Proactive Energy-Aware Adaptive Video Streaming on Mobile Devices

Jiayi Meng, Qiang Xu, and Y. Charlie Hu, *Purdue University*

https://www.usenix.org/conference/atc21/presentation/meng

This paper is included in the Proceedings of the
2021 USENIX Annual Technical Conference.

July 14–16, 2021

978-1-939133-23-6

Open access to the Proceedings of the
2021 USENIX Annual Technical Conference
is sponsored by USENIX.

Proactive Energy-Aware Adaptive Video Streaming on Mobile Devices

Jiayi Meng
Purdue University

Qiang Xu
Purdue University

Y. Charlie Hu
Purdue University

Abstract

Energy-aware app adaptation enables mobile apps to dynamically adjust data fidelity such as streaming video quality to meet a user-specified goal for battery duration. Traditional energy-aware app adaptation is reactive in nature where the operating system monitors the app energy drain and signals the app to adapt upon detecting energy drain deviation from the pre-specified energy budget which can cause high oscillation and poor quality-of-experience (QoE).

In this paper, we observe that modern power-hungry apps such as video streaming and offloading-based apps already come with sophisticated app adaptation to deal with resource changes such as network dynamics and propose proactive energy-aware adaptation where the user-specified energy budget is integrated with the app adaptation logic. The potential benefit of such an approach is that app energy drain adaptation is no longer an "after-effect", and hence the approach is likely to reduce the oscillation in app adaptation and improve the app QoE.

In this paper, we study the design, implementation and performance tradeoffs of reactive and proactive energy-aware app adaptation in the context of one of the most power-hungry classes of mobile apps, ABR-based video streaming. Our study shows that proactive energy-aware ABR video streaming is easy to implement by leveraging the built-in adaptation of modern apps and can improve the QoE of reactive approach by 44.8% and 19.2% in streaming 360^o videos to Pixel 2 and Moto Z3 phones under low power budget.

1 Introduction

For enriched user experience, modern mobile apps utilize a large number of power-hungry hardware components such as the CPU, GPU, WiFi and 4G, and hardware decoder and as a result draw significant amount of power. As a result, the user experience of such feature-rich apps is often limited by the shortened battery life from increased power draw [7, 8].

Energy-aware app adaptation [31, 88] exploits a key observation that many apps can reduce their power draw by reducing their data fidelity such as the size and quality of a video in video streaming apps or the filtering level of a map in navigation apps, and enables apps to dynamically adjust their data fidelity to meet a user-specified goal for battery duration, *e.g.,* a four-hour plane ride.

There are two components in building an energy-aware app adaptation system: energy accounting and control. First, an energy accounting subsystem is needed to monitor the energy drain during the app execution and detect any significant deviation, *e.g.,* exceeding a threshold, from the user-specified energy budget. Second, the app needs to implement some adaptation logic to stay within the energy drain budget or correct energy drain deviation.

Traditional energy-aware app adaptation schemes such as PowerScope [31], ECOSystem [88], Cinder [68], and Nemesis [62] have mainly focused on system-level energy accounting and control, treating the app as a "black-box". In particular, in such systems, the operating system (OS) monitors the app energy drain during app execution, and upon detecting energy drain deviation from the pre-specified budget either throttles the execution of the app process or threads or issues an upcall to the app to trigger app reactive adaptation. The benefit of such a black-box approach is simplified app implementation, as real-time energy accounting is provided as an OS service. The downside of such an approach is that the disintegrated and reactive nature prevents jointly optimizing the app QoE and meeting the energy drain budget. In particular, correcting energy drain deviation as an after-effect can lead to app fidelity oscillation and negatively affect the QoE.

In this paper, we make a key observation that compared to the mobile apps studied two decades ago [31], mobile apps today not only are more power hungry, but also often come with sophisticated adaptation built in to optimize the user-perceived QoE in reaction to network dynamics or other system constraints. For example, fidelity adaptation such as adaptive bitrate (ABR) is now widely adopted in video streaming systems [15, 38, 45–47, 54, 59, 61, 65, 69, 72–74, 76, 83, 85], and adaptive offloading of computation to edge servers has been proposed for deep learning enhanced tasks such as video analytics [21, 37, 51, 57, 66].

We argue that the built-in QoE optimization frameworks in such modern mobile apps naturally lend themselves to *integrated, proactive* energy-aware app adaptation where the energy drain budget is seamlessly integrated into the pre-existing QoE adaptation as a constraint. The key benefit of such an integrated, proactive approach is that app energy drain adaptation is no longer an after-effect and hence likely to reduce the oscillation in app adaptation and improve the app QoE. Compared to the reactive approach, the integrated approach faces two design challenges: (1) the app needs to predict the power consumption for each adaptation candidate beforehand, and (2) the app needs to incorporate the energy budget into its QoE optimization logic.

In this paper, we study the design, implementation, and performance tradeoffs of reactive and proactive energy-aware

app adaptation in the context of one of the most power-hungry classes of mobile apps, ABR-based video streaming [41,79,87,89]. We focus on a state-of-the-art ABR scheme, Robust MPC [85], that employs receding horizon control [19] to maximize the QoE for the next few video chunks based on the predicted network throughput in the moving horizon of video chunk downloading intervals.

We design Energy-aware ABR, an energy-aware ABR video streaming system to demonstrate how to address the two challenges in designing proactive energy-aware app adaptation. First, proactive energy-aware app adaptation requires a power predictor that can *predict* the average power draw of the streaming app in the next time interval in fetching a chunk of any candidate chunk quality. Traditional mobile device power models [22,23,26,70,87,91] cannot be used as they take component utilization logged as input which are not available *before* app execution. We propose a novel function-level power modeling methodology that accurately predicts the app energy drain in the next interval.

Second, integrating an energy budget into the app adaptation logic, in particular, the model predictive control (MPC)-based QoE optimizer for MPC-based ABR, faces a unique challenge. Unlike other constraints such as network throughput in the QoE optimization problem, *app energy drain is cumulative and hence elastic over time.* At any time interval during a streaming session, the app may have accumulated energy drain surplus or deficit from past time intervals, *e.g.,* from selecting low chunk formats limited by transient low network bandwidth. We explore several design options on how to integrate such energy surplus/deficit with the QoE optimization framework of MPC-based ABR controllers.

Since MPC-based controllers are notoriously hard to analyze [17,34,67], we evaluate the end-to-end performance of reactive and proactive energy-aware ABR video streaming designs using testbed experiments. We have implemented both designs on top of Puffer [11], an open-sourced video streaming platform, and evaluated different design options by streaming 360^o videos from a media server to a mobile client, while varying the network condition using network traces from two large datasets, YTrace and FCC [4].

Our evaluation results show that (1) Our function-wise power predictor achieves low mean per-interval (2-second) energy prediction error of 4.87% (Pixel 2) and 5.86% (Moto Z3). (2) Under dummy power budget, *i.e.,* using the average power draw of the energy-oblivious ABR as the power budget, both proactive and reactive Energy-aware ABR achieve only slightly lower QoE than that of the energy-oblivious ABR, with proactive Energy-aware ABR achieving slightly higher QoE than reactive Energy-aware ABR. (3) Under low power budget, proactive Energy-aware ABR improves the QoE by 44.8% (Pixel 2) and 19.2% (Moto Z3) over reactive Energy-aware ABR. (4) The majority of the improvement comes from significantly reduced video quality variation component of the QoE, of 85.2% (Pixel 2) and 87.4% (Moto Z3), showing that proactive Energy-aware ABR can effectively mitigate the oscillation drawback of reactive energy-aware adaptation.

In summary, this paper makes the following contributions:

- We show prior reactive energy-aware app adaptation can lead to app fidelity oscillation which can negatively affect user-perceived QoE.
- We propose to our knowledge the first proactive energy-aware app adaptation and show that it can be easily implemented by integrating user-specified energy budget with the built-in app adaptation logic of modern apps such as MPC-based ABR systems for video streaming.
- We present a novel function-wise power predictor that can be used for what-if power draw analysis needed in proactive energy-aware app adaptation, *e.g.,* Energy-aware ABR.
- We experimentally compare the end-to-end performance of reactive and proactive energy-aware adaptive streaming of 360^o videos on Pixel 2 and Moto Z3 phones.
- Our results show proactive energy-aware video streaming improves the QoE by 44.8% (Pixel 2) and 19.2% (Moto Z3) over the reactive approach under low power budget.

To further the research on energy-aware app adaptation, we have open-sourced Energy-aware ABR implementation.[1]

2 Motivation

To motivate how energy-aware adaptation can help to reduce app power draw and elongate battery duration, we performed a measurement study of one of the most power-hungry classes of apps, video streaming, with example apps such as Youtube and Netflix consistently ranked among the top 10 battery draining apps [8, 12, 13].

Experimental Setup. We streamed 6 popular panoramic videos in full screen mode in the Youtube app on a Pixel 2 phone, which was connected to a Monsoon power monitor to measure the total phone power draw during video streaming. The videos were categorized into 3 groups, slow, medium and high, based on the moving speed of the camera. Each video was encoded into different resolutions and frame rates, including (4K, 60FPS), (1440p, 60FPS), (1080p, 60FPS), (720p, 60FPS), (480p, 30FPS) and (360p, 30FPS). Table 1 shows the bitrates of the videos at different quality settings.

We streamed each video in Youtube at different qualities with and without screen on, respectively. When the screen was on, the brightness level was set at 60%. To prevent the app from directly retrieving video frames from the cache rather than through the Internet, we cleared the cache of Youtube app on the phone before each experiment.

To understand the power breakdown of 360^o video streaming, we built component-wise power models [22, 26, 44, 75,

[1] https://github.com/meng72/Proactive-Energy-Aware-Adaptive-Video-Streaming

Table 1: Average bitrate (Mbps) of 6 panoramic videos at different resolutions and frame rates from Youtube.

Videos	4K/60	1440p/60	1080p/60	720p/60	480p/30	360p/30
Slow						
Planets [1]	18.10	6.20	2.06	1.01	0.35	0.19
Solar [6]	22.10	5.38	1.70	0.76	0.22	0.10
Medium						
Orleans [5]	25.70	12.00	3.72	2.13	0.61	0.33
Chicago [9]	25.60	10.60	3.41	1.93	0.58	0.31
High						
Monster [2]	26.20	12.60	4.20	2.49	0.71	0.39
Optical [10]	24.40	12.60	4.16	2.47	0.71	0.39

81, 87] for major hardware components in the Pixel 2 phone, profiled hardware component usage while streaming 360^o videos at (4K, 60FPS) in Youtube on Pixel 2, and finally estimated the corresponding power draw for each hardware component usin the power models.

Findings. Figure 1 shows the average power consumption of streaming the 6 panoramic videos at different quality settings over the Internet in Youtube. We derived the screen power by subtracting the total power of video streaming when the screen was off from that when the screen was on. We see that (1) 360^o video streaming is power-intensive on modern mobile devices. Streaming videos at (4K, 60FPS) consumes the highest total power of 579.73mA.[2] At this rate, a fully charged Pixel 2 phone (with a 2700 mAh battery) will drop to 15% in 3 hours and 40 minutes when the Battery Saver mode will turn on. (2) Thanks to the OLED technology, the power consumption of the display is relatively small, only 41.66−50.36mA, or 7.0%−12.0% of the total power across different settings. (3) Reducing the data fidelity, *i.e.,* the video resolution and frame rate, can significantly lower the total app power draw. For example, the total power draw decreases by 33.1% from 579.73mA to 387.79mA when the video quality changes from (4K, 60FPS) to (360p, 30FPS).

Figure 2 shows the power breakdown of streaming 360^o videos at (4K, 60FPS). We see that (1) the CPU, GPU and screen consume relatively low power, of 82.11mA (14.0%), 74.49mA (12.7%) and 46.92mA (8.0%), respectively; (2) in contrast, the NIC (network interface card) and hardware decoder, two primary hardware components involved in video streaming, dominate the total power draw of streaming, *i.e.,* 220.53mA (37.6%) and 162.47mA (27.7%), respectively. These results suggest that adapting the data fidelity of video chunks to control the power draw of networking and decoding will be effective in controlling the total power draw of the app.

3 Prior Work on Energy-aware App Adaptation

The large body of research on managing the energy drain of applications on mobile systems has mainly focused on

[2] In this paper, for power measurement we directly report the current drawn in milli-Amperes (mA); the actual power consumed would be the current drawn multiplied by 3.7V, the voltage supply of the battery. The smartphone batteries are rated using these metrics and hence are easy to cross reference.

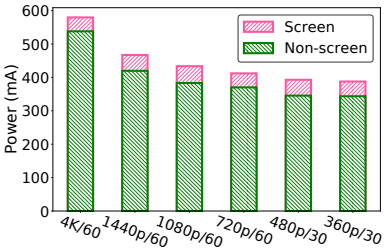


Figure 1: Average power consumption of streaming 360^o videos with different qualities in Youtube on Pixel 2.

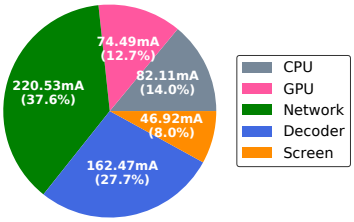


Figure 2: Power breakdown of streaming 360^o videos at 4K/60FPS on Pixel 2.

system-level energy accounting and control, treating the app as a "black-box".

Since managing the energy drain of apps requires accurately monitoring the energy drain during application execution, a large body of work proposed solutions to the resource container-level, process-level, or thread-level energy accounting problem in the OS, such as PowerScope [32], ECOSystem [88], Cinder [68], and Nemesis [62]. In a nutshell, energy accounting in these systems is achieved via either aligning external power measurement with interrupt-triggered program sampling as in early systems such as PowerScope [32] and Quanto [33], or using a pre-trained power model that captures the correlation between utilization of each hardware component in each of its power states and the resulting power draw and feeding the power model with hardware component usage logged during app execution to estimate the app energy drain, *e.g.,* in ECOSystem [88] and Cinder [68].

Upon detecting that an app's energy drain has exceeded a predetermined budget, the system needs a way to throttle the app's energy drain. In ECOsystem [88], Currentcy [88], and Cinder [68], the kernel would enforce energy budget by halting or throttling app threads, processes, or resource containers from execution. The Odyssey extension [31] and Nemesis [62] do not throttle applications, but issue upcalls or provide feedbacks to the applications to trigger fidelity adaptation to adjust their energy drain rate.

4 Reactive vs. Proactive Energy-aware App Adaptation

We discuss the drawbacks of prior reactive energy-aware app adaptation and propose proactive energy-aware app

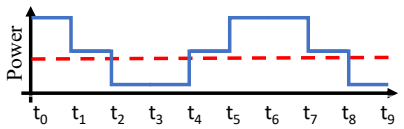


Figure 3: Reactive approach causes power oscillation.

adaptation.

Reactive energy-aware app adaptation. The prior system-level app energy control solutions are *reactive* and *disintegrated*. They are reactive because they treat applications as black-boxes and passively monitor their energy drain, and inform apps of the need to perform adaptation reactively upon detecting any deviation of the app energy drain from the pre-specified budget. They are disintegrated because the two tasks are performed in isolation: the OS monitors the app energy drain while the app performs adaptation.

The benefit of such a reactive approach is simplified app implementation, as real-time energy accounting can be provided as an OS service and the apps can focus on reactive adaptation, although fine-grained model-based energy monitoring relies on collecting fine-grained hardware component usage which can incur high runtime overhead.

The downside of a reactive approach is that the disintegrated and reactive nature deprives the opportunity of jointly optimizing the app QoE and meeting the energy drain budget at each time interval. In particular, the app not only performs adaptation *after* deviating from the energy drain target, but also typically does not have specific guidance on *how much* app fidelity to adapt in the next time interval, which can result in power draw and app fidelity oscillation.

Figure 3 shows an example of how the power draw of a disintegrated, reactive app adaptation scheme can cause oscillation in app power draw. Assume an app has three fidelities, high, medium, and low, drawing correspondingly high, medium, and low power. An energy-aware OS like Odyssey [31] tries to steer the app towards a target power budget specified in the dashed line to ensure a target battery duration. The app starts running in high fidelity, drawing high power. At t_1, the OS performs an upcall informing the app of an energy drain deficit, and the app lowers the fidelity to medium. At t_2, the OS informs the app of energy drain deficit again as the average power is still above the budget, and the app lowers the fidelity to low. The trend reverses in the next three intervals, and the oscillation would continue as a result of the reactive correction without concrete guidance on app adaptation due to the disintegrated approach.

Proactive energy-aware app adaptation. Our proposal of proactive energy-aware app adaptation is motivated by the above drawbacks of the reactive approach and an observation about modern mobile apps. Compared to the mobile apps studied two decades ago (*e.g.,* [31]), mobile apps today are more power hungry, but also often come with sophisticated proactive adaptation built in to optimize the user-perceived QoE under network dynamics or other system constraints. For example, proactive fidelity adaptation such as adaptive bitrate (ABR) (*e.g.,* DASH) is now a standard feature in regular video streaming systems [15, 45, 46, 54, 59, 61, 72, 73, 76, 83, 85] as well as 360^o video streaming systems [38, 47, 65, 69, 74]. Similarly, adaptive offloading of computation to an edge server according to the network dynamics has been proposed in many systems [21, 37, 51, 57, 66]. More recently, adaptive offloading of machine learning inference has been proposed to adaptively offload a subset of DNN layers from the mobile device to the edge server [30, 49, 52, 92].

Motivated by the above two observations, we argue that the built-in QoE optimization frameworks in many modern mobile apps lend themselves to integrated, proactive energy-aware app adaptation that potentially overcomes the oscillation drawback of reactive approaches. In such an approach, at every step of the QoE optimization for selecting the quality of the chunk to be fetched in the next time interval, the power budget is explicitly taken into consideration so that the QoE optimization will directly output the optimal chunk format that maximizes the QoE for the next chunk and satisfies the energy budget in the next interval.

The key benefit of such an integrated, proactive approach is that app energy drain adaptation is no longer an after-effect correction and hence likely to reduce the oscillation in app adaptation. The challenge of such an approach is that each app needs to (1) predict the power consumption for each adaptation candidate beforehand, and (2) incorporate the energy budget in its QoE optimization logic.

In this paper, we investigate the design and performance tradeoffs of reactive and proactice energy-aware app adaptation in the context of one of the most power-hungry classes of mobile apps – ABR-based video streaming.

5 Energy-aware ABR

We briefly review state-of-the-art ABR algorithms and state the energy-aware QoE maximization problem.

Background on ABR. Adaptive bitrate (ABR) algorithms embody a primary technique of streaming videos over the Internet [71], where each video is encoded into multiple "tracks" with different quality bitrates and each track is segmented into "chunks" (*e.g.,* 2-second each). These algorithms aim at optimizing the video QoE by dynamically selecting which chunk to fetch based on network conditions. Earlier ABR schemes were either "buffer-based" [45, 73], or "rate-based" which pivot on estimating available network throughput and finding a matching video bitrate [46, 76]. MPC [85] unifies the QoE objective of chunk k as a weighted sum of three key elements, (1) video quality, (2) video quality variation, and (3) stall time:

$$QoE_k = Q_k - \lambda |Q_k - Q_{k-1}| - \mu T_k \quad (1)$$

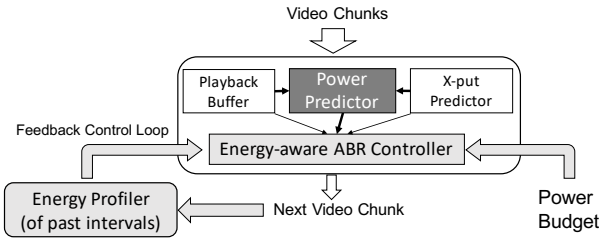


Figure 4: Architecture of energy-aware ABR. Lightly shaded components are new to both reactive and proactive designs, and the dark shaded one is new to the proactive design.

where Q_k represents the quality of video chunk k, T_k represents the stall time experienced by fetching chunk k and λ, and μ are weighting parameters for quality variation and rebuffering, respectively. Some papers (*e.g.,* [85]) quantify Q_k as the bitrate of chunk k, while others (*e.g.,* [83]) use perceptual quality metrics, *e.g.,* SSIM [77]. The ABR problem is then formulated as maximizing the QoE of all the chunks of a video downloaded in a streaming session:

$$\text{maximize} \sum_k QoE_i \text{, subject to buffer and network dynamics} \quad (2)$$

Since future throughput is unknown, practical algorithms like Robust MPC [85] employ receding horizon control [19] to maximize the QoE for the next few chunks (*e.g.,* 5 future chunks). Such algorithms take estimated network throughput for the next few intervals and client playback buffer occupancy as input and output the quality format for the next video chunk to be downloaded from the server while maximizing the QoE of next few chunks.

Energy-aware QoE maximization problem. We consider the canonical energy-aware app adaptation scenario in previous work [31,68,88], where the phone user specifies the total energy budget E_b (*e.g.,* 50% of batter level drop) over a fixed amount of time T_d (*e.g.,* duration of a train ride) and hence an average power target $P_b = E_b/T_d$. The energy-aware ABR problem is then formulated as maximizing the QoE of all the chunks of a video downloaded in a streaming session:

$$\text{maximize} \sum_k QoE_i \text{, subject to buffer and network dynamics and total energy constraint} \quad (3)$$

Energy drain is elastic. We note that in the above problem statement, the energy constraint is different from other constraints like network throughput in that *app energy drain is cumulative and elastic over time.* Since the user-specified energy budget is the total energy drain over a streaming duration, the streaming app may accumulate some surplus or deficit based on the energy drain so far during the streaming session. For example, due to network bandwidth fluctuation, there can be intervals during which the network bandwidth is low and the ABR algorithm is forced to pick low video quality formats which result in low power draw and hence low energy drain during those intervals. In such intervals, the app effectively accumulates an energy surplus, which can be spent in later intervals, *i.e.,* when the network bandwidth is high and high quality video chunks can be downloaded and played, to improve the total QoE.

Algorithm 1: Reactive Energy-Aware ABR – RA

Input : Power budget P_b; Energy draw E_{actual} over streaming time so far T; Selected format F_{k-1} for interval $k-1$; Average interval duration t
Output : Format F_k for next interval k

// if energy deficit, downgrade format
if $E_{actual} - P_b T > \gamma P_b t$ **then**
  $F_k = min(F_{k-1} - 1$, predicted format by ABR);
else $F_k =$ predicted format by ABR ;

6 Reactive Energy-Aware ABR Design

A reactive energy-aware ABR can be easily implemented by adding an energy profiler and adding reactive adjustment to the QoE optimizer output of the ABR controller, as shown in Figure 4. The energy profiler monitors the app energy drain and is decoupled from the ABR controller, either implemented in the OS as in [36,63,86] or in an app-agnostic library linked with the app. It sends an upcall to the ABR player either reactively upon detecting significant deviation of the predetermined energy budget or periodically to inform the app of its energy drain so far which is used by the ABR controller to monitor the energy surplus or deficit and to decide when and how to adapt. We assume the later approach which gives the ABR controller more flexibility.

In particular, the ABR controller accumulates a running energy drain balance as the difference between the expected energy drain so far $P_b \cdot T$ and the actual energy drain E_{actual} so far which is monitored by the energy profiler.

The goal of the reactive adjustment is to adjust the chunk format selected by the ABR QoE optimizer to try to correct the energy drain deviation from that according to the prespecified average power budget. To achieve this, the final chunk quality format is adjusted as no higher than the previous chunk format if there is an energy deficit greater than a threshold, as shown in Algorithm 1. We experimentally found a threshold of 10% of the energy budget to work well.

7 Proactive Energy-aware ABR Design

7.1 Integrated Energy-aware MPC Algorithm

Since an MPC-based ABR algorithm is proactive by nature, *i.e.,* it calculates the video chunk quality to be fetched next based on the predicted network throughput in the next time interval, a proactive energy-aware ABR can be naturally realized by integrating the energy constraint with a practical

MPC algorithm such as Robust MPC [85]. In particular, the new energy-aware MPC algorithm optimizes the worst-case QoE assuming that the throughput in the future can take any value in range $[\hat{C}_t, \hat{C}_t{}']$, by solving the following optimization problem at time t_k to derive the quality format for fetching the next chunk $F_k = f_{eampc}(F_{k-1}, B_{k-1}, \hat{C}_t)$:

$$max_{F_k,\ldots,F_{k+N-1}} \; min_{\hat{C}_t \in [\hat{C}_t, \hat{C}_t']} \quad \sum_{k}^{k+N-1} QoE_i$$

subject to buffer and throughput dynamics and

$$E_k + \ldots + E_{k+N-1} < N \cdot P_b \cdot \delta t \quad (4)$$

where $\hat{C}_t$ is the low bound in the predicted throughput range $[\hat{C}_t, \hat{C}_t{}']$, B_{k-1} is the buffer occupancy after downloading chunk $k-1$ and δt is the interval duration, *e.g.,* 2 seconds.

We note that since app energy drain is cumulative, converting the total energy constraint into a constant energy constraint per time interval can be conservative. As with MPC [85], we do not claim this energy-constrained MPC is necessarily the optimal control algorithm for the energy-constrained bitrate adaptation problem, but one that is practical and can leverage accurate network throughput prediction and power draw prediction in the near horizon.

7.2 Architecture Overview

Figure 4 (adding the power predictor) shows the architecture of our proposed integrated, proactive Energy-aware ABR. As with the original ABR, Energy-aware ABR derives the client-side playback buffer status and estimated network throughput from two generic modules of ABR, the playback buffer module and throughput predictor module, respectively. The energy profiler module estimates the energy drain in past intervals which is used to maintain the energy surplus/deficit.

To select the video format for the next video chunk, Energy-aware ABR uses a new power predictor to predict the average power draw of the next video chunk of each candidate format. Its controller then filters out the formats whose predicted energy drain exceeds the energy budget adjusted for energy surplus or deficit so far and chooses the one that maximizes the QoE stated in Equation 1.

The proactive Energy-aware ABR system design faces two challenges: (1) how to predict the power consumption in fetching future video chunks of different candidate formats in the power predictor? (2) how to incorporate the energy constraint into the MPC algorithm in the Energy-aware ABR controller to facilitate maximizing the QoE for the future chunks? We start with discussing our solution to (2).

7.3 Energy-aware QoE Maximization

The basic design for incorporating the expected energy drain for the N future intervals in Eqn. 4 is straight-forward. To exploit dynamic energy surplus/deficit, the Energy-aware ABR controller accumulates the energy surplus/deficit as in the reactive approach, which is then exploited by the QoE maximization module to maximize the QoE of future video chunks. We explore three design options for incorporating energy surplus/deficit into the QoE maximization.

Algorithm 2: Design option 3 – LA(N)+LB

Input : Power budget P_b; Energy draw E_{actual} over streaming time so far T; Energy surplus/deficit $E_s = P_b T - E_{actual}$; Current buffer level B_{k-1}; Predicted power array $P_k[f][N]$ for all the formats from next interval k to interval $k+N-1$; Interval duration δt

Output : Format F_k for next interval k

Call recursiveABR(0, 0, B_{k-1}, 0) to derive F_k;

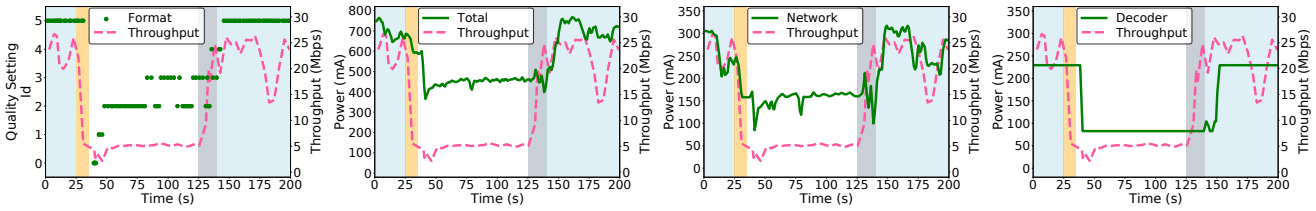
Function recursiveABR(n, F_{k+n-1}, B_{k+n-1}, E):

```
if n == N then
    if E > N · P_b · δt + E_s then return −∞ ;
    else return Quality of F_{k+n−1} ;
end
max_QoE = −∞;
for i = 0 to f do
    q = QoE between chunk n and n + 1;
    B = Buffer level after downloading chunk n + 1;
    E' = E + P_k[i][n] · δt;
    Q = q + recursiveABR(n + 1, i, B, E').QoE;
    if Q > max_QoE then
        F_{k+n} = i; max_QoE = Q;
    end
end
return < max_QoE, F_{k+n} >
```

(1) Look ahead 1 (LA(1)). The strawman design is to ignore energy surplus/deficit, and choose among all the candidate chunk formats with which the predicted app power draw in the next interval will not exceed the average power budget P_b, the one that maximizes the total QoE for the horizon (*e.g.,* 5 intervals). Such a design can be conservative in terms of QoE from not exploiting potential energy drain surplus accumulated in the past intervals when the network bandwidth fluctuates up and down.

(2) Look ahead 1 and look back (LA(1)+LB). Design option 2 extends LA(1) by exploiting the amount of energy surplus/deficit during past intervals. In selecting the video format for the new video chunk to fetch, the controller increases the energy budget for the next interval to $P_b \cdot \delta t + E_s$ (energy surplus/deficit). However, such a design may sacrifice video smoothness, as the energy surplus from past intervals may be large and allow some high quality chunk to be fetched in the next interval, followed by video chunks that go back to some low format.

(3) Look ahead N and look back (LA(N)+LB). To overcome the potential smoothness problem of LA(1)+LB, we extend it by allowing the energy surplus/deficit and the N-chunk energy budget to be spread over the next N chunks. Since the basic MPC already looks ahead N chunks in pick-



(a) Format selection (b) Total power (c) Network power (d) Decoder power

Figure 5: Network throughput, ABR decision, total and individual hardware component power in different network states (**H:** blue; **L:** white; **H-L:** orange; **L-H:** gray). Quality setting: 5: (4K, 60 FPS); 4: (1440p, 60 FPS); 3: (1080p, 60 FPS); 2: (720p, 60 FPS); 1: (480p, 30 FPS); 0: (360p, 30 FPS). The client buffer size is 7s, the same as 4K streaming on Youtube mobile app stated in §9.

ing the next chunk k to optimize the total QoE for them, allowing spreading the energy surplus over the N chunks can be easily incorporated in the modified MPC. In particular, Algorithm 2 adds the total energy $N \cdot P_b \cdot \delta t + E_s$ as the total energy constraint to the dynamic programming which will search among all possible ways to spread the energy surplus among the N intervals to find the one that gives the maximal total QoE for the N intervals. As in the basic MPC, optimizing the total QoE for the next N intervals will take smoothness in the next N chunks into consideration.

7.4 Function-wise Power Prediction

We next describe a practical and accurate function-wise power prediction methodology for use with any proactive energy-aware ABR adaptation such as Energy-aware ABR.

7.4.1 Why Function-wise Power Prediction?

The obvious choice of using traditional component-wise power models for mobile devices which have been well studied [22,23,26,29,44,64,70,75,81,87,91] does not work. Such a model derives the correlation between the utilization of each phone component in each of its power states, *e.g.,* utilization and operating frequency, and the resulting power draw using carefully designed microbenchmarks. To use such a model, the hardware component usage is logged *during* app execution and afterwards fed into the power model as input to estimate the component-wise power draw that happened during the app execution. Thus such traditional power models are *postmortem*; they are suitable for post-processing, *e.g.,* monitoring the actual energy drain of a past interval (or calculating the energy surplus/deficit) in reactive or proactive energy-aware app adaptation, but not for predicting the app energy drain in future intervals.

The other design choice is to treat the video streaming app as a black-box and measure and tabulate its average power draw offline when streaming all possible video bitrates under all possible network bandwidth. However, this is not practical as there are potentially infinite number of such combinations. More importantly, such an approach cannot easily model the power draw of *asynchronous component behavior* discussed below where different phone components, *e.g.,* the decoder and the network interface, can be processing different video chunks (of different bitrates) in a time interval, due to the playback buffer delay effect explained below.

Asynchronous Component Behavior. To illustrate the asynchronous component power behavior in video streaming, we profile the component power draw in an ABR-based 360^o video streaming session.

Our experimental setup is the same as in §2 except two differences for enabling power profiling of phone components. First, we build our own ABR server for 360^o video streaming with Robust MPC [85] as the default ABR algorithm, and implement our own mobile client using ExoPlayer [3]. Second, we derive the traditional component-wise power model for our experimental phone, Pixel 2, by running microbenchmarks and measuring the phone power draw using an external high-resolution Monsoon power monitor.

Figure 5 shows the profiling result. We see that the network bandwidth went through five stages: high (the H stage), high transitioning to low (the H-L stage), low (the L stage), low transitioning to high (the L-H stage), and finally high again. Figure 5a shows that the network bandwidth change causes the chunk format selected to change almost immediately, *e.g.,* from format 5 during the H stage dropping to format 1 during the H-L stage. However, Figure 5b shows that there is a lag of the total power draw change in following the network bandwidth change, mostly notably at 25-40s and 130-150s. To understand this lag, we zoom into the per-component power draw timeline.

The lag cannot be explained by the CPU and GPU power, both staying almost constant during the session (not shown due to page limit) regardless of the chunk format because of their constant load. The lag also cannot be explained by the network interface (NIC) power which, as shown in Figure 5c, is directly affected by the chunk format and closely follows the format change, *e.g.,* during the H stage and the L stage. In the L-H stage, the NIC power first goes up due to chunk size increase but then goes down as it spends an increasing fraction of the 2-second interval in the idle state as the network bandwidth goes up sharply.

Finally, Figure 5d shows that although in general the hardware decoder power in the H stage is higher than in the L stage because the chunk format and hence decoding load are higher in the H stage, the change of the decoder power

shows a prominent delay behind the network bandwidth change, *e.g.,* when the network bandwidth drops sharply around 24-40s and increases sharply around 130-150s. This explains the lag of the total power draw curve behind the network bandwidth curve. This happens due to the *buffer delay effect* which results in asynchronous power behavior of phone components: while the NIC is downloading the next chunk (*e.g.,* in a low format), the decoder decodes the video chunk at the head of the playback buffer (*e.g.,* in a high format) which was downloaded several intervals ago. The extent of the delay depends on the occupancy of the client buffer as well as network throughput.

7.4.2 Function-wise Power Prediction

The above asynchronous hardware component power behavior suggests that if we cluster the hardware components according to the common video chunk they process at each time interval, the components within each cluster will have synchronous power behavior, *i.e.,* which only depends on the properties of the chunk they process, and such power behavior can be modeled using a power predictor that only uses chunk properties as input. Semantically, each cluster typically corresponds to a high-level app function. We thus propose *function-wise power prediction* that models the power draw of each high-level app function.

In 360^o video streaming, there are two primary tasks: (1) *video decoding and displaying* which employs the CPU, GPU, hardware decoder, screen and game rotation vector sensor to process the same chunk in each time interval; (2) *network transmission* for fetching video chunks which involves the CPU and the network interface to process the same chunk (which is different from in task 1) in each interval. In function-wise power prediction, we build the power predictors for these two functions separately.

For the video decoding and displaying function, we make a key observation that the primary hardware components involved are the GPU and hardware decoder, and their power only depend on video properties such as resolution and frame rate but barely depend on video content (based on our measurements). Thus we should be able to develop a power prediction model using these video properties as input. We model the display power separately as a function of the brightness.[3] In particular, we measure the total power draw in playing pre-downloaded 360^o videos with the same six quality settings as in §2 using the Monsoon power monitor. In offline processing, the video decoding and displaying function power draw is modeled as a piecewise linear function,[4] $P_{vid} = P(b, res, fps)$, where b, res and fps represent the screen brightness level, video resolution and video frame rate, respectively.

[3] We did not use possibly more accurate, content-aware OLED power models [25, 28] as OLED display only draws a small amount of energy (§2).

[4] The hardware decoder power draw behaves as a step function of the resolution and FPS.

Algorithm 3: Smoothing in Energy-aware ABR

Input : Selected format FS_{k-1} for interval $k-1$;
Format F_k for next interval k selected by reactive or proactive energy-aware ABR;
Output: Final format FS_k for next interval k

if $F_k > FS_{k-1}$ **then** $FS_k = FS_{k-1} + 1$;
else $FS_k = F_k$;

For the network transmission function, we develop a linear model that models the power draw as a function of the downlink throughput and chunk size. We experimentally found that running network microbenchmarks does not capture well the share of CPU usage due to downloading when the whole streaming app is running. Instead, we directly stream 360^o videos over the Internet, while logging the inputs to both function-wise prediction models, *i.e.,* application events for each 2-second video chunk, including three video properties (resolution, frame rate, and size), start and end time of network transmission, and start and end time of decoding and displaying. We measure the total phone power draw using the power monitor, and then subtract from it the power consumed by the video decoding and displaying function (estimated using its power prediction model built above) as the power for the network transmission function. Finally, the network transmission function power draw is modeled as $P_{net} = P_{\triangle,net} \times \gamma + P_{base,net}$, where γ is the throughput of fetching the video chunk and $P_{\triangle,net}$ and $P_{base,net}$ are derived from linear regression. [5]

8 Adaptation Smoothing

The baseline reactive adaptation Algorithm 1 can result in significant oscillation in selecting the next chunk format, *e.g.,* under high network bandwidth which allows some high format that exceeds the power budget, followed by switching to some low format to compensate for temporary energy deficit (see Figure 3). A proactive adaptation algorithm like Algorithm 2 will not pick arbitrarily high format due to the fixed per-interval energy budget, but exploiting energy surplus from past low-bandwidth intervals can also result in the controller picking some high format transiently since it can only look ahead N chunks. In both cases, the sudden change in format can reduce the smoothness component of the QoE over time (*e.g.,* beyond the N chunk horizon).

To mitigate this potential effect, we propose a smoothing step that can be applied to both reactive and proactive algorithms: it imposes an incremental increase when the chosen format for the next chunk is much higher than the previous one, as shown in Algorithm 3. Note we cannot impose an incremental decrease when newly chosen format is much lower than the previous one, since such a choice is limited by the network bandwidth.

[5] Network conditions, *e.g.,* signal strength, are reflected in throughput which is one of the model predictors.

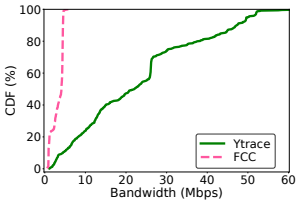


Figure 6: Distribution of network throughput over the traces of two datasets.

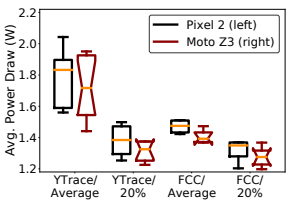


Figure 7: Distribution of average and 20th-percentile per-interval power draw over the traces of two datasets.

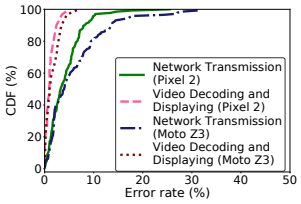


Figure 8: CDF of training accuracy of function-wise power modeling.

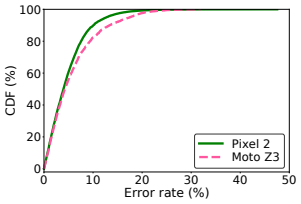
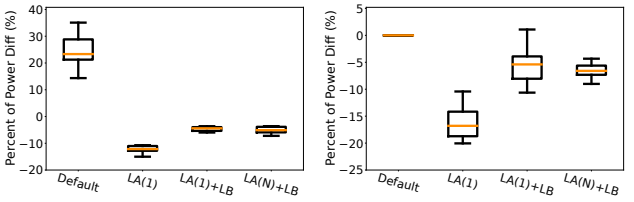


Figure 9: CDF of actual prediction accuracy on Pixel 2 and Moto Z3.



(a) Low power budget (b) High power budget

Figure 10: Percentage difference between the average power consumption of each streaming session and corresponding power budget for proactive approaches on Pixel 2.

9 Implementation

We implemented the Energy-aware ABR server on top of Puffer, an open-sourced platform for video streaming [11] in about 1800 lines of C++ code, and built a simple Energy-aware ABR client that enables 360^o video streaming on top of Exoplayer [3] by adding 1.2 KLOC in Java. To save energy on mobile client devices, we implemented both reactive and proactive Energy-aware ABR on the server side (following [11]). For convenience, we used function-wise power prediction to implement the energy profiler module (using actual throughput) in both types of adaptation schemes and the power predictor module (using predicted throughput) in proactive schemes.

In proactive schemes, the Energy-aware ABR client checks its buffer occupancy every 0.25 seconds. If it is below the buffer threshold, it reports the current buffer occupancy back to the server. The server then runs the Energy-aware ABR algorithm to predict the format for the next video chunk, and transmits the video chunk with the selected format to the client. We choose the buffer threshold of 7 seconds based on the observation (using the Youtube built-in tool *stats-for-nerds* [14]) that the Youtube mobile app on the Pixel 2 phone, when streaming 4K 360^o videos, requests for the next video chunk when the client-side buffer size is below 7 seconds.

10 Evaluation

In this section, we evaluate the end-to-end performance of reactive and proactive energy-aware ABR streaming players for 360^o videos. Our evaluation seeks to answer the following questions: (1) How effective is incorporating energy surplus/deficit in proactive app adaptation? (2) How much does proactive energy-aware adaptatin mitigate oscillation and improve QoE compared to reactive approaches? (3) How effective is adaptation smoothing?

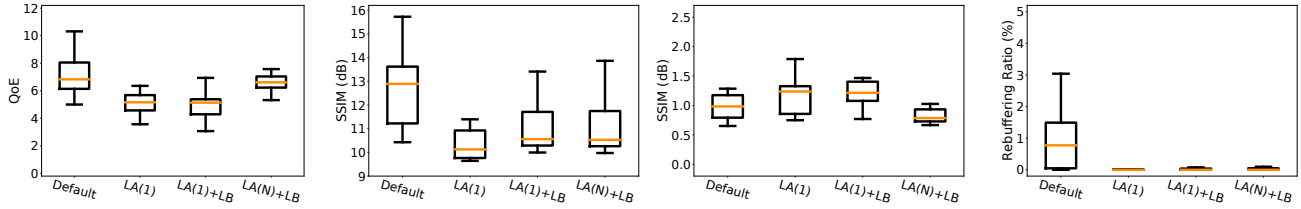
10.1 Experimental Setup

We evaluate Energy-aware ABR performance by streaming videos from a media server to a mobile client, while varying the network condition using network traces from two datasets, YTrace and FCC. We collected *YTrace* by logging the 2-second average throughput of real users watching 360^o videos on Youtube on mobile devices over around 43200 seconds. *FCC* [4] is a broadband dataset that has been used in many recent ABR work [15, 59]. Figure 6 shows the distribution of the network throughput of the traces in the two datasets; the average throughput across the traces in the two datasets are 22.89 Mbps and 3.24 Mbps, respectively.

To compare the performance of different Energy-aware ABR designs under the same network condition, we use the Linux tc tool to throttle the throughtput along with an 80ms RTT between the server and the client. The video hosting server runs on an Intel i5 2.5GHz processor and runs Ubuntu 18.04. The mobile client streams videos over 802.11n on the Pixel 2/Moto Z3 phones which are connected to a Monsoon power monitor to measure the total phone power draw as the ground truth. Each streaming session lasts for 5 minutes.

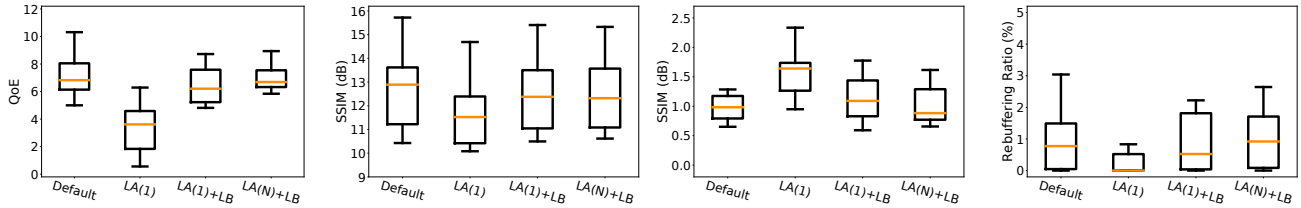
The 360^o videos chosen from Youtube are characterized into the same three groups as in §2. Each video is segmented into 2-second chunks and encoded into the same six quality settings as in §2. We calculate each encoded video chunk's SSIM [77] relative to the canonical source as the quality Q_k of the chunk used in the QoE function (Eqn. 1). As the default QoE function, we use the weights $\lambda = 5, \mu = 20$. We also run sensitivity experiments that vary the QoE weights.

We evaluate various streaming approaches under two average power budgets selected as follows. We first stream the 360^o videos with the default ABR algorithm without any power constraint, and measure the average power draw of each streaming session and the per-interval average power draw within the same streaming session. Then, for each network trace, we select the 20th-percentile per-interval average power draw as its *low power budget* and the average power draw over the streaming session as its *high power budget* in all our experiments. Figure 7 shows the distribution of the



(a) QoE (b) Quality (c) Smoothness (d) Rebuffering

Figure 11: QoE and breakdowns of proactive Energy-aware ABR under the low power budget on Pixel 2.



(a) QoE (b) Quality (c) Smoothness (d) Rebuffering

Figure 12: QoE and breakdowns of proactive Energy-aware ABR under the high power budget on Pixel 2.

two power budgets chosen this way; they vary across the traces of the two data sets. We choose the high power budget in this way to assess the potential penalty of being energy-aware by calculating the performance difference between Energy-aware ABR and the default ABR.

10.2 Accuracy of Function-wise Power Modeling

We first evaluate the training accuracy of function-wise power modeling for the two functions seperately. To train the model for the video decoding and displaying function, we randomly select one video from every video group and play it locally on the Pixel 2 and Moto Z3 phones, respectively. To train the model for the network transmission function, we randomly select 10% network traces from each network dataset and we stream one 360^o video for 1.2 hours in total. Figure 8 shows that the average error rate of per-interval power draw in training for the two functions are 4.21% and 1.16% on Pixel 2 and 6.11% and 1.65% on Moto Z3, respectively.

We next validate our power model by comparing the estimated average per-interval power consumption of 360^o video streaming against the power monitor readings over 50% of the remaining network traces. Figure 9 shows that function-wise power predictor achieves mean estimation accuracy of 4.87% and 5.86% in estimating the per-interval average power consumption on Pixel 2 and Moto Z3 phones, respectively.

10.3 Proactive Energy-aware ABR

We first evaluate the three designs of proactive Energy-aware ABR. We focus on Pixel 2; the results for Moto Z3 are very similar and are omitted due to page limit.

Power consumption. Figure 10a shows that for the low power budget, all three proactive designs consume less power than the power budget. The average power consumption for LA(1)+LB and LA(N)+LB are only 4.09% and 4.80% below the given power budget, respectively, suggesting both designs efficiently exploit the energy saved during past intervals in downloading future video chunks; the small gap to the given power budget comes from discretized chunk formats. In contrast, LA(1) significantly under-utilizes the power budget by 12.20% on average, from not exploiting energy surplus accumulated from low network bandwidth intervals.

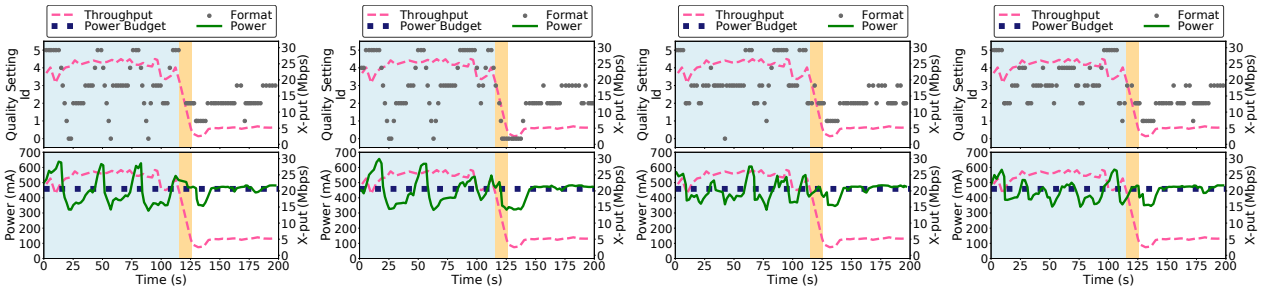
Table 2: Comparison between reactive and proactive Energy-aware ABR without and with smoothing under low power budget on Pixel 2 and Moto Z3 (average/standard deviation).

	RA	RA+S	LA(N)+LB	LA(N)+LB+S
Pixel 2				
Power Diff (%)	-3.50/0.96	-3.40/1.43	-4.80/1.90	-3.78/1.61
QoE	4.02/1.31	4.91/0.39	6.74/0.65	7.11/0.60
Quality (dB)	11.19/1.05	11.33/1.09	11.57/1.23	11.59/1.23
Smoothness (dB)	1.43/0.27	1.27/0.20	0.96/0.30	0.90/0.25
Rebuffering (%)	0.00/0.00	0.12/0.29	0.02/0.05	0.00/0.00
Moto Z3				
Power Diff (%)	0.42/1.53	0.24/1.94	-1.73/1.83	-0.59/1.56
QoE	4.27/1.03	5.36/0.81	5.79/0.97	6.39/0.81
Quality (dB)	11.46/0.52	11.52/0.42	11.50/0.55	11.55/0.52
Smoothness (dB)	1.47/0.28	1.21/0.22	1.13/0.27	1.03/0.18
Rebuffering (%)	0.14/0.32	0.22/0.48	0.13/0.28	0.03/0.06

Figure 10b shows that for the high power budget, the trend is similar; LA(1)+LB and LA(N)+LB only under-utilize the power budget by 5.65% and 6.58%, while LA(1) under-utilizes by 15.39%. The larger gaps compared to the low power budget scenario are because all schemes are juggling among higher chunk formats due to the high power budget, which have larger discretization effects of video encoding.

User experience. We next compare the user experience of the three designs under the two power budgets. Figure 11a shows that for the low power budget, LA(N)+LB achieves the highest average QoE of 6.61, compared with 4.75 for LA(1) and 4.83 for LA(1)+LB.

To understand how different designs affect the QoE for the low power budget, we break down QoE into its three components: video quality, smoothness and rebuffering (Eqn. 1). (1)



(a) RA (b) RA+Smoothing (c) LA(N)+LB (d) LA(N)+LB+Smoothing

Figure 13: Low power budget case study: format selection and power behavior of reactive and proactive designs with and without smoothing under a sample network trace on Pixel 2.

Figure 11b shows that LA(1)+LB and LA(N)+LB have similar video quality, 11.18 dB and 11.14 dB, respectively, both higher than the quality of LA(1) of 10.57 dB. It suggests that exploiting energy surplus saved during past intervals in LA(1)+LB and LA(N)+B improves the video quality of future intervals. (2) Figure 11c shows that LA(N)+LB has the smallest mean quality change of 0.89 dB, compared with 1.15 dB for LA(1) and 1.26 dB for LA(1)+LB. It is the same as that of the default ABR control. It suggests that looking ahead the power consumption of future N intervals effectively smooths the quality switching. (3) Figure 11d shows that all three energy-aware designs have similarly low average rebuffering ratio of around 0.14%, since the low power budget leads to lower chunk formats selected by the Energy-aware ABR controller which reduce the rebuffering time for all designs.

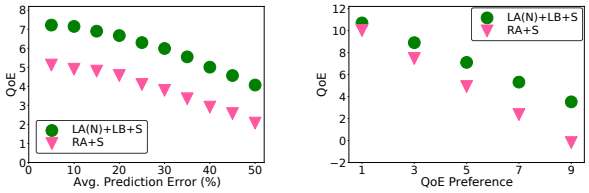
Figure 12 shows that the high power budget scenario has similar user experience results as the low power budget scenario, where LA(N)+LB achieves the highest average QoE of 6.94 and the closest gap with the default ABR of only 4.1%. The slightly low QoE comes from 0.28 dB video quality reduction (2.2%) as shown in Figure 12b. This shows the penalty of proactive energy-aware adaptation compared to the energy-oblivious default ABR is really small.

10.4 Reactive vs. Proactive Energy-aware ABR

We next evaluate the benefits of proactive energy-aware app adaptation by comparing reactive and proactive Energy-aware ABR, with and without adaptation smoothing, under the low power budget, on Pixel 2 and Moto Z3 phones.

Power consumption. Table 2 shows that all four designs satisfy the power budget with a small gap of 4.80%−3.40% below it on Pixel 2 and -1.73%−0.42% on Moto Z3. The small gap can be explained by the small discretization effects among the low chunk formats selected in the low budget scenario.

User experience. Table 2 shows when the power budget is lower than the default app energy power draw, proactive energy-aware app adaptation shows significant benefits over reactive adaptation on both phones. We next elaborate on the results on Pixel 2. (1) Without smoothing, LA(N)+B achieves much higher (67.7%) mean QoE than RA, 6.74 over



(a) Throughput prediction error (b) QoE preferences

Figure 14: Sensitivity analysis under low power budget on Pixel 2.

4.02. As expected, the improvement mainly comes from significantly improved smoothness, 0.96 dB for LA(N)+B and 1.43 dB for RA. (2) Smoothing improves QoE for both reactive and proactive design, by 0.89 and 0.37, respectively, primarily from improved smoothness of 0.16 dB for RA and 0.06 dB for LA(N)+B. (3) With smoothing, LA(N)+B+Smoothing achieves 44.8% higher mean QoE than RA+Smoothing, 7.11 over 4.91. The improvement mainly comes from significantly reduced quality switching (0.37 dB reduction) and significantly lower rebuffering ratio (0.12% reduction) and to a small extent the 0.26 dB higher quality. Table 2 also shows LA(N)+B+Smoothing achieves 19.2% higher mean QoE than RA+Smoothing, 6.39 over 5.36, on Moto Z3.

Case study. Figure 13 shows a case study on Pixel 2 to explain how proactive designs reduce format oscillation compared to reactive designs. The streaming session was under the low power budget, and the network bandwidth went through 3 stages: high (H), high-to-low (H-L), and low (L).

The comparison at the H stage shows that proactive designs can reduce the oscillation when the network bandwidth is high. RA adjusts the format without knowing how much to adapt, which leads to frequent format oscillation to correct energy drain deviation as seen around 20-30s, 50-70s, 80-100s. Instead of aggressively increasing the format when there is no energy deficit, RA+Smoothing increases the format by 1 at each step at around 30-50s and 70-90s. In contrast, LA(N)+B with and without smoothing do not rapidly go up and down all formats and stay in each format longer, from incorporating the power budget in selecting chunk formats. While LA(N)+LB occasionally jumps formats, *e.g.*, to format 5 from format 3 around 60-70s and 105-115s, LA(N)+LB+Smoothing

further improves smoothness by gradually increasing the format when there is energy surplus, *e.g.,* from format 3 to format 4 around 60-70s and 90-100s,

10.5 Sensitivity Analysis

Throughput prediction accuracy. To study the impact of network throughput prediction error on QoE, we evaluate reactive and proactive designs with smoothing under the low power budget by modeling the throughput prediction as a combination of the ground truth throughput with random noise according to the given average error level. Figure 14a shows that the throughput prediction error influences both reactive and proactive approaches, but the gaps remain similar. In parcular, LA(N)+LB+S performs better than RA+S by 41.0%~96.6% for the low power budget on Pixel 2.

User QoE preference. We compare QoE of RA+S and LA(N)+LB+S under 5 different QoE weights for smoothness, {1, 3, 5, 7, 9}, while keeping the weights for quality and rebuffering at 1 and 20, respectively. Figure 14b shows that as users put more penalty weight of smoothness, the difference between QoE of LA(N)+LB+S and RA+S increases from 0.68 (1.07X) to 3.69 (19.4X) for the low power budget on Pixel 2.

11 Discussion

Multiple apps competing for the energy budget. In this paper, we focused on energy-aware adaptation of a single app, *e.g.,* one that dominates the phone energy drain in a four-hour plane ride. In practice, the user may be switching among multiple apps, and there could be unexpected background apps that also drain energy from the total energy budget. In the first case, the integrated proactive app adaptation can be applied to each app while it is running, *e.g.,* constrained by the average power budget for the plane ride. If several apps run concurrently, *e.g.,* some in background and some in foreground, the user can potentially provide input on how the total energy/power budget should be split among the apps. Alternatvely, a global energy-aware controller could be developed, *e.g.,* in the OS, to jointly optimize the QoE of concurrently running apps while satisfying the total energy/power budget.

Leveraging hysteresis in proactive adaptation. Reactive adaptation (*e.g.,* [31]) mitigates the oscillation problem by leveraging hysteresis, *i.e.,* imposing a threshold that the energy surplus/deficit must exceed in order to trigger fidelity adaptation (§6). In contrast, our proactive adaptation design, LA(N)+LB, already alleviates oscillation by allowing the energy surplus/deficit and the N-chunk energy budget to be spread over the next N chunks in maximizing the QoE. Incorporating hysteresis into proactive adaptation designs, *e.g.,* leveraging the energy surplus/deficit (while looking ahead N chunks) only when exceeding some threshold, may result in further smoothness but also lower video quality because of its conservativeness.

12 Related Work

We already discussed previous wok on reactive energy-aware app adaptation in §3. Below, we discuss related work on normal and 360^o video streaming.

Energy measurements and optimization of video streaming. Many works [41, 79, 89] measure the energy consumption of commercial regular video streaming services in WiFi and LTE networks. Recent studies focus on power measurement of 360^o video streaming [48, 87]. Many works [16, 20, 24, 27, 35, 40, 43, 53, 55, 56, 58, 60, 78, 90] propose techniques to minimize energy drain for video streaming via bandwidth control, packet scheduling, screen brightness scaling, *etc.* RnB [84] studies the problem of jointly adapting video bitrate and display brightness to reduce energy consumption while maintaining a quality goal. These works are orthogonal to our work; they focus on energy drain measurement or optimization, while our work focuses on energy-aware adaptation, *i.e.,* how to maximize QoE while satisfying user-configurable power constraint.

360^o video streaming. Many works study supporting 360^o video streaming on head-mounted displays or commodity phones. Several works [18, 50] propose to pre-cache panoramic frames to provide the clients the freedom of changing orientation during playback. Other works [39, 42, 65, 69, 80, 93] exploit different projection and tile-based or viewport-based video encoding schemes to save network bandwidth. They perform network-aware adaptation rather than energy-aware adaptation. We did not evaluate viewport-based 360^o video streaming because commercial video streaming like Youtube does not use it and viewport prediction in the 2-second scale has been shown to be inaccurate; the median longitude error is about 20 degrees [82].

13 Conclusion

In this paper, using 360^o video streaming as a case study, we showed that proactive energy-aware app adaptation that integrates the user-specified energy drain budget into the QoE optimizer of modern apps can significantly reduce app fidelity oscillation and improve the app QoE over traditional reactive energy-aware app adaptation. We believe that proactive energy-aware app adaptation is rather general and as future work, we will validate its applicability and effectiveness for other power-hungry modern apps with built-in adaptation logic such as the class of mobile apps that adaptively offload computation to edge servers.

Acknowledgement We thank our shepherd Amy Lynn Murphy and the anonymous reviewers for their helpful comments. This work was supported in part by NSF/Intel grant 1719369.

References

[1] Planets. https://www.youtube.com/watch?v=qhLExhpXX0E, 2017.

[2] Monster. https://www.youtube.com/watch?v=a9nV5JvM9Tw, 2018.

[3] Exoplayer. https://exoplayer.dev/, 2019.

[4] Federal communications commission. measuring broadband america. https://www.fcc.gov/general/measuring-broadband-america, 2019.

[5] Orleans. https://www.youtube.com/watch?v=bSV8qc2_qFs, 2019.

[6] Solar. https://www.youtube.com/watch?v=MnmU_NwhTFU, 2019.

[7] Top mobile app development trends in 2020. https://www.smartinsights.com/mobile-marketing/app-marketing/top-mobile-app-development-trends-in-2020-infographic/, 2019.

[8] Battery draining quickly? find out which apps are to blame. https://www.komando.com/smartphones-gadgets/battery-draining-quickly-find-out-which-apps-are-to-blame/698674/, 2020.

[9] Chicago. https://www.youtube.com/watch?v=Gu1D3BnIYZg, 2020.

[10] Optical. https://www.youtube.com/watch?v=x_rN5YUXZi8, 2020.

[11] Puffer. https://puffer.stanford.edu/, 2020.

[12] Techengage. top 10 battery draining apps to avoid 2020. https://techengage.com/top-battery-draining-apps-to-avoid/, 2020.

[13] Techrepublic. the most battery-draining apps of 2020. https://www.techrepublic.com/article/the-most-battery-draining-apps-of-2020/, 2020.

[14] Youtube stats-for-nerds. https://support.google.com/youtube/thread/3284269?hl=en, 2020.

[15] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang. Oboe: auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[16] F. Albiero, J. Vehkaperä, M. Katz, and F. Fitzek. Overall performance assessment of energy-aware cooperative techniques exploiting multiple description and scalable video coding schemes. In *6th Annual Communication Networks and Services Research Conference (CNSR 2008)*, pages 18–24. IEEE, 2008.

[17] A. Bemporad and M. Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[18] M. Berning, T. Yonezawa, T. Riedel, J. Nakazawa, M. Beigl, and H. Tokuda. parnorama: 360 degree interactive video for augmented reality prototyping. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 1471–1474, New York, NY, USA, 2013. ACM.

[19] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[20] N. Chang, I. Choi, and H. Shim. Dls: dynamic backlight luminance scaling of liquid crystal display. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(8):837–846, 2004.

[21] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168, 2015.

[22] X. Chen, N. Ding, A. Jindal, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone energy drain in the wild: Analysis and implications. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):151–164, 2015.

[23] X. Chen, J. Meng, Y. C. Hu, M. Gupta, R. Hasholzner, V. N. Ekambaram, A. Singh, and S. Srikanteswara. A fine-grained event-based modem power model for enabling in-depth modem energy drain analysis. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):1–28, 2017.

[24] L. Cheng, S. Mohapatra, M. El Zarki, N. Dutt, and N. Venkatasubramanian. Quality-based backlight optimization for video playback on handheld devices. *Advances in Multimedia*, 2007, 2007.

[25] P. Dash and Y. C. Hu. How much battery does dark mode save? an accurate oled display power profiler for modern smartphones. In *Proceedings of ACM MobiSys*, 2021.

[26] N. Ding and Y. C. Hu. Gfxdoctor: A holistic graphics energy profiler for mobile devices. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 359–373, 2017.

[27] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 107–122, 2010.

[28] M. Dong, Y.-S. K. Choi, and L. Zhong. Power modeling of graphical user interfaces on oled displays. In *Proceedings of the 46th Annual Design Automation Conference*, pages 652–657. ACM, 2009.

[29] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348, 2011.

[30] A. E. Eshratifar, M. S. Abrishami, and M. Pedram. Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 2019.

[31] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, 1999.

[32] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 2–10. IEEE, 1999.

[33] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *OSDI*, volume 8, pages 323–338, 2008.

[34] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[35] Y. Go, O. C. Kwon, and H. Song. An energy-efficient http adaptive video streaming with networking cost constraint over heterogeneous wireless networks. *IEEE Transactions on Multimedia*, 17(9):1646–1657, 2015.

[36] L. Guo, T. Xu, M. Xu, X. Liu, and F. X. Lin. Power sandbox: Power awareness redefined. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[37] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 68–81. ACM, 2014.

[38] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-degree streaming for smartphones. In *MobiSys*, 2018.

[39] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, pages 482–494, New York, NY, USA, 2018. ACM.

[40] M. A. Hoque, M. Siekkinen, and J. K. Nurminen. Using crowd-sourced viewing statistics to save energy in wireless video streaming. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 377–388, 2013.

[41] M. A. Hoque, M. Siekkinen, J. K. Nurminen, and M. Aalto. Dissecting mobile video services: An energy consumption perspective. In *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 1–11. IEEE, 2013.

[42] M. Hosseini and V. Swaminathan. Adaptive 360 VR video streaming: Divide and conquer! *CoRR*, abs/1609.08729, 2016.

[43] W. Hu and G. Cao. Energy-aware video streaming on smartphones. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1185–1193. IEEE, 2015.

[44] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Mobisys*, 2012.

[45] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[46] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108, 2012.

[47] N. Jiang, Y. Liu, T. Guo, W. Xu, V. Swaminathan, L. Xu, and S. Wei. Qurate: power-efficient mobile immersive video streaming. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 99–111, 2020.

[48] N. Jiang, V. Swaminathan, and S. Wei. Power evaluation of 360 vr video streaming on head mounted display devices. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60, 2017.

[49] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[50] E. Kuzyakov and D. Pio. Next-generation video encoding techniques for 360 video and vr.(2016). `https://code.facebook.com/posts/1126354007399553/nextgeneration-video-encoding-techniques-for-360-video-and-vr`, 2016.

[51] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane. Spinn: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2020.

[52] E. Li, L. Zeng, Z. Zhou, and X. Chen. Edge ai: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[53] X. Li, M. Dong, Z. Ma, and F. C. Fernandes. Greentube: power optimization for mobile videostreaming via dynamic cache management. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 279–288, 2012.

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

[55] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh. Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications. *IEEE Transactions on Computers*, 63(2):335–348, 2012.

[56] J. Liu and L. Zhong. Micro power management of active 802.11 interfaces. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, pages 146–159, 2008.

[57] L. Liu, H. Li, and M. Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[58] Y. Liu, M. Xiao, M. Zhang, X. Li, M. Dong, Z. Ma, Z. Li, and S. Chen. Gocad: Gpu-assisted online content-adaptive display power saving for mobile devices in internet streaming. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1329–1338, 2016.

[59] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[60] S. Mohapatra, R. Cornea, H. Oh, K. Lee, M. Kim, N. Dutt, R. Gupta, A. Nicolau, S. Shukla, and N. Venkatasubramanian. A cross-layer approach for power-performance optimization in distributed mobile systems. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 8–pp. IEEE, 2005.

[61] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang. Qdash: a qoe-aware dash system. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 11–22, 2012.

[62] R. Neugebauer and D. McAuley. Energy is just another resource: Energy accounting and energy pricing in the nemesis os. In *Proceedings Eighth Workshop on Hot Topics in Operating Systems*, pages 67–72. IEEE, 2001.

[63] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 29–42, 2012.

[64] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168, 2011.

[65] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pages 99–114, New York, NY, USA, 2018. ACM.

[66] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1421–1429. IEEE, 2018.

[67] J. B. Rawlings and D. Q. Mayne. *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[68] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, pages 139–152, 2011.

[69] S. Shi, V. Gupta, and R. Jana. Freedom: Fast recovery enhanced vr delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 130–141, New York, NY, USA, 2019. ACM.

[70] A. Shye, B. Scholbrock, and G. Memik. Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 168–178, 2009.

[71] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE multimedia*, 18(4):62–67, 2011.

[72] K. Spiteri, R. Sitaraman, and D. Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2s):1–29, 2019.

[73] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[74] L. Sun, Y. Mao, T. Zong, Y. Liu, and Y. Wang. Flocking-based live streaming of 360-degree video. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 26–37, 2020.

[75] L. Sun, R. K. Sheshadri, W. Zheng, and D. Koutsonikolas. Modeling wifi active power/energy consumption in smartphones. In *2014 IEEE 34th International Conference on Distributed Computing Systems*, pages 41–51. IEEE, 2014.

[76] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285, 2016.

[77] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[78] S. Wei, V. Swaminathan, and M. Xiao. Power efficient mobile video streaming using http/2 server push. In *2015 IEEE 17th International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6. IEEE, 2015.

[79] Y. Xiao, R. S. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *2008 The Second International Conference on Next Generation Mobile Applications, Services, and Technologies*, pages 61–69. IEEE, 2008.

[80] X. Xie and X. Zhang. Poi360: Panoramic mobile video telephony over lte cellular networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, pages 336–349, New York, NY, USA, 2017. ACM.

[81] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 43–55, Lombard, IL, 2013. USENIX.

[82] T. Xu, B. Han, and F. Qian. Analyzing viewport prediction under different vr interactions. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 165–171, 2019.

[83] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein. Learning in situ: a randomized experiment in video streaming. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 495–511, 2020.

[84] Z. Yan and C. W. Chen. Rnb: Rate and brightness adaptation for rate-distortion-energy tradeoff in http adaptive streaming over mobile devices. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 308–319, 2016.

[85] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[86] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha. Appscope: Application energy metering framework for android smartphone using kernel activity monitoring. In *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12)*, pages 387–400, 2012.

[87] C. Yue, S. Sen, B. Wang, Y. Qin, and F. Qian. Energy considerations for abr video streaming to smartphones: measurements, models and insights. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 153–165, 2020.

[88] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. *ACM SIGOPS operating systems review*, 36(5):123–132, 2002.

[89] J. Zhang, G. Fang, C. Peng, M. Guo, S. Wei, and V. Swaminathan. Profiling energy consumption of dash video streaming over 4g lte networks. In *Proceedings of the 8th International Workshop on Mobile Video*, pages 1–6, 2016.

[90] J. Zhang, Z.-J. Wang, Z. Quan, J. Yin, Y. Chen, and M. Guo. Optimizing power consumption of mobile devices for video streaming over 4g lte networks. *Peer-to-Peer Networking and Applications*, 11(5):1101–1114, 2018.

[91] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 105–114, 2010.

[92] Z. Zhao, K. M. Barijough, and A. Gerstlauer. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2348–2359, 2018.

[93] C. Zhou, Z. Li, and Y. Liu. A measurement study of Oculus 360 degree video streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 27–37. ACM, 2017.