# Energy-Efficient 360-Degree Video Streaming on Multicore-Based Mobile Devices

Xianda Chen and Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University
E-mail: {xuc23, gxc27}@psu.edu

***Abstract*—360° video streaming becomes increasingly popular on video platforms. However, streaming (downloading and processing) 360° video consumes a large amount of energy on mobile devices, but little work has been done to address this problem, especially considering recent advances in the mobile architecture. Through real measurements, we found that existing systems activate all processor cores during video streaming, which consumes a great deal of energy, but this is unnecessary since most heavy computations in 360° video processing are handled by the hardware accelerators such as hardware decoder, GPU, etc. To save energy, we propose to selectively activate the proper processor cluster and adaptively adjust the CPU frequency based on the video quality. We model the impacts of video resolution and CPU frequency on power consumption, and model the impacts of video features and network effects on Quality of Experience (QoE). With the developed models, we formulate the energy and QoE aware 360° video streaming problem as an optimization problem with the goal of maximizing QoE and minimizing energy. We then propose an efficient algorithm to solve this optimization problem. Evaluation results demonstrate that the proposed algorithm can significantly reduce the energy consumption while maintaining QoE.**

## I. Introduction

360° video has become increasingly popular on many video platforms such as YouTube [1, 2]. 360° video is created by capturing scenes in all directions (i.e., panoramic views), and it provides immersive experience to users by allowing them to freely change the viewing orientation during video playbacks. Due to the panoramic nature of 360° videos and the limited Field of View (FoV) of the end devices (typically 90° to 120° [3]), only a small portion of the downloaded video is viewed by the user at a given time. Thus, 360° videos are much larger than conventional videos for the same user perceived quality [4, 5]. For example, for a FoV of 90°, to achieve the same perceived quality as viewing a planar video with resolution of 960x960, the 360° video has to be encoded at resolution of 3840x1920 (i.e., eight times larger than the planar video). Therefore, streaming (downloading and processing) 360° video over wireless networks consumes a great deal of energy on mobile devices.

The energy consumption of video processing (decoding and rendering) is mainly related to the complexity of video processing, which is determined by the resolution and the frame rate of the video [5]. As 360° video is much larger, i.e., with much higher resolution, video processing consumes a large amount of energy. Unfortunately, little work has been done to address this problem, especially considering recent advances in the mobile architecture.

Modern mobile devices feature multiple cores in a tri-cluster processor architecture which includes three processor clusters and each cluster is designed for efficiently handling different kinds of workloads. For example, Samsung S20 features eight cores in a tri-cluster configuration consisting of two big cores, two middle cores, and four little cores. Existing systems activate all these cores during video streaming, which consumes a large amount of energy, but unnecessary. This is because, in modern mobile devices, due to the use of many hardware accelerators such as hardware decoder, GPU, and display processing unit, the workload of the CPU can be significantly reduced. As a result, it is not necessary to activate all eight cores of CPU. In fact, based on real experiments and measurements, we found that the little core processor cluster, with the help of hardware accelerators, has enough computational capacity to handle the workload of 360° video processing. Moreover, the default CPU governor sets the CPU at high frequency even when processing low resolution videos. To further save energy, we should properly adjust the CPU frequency based on the video resolution and the hardware characteristics of the mobile device.

To support 360° video streaming on mobile devices, both energy efficiency and user perceived Quality of Experience (QoE) are important. The energy consumed for video downloading and video processing is related to the video quality and the CPU frequency used to process the video. Thus, to save energy while maintaining QoE, we should download video at the right quality, and adjust the CPU frequency based on the video resolution and the hardware characteristics of the mobile device. Base on real measurements, we model the impact of video resolution and CPU frequency on power consumption, and model the impact of video features (i.e., video resolution and video bitrate) and network effects (i.e., rebuffering events) on QoE. With these developed models, we formulate the energy and QoE aware 360° video streaming problem as an optimization problem with the goal of maximizing QoE and minimizing energy. Since the optimal solution to this optimization problem requires perfect knowledge of future tasks which is not available in practical scenarios, we further propose a heuristic based algorithm for 360° video streaming.

The paper has the following main contributions.

- Through real measurements, we identify the energy inef-

ficiency problem of 360° video streaming on multicore-based mobile devices, and propose to save energy by selectively activating the proper processor cluster and adaptively adjusting the CPU frequency.
- We formulate the energy and QoE aware 360° video streaming problem as an optimization problem, and propose an efficient algorithm to solve it.
- We evaluate the performance of the proposed algorithm with real measurements. The evaluation results demonstrate that the proposed algorithm can significantly reduce the energy consumption while maintaining QoE.

In the reminder of this paper, we introduce background and motivation in Section II, and present the video, QoE, and power models in Section III. Section IV formalizes and solves the energy and QoE aware 360° video streaming problem. Section V presents evaluation results. We discuss related work in Section VI, and conclude the paper in Section VII.

## II. Background and Motivation

Different from conventional video, 360° video provides users with immersive experience, i.e., a user can navigate in a virtual world by looking around and interact with the virtual world. 360° video can be viewed with dedicated head mounted display, like Oculus [6] and HTC Vive [7], or by placing smartphones in headsets such as Google Cardboard [8] and Samsung Gear VR [9].

Fig. 1 shows the 360° video processing pipeline, where hardware accelerators in modern mobile devices, such as hardware decoder, GPU, and display processing unit, are leveraged for processing 360° video. More specifically, the device first receives encoded 360° video from the video servers (e.g., YouTube [1], Facebook-360 [2], etc), or loads the video from local storage. With a hardware decoder, the video is decoded to retrieve the original 360° frames, which are buffered in the video buffer, waiting to be rendered. Unlike the conventional 2D video processing where the decoded frames can be directly displayed on the screen, in 360° video processing, the actual viewing video content (called FoV frame) is rendered before being displayed. Based on the user's head orientation, a coordinated projection is performed to map the 3D coordinates of the viewing area to 2D coordinates, based on which the FoV frames are generated. In modern mobile devices, GPU is leveraged to accelerate the rendering process. After projection, the display processor reads the generated FoV frames from the video buffer and displays on the screen.

Recently, more and more smartphone manufactures such as Samsung, LG, Huawei, etc., are adopting the tri-cluster processor architecture which consists of three processor clusters, suitable for different kinds of workloads. For example, the Samsung Galaxy S20 smartphone uses Exynos 990 SoC with quad-core ARM Cortex-A55 (i.e., little cores 0 to 3), dual-core ARM Cortex-A76 (i.e., middle cores 4 and 5), and dual-core Mongoose-M5 (i.e., big cores 6 and 7). The Cortex-A55, Cortex-A76, and Mongoose M5 cores can operate in a frequency range from 442 MHz to 2.002 GHz, from 507 MHz to 2.504 GHz, and from 546 MHz to 2.73 GHz, respectively.

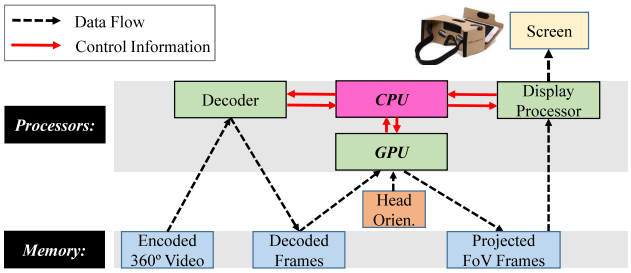



Fig. 1: A 360° video processing pipeline on a mobile device.

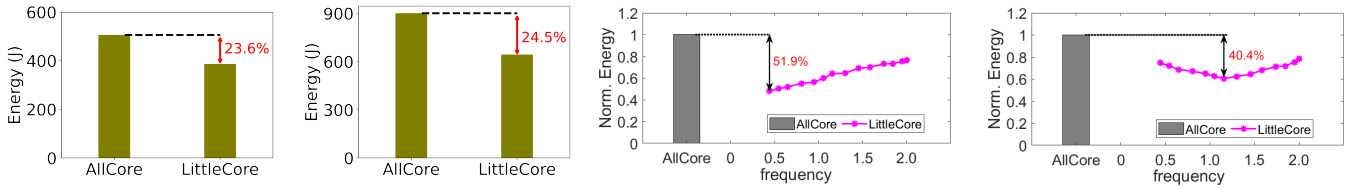
To better understand the characteristics of heterogeneous CPU processor architecture, we measure the average energy consumption of 360° video processing (i.e., local playing video). The experiment was conducted using a rooted Samsung S20. The energy consumption was calculated as the difference between the energy consumed for 360° video local playback and the energy consumed when the video player is turned on but no video is played. The videos are locally cached in the smartphone, and encoded at different qualities, i.e., resolution of 144p and 2160p. The videos are encoded at 30 frames per second. We compare two approaches, AllCore which is the default configuration, and LittleCore which only uses little cores.

As shown in Figure 2(a) (b), compared to the AllCore approach, using only the cores within the same processor cluster can save energy for processing 360° video. Compared to the AllCore approach, the LittleCore approach can reduce the energy consumption by 23.6% and 24.5% for video encoded at resolution of 144p and 2160p, respectively.

**Observation and Insight:** *In modern mobile devices, due to the use of many hardware accelerators such as hardware decoder, GPU, and display processing unit, the workload of the CPU can be significantly reduced. As a result, it is not necessary to activate all eight cores of CPU. Using only the little cores has enough computational capacity to handle the workload of 360° video processing, and hence save energy.*

The energy can be further reduced by CPU frequency scaling. The CPU frequency and the CPU voltage can be adjusted at run-time, which is referred to as Dynamic Voltage and Frequency Scaling (DVFS). The CPU frequency can be adjusted based on the CPU governor. However, the default CPU governor in most smartphones sets the CPU at high frequency to provide better performance, which may waste a great deal of energy. To better understand the relationship between CPU frequency and energy consumption for processing 360° video, we conducted experiments by setting different CPU frequencies (details see Section III-B).

Figure 2(c) illustrates the energy consumption of the LittleCore approach with different CPU frequencies for playing a 360° video with resolution of 144p. Here, the energy is normalized based on that of the AllCore approach. When setting the CPU cores at 442 MHz, the LittleCore approach can save energy by 51.9% compared to AllCore. That is, by reducing the CPU frequency to 442 MHz, LittleCore can further save 28.3% energy, compared to working at a higher CPU frequency based on the default CPU governors. Note




(a) Energy consumption (144p) (b) Energy consumption (2160p) (c) Energy w.r. CPU frequency (144p) (d) Energy w.r. CPU frequency (2160p)

Fig. 2: Comparisons of energy consumption for processing 360° video using all cores (default configuration) and using only the little cores. There are no video stalls in both approaches.

that in both approaches, there are no video stalls and hence the Little cores have enough computation capacity to handle the video processing workload with the help of hardware accelerators.

Fig. 2(d) compares the energy consumption for playing 2160p 360° video. Processing high resolution video requires more computational capacity. Although the computation intensive tasks such as decoding and rendering are handled by the dedicated hardware accelerators, the CPU has to coordinate them, which increases its workload. As the workload increases, using the lowest CPU frequency may not always be the best option to save energy, since low CPU frequency may mean more CPU utilization and then increase the energy consumption. As a result, the best CPU frequency to save energy may not be the highest or lowest. As shown in Fig. 2(d), setting the CPU frequency at 1.157GHz, the LittleCore approach can save the energy consumption by 40.4% compared to AllCore. That is, the LittleCore approach can further save 15.9% energy for processing 2160p 360° videos by dynamically adjusting the CPU frequency.

**Observation and Insight:** *The default CPU governor uses high CPU frequency even when processing low resolution videos, which consumes a great deal of energy. To further save energy, we should properly adjust the CPU frequency based on the video resolution and the hardware characteristics of the mobile device.*

Based on the above observations and insights, to save energy while maintaining QoE, we should download video at the right quality, and adjust the CPU frequency based on the video resolution and the hardware characteristics of the mobile device. In this paper, base on real measurements, we model the impact of video resolution and CPU frequency on power consumption, and model the impact of video features and network effects on QoE. With these developed models, we formulate the energy and QoE aware 360° video streaming problem and propose an efficient algorithm to solve it.

## III. Video, Power and QoE Models

### A. Video Model

On the server side, the video is divided into a sequence of $n$ video segments, where each segment contains a fixed duration of video content (e.g., $L$ seconds). Each video segment is encoded as $V$ copies corresponding to $V$ different qualities (i.e., different video resolutions and encoding bitrates). For example, similar to the common settings in YouTube, the videos are encoded into eight quality levels. Based on the network condition, the video player at the client requests a segment with a specific quality level. The 360° video streaming process can be considered as $n$ tasks which corresponds to downloading and transmitting $n$ video segments. Let $T_k$ denote the $k^{th}$ task which streams the $k^{th}$ video segment. Let $T_k^v$ denote the $k^{th}$ task where the video segment is encoded at quality level $v$ ($v \in \{1, 2, ..., V\}$). Let $B_k$ denote the video length (in seconds) of the downloaded but not yet viewed video in the buffer, when the client requests the $k^{th}$ video segment. To avoid rebufferings (or video stall), the video segment should be completely downloaded before the video player runs out of buffer.

### B. Power Model

The power consumed for 360° video streaming on mobile devices includes two parts: video downloading ($P_d$) and video processing ($P_p$). The downloading energy is related to the video quality level and the wireless link interface, and the processing energy is related to the video quality level and the hardware characteristics of the mobile device. To measure the power consumed by the wireless interface $P_d$, we conduct a number of experiments with a wget daemon running in the background (the screen is off) to download data from the server. Since all other background tasks are turned off and only one data downloading app is running, the measured power for data downloading minus the base power without data downloading will be the power for the wireless link interface. The measurement is based on Samsung Galaxy S20, and the power level is calculated as the battery voltage times the current, which can be read from the */sys/class/power_supply/battery/* files in the virtual file system. To model $P_p$, we measure the power consumption when using little cores with different CPU frequencies for playing 360° video encoded at different quality levels. Table I shows the encoding details of the videos. Specifically, we use the same video resolutions as YouTube, which has eight resolutions for 360-degree video, i.e., 144p, 240p, 360p, 480p, 720p, 1080p, 1440p, 2160p. Since the videos are locally cached, $P_p$ models the impact of the video quality and the CPU frequency on power consumption.

When watching 360° video with a specific video quality, we consider the following two cases. In **case 1** (i.e., the baseline case), the video player is turned on but no video is played. In **case 2** (i.e., the playback case), the CPU cores are manually controlled by setting the CPU frequency value

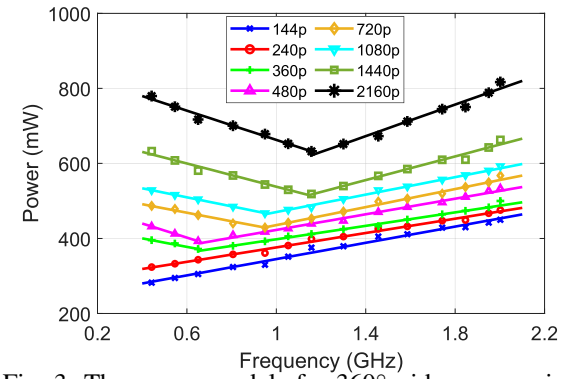


Fig. 3: The power models for 360° video processing.

TABLE I: The resolution and bitrate for video encoding.

| Level | Resolution | Bitrate (Mbps) |
|---|---|---|
| 2160p | 3840x2160 | 19.69 |
| 1440p | 2560x1440 | 13.24 |
| 1080p | 1920x1080 | 9.89 |
| 720p | 1280x720 | 7.97 |
| 480p | 852x480 | 4.82 |
| 360p | 640x360 | 2.75 |
| 240p | 424x240 | 1.78 |
| 144p | 256x144 | 0.98 |

in $/sys/devices/system/cpu/[cpu\#]/cpufreq$. The power difference between case 2 and case 1 represents the power consumption of video processing under a specific CPU frequency (i.e., $P_p(f)$). Note that $P_p(f)$ does not consider the power consumption of the screen (i.e., since the screen is turned on for both case 1 and case 2), since it depends on the screen size of the specific smartphone and the screen brightness set by the user. By running the least squares regression method, we can draw the fitted curve in Figure 3. The power models for processing 360° videos with different quality levels are summarized in Table II, where the CPU frequency $f$ is in GHz and the power is in mW.

With the developed power models, the energy consumed for downloading and processing a video segment can be calculated as follows.

$$E(T_k^v) = E_d(T_k^v) + E_p(T_k^v) \tag{1}$$

where $E_d(T_k^v)$ and $E_p(T_k^v)$ are the energy consumption for downloading and processing the $k^{th}$ video segment, respectively. $E_d(T_k^v)$ which can be calculated as $E_d(T_k^v) = P_d \cdot \frac{S(T_k^v)}{R_k}$, where $P_d$ is the wireless interface power, as shown in Table III, $S(T_k^v)$ is the segment data size when the $k^{th}$ video segment is encoded at quality level $v$, and $R_k$ is the network bandwidth used to download the video segment. $E_p(T_k^v)$ are related to the duration of each video segment $L$, i.e., $E_p(T_k^v) = P_p(f) \cdot L$.

### C. QoE Model

The user's perceived QoE for watching a video is defined as the average QoE values for all video segments. For each video segment $k$, similar to [4, 10, 11, 12], the QoE model quantifies the user perceived quality by considering the following metrics: average video quality, quality variation, and rebuffering. The QoE model is defined as follows:

TABLE II: The power models for video processing.

| Video Quality | Power (mW) |
|---|---|
| 144p | $P_p(f) = 108.4f + 236.5$ |
| 240p | $P_p(f) = 95.9f + 280.2$ |
| 360p | $P_p(f) = \begin{cases} -113.5f + 445.6, & \text{if } f \le 0.65 \\ 89.6f + 308.2, & \text{otherwise} \end{cases}$ |
| 480p | $P_p(f) = \begin{cases} -188.7f + 515.3, & \text{if } f \le 0.65 \\ 104.1f + 318.8, & \text{otherwise} \end{cases}$ |
| 720p | $P_p(f) = \begin{cases} -113.7f + 536.7, & \text{if } f \le 0.949 \\ 120.2f + 315.4, & \text{otherwise} \end{cases}$ |
| 1080p | $P_p(f) = \begin{cases} -120.6f + 581.9, & \text{if } f \le 0.949 \\ 116.5f + 353.8, & \text{otherwise} \end{cases}$ |
| 1440p | $P_p(f) = \begin{cases} -154.4f + 692.4, & \text{if } f \le 1.157 \\ 157.9f + 334.3, & \text{otherwise} \end{cases}$ |
| 2160p | $P_p(f) = \begin{cases} -172.8f + 832.4, & \text{if } f \le 1.157 \\ 220.8f + 358.7, & \text{otherwise} \end{cases}$ |

TABLE III: The power models for wireless interfaces.

| Interface | Power (mW) |
|---|---|
| Wifi | $P_d = 1201.8 \pm 29.9$ |

$$Q(T_k^v) = Q_o(T_k^v) - \omega_c I_c(T_k^v) - \omega_r I_r(T_k^v) \tag{2}$$

where $Q_o(T_k^v)$ is the "original" video quality without considering any quality impairment, $I_c(T_k^v)$ is the quality impairment caused by quality change between two consecutive segments, $I_r(T_k^v)$ is the quality impairment caused by rebuffering event, and $\omega_c$ and $\omega_r$ are the weights for quality change and rebuffering, respectively. The following gives the details of $Q_o$, $I_c$, and $I_r$.

- *Average Quality*. Since only video content in the viewing area contributes to user perceived quality, the average quality in the viewing area is considered to calculate $Q_o$.

$$Q_o(T_k^v) = q(\overline{\mathcal{V}_k}) \tag{3}$$

where $\overline{\mathcal{V}_k}$ denotes the average video quality (i.e., video bitrate, in Mbps) in the viewing area, $q(.)$ is a mapping function which maps the video quality to the user perceived quality. Similar to [10, 13], $Q_o$ is modeled with a Michaelis-Menten function, i.e., $Q_o = max(1, min(5, 1+4\cdot\frac{c_1\cdot\mathcal{V}_k}{c_2+\mathcal{V}_k}))$, where $\mathcal{V}_k$ is the bitrate, and $c_1$ and $c_2$ are the model parameters. We use the parameter values from [10], i.e., $c_1 = 1.036$ and $c_2 = 0.429$. These values are set up based on subjective quality assessment experiments.
- *Quality variation*. With quality variations between consecutive video segments, users may feel discomforts such as dizziness. Thus, the QoE model should consider quality variations.

$$I_c(T_k^v) = \frac{max(\overline{\mathcal{V}_{k-1}} - \overline{\mathcal{V}_k}, 0)}{\overline{\mathcal{V}_k}} \cdot Q_o(T_k^v) \tag{4}$$

where $\overline{\mathcal{V}_{k-1}}$ and $\overline{\mathcal{V}_k}$ are the video bitrate in the viewing area for the $(k-1)^{th}$ and $k^{th}$ video segment. $I_c$ quantifies the quality impairment due to bitrate drop (i.e., $max(\overline{\mathcal{V}_{k-1}} - \overline{\mathcal{V}_k}, 0) < 0$; a sudden large bitrate drop may

lead to severe QoE degradation. For the case of bitrate increase, due to network variations, a sudden large bitrate increase may generate rebuffering events.

- *Rebuffering Effect*. When rebuffering occurs, the video will freeze and hence significantly affecting the QoE.

$$I_r(T_k^v) = \frac{max(S(T_k^v)/R_k - B_k, 0)}{B_k} \cdot Q_o(T_k^v) \quad (5)$$

where $S(T_k^v)$ is the segment data size when the $k^{th}$ video segment is encoded with quality level $v$, $R_k$ is the downloading throughput, and $B_k$ is the amount of video data in the buffer when the client starts to request the $k^{th}$ segment.

For 360° video streaming on mobile devices, streaming a video segment encoded at higher quality can improve the $Q_o$ factor in the QoE model. However, because of network variations, a sudden large bitrate increase may cause frequent quality changes (i.e., larger $I_c$) and more rebuffering events (i.e., larger $I_r$), and then in turn affect the user QoE. Moreover, streaming video at higher quality requires much more data to be downloaded and processed on smartphones and thus consuming a great deal of energy.

## IV. Energy and QoE Aware 360° Video Streaming

In this section, we formulate and solve the energy and QoE aware 360° video streaming problem.

### A. Problem Formulation

We use two binary variables ($\eta_{kv}$ and $\delta_{kvf}$) in our problem formulation, where $\eta_{kv} = 1$ if the $k^{th}$ video segment is encoded at quality level $v$. Different implementations of heterogeneous processor architecture may have different clusters of CPU cores, with different number of CPU frequencies. To simplify the formulation, we map different pairs of core and frequency to a set of processing levels $\{1, 2, ..., F\}$, and $\delta_{kvf} = 1$ if the CPU cores are set at frequency $f \in \{1, 2, ..., F\}$ to process video segment $k$. Then, we have the following optimization problem, where the goal is to minimize energy and maximize QoE by selecting the right quality level and processing level for each video segment.

$$\textbf{min} \quad \sum_{k=1}^{n}\sum_{v=1}^{V}\eta_{kv}(\gamma\sum_{f=1}^{F}\delta_{kvf}\frac{E(T_k^{v,f})}{E(T_k^{V,F})} - (1-\gamma)\frac{Q(T_k^v)}{Q(T_k^V)}) \quad (6)$$

$$\textbf{s.t.} \quad \sum_{v=1}^{V}\sum_{f=1}^{F}\eta_{kv}\cdot\delta_{kvf} = 1, \quad \forall k \quad (6a)$$

$$\sum_{v=1}^{V}\eta_{kv}\cdot S(T_k^v) \le R\cdot B_k, \quad \forall k \quad (6b)$$

where $E(.)$ and $Q(.)$ are the energy model and QoE model. $T_k^v$ is the $k^{th}$ task to stream the video segment encoded with quality level $v$, $S(T_k^v)$ is the data size, $R$ is the network bandwidth, $B_k$ is the duration of video content that are buffered but not yet viewed at the time of requesting video segment $k$. Constraint (6a) enforces that only one quality level is selected for every video segment, and only one processing level is used to process the downloaded video segment. Constraint (6b) ensures that the video segment is successfully transmitted before being viewed. Because $E(T_k^{v,f})$ and $Q(T_k^v)$ use different units, we normalize them with the highest energy consumption ($E(T_k^{V,F})$) and QoE ($Q(T_k^V)$) obtained by using the highest processing level and the highest quality level. We also introduce a weighting factor $\gamma$, where more weights are given to maximize QoE with a smaller $\gamma$, and more weights are given to minimize energy when $\gamma$ is larger.

### B. The Optimal Solution

In 360° video streaming, finding the optimal quality level and processing level for each video segment (task) can be mapped to the shortest path problem [14, 10].

Let node $T_s$ represent the start and let $T_e$ represent the end of the 360° video streaming process. With $V$ bitrates and $F$ frequencies, $V * F$ nodes are added for each task ($T_k$). Let $T_k^{v,f}$ denote the video segment downloaded in $T_k$, encoded with bitrate index $v$, and processed with CPU frequency $f$. Fig. 4 illustrates how the graph is constructed. More specifically, edges are added from $T_s$ to each node in task $T_1$, and from each node in task $T_n$ to $T_e$. An edge is also added from each node in task $T_k$ to all nodes in the next task. By taking into consideration of QoE and energy, the edge weight is as follows: $(\gamma\frac{E(T_k^{v,f})}{E(T_k^{V,F})} - (1-\gamma)\frac{Q(T_k^{v,f})}{Q(T_k^{V,F})})$. For edges from nodes of the last video downloading task $T_n$ to node $T_e$, we set their weights to 0.

The constructed graph considers all possible cases of $(v, f)$ tuples for each task and all schedule paths between tasks, and hence each path from $T_s$ to $T_e$ can be mapped to the selection of bitrate and frequency in video streaming, and vice versa. Thus, the shortest path from $T_s$ to $T_e$ will be the optimal selection of $(v, f)$ tuples for all tasks in the 360° video streaming to achieve the goal of maximizing QoE and minimizing energy.

Dijkstra's algorithm can be applied to find the shortest path in this graph. For a video streaming problem with $n$ tasks, the graph has $O(nVF)$ nodes and $O(n(VF)^2)$ edges. Dijkstra's algorithm can be solved with the complexity of $O((N+E)logN)$, where $N$ denotes the number of nodes and $E$ denotes the number of edges, and hence the optimal solution has a time complexity of $O(n(VF)^2log(nVF))$. Since $V$ and $F$ are small, the algorithm can quickly find the optimal selection of $(v, f)$ tuples.

### C. The Heuristic Based Algorithm

The optimal algorithm can only provide a performance upper bound for comparison purposes, and it cannot be implemented in practice since it requires information about all future tasks. As a result, we propose a heuristic based algorithm which is described in Algorithm 1. In this algorithm, similar to [15, 10], the network bandwidth is estimated with the harmonic mean of the downloading throughput of the last

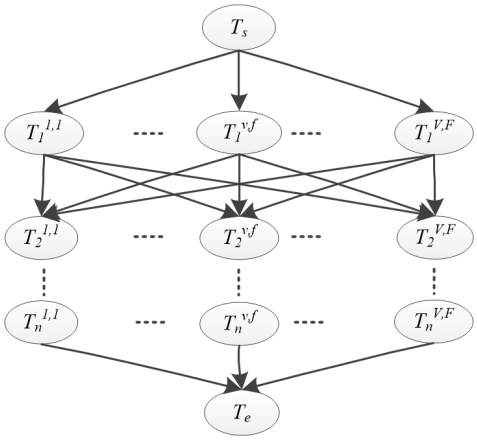



Fig. 4: The energy and QoE aware 360° video streaming problem is mapped to a shortest path problem.

several segments. More bandwidth estimation techniques can be found in [16, 17, 18], which is not the focus of this paper.

Based on the available bandwidth, the algorithm finds the right bitrate and processing frequency for each video segment. More specifically, it calculates the energy consumption ($E(T_k^{v,f})$) and the QoE ($Q(T_k^{v,f})$) when downloading the $k^{th}$ video segment encoded with bitrate $v$ and the video segment is processed with CPU frequency $f$. To achieve the goal of maximizing QoE and minimizing energy, the algorithm determines the quality level and the processing frequency in a greedy manner, i.e., based on the values of $(\frac{(1-\gamma)\cdot Q(T_k^{v,f})}{Q(T_k^{V,F})})/(\frac{\gamma\cdot E(T_k^{v,f})}{E(T_k^{V,F})})$ (called objective values).

Since the network bandwidth may frequently change, increasing the bitrate too much may create more video stalls and frequent bitrate changes, and decreasing the bitrate too much may lead to severe QoE degradation. To address these issues, we apply a gradual bitrate change strategy. Specifically, the algorithm sorts the tuples of $(v, f)$ in descending order of their objective values, and if two tuples of $(v, f)$ have the same objective values, the one with larger $v$ will be sorted first. The $(v, f)$ tuple with maximum objective value is called the *reference* tuple. The algorithm searches from the reference tuple to determine the final $(v, f)$ tuple, considering the following two cases.

The first case is related to bitrate increase. If the reference bitrate is one level higher than that of the previous video segment, the reference $(v, f)$ tuple is selected. Otherwise, if the reference bitrate is multiple levels higher than that of the previous segment, a search starts from the reference tuple to find the first $(v, f)$ tuple whose bitrate is one level higher than its previous video segment (lines 4-5 in Algorithm 1). This gradual bitrate change can reduce the QoE degradation caused by frequent bitrate updates. If the network bandwidth remains high for multiple continuous segments, the video bitrate will be gradually increased to the reference bitrate.

The second case is related to bitrate drop. If the reference bitrate is one level lower than that of the previous video segment, the reference $(v, f)$ tuple is selected. Otherwise, if the reference bitrate is several levels lower than its previous segment, a search starts from the bitrate of the previous segment to the reference bitrate, to find the first $(v, f)$ tuple whose bitrate is capable of being used to successfully download the video segment before the buffer runs out (lines 6-9 in Algorithm 1). If the network bandwidth remains low for multiple continuous segments, the video bitrate will be gradually reduced to the reference bitrate.

**Algorithm 1:** The Heuristic Based Algorithm

**Input :** $\gamma$, $B_k$, previous bitrate $v_{k-1}$
**Output:** $(v_k, f_k)$: bitrate level and processing frequency for video segment $k$

1 Estimate available bandwidth $\widehat{R}_k$
2 Create a set $\Pi$ of $(v, f)$ tuples in descending order of their objective values and in descending order of bitrate level if they have the same objective values
3 **foreach** $(v_k^\star, f_k^\star) \in \Pi$ **do**
4  **if** $v_k^\star > v_{k-1}$ **then**
5  $v_k \leftarrow v_{k-1} + 1$
6  **else**
7  $v_k \leftarrow \max\{v | v \in \{v_k^\star, ..., v_{k-1}\}$ and $(\frac{S(T_k^v)}{\widehat{R}_k} \leq B_k)\}$
8  // $S(T_k^v)$: data size of segment $k$ at bitrate $v$
9  **end**
10 **end**
11 **return** $(v_k, f_k)$

## V. Performance Evaluations

In this section, we evaluate the performance of our energy and QoE aware 360° video streaming algorithm, and compare it with other approaches.

### A. Experiment Setup

We collect traces by watching seven videos using a Samsung S20 smartphone. The videos have various lengths and they are encoded with different quality levels. Table IV shows the details of these seven videos, where the video bitrates correspond to the common video resolutions in YouTube, i.e., {144p, 240p, 360p, 480p, 720p, 1080p, 1440p, 2160p}. Similar to [4, 19, 12], each video is divided into a sequence of video segments, and each has one second of video. The user viewing area is determined by the viewing center and the FoV of the end device, which is set to 100 degrees horizontally and vertically [20, 21, 22]. For the QoE model, similar to [4, 11], the weights are set to ($\omega_v$, $\omega_r$) = (1, 1). The power models are built based on real measurements using a rooted Samsung Galaxy S20, as shown in Section III-B. We set the playback buffer to five seconds, and set the weight factor $\gamma = 0.5$.

The network traffic is generated based on forty throughput traces with varying patterns [23]. To consider different network conditions, we linearly scale the trace to generate three traces, called *low*, *medium*, and *high* bandwidth traces. Fig. 5 shows the bandwidth CDF of the three traces. For the *high* bandwidth traces, we take the original forty throughput traces and eliminate the throughput records which are less than 2Mbps. Then, the *medium* bandwidth traces are linearly down scaled by half from the *high* bandwidth traces, and the *low* bandwidth traces are a quarter of the *high* bandwidth traces.

TABLE IV: Video traces for performance evaluations.

| Video | Length (sec) | Video bitrates (Mbps) |
|---|---|---|
| 1 | 243 | [0.78, 1.11, 2.15, 3.68, 6.78, 8.45, 10.28, 18.62] |
| 2 | 337 | [1.57, 2.34, 3.25, 5.38, 7.82, 9.56, 13.24, 20.36] |
| 3 | 435 | [1.17, 2.23, 3.25, 5.86, 9.72, 11.12, 14.53, 19.46] |
| 4 | 538 | [0.77, 1.31, 2.45, 4.48, 7.87, 9.31, 12.22, 18.96] |
| 5 | 674 | [1.23, 2.45, 3.24, 5.46, 8.86, 10.12, 13.65, 20.75] |
| 6 | 734 | [0.87, 1.83, 3.11, 4.36, 9.24, 10.17, 12.54, 18.44] |
| 7 | 863 | [0.98, 1.78, 2.75, 4.82, 7.97, 9.89, 13.24, 19.69] |

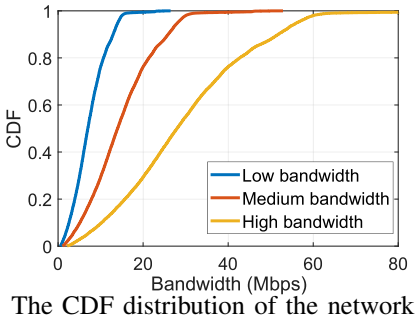


Fig. 5: The CDF distribution of the network traces.

With the collected traces, we evaluate and compare the following approaches.

- `Baseline`: The default 360° video streaming approach used in many video platforms, where the video segments are downloaded at their highest possible bitrate under the current network condition, and processed using the default CPU governor on mobile devices (i.e, using all CPU cores).
- `DefFreq`: The video segments are downloaded at their highest possible bitrate under the current network condition, and the video is processed using only the little cores. The CPU frequency is determined by the default CPU governor, which tends to set the CPU at high frequency.
- `AdaFreq`:The video segments are downloaded at their highest possible bitrate. The CPU frequency is determined based on the video resolution, i.e., for a specific video quality, the CPU frequency is set to be optimal, i.e., can minimize the energy consumption according to the power models shown in Section III-B.
- `EQA`: The proposed Energy and QoE Aware (EQA) 360° video streaming algorithm, which selects the right video bitrate and CPU frequency for each video segment such that the energy is minimized and the QoE is maximized.
- `Optimal`: The optimal requires complete knowledge of all future tasks. Note that it is impossible to achieve the optimal in practice, and it only provides a performance upper bound for comparison purposes.

### B. Energy Comparisons

The energy consumed for video streaming consists of the energy consumed for video processing and video downloading. For *EQA* and *Optimal*, the power models presented in Section III-B are used to calculate the energy of video processing. For *Baseline*, *DefFreq*, and *AdaFreq*, Table V is used, which shows the measured power consumption when different quality 360° videos are processed. The energy of video downloading for these approaches is calculated based on the power model of the wireless interface in Table III.

TABLE V: Power consumption (mW) for processing 360° videos using different approaches.

| Type | 144p | 240p | 360p | 480p | 720p | 1080p | 1440p | 2160p |
|---|---|---|---|---|---|---|---|---|
| *Baseline* | 586.8 | 614.5 | 623.9 | 694.9 | 728.7 | 808.3 | 878.5 | 987.6 |
| *DefFreq* | 448.3 | 491.8 | 529.2 | 570.3 | 586.6 | 647.6 | 681.8 | 745.7 |
| *AdaFreq* | 282.1 | 323.6 | 371.8 | 392.7 | 429.1 | 467.2 | 518.5 | 622.2 |

In Figure 6, we compare the energy consumption of different approaches under various network conditions, i.e., with *low*, *medium*, and *high* bandwidth traces. In general, our EQA algorithm has very low energy consumption which is very close to *Optimal*, and significantly outperforms other approaches.

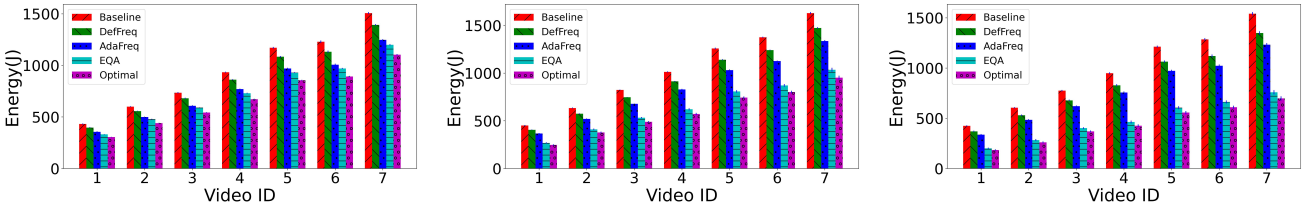
As shown in Fig. 6, *Baseline* consumes the highest amount of energy, since it downloads video using the highest possible bitrate and processes video using the default CPU governor. The default CPU governor activates all CPU cores across different processor clusters, and thus consumes more energy. By only activating the little cores, *DefFreq* can save energy compared to *Baseline*. Compared to *DefFreq*, *AdaFreq* can further save energy by adjusting the CPU frequency based on the video quality. *EQA* significantly outperforms other approaches, since it selects the right video bitrate and CPU frequency to minimize energy and maximize QoE.

From Figure 6, we can see that the energy saving increases when the network bandwidth increases, especially for *EQA*. For example, for video 7, as the network bandwidth increases from *low* to *high*, the energy saving of *EQA* compared to *Baseline* increases from 20.4% to 50.8%. This is because, when the network bandwidth is high, *Baseline* downloads video at very high bitrate which leads to more energy consumption for video downloading, and processes the video at very high CPU frequency which leads to higher energy consumption for video processing. In contrast, *EQA* selects the right video bitrate and CPU frequency to minimize energy and maximize QoE.

Figure 7 shows the overall energy saving compared to the *Baseline* approach. Since the *Baseline* approach has the highest energy consumption, all four approaches can further save energy. The proposed *EQA* can save similar amount of energy as the optimal approach, and both perform much better than *DefFreq* and *AdaFreq*. Specifically, the *EQA* approach can save energy by 21.0%, 37.0%, and 50.7% on average when the network bandwidth is *low*, *medium*, and *high*, respectively.

*Video Downloading vs. Video Processing:* To further analyze the energy consumption of different components, we divide it into two parts: energy consumed for video downloading and video processing. The energy consumed for downloading is calculated as the power of the wireless interface multiplies the time duration of video downloading. The energy consumed for processing is calculated as the video processing power multiplies the length of the video.

In Figure 8, we compare the energy consumed for video downloading and video processing by different approaches, using video 1 as an example. For video downloading, *Baseline*, *DefFreq*, and *AdaFreq* download the highest possible bitrate under the current network conditions, and thus consume much more energy than *EQA*. *EQA* consumes less energy with




(a) Low bandwidth (b) Medium bandwidth (c) High bandwidth

Fig. 6: Energy consumption for videos under different bandwidth traces.

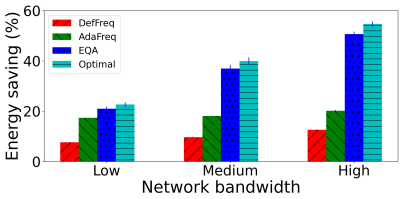



Fig. 7: Energy saving over all videos (Samsung S20).

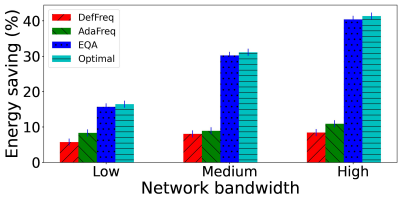
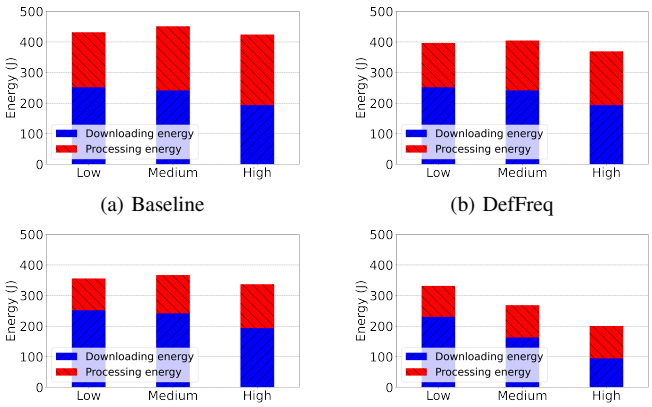



Fig. 9: Energy saving over all videos (Pixel 6).




(a) Baseline (b) DefFreq

(c) AdaFreq (d) EQA

Fig. 8: Energy consumed for downloading and processing video 1.

high bandwidth trace than that with low bandwidth trace. This is because the average downloading data rate in the high bandwidth trace is four times higher than that in the low bandwidth trace. Then, the downloading time, and the downloading energy, can still be reduced even when higher bitrate video is downloaded.

For video processing, *Baseline* consumes the highest amount of energy, because it has to process a large amount of data when downloading high quality video, and process the video using all cores across different processor clusters. Compared to *Baseline*, *DefFreq* can save energy since it uses only the little cores for video processing. Compared to *DefFreq* which always sets the CPU at high frequency, *AdaFreq* can adaptively adjust the CPU frequency based on the video quality and then save energy. *EQA* significantly outperforms other approaches, since it considers minimizing energy and maximizing QoE for determining the video bitrate for video segments, and adaptively adjusts the CPU frequency for processing the video.

*Energy Consumption for Other Phones:* We also conducted experiments using other Android based phones such as Pixel devices. Figure 9 shows the energy saving of various approaches compared to the *Baseline* approach using Google Pixel 6, which adopts the tri-cluster processor architecture similar to Samsung S20, consisting of little cores, middle cores and big cores. As shown in the figure, by leveraging the proposed optimization techniques, energy can be significantly reduced for 360° video streaming on Pixel 6. Specifically, the overall energy saving for the *EQA* approach is 15.7%, 30.2%, and 40.4% on average when the network bandwidth is *low*, *medium*, and *high*, respectively. We can also see that the proposed *EQA* can save similar amount of energy as the optimal approach, and both perform much better than *DefFreq* and *AdaFreq*, especially when the network bandwidth is high.

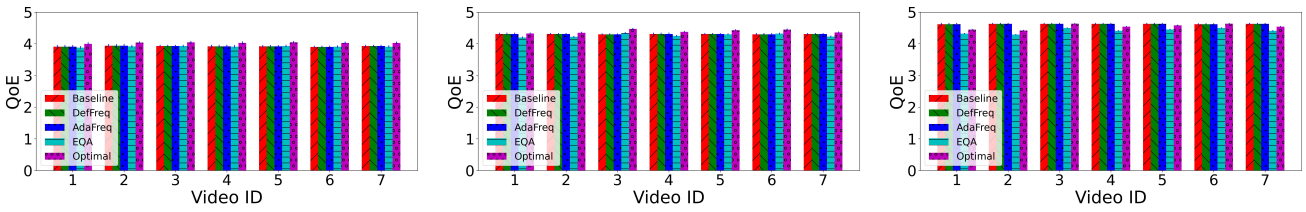
### C. QoE Comparisons

In Figure 10, we compare the QoE of all approaches under different network traces. As shown in the figure, our *EQA* algorithm can achieve very high QoE, and the QoE gap between *EQA* and others is very small.

Figure 11 draws the overall QoE degradation of different approaches compared to *Baseline*. In general, the QoE degradation for *EQA* is very small (almost zero), especially for *low* bandwidth traces where all approaches request videos encoded at low bit rate. When network conditions become better, for example, for *high* bandwidth traces, *Baseline* downloads videos at the highest possible bit rate, while *EQA* determines the right video bit rate and CPU frequency for video segments to minimize energy and maximize QoE. From Figure 11, we can see that the average QoE degradation for *EQA* is 0.2%, 1.1%, and 4.6% for *low*, *medium*, and *high* bandwidth traces, respectively. Thus, our *EQA* approach significantly saves energy (i.e., 50.7% for *high* bandwidth traces) at the cost of very small QoE degradation (i.e., 4.6% for *high* bandwidth traces).

## VI. Related Work

**Tiled-based 360° Video Streaming**. There has been considerable research on tile-based 360° video streaming [4, 5,




(a) Low bandwidth (b) Medium bandwidth (c) high bandwidth

Fig. 10: QoE when watching videos under different bandwidth traces.

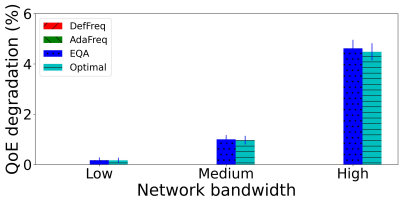



Fig. 11: QoE degradation over all videos.

11, 12, 21, 24, 25]. By cutting the video segment into tiles, only the tiles covering users' viewing area are downloaded with high quality, and other tiles are downloaded with low quality or not delivered at all to save bandwidth. Although tile-based 360° video streaming can improve QoE with limited wireless bandwidth, processing many small tiles consume a large amount of energy on mobile devices. This is because applying many concurrent decoders to accelerate video decoding makes the video decoding pipeline much complex, which leads to high computational overhead and thus high energy consumption on mobile devices. In this paper, for performance evaluations, we consider the 360° video format similar to that used in common 360° video platforms such as YouTube [1] and Facebook [2], where the video is encoded at different resolutions matching the corresponding video qualities. Note that our algorithms can also be applied to other video formats such as tile-based 360° video streaming, since the pipeline for processing tile-based 360° video is the same as that for general 360° video streaming, except that the dedicated hardware decoder will have to handle the tasks of decoding multiple tiles.

**Energy Consumption**. There are a number of studies on saving energy for video streaming in mobile environments. One way is to reduce the power consumed by the wireless interface when streaming video from the video server [26, 27]. For instance, Hu *et al.* [26] proposed techniques to reduce energy consumption based on whether users will stick on watching the video, skip video watching or abandon watching early. Wu *et al.* [27] proposed techniques to save energy for video streaming in heterogeneous networks. Other researchers studied how to save energy for video processing on the client devices [14, 10, 28]. In [14], an adaptive CPU frequency adjustment approach was proposed to reduce the energy consumption of video streaming. In [10], the environment factor (vibration or shaking impact) during video streaming is leveraged to design video bitrate adaptation algorithms to save energy. In [29], considering the distance between the user's eyes and the screen, a resolution dynamic scaling method is proposed to reduce energy consumption. In [28], the brightness of the display screen is dynamically adjusted to save energy.

Recently, some researchers start to look into energy efficiency issues in 360° video streaming or virtual reality applications [30, 31]. In [30, 32], the authors proposed to save energy by reducing the less important video frames in each segment. In [31], energy is saved by reusing pixel values to reduce the computation overhead during video rendering. However, none of them considers to save energy by leveraging today's heterogeneous multicore architecture on mobile devices, which is the focus of this paper.

## VII. Conclusions

In this paper, we identified the energy inefficiency problem of 360° video streaming on mobile devices, and proposed energy efficient 360° video streaming algorithms. Based on extensive experiments, we found that existing systems activate all CPU cores during video streaming, which consumes a large amount of energy, but it is unnecessary since most heavy computations in 360° video processing are handled by the hardware accelerators such as hardware decoder, GPU, and display processing unit. We also found that the default CPU governor sets the CPU at high frequency even when processing low resolution videos, which consumes more energy. To save energy, we propose to selectively activate the proper processor cluster and adaptively adjust the CPU frequency based on the video quality. We modeled the impact of video resolution and CPU frequency on power consumption, and modeled the impact of video features and network effects on QoE. Then, we formulated the energy and QoE aware 360° video streaming problem as an optimization problem, and proposed an optimal solution. Since the optimal solution is impossible in practice, we further proposed a heuristic based algorithm. Through extensive evaluations, we demonstrated that the proposed *EQA* algorithm can dramatically reduce the energy consumption (e.g., 21% for *low* bandwidth and 50.7% for *high* bandwidth traces) with very small QoE degradation (e.g., 0.2% for *low* bandwidth and 4.6% for *high* bandwidth traces).

## VIII. Acknowledgments

This work was supported in part by the National Science Foundation under grants 2215043 and 2125208.

## REFERENCES


[1] Google. YouTube Live in 360 Degrees Encoder Settings, July 2022. https://support.google.com/youtube/answer/6396222.

[2] Facebook. Facebook 360, July 2022. https://facebook360.fb.com/.

[3] Comparison of Virtual Reality Headsets. https://en.wikipedia.org/wiki/Comparison_of_virtual_reality_headsets.

[4] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *ACM MobiCom*, 2018.

[5] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-Degree Video Streaming for Smartphones. In *ACM MobiSys*, 2018.

[6] Oculus. https://www.oculus.com/.

[7] HTC Vive. https://www.vive.com/.

[8] Google Cardboard. https://arvr.google.com/cardboard.

[9] Samsung Gear VR. https://www.samsung.com/global/galaxy/gear-vr/.

[10] X. Chen, T. Tan, G. Cao, and T. La Porta. Context-Aware and Energy-Aware Video Streaming on Smartphones. *IEEE Trans. on Mobile Computing*, March 2022.

[11] Y. Zhang, P. Zhao, K. Bian, Y. Liu, L. Song, and X. Li. DRL360: 360-Degree Video Streaming with Deep Reinforcement Learning. In *IEEE INFOCOM*, 2019.

[12] X. Chen, T. Tan, and G. Cao. Popularity-Aware 360-Degree Video Streaming. In *IEEE INFOCOM*, 2021.

[13] K. Yamagishi and T. Hayashi. Parametric Quality-Estimation Model for Adaptive-Bitrate-Streaming Services. *IEEE Trans. on Multimedia*, February 2017.

[14] Y. Yang, W. Hu, X. Chen, and G. Cao. Energy-Aware CPU Frequency Scaling for Mobile Video Streaming. *IEEE Trans. on Mobile Computing*, November 2019.

[15] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming With FESTIVE. *IEEE/ACM Trans. on Networking*, February 2014.

[16] A. H. Zahran, D. Raca, and C. Sreenan. ARBITER+: Adaptive Rate-Based Intelligent HTTP Streaming Algorithm for Mobile Networks. *IEEE Trans. on Mobile Computing*, December 2018.

[17] C. Yue, R. Jin, K. Suh Y. Qin, B. Wang, and W. Wei. Linkforecast: Cellular Link Bandwidth Prediction in LTE Networks. *IEEE Trans. on Mobile Computing*, July 2018.

[18] Gerui Lv, Qinghua Wu, Weiran Wang, Zhenyu Li, and Gaogang Xie. Lumos: towards Better Video Streaming QoE through Accurate Throughput Prediction. In *IEEE INFOCOM*, 2022.

[19] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang. Pano: Optimizing 360 Video Streaming with a Better Understanding of Quality Perception. In *ACM SIGCOMM*, 2019.

[20] A. Mahzari, A. T. Nasrabadi, A. Samiei, and R. Prakash. FoV-Aware Edge Caching for Adaptive 360° Video Streaming. In *ACM Int'l Conf. on Multimedia*, 2018.

[21] C. Zhou, M. Xiao, and Y. Liu. ClusTile: Toward Minimizing Bandwidth in 360-Degree Video Streaming. In *IEEE INFOCOM*, 2018.

[22] M. Xiao, S. Wang, C. Zhou, L. Liu, Z. Li, Y. Liu, and S. Chen. MiniView Layout for Bandwidth-Efficient 360-Degree Video. In *ACM Int'l Conf. on Multimedia*, 2018.

[23] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. Rondao Alface, T. Bostoen, and F. De Turck. HTTP/2-Based Adaptive Streaming of HEVC Video over 4G/LTE Networks. *IEEE Communication Letters*, November 2016.

[24] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang. Flocking-based Live Streaming of 360-Degree Video. In *ACM Multimedia Systems Conference (MMSys)*, 2020.

[25] Lei Zhang, Yanyan Suo, Ximing Wu, Feng Wang, Yuchi Chen, Laizhong Cui, Jiangchuan Liu, and Zhong Ming. TBRA: Tiling and Bitrate Adaptation for Mobile 360-Degree Video Streaming. In *ACM Int'l Conf. on Multimedia (ACMMM)*, 2021.

[26] W. Hu and G. Cao. Energy-Aware Video Streaming on Smartphones. In *IEEE INFOCOM*, 2015.

[27] J. Wu, B. Cheng, M. Wang, and J. Chen. Energy-Efficient Bandwidth Aggregation for Delay-Constrained Video over Heterogeneous Wireless Networks. *IEEE J. Selected Areas in Communications*, January 2017.

[28] Z. Yan and C. W Chen. RnB: Rate and Brightness Adaptation for Rate-Distortion-Energy Tradeoff in HTTP Adaptive Streaming over Mobile Devices. In *ACM MobiCom*, 2016.

[29] S. He, Y. Liu, and H. Zhou. Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling. In *ACM MobiCom*, 2015.

[30] X. Chen and G. Cao. Energy-Efficient and QoE-Aware 360-Degree Video Streaming on Mobile Devices. In *IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2022.

[31] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Déja view: Spatio-temporal compute reuse for 'energy-efficient 360 vr video streaming. In *ACM/IEEE Int'l Symposium on Computer Architecture (ISCA)*, 2020.

[32] X. Chen, T. Tan, and G. Cao. Macrotile: Toward QoE-Aware and Energy-Efficient 360-Degree Video Streaming. *IEEE Trans. on Mobile Computing*, To appear.