$= 2 \tan^{-1} \left(\tan \frac{p}{2} + \tan \frac{O}{2} \right) \tag{8}$$

According to Equation (8), the enlarged viewport O' should be $O' = 2 \tan^{-1} (\tan 30/2 + \tan 90/2) \approx 103.48°$. Since we cannot set arbitrary viewport size in FFmpeg360, we set $len_{vp} = 800$, $len_{pred} = 1400$, $O = 90°$, and $O' = 120°$. This leads to two r values $r = 400$ and $r \approx 404$, which should be the same value theoretically. We use $r = 400$ to match the final viewport in our implementation.

4.1.4 Super-resolution. The super-resolution module on the edge server plays the role of boosting the received low-resolution VR video to high resolution. To balance the performance and the output quality, we utilize both ESPCN [32] and bi-linear interpolation [35] to accomplish the super-resolution operation. The input low-resolution VR video from the upstream server with the size of $w \times h$ is processed by ESPCN first via the PyTorch framework to boost the resolution to $2w \times 2h$. Then, the bi-linear interpolation further increases the resolution to $4w \times 4h$.

4.2 Client

4.2.1 Video Player. With edge offloading, the client component of *EdgeVR* plays the role of a lightweight 2D video player to display the viewport to the user. We adopt an open-source, FFmpeg-based player, *ijkplayer* [9], in the implementation of video player. The player receives the pre-rendered VR video frames from the edge server using the RTMP protocol [16]. Then, it decodes and crops the received frame based on the actual head orientation data for display.

4.2.2 Viewport Cropping. As described previously, our viewport prediction algorithm returns an enlarged viewport to mitigate the potential prediction errors or other noises. Therefore, before playing back the video, the video player must crop the target frame to the original size based on the actual user head orientation. Due to the projection during the rendering, the offsets of the cropped viewport are based on the deviation between the predicted and actual user head orientations, which can be represented as follows:

$$\Delta_{yaw} = r \tan \Delta\beta \tag{9}$$

$$\Delta_{pitch} = r \tan \Delta\theta \tag{10}$$

where $\Delta\beta$ and $\Delta\theta$ are the deviations of the head orientation in the *yaw* and *pitch* directions, respectively. Considering the rotation during the rendering, the deviation of the head orientation reflected in the enlarged frame satisfies the following relationship:

$$\Delta\beta = (\beta_{real} - \beta') \cos\alpha' - (\theta_{real} - \theta') \sin\alpha' \tag{11}$$

$$\Delta\theta = (\beta_{real} - \beta') \sin\alpha' + (\theta_{real} - \theta') \cos\alpha' \tag{12}$$

where β_{real} and θ_{real} are the user head orientation in the *yaw* and *pitch* directions, respectively. Using the central point of the enlarged frame as an origin, the coordinate of the actual user head orientation is (Δyaw, $\Delta pitch$). With the quantified offset, we can crop the rendered frame using FFmpeg for display.

Additionally, since we use $O' = 120°$ as the enlarged viewport, we have an extra angular error tolerance beyond the originally designed tolerance $p = 30°$. Based on Equation (8), we can calculate the actual tolerance p' as follows:

$$p' = 2 \tan^{-1} \left(\tan \frac{O'}{2} - \tan \frac{O}{2} \right) \tag{13}$$

$$= 2 \tan^{-1} \left(\tan \frac{120°}{2} - \tan \frac{90°}{2} \right) \approx 72.41° \tag{14}$$

Considering that the distortion becomes worse when the content is closer to the edge of the viewport, we limit the maximum offset to ±15° in the current implementation. Meanwhile, we maintain the ability to transmit the enlarged viewport with extra tolerance.

4.3 Communication between Edge and Client

There are two communication channels between the client and the edge server. First, the client sends the instant user head orientation data to the edge server via HTTP POST, which is required for the velocity-based viewport prediction on the edge server. Second, We use RTMP [16] to deliver the rendered video frames from the edge server to the client to minimize the communication delay. In addition to the rendered frame, the client requires the predicted head orientation corresponding to the frame, which is viewed as metadata associated with the predicted viewport. We adopt the *Supplemental Enhancement Information* (SEI) [8] in H.264 to transmit the metadata together with the rendered frame in order.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

Hardware and system setup. We evaluate the power efficiency of *EdgeVR* using two smartphones as the VR headsets: *(i)* LG V20 with 1.6 & 2.15 GHz CPU frequency, 4 GB RAM, and a 3300mAh removable battery; and *(ii)* Google Pixel 3 with 1.6 & 2.5 GHz CPU frequency, 4GB RAM, and a 2915 mAh non-removable battery. The smartphone under test connects to a wireless router from an approximate distance of 12 feet using the IEEE 802.11ac protocol. The edge server (dual E5-2623 CPU, 32GB RAM, and GTX TITAN X GPU) connects to the wireless router through a wired Ethernet connection. We conduct experiments to compare the proposed *EdgeVR* system with the following two baseline systems: (1) ***Original*** refers to the original VR video streaming system without power optimization. We implemented this system using the WebVR library [13], the DASH protocol [33], and the ESPCN [32] via the ONNX-js library [15]. It serves as baseline to evaluate the power benefit of edge offloading adopted in *EdgeVR*. (2) ***EdgeVR-NP*** refers to our proposed *EdgeVR* system but without the viewport prediction for MtoP latency compensation. It serves as a baseline to evaluate our viewport prediction-based latency compensation in *EdgeVR*.

Dataset. We use a VR head movement dataset [38] to emulate the user head movements, excluding a small number of data entries where the timestamps of the user traces are non-monotonic. We adopt the 48 users and 9 videos in the first group of the dataset, which represents the scenario where users freely watch the videos without being asked to accomplish any specific task.

Power Measurement Method. We adopt the Monsoon power monitor [12] for power measurements. To establish a relatively stable measurement environment, we mute the smartphone, set the brightness to the lowest level, and turn on the airplane mode with only WiFi enabled. For the *EdgeVR* and *EdgeVR-NP* cases, we directly measure the power consumption on our implemented end-to-end system. In the *Original* case, due to the high complexity of super-resolution computations, we were not able to achieve real-time super-resolution on the two smartphones and, therefore, we measure the following two components separately: *(i)* the VR rendering power is measured on a WebVR-based end-to-end video streaming system, following the same setup adopted in the literature [23]; and *(ii)* the super-resolution power is measured by a non-realtime implementation of super-resolution on the two smartphones using the ESPCN [32] via the ONNX-js library [15].

5.2 Power Evaluations

Table 1 shows the average power consumption of the three systems measured over 48 users and 9 videos on 2 different phones. We observe that *Original* consumes the highest average power among the three systems, and *EdgeVR-NP* consumes the lowest thanks to the benefit of edge offloading but would incur huge MtoP (as discussed in Section 5.3). The power consumption of *EdgeVR* is slightly higher than *EdgeVR-NP* as the headset needs to conduct the extra cropping operation before displaying the video. In summary, compared to *Original*, *EdgeVR* achieves significant power savings, i.e., 57% (LG V20) and 64% (Pixel 3). Figure 3 illustrates the power distribution of the three systems over 9 videos and 48 users on 2 different phones. For each video, we measure VR rendering for 48 users (1 minute each), and we measure super-resolution for 10 minutes as it is independent of user traces. We then plot the sum of (1) average VR rendering power over 48 users, and (2) a moving window average of super-resolution power (length = 10 samples and step = 10 samples), as the total power consumption of *Original*. The distribution results indicate that the most common power value of *EdgeVR* is significantly lower than *Original*, i.e., 1.93W vs. 4.81W (LG V20), and 1.61W vs. 4.40W (Pixel3).

Table 1: Average power consumption (48 users, 9 videos).

System	Average Power (Watt)	
	LG V20	Pixel 3
Original	5.01	4.58
EdgeVR-NP	1.65	1.41
EdgeVR	2.14	1.65

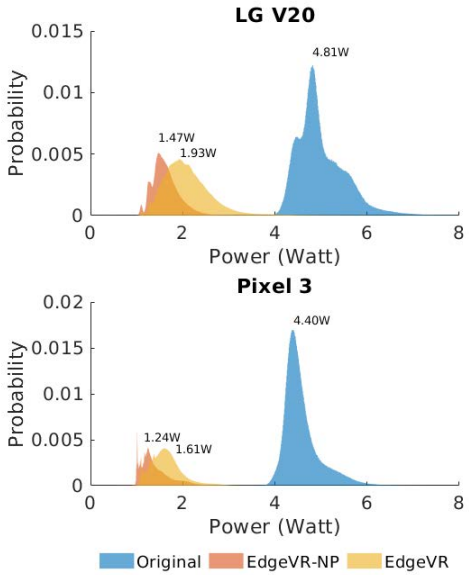

Figure 3: Power distribution comparison between *EdgeVR* and two baselines (48 users, 9 videos).

5.3 Latency Overhead

We measure the MtoP latency by calculating the time difference between the user head movement event and the corresponding frame display event. Figure 4 shows the distribution of the MtoP latencies for all the 9 videos and 48 users. The overall average MtoP latency of *EdgeVR* is 11 ms, versus 231 ms for *EdgeVR-NP*. Also, 95% of the MtoP latency values are within the 20 ms boundary, benefiting from the effective live viewport prediction in *EdgeVR*.

5.4 Quality Overhead

We note that the remote rendering and local cropping mechanisms adopted by *EdgeVR* may result in quality degradations of the VR

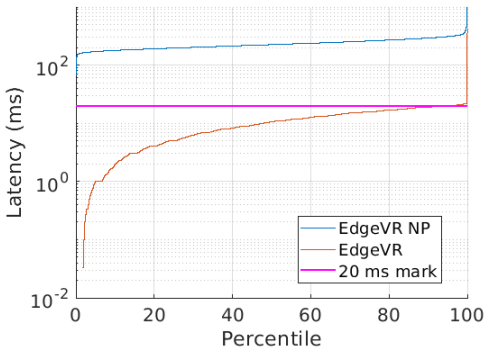

Figure 4: Distribution of the MtoP latency (ms) comparing *EdgeVR-NP* and *EdgeVR* for 9 videos and 48 users on LG V20.

Table 2: Per-video angular error distribution (in degrees) for *EdgeVR* in pitch, yaw, and roll directions over 48 users.

	Percentile	5	10	15	20	80	85	90	95
Video 1	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	3.15	12.55
	roll	0.06	0.12	0.19	0.26	4.89	6.40	8.72	13.22
Video 2	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	3.87	14.24
	roll	0.09	0.18	0.28	0.40	6.34	8.12	10.85	15.85
Video 3	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	2.73	13.27
	roll	0.06	0.12	0.18	0.25	4.26	5.58	7.70	12.05
Video 4	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	2.27	10.38
	roll	0.06	0.11	0.17	0.24	4.70	6.21	8.57	13.16
Video 5	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0.99	6.16	16.08
	roll	0.08	0.15	0.24	0.35	5.99	7.62	10.14	14.72
Video 6	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	3.96	12.66
	roll	0.05	0.09	0.15	0.21	4.15	5.43	7.44	11.47
Video 7	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	1.46	13.57
	roll	0.03	0.06	0.10	0.12	2.42	3.28	4.64	7.25
Video 8	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	1.82	7.20	17.33
	roll	0.06	0.12	0.19	0.27	4.47	5.75	7.75	11.51
Video 9	pitch	0	0	0	0	0	0	0	0
	yaw	0	0	0	0	0	0	2.20	11.01
	roll	0.04	0.09	0.14	0.20	4.41	5.86	8.11	12.51

video presented to the end user. It is because the central point of the actual viewport required by the end user may be shifted from that used for rendering the enlarged viewport at the edge server, due to the imperfect viewport predictions. The cropping helps reduce the mismatch but would introduce additional quality overhead, since it changes the content without changing the projection point.

To evaluate the quality impact of the viewport prediction, we adopt the angular distance (a.k.a., angular error) between the final viewport and actual user head orientation as our quality evaluation metric. Table 2 shows the detailed angular error distribution results for the 9 test videos over 48 users in the pitch, yaw, and roll directions. We observe that *EdgeVR* achieves 0 angular errors in 95% and 80% of the cases in pitch and yaw directions, respectively, and around 95% of the roll direction errors are within 15 degrees.

We adopt the SSIM metric to evaluate the impact of cropping (i.e., the potential distortion), as it can represent the similarity between the ground truth image and the one generated after cropping. Figure 5 shows the distribution of the per-frame SSIM results for each video over 48 users based on the dataset [38]. The top and bottom box boundaries indicate the 25% and 75% percentile of the results (i.e., $Q1$ and $Q3$), respectively; the top and bottom whisker marks indicate $Q1 - 1.5 \times (Q3 - Q1)$ and $Q3 + 1.5 \times (Q3 - Q1)$, respectively, which represent the non-outlier range; and the rest of the data points are outliers. We observe that the minimal median SSIM is 0.60 over 9 videos, and the average SSIM ranges from 0.62 to 0.80. Note that the interpretation of SSIM for visual quality may be different for the VR use case than the traditional 2D images.

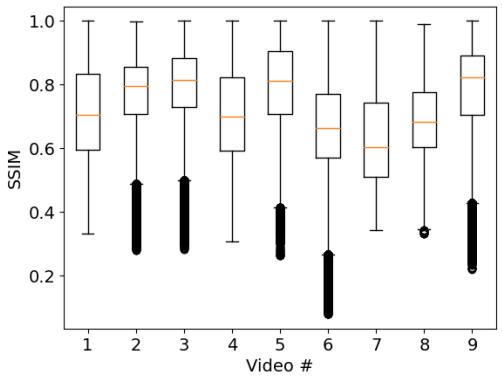

Figure 5: Per-frame SSIM results for 9 videos over 48 users.

6 CONCLUSION

We have developed *EdgeVR*, a power optimization framework targeting live VR streaming. *EdgeVR* leverages edge offloading to reduce the dominant power consumption caused by the on-device VR rendering and deep learning computations. Also, it compensates for the increased MtoP latency by dynamically predicting and pre-rendering the user's viewport of interest on the edge. The proposed edge offloading and live viewport prediction approaches achieve significant power savings while keeping a low MtoP latency to meet the real-time requirement. The repository of the project is at https://github.com/hwsel/EdgeVR.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive feedback. This work was partially supported by the National Science Foundation under awards CNS-1910085 and CNS-1755659.

REFERENCES

[1] 2015. Facebook to Support Spherical Video in News Feed and Oculus. https://techcrunch.com/2015/03/25/facebook-to-support-spherical-video-in-news-feed-and-oculus/.

[2] 2015. You Can Now Watch and Upload 360-Degree Videos on YouTube. https://www.wired.com/2015/03/youtube-360-degree-video/.

[3] 2016. Keeping the virtual world stable in VR, Qualcomm OnQ Blog. https://www.qualcomm.com/news/onq/2016/06/29/keeping-virtual-world-stable-vr.

[4] 2017. FOX Sports Unveils Social Virtual Reality with Three Matches at CONCACAF Gold Cup. https://www.foxsports.com/presspass/latest-news/2017/06/28/fox-sports-unveils-social-virtual-reality-three-matches-concacaf-gold-cup.

[5] 2018. Motion to Photon Latency in Mobile AR and VR. https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926.

[6] 2018. Niche or next? Utilizing VR for live sports - Will virtual reality catch on with sports audiences? https://www2.deloitte.com/us/en/pages/technology-media-and-telecommunications/articles/vr-live-sports.html.

[7] 2018. Samsung GearVR. https://www.samsung.com/global/galaxy/gear-vr/.

[8] 2019. H.264: Advanced Video Coding for Generic Audiovisual Services. https://www.itu.int/rec/T-REC-H.264.

[9] 2020. bilibili/ijkplayer. https://github.com/bilibili/ijkplayer.

[10] 2020. FFmpeg. http://ffmpeg.org/.

[11] 2020. Google Cardboard. https://vr.google.com/cardboard/.

[12] 2020. High Voltage Power Monitor. https://www.msoon.com/high-voltage-power-monitor.

[13] 2020. WebVR - Bringing Virtual Reality to the Web. https://webvr.info/.

[14] 2021. Oculus Quest 2. https://www.oculus.com/quest-2/.

[15] 2021. ONNX.js: run ONNX models using JavaScript. https://github.com/microsoft/onnxjs.

[16] 2021. RTMP Streaming: The Real-Time Messaging Protocol Explained. https://www.wowza.com/blog/rtmp-streaming-real-time-messaging-protocol.

[17] 2022. DODOcase VR. https://www.dodocasevr.com/.

[18] Jiawen Chen, Miao Hu, Zhenxiao Luo, Zelong Wang, and Di Wu. 2020. SR360: Boosting 360-Degree Video Streaming with Super-Resolution. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 1–6.

[19] Mallesham Dasari, Arani Bhattacharya, Santiago Vargas, Pranjal Sahu, Aruna Balasubramanian, and Samir R Das. 2020. Streaming 360-Degree Videos Using Super-Resolution. In *IEEE Conference on Computer Communications (INFOCOM)*. 1977–1986.

[20] Xianglong Feng, Viswanathan Swaminathan, and Sheng Wei. 2019. Viewport Prediction for Live 360-Degree Mobile Video Streaming Using User-Content Hybrid Motion Tracking. *ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)* 3, 2 (2019), 1–22.

[21] Wenjie Hu and Guohong Cao. 2015. Energy-Aware Video Streaming on Smartphones. In *IEEE Conference on Computer Communications (INFOCOM)*. 1185–1193.

[22] Nan Jiang, Yao Liu, Tian Guo, Wenyao Xu, Viswanathan Swaminathan, Lisong Xu, and Sheng Wei. 2020. QuRate: Power-Efficient Mobile Immersive Video Streaming. In *ACM Multimedia Systems Conference (MMSys)*. 99–111.

[23] Nan Jiang, Viswanathan Swaminathan, and Sheng Wei. 2017. Power Evaluation of 360 VR Video Streaming on Head Mounted Display Devices. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 55–60.

[24] Steven M. LaValle, Anna Yershova, Max Katsev, and Michael Antonov. 2014. Head Tracking for the Oculus Rift. In *IEEE International Conference on Robotics and Automation (ICRA)*. 187–194.

[25] Yue Leng, Chi-Chun Chen, Qiuyue Sun, Jian Huang, and Yuhao Zhu. 2019. Energy-Efficient Video Processing for Virtual Reality. In *International Symposium on Computer Architecture (ISCA)*. 91–103.

[26] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. 2017. Enhanced Deep Residual Networks for Single Image Super-Resolution. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 136–144.

[27] Xing Liu, Bo Han, Feng Qian, and Matteo Varvello. 2019. LIME: Understanding Commercial 360° Live Video Streaming Services. In *ACM Multimedia Systems Conference (MMSys)*. 154–164.

[28] Yao Liu, Mengbai Xiao, Ming Zhang, Xin Li, Mian Dong, Zhan Ma, Zhenhua Li, and Songqing Chen. 2015. Content-Adaptive Display Power Saving in Internet Mobile Streaming. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 1–6.

[29] Yao Liu, Chao Zhou, Shuoqian Wang, and Mengbai Xiao. 2019. FFmpeg360 for 360-Degree Videos: Edge-Based Transcoding, View Rendering, and Visual Quality Comparison: Poster. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 337–339.

[30] J. Logan Olson, David M. Krum, Evan A. Suma, and Mark Bolas. 2011. A Design for a Smartphone-Based Head Mounted Display. In *IEEE Virtual Reality Conference (VR)*. 233–234.

[31] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. 2019. Mobile VR on Edge Cloud: A Latency-Driven Design. In *ACM Multimedia Systems Conference (MMSys)*. 222–231.

[32] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1874–1883.

[33] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia* 18, 4 (2011), 62–67.

[34] Anthony Steed and Simon Julier. 2013. Design and Implementation of an Immersive Virtual Reality System Based on a Smartphone Platform. In *IEEE Symposium on 3D User Interfaces (3DUI)*. 43–46.

[35] Richard Szeliski. 2011. *Computer Vision: Algorithms and Applications*. Springer.

[36] Guibin Tian and Yong Liu. 2013. On Adaptive HTTP Streaming to Mobile Devices. In *International Packet Video Workshop (PV)*. 1–8.

[37] Sheng Wei, Viswanathan Swaminathan, and Mengbai Xiao. 2015. Power Efficient Mobile Video Streaming Using HTTP/2 Server Push. In *International Workshop on Multimedia Signal Processing (MMSP)*. 1–6.

[38] Chenglei Wu, Zhihao Tan, Zhi Wang, and Shiqiang Yang. 2017. A Dataset for Exploring User Behaviors in VR Spherical Video Streaming. In *ACM Multimedia Systems Conference (MMSys)*. 193–198.

[39] Zhisheng Yan, Chen Song, Feng Lin, and Wenyao Xu. 2018. Exploring Eye Adaptation in Head-Mounted Display for Energy Efficient Smartphone Virtual Reality. In *International Workshop on Mobile Computing Systems & Applications (HotMobile)*. 13–18.

[40] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural Adaptive Content-aware Internet Video Delivery. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 645–661.

[41] Jun Yi, Shiqing Luo, and Zhisheng Yan. 2018. A Measurement Study of YouTube 360° Live Video Streaming. In *ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 49–54.

[42] Jingbo Zhao, Robert S. Allison, Margarita Vinnikov, and Sion Jennings. 2017. Estimating the Motion-to-Photon Latency in Head Mounted Displays. In *IEEE Virtual Reality Conference (VR)*. 313–314.

[43] Zichen Zhu, Nan Jiang, Tian Guo, and Sheng Wei. 2019. Virtual Reality Streaming at the Edge: A Power Perspective: Poster. In *ACM/IEEE Symposium on Edge Computing (SEC)*. 316–318.